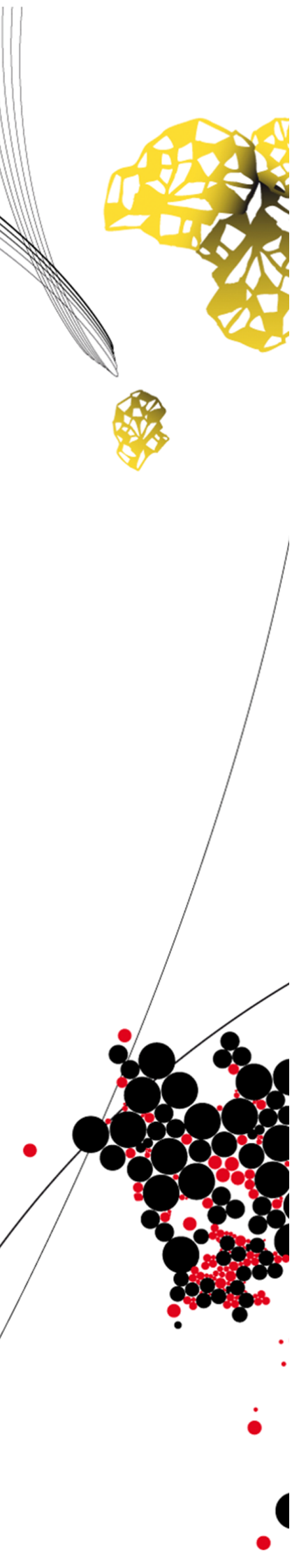


UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Design of a model-driven platform for IoT event stream processing

Mark Alexander de la Court

M.Sc. Final Project
December 2020

Supervisors:
Dr. ir. Marten van Sinderen
Dr. Ansgar Fehnker
Samet Kaya

Formal Methods and Tools Group
Faculty of Computer Science,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Executive Summary

Internet of Things (IoT) is the concept of connecting any device to the internet. Companies are increasingly adopting this concept to allow for new smart business processes and services based on data from connected devices. However, the implementation of IoT is not without its challenges. Transforming raw sensor data into useful information requires high-volume, real-time processing of heterogeneous data streams. Such processing tasks include cleaning, enrichment, generalisation and reduction of the data. The process responsible for transporting and pre-processing IoT sensor data, such that it can be used by applications, is referred to as an integration. Programming, deploying and managing these integrations manually is a time-intensive and difficult process. An integration platform addresses these challenges, by offering a model-driven interface that allows users to develop, deploy and manage integrations on a re-usable infrastructure, without the need to know how integrations are implemented. Integration platforms have proven to be successful in other domains, such as enterprise integration, however the current body of knowledge does not yet describe an architecture for an integration platform in the domain of IoT. This gap in knowledge hinders the development of IoT integration platforms.

To address this gap, this thesis proposes a novel architecture for an IoT integration platform. The requirements for the design are established through a semi-systematic literature review and through interviews with enterprises using IoT, which identify the challenges that organisations face during IoT integration that need to be addressed. To form a solid foundation for the design, existing solutions for IoT stream processing and model-driven integration development are identified using a systematic literature review. The design builds upon these existing solutions to provide an integrated architecture for an IoT integration platform. Key to the design is the model-driven editor, that allows users to visually model integrations, as well as components for deployment, and management of integrations. The integrations built using the editor are supported using dynamic embeddable runtime that allows the execution of scalable, stateful integrations on-premise, in the cloud or on the edge. To support implementation, this thesis also provides a comprehensive description of relevant IoT protocols, architectures, processing frameworks and model-driven interfaces.

The design has successfully been used to develop a prototype of the platform for a Dutch integration platform vendor seeking to implement IoT stream processing support. This prototype is the first model-driven editor for designing embedded stream processing integrations based on Kafka Streams. Early validation results show that end-users can successfully use this prototype to develop integrations, and that they have a high intention to use and adopt this prototype in practice for developing IoT integrations. Additionally, an evaluation with experts on Enterprise Architecture shows that the design provides high architectural quality, and that it can support organisations in developing an IoT integration platform architecture.

The findings presented contribute in several ways to existing IoT integration research and to practice. To academic research, it provides an overview of the current body of knowledge on IoT integration and combines this isolated knowledge into a unified architecture for an IoT integration platform and illustrates how, by applying this architecture, IoT integration challenges can be addressed. To practice, this research provides valuable guidelines for designing and implementing an IoT integration platform.

Contents

Chapter 1

Introduction

1.1 Background

The number of devices connected to the internet is rapidly growing [1], [2]. Smart lamps, thermostats, speakers and TVs are just a few examples of the many devices increasingly used in everyday households in an attempt to make our lives just a little bit easier, for example, by automatically turning the lights off when you leave your house. This ever-growing ecosystem of connected devices forms the Internet of Things, or "IoT". And this ecosystem of devices is not just limited to applications in our homes. In fact, IoT is increasingly used by businesses to optimise their business processes and offer new smart services. For instance, it is used in smart-logistics to track transport vehicles, for asset management to predict maintenance issues, or in hospitals to monitor patients. The enterprise applications of IoT appear to be countless, and this rapidly growing domain of IoT is also referred to as 'Industrial IoT'.

But how do we make use of IoT? Obviously, an IoT sensor or device by itself has no inherent value. A temperature sensor that just measures temperature, and then discards the value is worthless and will provide little value to the user. Therefore, IoT devices are only as valuable as the applications and services they enable. It is only once we connect our temperature sensor to an application that shows us the temperature, that our sensor can truly start delivering value. The IoT ecosystems, combined with the applications they enable, are often conceptualised as Cyber-Physical Systems (CPS). In a CPS devices in the physical world, represented by IoT, are controlled and/or monitored with IT systems and applications [3]. IoT can, through CPS, be used to optimise existing business processes or to unlock completely new value propositions. Concepts such as Industry 4.0, for example, rely on IoT and CPS to power the shift to increasingly digitized and automated daily operations in industries [4].

The implementation of such Cyber-Physical Systems is, however, not without its challenges. Consider our IoT temperature sensor application example; implementing this appears to be simple enough since we just want to get the data from our sensor, and show it in our app. However, as soon as we scale up our app into production we will run into various problems: How do we get thousands of measurements from the IoT devices to our application in real-time? What if these sensors are from a different brands and send completely different measurements? And how do we deal with errors and sudden spikes in the measurements? These are just some of the challenges we will face when implementing our application. Dealing with these 'IoT integration challenges' is one of the major challenges faced by companies when implementing IoT [5]–[9]. Cyber-physical

systems are essentially formed through tight integration of IoT data and applications. Integrating this data is complex since it requires the transport and processing of high-volumes of data in real-time. Additionally, IoT data is contextual, and frequently requires statefull processing (such as discarding outliers), enrichment and other resource intensive processing operations before it can be used. Since companies often have multiple applications with which IoT has to be integrated, integrations are also distributed throughout the enterprise. Adding to this complexity is that IoT is a volatile domain such that integrations are prone to changes [5].

Existing research does not yet provide a holistic approach for developing IoT integrations and addressing these IoT integration challenges. In practice, this gap in knowledge means that users that want to build IoT applications need to develop point-to-point integrations from the ground up, while manually addressing the integration challenges. This makes the development of IoT integrations a tedious and repetitive task, and the resulting integrations are hard to maintain.

1.2 Research objective

The research objective is to address the aforementioned IoT integration challenges by proposing a design for an IoT integration platform.

Currently, integration platforms assists businesses with integrating data, services and applications by providing a platform for users to manage, govern and model most of their business integrations. For instance, the platform assists businesses in retrieving data from one system, then transforming it to a specific data format, and then sending it to another system. These integrations can be managed and developed using model-driven or low-code approaches such that no to little programming skills are needed to design integrations. Model-driven approaches are the most common, and allow integration development through a graphical drag-and-drop interface. Through this approach, integration platforms are known to contribute to significant reductions in the complexity and costs of integrations [10], [11]. Enterprise integration platforms currently described in literature provide enterprise integration services using messaging-based event processors. With event processing, business data events are processed one-by-one which allows for complex routing and transformations. However, this technology is unsuitable for IoT data processing, which requires high-throughput, low-latency, statefull processing of streams, rather than just individual events.

To the best of our knowledge, no research into the design of an integration platform with stream processing support has been conducted. As a result, any organisation seeking to adopt an integration platform for IoT faces multiple design challenges with regard to what requirements the platform should have, what the architecture should be, and how the platform should be integrated and developed. This research addresses this gap in knowledge by proposing the design of a model-driven integration platform for IoT stream processing.

1.3 Research questions

The goals of this research are described using the design problem template proposed by Wieringa et al. [12]. This template describes the problem context, the artifact to be designed, the requirement and the goal for the research to aid the successful execution of the research such that the goals and requirements of stakeholders can be achieved. The template to define the research problem in the

form of a question is defined by Wieringa as follows:

How to *<(re)design an artifact>*
that *satisfies <requirements>*
so that *<stakeholder goals can be achieved>*
in *<problem context>*?

Applying the template to this research results in the following main research problem:

How to *design a model-driven integration platform*
that *allows the development and management of stream processing integrations*
so that *developers can preprocess data streams more efficiently*
in *IoT integration*?

To address this problem, this research first provides an overview of the context, and the challenges that are faced during IoT integration. Based on this, the requirements for the platform are defined. Next, existing solutions are researched, specifically frameworks for stream processing and platforms for graphical programming of event (stream) processing applications. Then, a new integrated design is proposed based on the existing solutions. And finally, the proposed design is evaluated by end-users and experts to evaluate the effects. Consequently, the following sub-questions can be identified:

- Research Question 1: What are the challenges that are faced by organisations during IoT integration?
- Research Question 2: What is the minimal set of requirements for a graphical model-driven programming platform to allow the effortless integration of event-streams?
- Research Question 3: What are the existing designs for stream processing, and graphical programming platforms for stream processing?
- Research Question 4: What combination of the alternative solutions would form the most optimal design for a model-driven programming platform for distributed event stream processing?
- Research Question 5: To what extent does the proposed design combination contribute to stakeholder goals?

1.4 Research methodology

This research is structured per the Design Science Methodology (DSM) described by Wieringa et al. [12]. This is an outcome oriented research paradigm that aligns with the research objective, since an artifact is developed to address a problem in the real world. The DSM is a proven methodology for outcome oriented research in information systems, providing formal and structured processes that can be used to execute the research. The DSM was used in this thesis to properly structure and guide the research to maximise the value and validity of research outcomes.

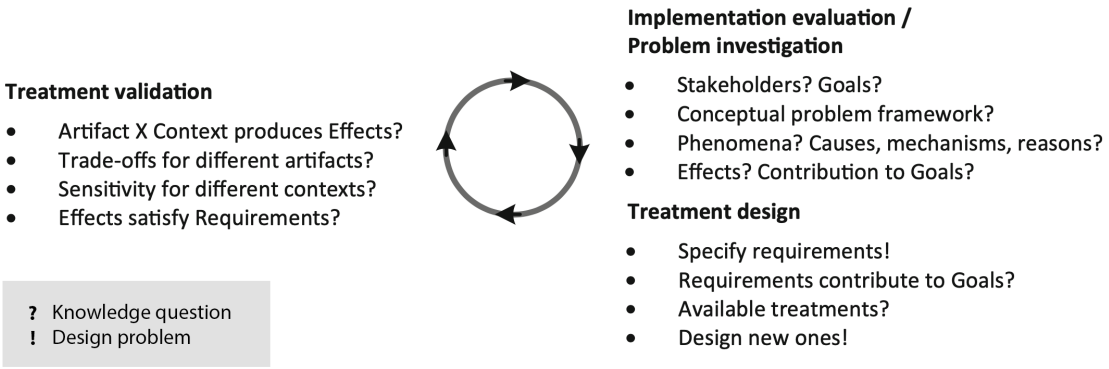


Figure 1.1: The design cycle of the Design Science Methodology, adopted from [12]

According to Wieringa, the design problem is split into two activities. The first activity is the design of the artifact in context, which involves addressing real world design problems. The other is the investigation of the artifact in the context, which involves the answering of knowledge questions. The first activity, the design of the artifact, is guided by the design cycle defined by Wieringa. The phases of the design cycle are depicted in Figure 1.1 and discussed in more detail below.

- **Problem investigation** The first phase is the problem investigation. The goal of this phase is to understand the problem before design is started and before the requirements have been established. In this research, the problem and its context are thoroughly investigated from both a theoretical as well as a practical perspective, using a literature study and interviews with problem owners. At the end of this phase the stakeholders and the design goals are identified.
- **Treatment design** During this phase, the requirements are identified and the artifact is designed. In this research, the requirements for the design are defined in collaboration with the stakeholders, and it is ensured that the requirements align with the problem and goal definitions from the previous phase. Next, a literature review is conducted to review existing and related solution designs. Finally, a novel solution design is proposed based on a combination of the existing designs to satisfy the design requirements.
- **Treatment validation** In this phase, it is demonstrated that the proposed framework can contribute to the stakeholder goals in the problem context. In this research, both a single-case mechanism experiment as well as expert opinion are used as validation models. In the single-case mechanism experiment the design is validated through the development of a prototype of the design, which is then validated with end-users using test scenario's. This prototype allows end users to interact with the design as if it were implemented in the problem context to confirm whether the design properly addresses end-user goals. In addition to validation with end-users, the expert opinion validation method is used to validate the design towards architectural goals. During the expert opinion validation, architecture experts imagine how the artifact would interact in the problem context [12]. Based on this interaction, the experts provide feedback on the design of the artifact.

Figure 1.2 shows which research questions, and other key activities, are applied in each phase in the design cycle. During the execution of the design cycle, knowledge questions are faced. For instance, questions about the problem context, or about the effects of the artifact in the problem context. Such knowledge questions are researched using a separate methodology, such as a literature study, an interview or experiment. Table 1.1 maps all the methodologies used in this thesis to the section that

Methodology	Description	Phase
DSRM	1.4	Throughout
Semi-systematic literature review	2.1.1	Problem investigation
Semi-structured interviews	2.2.1	Problem investigation
Systematic literature review	4.1	Design
TOGAF	5.1	Design
SCRUM	6.1	Validation
Single-case mechanism experiments	7.1	Validation
Expert opinion	7.1	Validation

Table 1.1: Overview of methodologies used

describes them in detail. The methodologies for stakeholder, goal and requirements analysis are not included as they are covered by the DSRM.

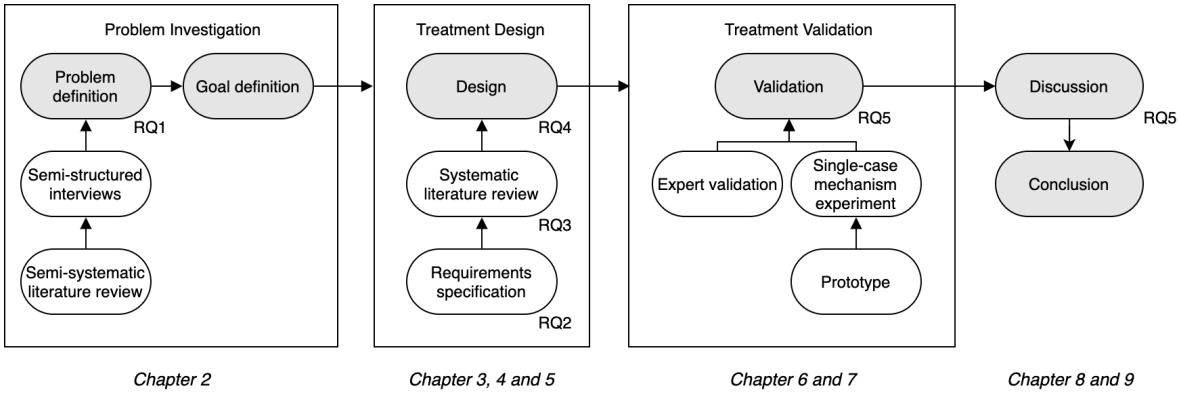


Figure 1.2: The design cycle applied to this research

1.5 Validation case

The research design is validated in the context of integration platform vendor "eMagiz". This vendor provides an integration platform as a service (iPaaS) to its customers to facilitate enterprise integration. In addition to serving customers directly, the vendor also provides consultancy services through an affiliated consultancy firm. This consultancy firm develops custom build applications as well as integrations using the vendors enterprise iPaaS, according to the needs of their clients.

The vendor observes that its customers are increasingly using IoT and is seeking to add IoT integration features to its platform to meet this demand. However, the vendor foresees significant research efforts to realise this extension, and would consequently benefit from the results of this thesis to accelerate the development of IoT support on the platform.

Therefore, this vendor provides an ideal problem context to validate the design. The validation will confirm whether the proposed design can accommodate the vendor in supporting IoT integrations. To this end, a prototype is developed as an instantiation of the design in the problem context of the vendor. This prototype is then validated with end-users to ensure it produces the intended effects. Additionally, the design itself is validated with the CTO of the integration platform to evaluate the anticipated effects of the architecture on the problem context. Additionally, to ensure external

validity, the design is also validated with independent experts from the University of Twente. Validation is discussed in more detail in Chapter 7.

1.6 Document structure

This thesis describes the research through several chapters. Chapter 2 introduces the problem context, presents the conceptual problem framework and overviews the IoT integration challenges. Based on these findings, Chapter 3 presents the requirements for the design to address the problem. Chapter 4 surveys existing solutions and building on these findings, Chapter 5 presents the solution design. Chapter 6 and Chapter 7 respectively validate the design through the development of a prototype, and through expert and user-based validations. Next, Chapter 8 relates the design to alternative solutions to the challenges initially identified and to the stakeholder goals. Finally, 9 presents the conclusions to the research questions and overviews the implications and limitations of the design.

Chapter 2

Problem Investigation

In order to design a treatment it is key that the problem to be treated is first understood [12]. Therefore, a comprehensive understanding of the problem and its context is needed in order to justify the design and to validate it before it is implemented. This chapter provides such an overview of the problem context, to answer research question 1. The chapter first overviews the problem through an extensive literature review, as well as through interviews with problem owners. Finally, this chapter provides a summary of the problem context and an overview of the challenges that are faced during IoT integration, based on the literature review and the interviews.

2.1 Literature Review

2.1.1 Methodology

This review provides the background for the design of the model-driven integration platform through the definition of the problem statement. To this end, several topics are addressed. First, background is provided on IoT applications and challenges. Second, current IoT architectures are described. This includes a description of the components and characteristics of IoT architectures, based on a review of architectures as well as actual IoT platforms, to understand how IoT ecosystems work. Third, an overview of (IoT) integration protocols and semantics is provided, to describe how applications and devices can communicate with the IoT middleware. Fourth, current IoT integration practices are described to obtain an understanding of how IoT is currently integrated with applications, and the challenges faced during integration. And finally, integration platforms and enterprise integration in general are described. These topics are addressed using the following research questions:

1. What are the application domains and challenges of IoT?
2. What are common reference architectures and instantiations for IoT?
3. What are the IoT communication protocols and semantics?
4. How is IoT currently integrated with applications?
5. How does an integration platform function?

Based on the answers from these questions, the problem statement as well as objectives for a solution are derived.

A structured, semi-systematic review approach is used for this review. Snyder et al. argue that a

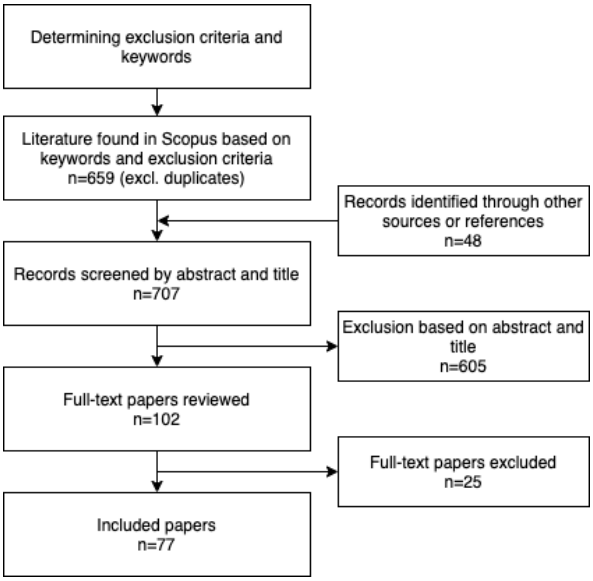


Figure 2.1: Literature review approach

semi-systematic approach is the most suitable approach for describing the state of knowledge and overviewing a research area [13]. Since this research does not focus on a specific domain where IoT is applied but rather aims to describe the state of knowledge across domains, providing a full review of all available literature is not feasible due to the high volume of publications [14].

The methodology for the literature review is adapted from the systematic approach proposed by Kitchenham et al. [15] and shown in figure 2.1. For this research, several systematic steps are excluded including the exhaustive search for all literature, exhaustive back and forward reference checking and the use of a formal data extraction protocol. Instead, generic process steps as proposed Snyder et al. are used for data extraction and record screening [13]. First, the keywords are defined for each of the research questions. Next, these keywords are entered into Scopus, which is provides the largest abstract database of (peer-reviewed) literature, and the exclusion criteria below are supplied as parameters.

- The paper must be written in English.
- The paper must be published in a scientific journal, magazine, or conference proceedings.
- The paper must be accessible. That is, it must be freely available, or available through the University of Twente library catalogue.
- The paper must be published within the domain of computer science.

For each query, only the first 50 results, sorted by relevance, are considered, this is because further refinement of the search query is in most cases unfeasible or not preferable. All the results are then added to a Scopus list. This is repeated for each query, such that all the found papers can be tracked. Papers are excluded if they were duplicates.

Next, all the found papers are reviewed based on their title and abstract. The following exclusion criteria were used for this:

- The study title and/or abstract must be relevant to the research question.

- The study must not be domain-specific (except for literature on IoT applications). For instance, an IoT architecture survey that is limited to the domain of health would be excluded.

All papers that are irrelevant based on the title and/or abstract are discarded, and the remaining papers are read. When no relevant papers are found for a query, Google Scholar is used instead of Scopus. Google Scholar was used exclusively in this specific scenario, as it provides more results but generally of lower quality.

During full-text reading, papers were excluded if they did not actually fulfil the inclusion criteria, or if they did not make any contributions compared to other included papers. If additional relevant literature arose during the examination of a paper, for instance, referenced literature, then this was added to the list of papers to be examined. Subsection 2.1.2 summarises the results of the literature review.

2.1.1.1 Keywords

What are the application domains and challenges of IoT?

```
"Internet of Things" OR IoT challenges
"Internet of Things" OR IoT applications
"Internet of Things" OR IoT survey
```

The third query has been added since survey papers typically include an overview of the challenges and applications of the surveyed technology. Because the queries are very generic, the results are limited to papers published since 2015, to limit the number of results.

What are common reference architectures and instantiations for IoT?

```
"Internet of Things" OR IoT architecture OR architectures
"Internet of Things" OR IoT architecture OR architectures survey
"Internet of Things" OR IoT middleware OR platform
"Internet of Things" OR IoT middleware OR platform survey
```

What are the IoT communication protocols and semantics?

```
"Internet of Things" OR IoT protocols
"Internet of Things" OR IoT semantics
```

How is IoT currently integrated with applications?

```
"Internet of Things" OR IoT integration
"Internet of Things" OR IoT Cloud OR (Distributed Application) Integration
"Internet of Things" OR IoT iPaas OR "Integration Platform"
"Internet of Things" OR IoT Enterprise
"Internet of Things" OR IoT Mining
```

The first three queries yield little relevant results, therefore, the fourth query has been included in an attempt to find overview and survey papers that may reflect on IoT application integration, and/or address the lack of research in this area. The last query has been added to research the gap between the IoT and applications.

How does an integration platform function?

```
"Integration Platform"
"Integration Platform as a Service" OR iPaaS
```

```
"Cloud-based integration platform"
Integration OR ESB Cloud
Enterprise Integration Patterns
```

In literature, integration platforms as a service are also commonly referred to as cloud-based integration platforms or integration clouds or simply as integration platforms. 'Enterprise Integration Pattern' is included as a keyword, since these patterns provide background on application integration. These patterns are independent of technology and remain to be very important for digital integration challenges, also for IoT [16].

2.1.2 Results

2.1.2.1 IoT Application Domains

Numerous papers provide a survey of IoT applications, including for example [7], [9], [17]–[20]. However, most of the studies do not provide an analytical assessment of IoT application domains, are dated, and do not present any clear selection or methodology for their results. Asghari et al. confirmed this, and have recently (2019) conducted a systematic review on IoT applications [20]. They identified healthcare, environmental, smart city, commercial and industrial applications as the most common application domains. Most of the surveys tend to agree with the domains identified by Asghari et al. For comparison, another IoT taxonomy proposed Sethi et al. [18] agrees on most application domains, however, Sethi identifies smart transport and entertainment as additional top-level application domains of IoT, while Ashari groups these under other domains.

Overall, IoT has countless of use-cases over a wide number of domains, and across these domains, IoT can be used to create value by combining digital and physical components [6]. Nonetheless, the requirements for IoT may strongly differ per domain. For example, in connected cars, reliability and response times are crucial while for smart farming reliability of individual sensors and response times may be lower. Razzaque et al. have identified the properties across IoT applications [21]. They found that diversity of the applications, the classification of real-time to non-real time, the service like nature of applications and the need for privacy and security of applications are the most significant properties. Since the applications are diverse and complex, they stress the need to abstract applications from the IoT devices, to ensure developers can focus on the task at hand rather than focusing on generic tasks related to the IoT infrastructure.

2.1.2.2 IoT challenges

Many challenges with regard to IoT can be identified, from architectural to technical and business-level challenges. From an architectural perspective, Alreshidi et al. provide a survey of the most common challenges and how to address them [22]. These architectural challenges represent the high-level design challenges involved with IoT. Based on a survey of 88 studies, they have established several research themes on architecting IoT based systems, most significantly cloud-based IoT, security and privacy, software-defined networking, agent-based systems, big data, autonomy and adaptivity. These patterns are discussed in more detail in Section 2.1.2.3 on IoT architecture.

Xu et al. provide an overview of challenges on a technical level [7]. Heterogeneity and integration are highlighted as key technical challenges to overcome. Distributed, heterogeneous devices need to be integrated, which poses technical challenges regarding connectivity, protocol integration,

device management and data abstraction [7], [23]. The integration of IoT into existing architectures poses a challenge on multiple levels. IoT often needs to be integrated with traditional data and legacy systems [6], [24]. Yet, these large amounts of data may not have much meaning unless it is analysed and understood. Analysing and mining information from this data is a technically complex task requiring proper infrastructure and strong data analytics skills. Security and privacy also pose challenges during implementation. Due to the diverse and low power nature of IoT devices, encryption is complex compared to other domains, while rather private information may be collected by IoT devices. Furthermore, challenges that apply to most IT systems, such as availability, performance, reliability and scalability also apply to the domain of IoT and should not be forgotten [25].

The challenges of IoT are not limited to technical ones. Wortmann and Flüchter stress the need to address strategic and operational challenges related to IoT [6]. Businesses need to evaluate the threats and opportunities of IoT and they need to adapt their business models and processes accordingly. At an operational level, IoT needs to be considered in product development, application development, marketing and support as IoT will impact how products are developed, supported and marketed. For example, the development of a connected washing machine will require hardware and software developers to collaborate. Support and maintenance systems need to be adjusted as well, for example, to support predictive maintenance. Corporate IT infrastructures need to be adapted to be able to support the seamless connection of resources, such as IoT devices, within and between organisations.

2.1.2.3 IoT architectures

This subsection discusses common patterns and components among IoT architectures. These IoT (reference) architectures act as guidelines for implementing IoT systems.

Layers In their survey of IoT references architectures, Moghaddam et al. concluded that most IoT architectures can be classified as layered architectures [26]. In a layered architecture, the architecture is represented by several layers with different responsibilities, the number of layers can vary depending on the architecture. The most common is the 4 layer architecture, which describes the presence of a sensor layer, a network layer, a processing layer and an application layer.
More layers have been introduced, however most these layers have not been widely adopted and the general consensus remains to be the four-layer architecture [6], [28]–[30]. In a survey of IoT reference architectures and platforms, Guth et al. confirm these four layers as the common demeanour between the reviewed middlewares [27] as visualised in Figure 2.2. An overview of these for layers is listed below.

1. **Sensor layer**: This layer is also known as the perception layer. This layer is responsible for translating physical information into data streams, or vice versa. There are many types of sensors, or 'IoT devices', and the requirements of these sensors depend on the applications [31]. Examples of sensors include location sensors, motion sensors, or information sensors embedded in traditional hardware such as industrial machines to sense the machine's state. Typically, sensors have little to no computation power, and limited internet access.

2. **Network layer**: This layer is responsible for connecting the IoT devices in the sensor layer with upper layers [32]. This network layer often consists of local gateways, that can connect with IoT devices over various network technologies such as Bluetooth, Zigbee, cellular or LAN.

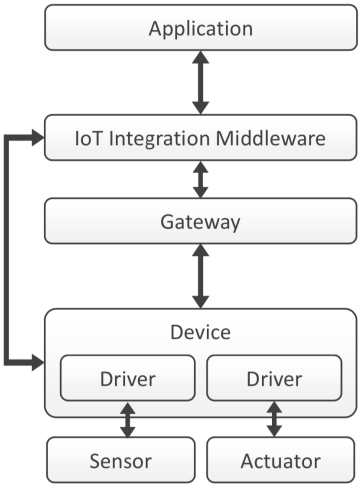


Figure 2.2: IoT reference architecture [27]

Incoming data is then converted from the incoming protocol and forwarded to the internet (TCP/IP protocol). In addition to forwarding, gateways may also perform basic operations, such as aggregation, translation and routing of data streams, as is discussed in Paragraph 2.1.2.3.

3. **Processing layer**: This is the middleware layer, responsible for the aggregation of IoT data. Features like device management and data forwarding are core functionalities of this layer. Additionally, the middleware may abstract incoming data, by translating heterogeneous incoming events into more generic events. Moreover, this layer may decide to perform actions based on these events, for example, it may decide to store the event, to destroy the event, or to generate new events. The middleware layer often exposes APIs such that applications can consume events and data produced by the middleware. Today, in many modern implementations, this layer is an online IoT platform, such as Amazon IoT, Google IoT, etc [26], [33], [34]. However, the middleware can also be located on premise or hybrid in the cloud and on-premise.

4. **Application layer**: The applications, data lakes, and other operations that run on IoT. These applications can vary from simple analytics dashboards overviewing all IoT events, to complex applications that mine information from IoT events and combine it with other information sources.

However, not all architectures can be classified as layered architectures, Ara et al. state that while layered architectures are the norm, other architectures are possible [19]. For example, one layer architectures where devices connect with each other without connecting to the cloud or a central network. Or a two-layer architecture where devices connect directly to applications in the cloud. For the current research, the focus, however, remains on the architectures that provide some sort of middleware layer that applications need to integrate with, as is the case for most IoT architectures. In other cases, such as described by Ara et al., IoT devices may connect directly to applications or the applications may be embedded in the middleware, and an integration platform may not, or can not, provide additional value for integrations.

Components Several functional components of IoT middlewares can be identified. These components give an insight into what functionalities and responsibilities the IoT middleware has. An overview of the most significant components is given below.

- **Device Management** [19], [33], [35]: The device manager maintains a connection with the IoT devices. Additionally the device manager may store device metadata in the database and set device configurations.
- **Message Broker** [19], [33]: The message broker, or event bus, is responsible for event distribution. For example, on the middleware, all gateways can publish to the message broker, and then the middleware can process these events. All processed events will then get published to all subscribed parties, such as external apps or other components.
- **Authentication** [19], [33], [35], [36]: This component covers security and access management. Security is a major concern in IoT, the authentication component ensures that devices and users (i.e. service consumers) are properly authenticated and have the correct right to access the data or configure the middleware.
- **(Metadata) storage** [33], [35], [37]: Here, sensor data can be stored as well as metadata about the device. For example the type of device, the unit of measurement, the location and the relation to other devices. This metadata can be used to enrich and standardize raw data points sent by devices to more contextual data points. Large cloud-based IoT platforms store more than only metadata. For example, they may store all events or the so-called 'Device Shadow'. The device shadow stores the last known state of a device, for example, the last temperature measured by a temperature sensor.
- **Rule Engine** [19], [33], [35]–[37]: Rule engines allow for automated decisions based on IoT events. For example, rules may describe threshold values for events to be stored or forwarded. For some middlewares, Rule engine's are complemented by machine learning algorithms.
- **Abstraction & Semantics** [19], [35]–[37]: Semantics represents the (contextual) knowledge or meaning of IoT data understanding of IoT data to facilitate interoperability, and are discussed in detail in Section 2.1.2.6. The components responsible for semantics or object abstraction can be included at the middleware level, but they can also be deployed at the network or device level, or both [37]. Some have conceptualised the network level as the object abstraction layer, as gateways provide some abstraction from the device and the protocol [36]. Noura et al. state that roughly one-fourth of all surveyed middleware frameworks provide semantic descriptions of data [38], however Petrakis et al. [33] argue that semantics and ontologies that contribute to a full automated understanding of messages and contexts have not yet been applied in large scale real-world implementations and only in conceptual frameworks within a research context.

In addition to the middleware layer, lower layers also provide components needed in IoT architectures, primarily in the network/gateway layer. Examples include configuration management and protocol translations. Since these lower layers are below the middleware, these layers are not directly involved in the application integration and not covered in this review. For an exhaustive overview of middleware components, one can instead refer to Ara et al. [19].

Distribution Patterns Middlewares can have distribution patterns. An analysis of middlewares shows that the most common designs are centralised, and distributed middlewares [26], [39]:

- **Centralised**: In centralised distribution patterns, there is a central component that represents the processing layer. All IoT gateways and devices, as well as all IoT applications, connect to this central component. The central component can be a cloud platform, a server, or a fog connected to the cloud. The centralised pattern is currently the most common [26], [40]. Mineraud et al. argue that centralised, cloud-based solutions, are especially suitable for large scale deployments of IoT [41].
- **Distributed**: In a distributed approach, the processing layer consists of multiple nodes that can work individually and communicate together to achieve a common goal. Truly distributed IoT middleware designs with no central or main nodes, are however sparse [26]. Weaker forms, such as collaborative networks and connected intranets also feature distributed computation, but there are still some dependencies or compromises. Moghaddam et al. argue that approximately one-fourth of all reference architectures offer some for a distributed design [26]. For implementations, Mineraud et al. [41] found that 5 of the 39 analysed frameworks were decentralised or distributed and Farahzadi et al. report that 7 of the 20 IoT middleware frameworks had support for some form of distribution [39].

Designs Middlewares can also be classified based on their design approaches. The most notable middleware designs are service-based and event-based [21]. Middlewares are not limited to a single design approach, and can also feature multiple design approaches. In a service-oriented architecture (SOA), all core middleware components are service-oriented [26]. Service-oriented designs may even conceptualise devices, gateways and internal components as services, and the functionality of the middleware is obtained through the composition of these services. Service-oriented architectures typically include three elements: First, service providers, which provide the services. Second, service brokers, that manage the services and their locations. And third service consumers, which use the services managed by the broker. Microservice-based architectures extend upon the principle of SOA and promote the decoupling of services, requiring that each service must be able to run independently without the need for service orchestration. Compared to SOA, microservice-based architectures allow for greater scalability. Event-based middlewares are based on a publish/subscribe architectural style. In such a middleware, there are several topics to which publishers can publish and subscribers can subscribe. For example, an IoT device can send a message on a certain topic, and then all subscribers to this topic will receive the message. A middleware can have multiple message brokers. While most IoT architectures are SOA based, IoT solutions are increasingly moving toward event-driven implementations [42].

Cloud Cloud-based architectures utilise the cloud for computation and storage. Moghaddam et al. argue that over fifty percent of all IoT architectures is cloud-based [26]. IoT produces large amounts of data, is distributed, and has limited storage and processing power. The cloud, however, provides a centralised place capable of virtually unlimited storage and computation, that is unprecedented by on-premise alternatives. Therefore, Botta et al. reason that the combination of cloud and IoT will be a disruptive paradigm that will enable many new applications. In the Cloud IoT paradigm devices can connect directly to the Cloud IoT middlewares, or cloud platforms. These cloud platforms allow for managing IoT devices and orchestrating IoT data in the cloud, providing unlimited scaling as needed, for example scaling of the message broker, computation, or storage. [43].

Fog / Edge The move of IoT to the cloud places a large burden on the cloud to process all IoT data, regardless of its relevance, because IoT data is produced at such a high velocity. Therefore, some

propose that IoT data can best be processed as close to the edge as possible [28]. That is, data deletion, processing and filtering can best be done close to the source device collecting the data (the 'edge'), to avoid overloading applications higher in the hierarchy with large volumes of data. Between the cloud and the IoT devices, a fog layer can be added to facilitate this. The fog is a virtual cloud that runs on the gateway, or between the gateway and the cloud [26]. The fog can filter the data to be sent to the cloud and can act upon the data to control devices locally. As IoT middlewares increasingly reside in the cloud, the fog can help to reduce the data flow to the cloud and increase performance and response times through local computation. Edge computation requires even deeper integration then fog integration, as it requires data processing to be integrated as close to the edge as possible, for instance on IoT devices themselves.

2.1.2.4 IoT platforms

Numerous papers provide a survey of IoT middleware [21], [34], [39], [41], [43]–[46]. Table 2.1 provides an overview of popular middlewares discussed in these papers, both open source and commercial, as well as some of their properties. Rather than providing a exhaustive list of all frameworks discussed in the surveys, a representative selection has been made. This includes the most popular open-source platforms identified by [46] et al. and for commercial platforms, the IoT solutions from the three leading[1] cloud providers, as well as the three IoT focused vendors that have the most frequent mentions in literature.

The middlewares in the table have been reviewed on the following properties:

- **API**: The API of the middleware, that is how applications can integrate with the middleware.
- **DM**: Whether or not device management is included in the middleware.
- **Rule**: Whether the platform has a rule engine (logic) that allows users to trigger events or other actions.
- **Sem**: This attribute reflects the abstraction / semantics support of the application.
- **Protocols**: The protocols the platform supports for incoming data from devices and gateways.
- **Shadow**: Whether the middleware maintains a digital twin, or virtual copy, of each device with its last know state.

Most middlewares have similar components and support similar protocols. Bader et al. confirm this, stating that most reference architectures recommend similar practices such as an MQTT broker, HTTP REST APIs and oAuth for authentication [14]. In their survey of over 30 IoT platforms and frameworks, Noura et al. found that all but three platforms offered a REST API, however, the majority of these REST APIs was platform unique and relied on internal information models [38]. In addition to a lack of uniformity in REST APIs, most platforms lack (uniform) event-based APIs. Mineraud et al. [41] conducted a gap analysis of IoT platforms, and confirmed the limitations in data processing and sharing. This lack of uniformity in IoT platforms hinders integration and use of multiple platforms.

All the reviewed non-open source IoT platforms are cloud-based and offered as a service. Most of these platforms, such as Amazon, Google, IBM and Microsoft, provide proprietary cloud services for storage, computation and more. For open-source platforms, such services are not available, and

[1] Leading cloud providers as identified by *Gartner*. Gartner has not yet identified any leaders for IoT platforms.

Middleware	API	DM	Rule	Sem	Protocols	Twin	Literature
LinkSmart (Hydra)	Proprietary	Yes.	Yes	No	MQTT, REST	No	[21], [39], [41], [45]
Thingspeak	REST	No	Yes	Yes	MQTT, REST	No	[40], [41], [44], [46]
OpenIoT	Proprietary	Yes	No	No	CoAP, X-GSN	Yes	[39], [41], [46], [47]
Kura	REST, P/S	Can	Yes	Can	MQTT	Can	[45], [46]
IoTivity (AllJoyn)	REST	Yes	No	Yes	CoAP, REST	Yes	[46]
Google IoT (Xively)	REST	Yes	Yes	No	MQTT, REST	Yes	[21], [34], [39], [41], [45]
Altair (Carriots)	REST	Yes	Yes	No	MQTT, REST	No	[21], [34], [39], [41], [44]
Exosite	REST	Yes	Yes	No	MQTT, REST	Yes	[34], [40], [41]
PTC ThingWorx	REST	Yes	Yes	Yes	MQTT, REST, CoAP	Yes	[34], [39]
AWS IOT	REST	Yes	Yes	Can	MQTT, REST	Can	[34]
Azure IoT	REST	Yes	Yes	Can	MQTT, AMQP, REST	Can	[34]

Table 2.1: IoT Platforms / Middleware

one would need additional platforms for storing and processing large amounts of heterogeneous data [48]. Examples for storage include Apache Hadoop, Druid and others. Examples for message processing include Spark, Storm, Kafka and RabbitMQ. Such platforms often run on IaaS technology to provide on-demand upscaling of resources when needed, OpenStack is an example of this.

All commercial middlewares provide platform functionalities for IoT. However, some middlewares also provide software-as-a-service applications for IoT. These can vary from analytics dashboards to node-based programming tools and full-fledged asset management applications. From all open source solutions, only ThingsSpeak facilitates itself as an application builder. While for the commercial cloud platforms, almost all vendors except Google, offer some SaaS IoT applications.

Most platforms are designed primarily to connect directly to IoT devices and gateways. However, some platforms, like Inter-IOT [47] and ThingsSpeak, support the integration of multiple middle-wares rather than only providing integrations with devices, to promote horizontal interoperability across middlewares.

2.1.2.5 Protocols

Multiple layers of communication can be identified, First, from IoT device to gateway, using protocols and radio technologies like Zigbee, Bluetooth, 802.15.4 and WiFi. Second, from gateway to middleware, most notably HTTP (REST), MQTT, AMQP, XMPP and CoAP. Event-based protocols such as MQTT are preferred, to prevent superfluous polling. MQTT and REST are the most used protocols [49]. And finally, from middleware to applications, most notably REST or proprietary APIs (sometimes event-based) [34].

Dizdarevic et al., Gazis et al. and Al-Fuqaha et al. all survey of protocols for IoT [1], [49], [50]. The protocols from these surveys are discussed below.

- **HTTP REST**: HTTP is the protocol that is also used for the web, providing wide compatibility with most network infrastructures and devices. HTTP guarantees the delivery of data but this may result in overhead in resource-constrained environments such as IoT. HTTP utilises requests and responses, a client can send an HTTP request and the server will send a response message. For web services, HTTP has been closely associated with REST. REST

defines operations such as create, post, put and get, which are mapped upon the respective HTTP operations. REST does not describe a specific data structure or format, however for IoT JSON is the most popular format used to present data.

- **CoAp**: CoAp could be considered as an alternative to HTTP for resource-constrained environments. Like HTTP, it supports the REST architecture. CoAp is lightweight, and has less overhead, but also less reliability, compared to HTTP. In addition to the HTTP like request/response interaction, CoAp allows clients to observe changes, allowing publish/subscribe like behaviour.
- **MQTT**: Similar to CoAp, MQTT is also designed as a lightweight messaging protocol. Because it is very simple and lightweight, it is often the protocol of choice for IoT [49], as is also found in Section 2.1.2.4. It allows for publish/subscribe interactions. MQTT defines three levels for quality of service, varying from no delivery guarantee to the guarantee that the message is delivered exactly once.
- **AMQP**: AMQP is another messaging protocol typically used for more power full IoT devices or gateways. The latest version of AMQP allows both request/response and publish/subscribe based messaging mechanisms. Compared to MQTT and CoAp, AMQP allows also peer-to-peer publish/subscribe messaging with the broker being only optional. Additionally, other paradigms such as broker-to-broker are supported for increased scalability. Similar to MQTT, it provides different service levels to guarantee delivery. Overall, AMQP is one of the heaviest protocols for IoT.
- **XMPP**: This is a messaging protocol initially designed for message exchange between applications. It was designed as a request/response-based protocol, however, publish/subscribe based messaging is possible using extensions. The protocol is quite heavy, however, optimisations are possible through extensions.

2.1.2.6 Semantics

Figure 2.3 provides an overview of the stages of interoperability, as defined by Jara et al [51]. First, interoperability on the network level is needed, such that all devices can connect with each other over the internet.

Next, interoperability requires inter-operable communication over the internet employing protocols such as MQTT, and CoAp described in Section 2.1.2.5. This also includes syntactical interoperability to ensure that a similar format, or 'grammar', is used such as JSON, XML or plain text [38]. Structured data, such as JSON or XML is in practice exchanged according to a schema specification, such as an XML schema or Avro schema. The use of schemas allows users to work with data at the object level. This is also referred to as the Web of Things, and such syntactic inter-operability is focused on vertically integration IoT with applications [51]. According to Jara et al. the first challenge for syntactic interoperability is overcoming the syntactic differences between heterogeneous IoT events [51]. This could for instance be resolved by translating heterogeneous events into a common language through object abstraction and harmonization. This allows users to work with data at the object level without concerns for the technical details such as the device used, or the exact format of the data.

Common access through inter-operable protocols and formats does not imply a common understanding since with syntactic interoperability contextual information about the meaning of the

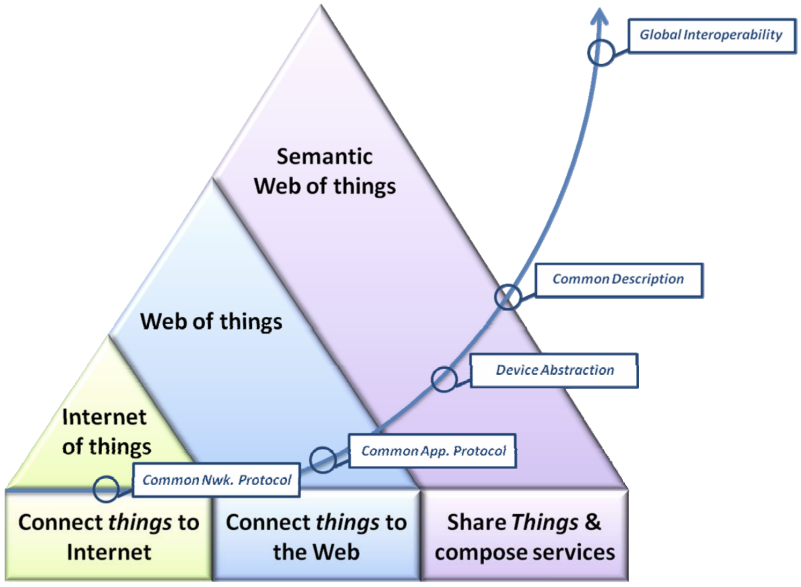


Figure 2.3: From Internet of Things to the Semantic Web Of Things. [51]

data is stored informally. True semantic interoperability, allows for unambiguous interpretation of data and relationships between data points through formal contextual descriptions of data. According to Sheth et al. true semantic interoperability can only be achieved when a formal description, such as an ontology, is provided that allows for a universal understanding of the data and the relations between data [52]. IoT with support for true semantic interoperability is also referred to as the Semantic Web of Things [38], [53].

Semantic knowledge in the Semantic Web of Things can be represented using informal agreements on standards, structures and formats to be used or, typically, using ontologies [38], [52]. Ontologies can act as a vocabulary for semantics, and they consist of objects and relationships which can be used to describe and explain the data of an area of interest. Ontologies are typically defined using a web ontology language such as OWL or RDF [2], OWL is preferred as it is more expressive. When ontologies are used together with hyperlinks to create references between ontologies, this is also referred to as Linked Data [53]. By using the use of hyperlinks, all information stored in ontologies becomes available as a single, global, graph of structured data. The W3C Semantic Sensor Network (SSN) ontology, as proposed by [53], is considered by the authors of the SSN as one of the most adopted IoT ontologies [38]. The SSN is domain-independent and describes sensors, observations, and deployments.

Ultimately, the Semantic Web of Things should lead to full interoperability of heterogeneous data, allowing automatic matchmaking and data exchange between compatible applications (alignment). However, the Semantic Web of Things has significant challenges that hinder adoption:

- Ontologies and (software) support for ontologies have not yet reached maturity. As described in Section 2.1.2.4, most middlewares do not provide semantic interoperability, and if they do, they only provide propriety data models and ontologies that do not fit the requirements of the Semantic Web of Things. Semantic standards, such as SSN are primarily used in research projects, such as OpenIoT [38]. Efforts have been made for the automated derivation of

semantic ontologies from data, and the automated alignment of ontologies [54], however, such tools are lacking and methodologies to use them are scarce [55].

- The majority of IoT ontologies are domain-specific and do not allow for cross-domain integration. As described by Noura et al, global ontology standard do not even exist within domains, and even if such a standard would exist chances industry wide adoption would be unlikely [38]. Others, including Szilagyi et al. and Rahman et al. also confirmed the domain-specific nature of IoT surveys [56], [57].
- Ontologies are complex to use. In a recent survey of the state-of-the art of IoT semantics and inter-operability by Rahman et al. concluded that one of the most challenging aspects of ontologies are the high-complexity and heavy weight of the ontologies [57]. Rahman found that it was extremely difficult to convert raw sensor data into RDF conforming to an ontology such as SSN.
- Ontologies guarantee a common, but limited understanding. According to Venceslau et al. [55], the use of shared ontologies solves issues with interpretation, but it may also lead to the loss of some domain specific information that is not captured by the ontology. Therefore, only some interoperability requirements can be addressed using Semantic Web of Things, due to the limit in which all domain aspects can be formally represented using corresponding ontologies. This issue especially arises with high-level ontologies such as the SSN.

Concluding, ontologies are currently primarily used within research contexts, to demonstrate domain-specific data exchange. Real-world use of cross-domain interoperability using ontologies has many limitations and challenges which are yet to overcome.

2.1.2.7 IoT Integration

Integration at the network level and at the middle-ware level have both been thriving topics in research. Mineraud et al. even state that, while there are of-course always remain challenges to overcome, there is currently an abundance of IoT middleware solutions and platforms [41]. IoT research has, however, put less focus on the final, and crucial, integration step: the integration between the processing layer of IoT and the applications that run on this layer [38].

The first challenge towards this integration is connecting applications with the specific APIs of the IoT middleware/platform. As discussed in Section 2.1.2.3 modern architectures provide a service or event broker, allowing IoT applications to integrate with IoT data. This requires developers to adapt their applications to the specific data models used by each middleware, and since event-based APIs are often not available or proprietary, they may need to write polling and change detection scripts to build event-based integrations. Frequently, integrations may even be device-specific, since the middleware does not provide harmonization of heterogeneous device data, as discussed in Section 2.1.2.4, and applications can then only access raw, device-specific, sensor data. This means that tasks such as harmonisation and enrichment based on device metadata have to be performed on a per-application basis as well. Additional challenges are faced when applications need to integrate with multiple middlewares, or when applications are ported from one middleware to another since each middleware provides its unique APIs and data models. Efforts have been made to integrate multiple IoT middlewares into a single platform by projects such as Inter-IoT [47], which proposes an IoT middleware-to-middleware layer that allows applications to connect with multiple middlewares through a unified service, or the FiWare context broker, which describes a universal API and context that middlewares can adapt [14]. Such solutions are a step forward to harmonize the APIs and data

models of IoT middlewares, however, the solutions do not have high adoptions in mainstream IoT architectures, and even if adopted, a gap between the data models of the applications and the Inter-IoT / FiWare middleware will remain. This is also referred to as the syntactic gap between the IoT data and the applications running on IoT.

A second challenge in the integration process is related to preprocessing. IoT produces raw, unfiltered, event data, applications which needs to be filtered, cleaned and enriched before it can be used by applications [58]. Shen et al. label this as the difference between data and knowledge, or the difference between primitive events and events [59]. Ma et al. label it the difference between primitive events and semantic events [60]. While Lempert et al. define the difference as a gradient from smart object information to business events [61]. Compared to business events, data events are highly contextual and relatively unreliable. A data event by itself contains little information and the interpretation depends on the context, like location and time. Business events, however, are self-contained, more reliable, and more independent of context. IoT data, or 'data events', need to go through a process called mining to form meaningful information as 'business events'. The process of mining varies from prepossessing tasks such as simple filtering, enrichment and transformations, to complex event processing over a window of events based on business rules and machine-learning algorithms [58]. The IoT integration is only responsible for preprocessing the data, for example to clean the IoT data-flow from errors and meaningless or irrelevant events to improve consistency and optimize throughput. Complex event processing is usually not considered as part of the IoT integration itself, but rather as an application of IoT. There is a large body of literature on tools and techniques available for mining [62], [63]. However the ability of IoT platforms to (pre)processs data is limited [41], leaving developers to manually implement such functionality or to integrate the data mining tools or custom processing applications. The difference between primitive IoT data, and preprocessed data or high-level knowledge extracted from this data, is also labelled as the abstraction gap by Janiesch et al [58].

To tackle the challenges listed above, from adapting to middleware specific APIs to preprocessing, developers need to develop middleware-specific point-to-point integrations with their applications [52]. These point-to-point integrations are inefficient, complex and expensive [10], [11]. This is because complex processing and mapping operations of data have to be re-developed for each integration. In their survey of IoT literature, Botta et al. argue the same "While Cloud IoT solutions have been already built around specific applications, little effort has been spent to derive a common methodology to integrate Cloud and IoT systems [..] a generic and flexible platform could be the starting point for implementing such workflows more easily." [43]. He et al. confirm the same [64] , they found that while several solutions have been proposed, that these solutions are typically domain-specific. They also state that there is a lack of standard architecture and guidelines for allowing integrations between IoT and the applications utilising IoT, while increasing re-usability.

Several related papers were found that attempted to resolve the need addressed above. Two papers present a full IoT architecture, Kutzias et al. [23] and Sarkar et al. [65]. Both acknowledge the lack of research on IoT application integration and present an architecture that aims to address IoT application integration concerns. However, rather than relying on existing IoT frameworks as building blocks for IoT application integration, both present new architectures that do not extend upon existing frameworks. This will require developers to completely re-design their IoT infrastructure from the bottom up, which is in most cases unfeasible and has major disadvantages. The authors acknowledge that the implementation of the architecture includes many challenges

related to, for example, protocol support, device management and security [23], while these challenges have already extensively been addressed in existing IoT frameworks. The architecture proposed by Sarkar et al. even requires the deployment of a custom daemon to every IoT device in the ecosystem [65], which would be a major challenge and sometimes even impossible depending on the IoT vendor. Therefore the research of Kutzias et al. and Sarkar et al would primarily be valuable for the long term development of IoT architectures, rather than as a practical architecture for building IoT integrations today using existing building blocks. Schel et al. suggest adding an integration layer on top of existing data service layers [66], this approach could be applied to IoT as well, conceptualising IoT as a data service. However, Schel et al. do not discuss the integration IoT specifically, therefore leaving integration challenges specific to IoT, for example, the abstraction gap, unanswered.

Overall, research does not yet seem to provide an alternative to point-to-point integrations to close the syntactic and abstraction gaps. As a result, there are no guidelines on how to create flexible, maintainable and re-usable integrations, and development of IoT integrations with cloud & legacy applications remains to be a challenge [7]–[9].

2.1.2.8 Enterprise Integration

The use of software and architectural principles by business to integrate enterprise application is referred to as Enterprise Application Integration (EAI). Before EAI, applications existed primarily as large silo's [10] which were connected using point-to-point integrations. However, businesses were limited by these silo's and fixed integrations in their ability to quickly respond to market changes and hence newer, more flexible IT systems were needed. With EAI, concepts such as the enterprise service bus (ESB) were introduced, which allowed more flexible integration of applications through a centralized information broker, employing standard connectors and centrally defined integration logic. To allow for even more flexibility, organisations increasingly adopted service-oriented architectures (SOA) together with ESB. In SOA, business functionalities are exposed as a service, such that they can be reused in multiple applications [67] to accelerate application development. While for some businesses and use cases, point to point integrations are still best suited, business with more demanding integration rely on integration platforms that can provide a full range of integration features from ESB, to features such as API management, and business workflow management.

As businesses moved to the cloud, new challenges arose related to the integration of cloud-based applications. Integration platforms as a service or 'iPaaS' aim to address the integration challenges brought by the integration of cloud-based applications [68]. iPaaS offers EAI in a cloud-based software suite, while also aiming to reduce the complexity that was traditionally involved with EAI by providing services for development, execution and governance of integrations [68]. Cloud-based integration platforms have since then evolved to support the integration of cloud, native and hybrid applications [11]. Section 2.1.2.9 provides more detail on integration platforms. The next generation of integration platforms is described by Gartner as the Hybrid Integration Platform (HIP) [2] yet, the concept of HIP has only been described broadly in literature by Palanimalai [69]. A HIP extends upon the principles of iPaaS, by providing a framework to manage integrations and applications as a whole, including for example IoT integration, API management, governance, and lifecycle management. Additionally, a HIP should support the integration of all kinds of data. iPaaS is

[2] https://www.gartner.com/smarterwithgartner/use-a-hybrid-integration-approach-to-empower-digital-transformation/

focused primarily on processing business events, business applications and business data [70], however information systems integration has to reach beyond classical business enterprise integration to support large scale enterprise information, for example, big data and IoT [68]. Research on iPaaS is, however, sparse and more recent developments have yet to be documented in literature. An academic unified description of what a HIP entails is also missing and further research is needed in this area.

While the technology to implement integrations is rapidly changing, the concepts of enterprise integration remain the same over time. Hohpe and Woolf present a collection of enterprise integration patterns that can be applied for data exchange between companies [71]. These patterns are independent of technologies and implementations. Several strategies for integration are identified:

- **Batch data exchange**: Files are moved in batches from one system to another during nighttime when the systems are not in use. Changes are not immediately available and this approach introduces overhead in the data exchanged.
- **Shared database**: By using a shared database in which multiple applications store data, all applications have to support the same data model, which is often not possible.
- **Raw data exchange**: Two systems can directly exchange binary data over a network in real-time. This approach, requires the recipient to be online, and developers are responsible for supporting and decoding the incoming data.
- **Remote procedure calls**: These methods simplify the point-to-point data exchange by providing a layer for sending complex data types. Examples of this are SOAP and REST.
- **Messaging events**: Messaging decouples the sender from the receiver using a publish/subscribe-like mechanism to asynchronously transfer and transform data records. Messaging between applications is handled by a messaging system that routes the messages to message queues, such that messages can be exchanged even when an application is temporarily offline. While messaging systems have limitations, especially regarding the exchange of very large data streams, it has numerous advantages such as interoperability, decoupling, and reliability.

Several kinds of message patterns have been identified. These patterns can be combined to create a messaging based integration on top of a message system. An exhaustive overview of all patterns can be found in [71]. The patterns are categorised as follows:

- **Endpoint patterns**: These patterns describe how an application connect to a messaging system. Examples are event-based endpoints that push messages, or messaging gateways that translate internal events of an application to messages. Endpoint are usually represented as connectors, and integration platforms often offer built-in connectors from/to specific protocols and technologies [68]. Kritikos et al. state that for example at least SOAP, REST and the Java Message Service (JMS) should be supported [72].
- **Construction patterns**: These patterns describe how messages are constructed. For example, a message may be in a request-response pattern, or as an asynchronous event pattern. Message patterns also describe the data itself, it may contain binary data, it may contain structured documents, or message may even be chucked. Also, metadata such as the return address and the expiration date are described.

- **Channel patterns**: Messages are transmitted through message channels. These channels (or pipes) connect senders to receivers. Channels can be point-to-point, ensuring that only a single receiver receives the message, or for example, publish-subscribe based, which ensures that all subscribers will receive the message. Channels can also be classified based on data types, validity (collecting invalid/dead messages) and delivery guarantees.
- **Routing patterns**: These patterns describe the logic based on which messages are routed towards target endpoints or channels. A message router consumes messages from one channel and appends them to certain other channels based on certain conditions. Other routing patterns include filters, that only pass through certain messages, and splitters and aggregators that can split up and combine messages.
- **Transformation patterns**: These patterns allow for the transformation of messages to different data formats. For example, the message translator can map messages from one format to another. The Canonical Data Model (CDM) pattern can be used with the translator pattern, to translate all incoming messages into a common data format, and then from the common data format to the target data format. In complex integrations, the CDM pattern can avoid an exponential increase in the amount of translators needed.

2.1.2.9 Integration Platforms

Serrano et al. describe an iPaas as "a suite of cloud services that enable users to create, manage and govern integration flows connecting a wide range of applications or data sources without installing or managing any hardware or middleware" [67]. Integration platforms allow users to create these integration flows by providing a managed implementation of the enterprise integration patterns previously discussed. Using these patterns (typically message-based), as well as through pre-built connectors for common data models and sources, the user can specify the integration logic that defines which data is exchanged with which applications and how. The integration platform will provide a user interface that allows users to set up this functionality. Additionally, the integration platform will allow the users to execute and deploy and manage the integrations.

Ebert et al. distinguish two kinds of iPaaS, basic platforms like *Zapier* and *IFTTT* that allow for simple connections between built-in connectors, and iPaaS for large enterprises [10]. Integration platforms for large enterprises allow for more complex integration use cases, including full support for the enterprise integration patterns and on-premise application support. According to Ebert et al., functions of an enterprise iPaaS include:

- Process modelling functionalities to design complex data flows with logic, branches and transformations.
- Multiple messaging mechanisms such as message queues, synchronous and asynchronous messaging support and event/batch-based integrations.
- Both pre-built connectors as well as custom connectors, supporting a wide variety of inputs from file-based to ESB.

Enterprise integration platforms consist of a development platform where integrations are designed, and a runtime where integrations are executed. Both the runtime and the development platform can have different deployment patterns. Some vendors may host both the runtime and the development platform in the cloud, possibly supported by locally deployed connectors, others may choose to only host the development or the runtime in the cloud and deploy the other component locally [10]. The

advantage of cloud-based deployments includes scalability, and maintainability, as the integration platform vendor will take responsibility for managing the infrastructure, updates and other maintenance.

Kleeberg et al. provide a high-level overview of iPaaS systems distinguishing several features [68]:

- **Fundamental integration features**: Consist of core integration features, that provide the infrastructure such as the mappings, connectors, and routing, and additional features related to the development, operation and governance of the integrations.
- **Cloud enabling features**: Support the virtualisation and execution of the fundamental integration features as cloud services
- **Service federation features**: An iPaaS by itself cannot, and for complexity reasons should not, provide all possible functionalities. Service federation reflects the ability of iPaaS to integrate additional models possibly from other integration providers, to provide additional functionality such as analytics and IoT.

While each integration platform will provide different functionalities, integration platforms will generally share certain core components. Singh et al. conducted a literature review over several integration platforms, and provides an overview of common components [73]. The integration platforms surveyed by Singh are domain-specific, not SaaS-based, and developed on a case by case scenario. Therefore, not all components identified by Singh are relevant for iPaaS. First, is the data layer. This layer covers the enterprise integration patterns, such as the connectors, transformers, filters and event busses that can be used in an integration. Next, Singh identifies the application layer which includes, for example, the workflows that define the event processing logic. Finally, a service or interface layer is identified, which exposes the functionalities of the integration platform to users and external applications. The platform management tool provides an administrator interface, while the dashboard provides the user interface in which the user can manage and model integrations [68]. Third, this layer provides web services, such as APIs that external applications can access.

The architectures described by Singh are typically powered by message based brokers such as ActiveMQ and RabbitMQ. However, event based architectures are increasingly moving from message based systems towards event streaming, powered by distributed event streaming solutions such as Apache Kafka [74]. Messaging and streaming both share the same fundamental concept, sending messages from publishers to subscribers, however they are fundamentally different in their implementation and properties. Messaging follows a smart-broker/dumb-consumer principle, pushing messages from the queues to subscribers and removing messages from the queue as subscribers acknowledge receiving it. Streaming follows the dumb-broker/smart-consumer principle, the broker stores an immutable log of messages that consumers can read. Stream-based brokers support message storage, replaying messages and high-throughput while event-based brokers provide better support for advanced message routing and different patterns such as request-reply [75]. While scaling is also supported with messaging brokers, Bakulev argue that the use of a streaming broker results in easier scaling and simplified processing typologies [74]. Message and streams are processed respectively by event processors and stream processors. The scalability and persistency makes stream processing more suitable for high-volume and statefull processing of data events, while event processing is more suitable for transforming and routing of self-contained business events. Therefore, an integration platform should ideally support a

combination of both processors [75] such that either one can be used depending on the use case and requirements of the integration. While Bakulev et al. recommend the support of event stream processing, they only provide a conceptualisation and no guidelines or architecture on how this can be implemented in practice. Hence this paradigm needs further research in the context of integration platforms.

Singh et al. only covers integration platforms, and not integration platforms as a service. For a perspective on how integration platforms can be offered as a service, one can consider the work of Kritikos et al. They describe high-level architectural components for multi-cloud application management platforms, which can manage the deployments of applications over multiple clouds. Such platforms have a high similarity to integration platforms [76], as integrations could be conceptualised as applications to be deployed and managed on the hybrid cloud. The components identified by Kritikos are discussed below and related to iPaaS.

- **Executionware**, or a runtime: The runtime abstracts the integrations from the specific infrastructure its deployed on. This runtime can be installed on-premise or in the cloud, and on the core integration features as proposed by Kleeberg can be executed on this runtime.
- **Model repository**: This repository contains models of the integrations. This could be considered an abstraction of the database identified by Singh. Runtimes can connect with this repository to obtain the integration models.
- **Upperware**: The responsibility of the upperware is threefold: First, to produce a deployment plan for the integrations. Second, to orchestrate the execution of the integration at run-time. Third to update the deployment plan when needed, for instance when more resources are required.
- **Monitoring plane**: This component allows users to manage deployed integrations and see application measurements and logs.
- **Control plane**: This component coordinates the execution of different platform components and the different models deployed on the run-times, to support the configuration of integrations.

Overall, this section has provided an overview of integration platform architectures, the available literature on this topic is, however, scarce. Future research is needed to provide a more exhaustive architecture of iPaaS. A lack of cross-referencing and literature on iPaaS architecture hinders research in the field of iPaaS. Without a common ground to rely on divergence in the respective field, in this instance iPaaS, is likely [73].

2.2 Interviews

2.2.1 Methodology

Three one-hour interviews were held with stakeholders from three companies using IoT, to identify use cases and challenges with integrating and managing IoT. The first company is one of the top 3 largest infrastructure construction companies in the Netherlands (by revenue) which is managing over 10 IoT projects. The second company is affiliated to the first company, and one of the largest real estate construction companies in the Netherlands, with many running IoT projects, of which 3 major projects covered by the interview. The third company is an IT consultancy firm, primarily active in the transport industry, but also in asset management and other sectors. The consultancy

firm recently took on several projects for IoT integration to be built using an integration platform, two of these projects are covered by the interview. For the sake of anonymity, the first company will be referred to as InfraCorp, the second company as EstateCorp and the third company as Consultancy Firm. With InfraCorp, the interview was held with the enterprise architect and a data scientist overseeing the IoT projects, with EstateCorp the interview was held with the enterprise architect and the integration architect. With the Consultancy Firm, the interview was held with the consultant responsible for the design of the two major IoT integration projects.

The interviews were conducted per the protocol in Appendix A.1. The interview is semi-structured, allowing diversion from these questions based on the course of the interview to accommodate the exploratory nature of these interviews.

2.2.2 Results

This section overviews the IoT challenges faced by each of the companies. A detailed description of the projects, and the challenges, for each of the companies can be found in Appendix A.

2.2.2.1 InfraCorp

With InfraCorp, IoT is used for a large variety of projects. The projects at InfraCorp are very similar in nature, and follow the following pattern: First data is collected, which is then processed and used to power an application that provides decision support or predictive maintenance.

Overall, InfraCorp identified the following challenges in working with IoT:

- **Connectivity**: Getting access to the data, and collecting the data in a central place is a challenge. When the data is retrieved from an external vendor, connectors need to be written for the specific infrastructure by which the data is delivered. In case data is collected with sensors owned by InfraCorp, this brings challenges related to the protocols to be used as well.
- **Data heterogeneity**: Each vendor, and each sensor has its own data format. Understanding the data format, and translating from each data format was found to be a major and time-intensive task.
- **Quality of the data**: Data can be missing, may contain error codes, or may otherwise be invalid. This has to be identified and accounted for. It was found to be a challenge to address these issues and ensure data quality for end-users.
- **Lack of standardisation in processing**: Currently for each IoT application, purpose-built scripts are used, leading to a large diversity in the scripts and approaches that are used. InfraCorp identified this variety in scripts as one of the most significant challenges and stressed that standardisation would be the best approach. However, they were not able to find any tools or methodologies to support such a standardised approach.

2.2.2.2 EstateCorp

The real-estate construction company also has a wide range of IoT projects, of which three were highlighted during the interview. Across these three projects and throughout the organisation EstateCorp identifies the following challenges:

- **Finding a business case**: EstateCorp claims that while there are many possibilities to apply IoT, finding a business case to fund a project is oftentimes more difficult than actually implementing the application. Partially, this is because high upfront investment costs.
- **Investigation and interpretation**: Every IoT device and API works differently, finding out how APIs work, for example to find out all the possible outputs from an API and how to interpret these, forms a major challenge. This does not only hold for new applications, but also for adding new sensors and data sources to existing applications. As a result, the addition of a new data source, such as the addition of a new new building to a building management system, is a project by itself, as each building will use different sensors and different software for which the IoT infrastructure must be adapted.
- **Management and Responsibilities**: The landscape of IoT integrations is complex. The physical sensors and their configuration, the IoT hub, data mining tools (like SkySpark), the digital data hub and the final applications all have a role in the IoT infrastructure. Since there is no central place to manage the infrastructure, and since there is no clear assignment of who's responsible for which components, this can lead to complications when problems occur. Sometimes, integrations break, sensors need to be replaced and/or configurations need to be updated, and the detection and management of these issues forms a challenge.
- **Reliability**: Reliability of the sensors is lower than expected. 50% of all sensors have an issue at least once a week, resulting in temporary loss of data. Reasons for this are unknown and may be related to the sensor or the connectivity of the sensor.
- **Integration with infrastructure**: Integration of IoT with the current IT infrastructure posed a challenge. Data warehouses and data processing was not suitable for data processing of IoT. There are many solutions offered by Azure, such as Cosmos, Data warehouses, etc, which all have their benefits and disadvantages. Selecting the appropriate technology for this while keeping cost and compatibility in mind poses a challenge.

In addition to the above list of company wide IoT challenges, project-specific challenges related to vendor lock-in, metadata and context are faced as well. Vendor lock-in is the result of hard-coding integrations to specific APIs, sensors and data vendors. In some instances, application code is built directly upon the vendor's specific data formats. This makes swapping and adding vendors a complex and expensive undertaking. Metadata, especially relations and hierarchies between devices and what they measure are complex to store, manage and integrate. And finally, contextual information about sensor data, such as the specifications, details and implicit knowledge about the data points is often lost since it is not stored and captured explicitly.

2.2.2.3 Consultancy Firm

Finally, the consultancy firm highlights two IoT integrations built on a messaging-based integration platform. Overall, the following challenges could be identified based on the interview:

- **Data transportation**: Currently, all integrations designed on the integration platform are message-based. While messaging provides many advantages (Section 2.1.2.9) it is sub-optimal for big data streaming, especially when all data has to be sent through a central component such as a message bus. Alternatives approaches, such as decentralised event streaming, are needed to provide efficient high-throughput data transportation for IoT.

- **Transformations**: Integration platforms provide excellent tooling for filtering and transforming messages using the enterprise integration patterns. Be that as it may, these tools are very inefficient for processing streaming IoT data. Some operations crucial for processing IoT data, like data aggregations, are not available at all on messaging-based platforms, and other operations, like enrichment, are complex to design. Therefore, the ability to preprocess IoT data is limited, or even impossible on the integration platform.
- **Monitoring and governance**: The integration platform is currently centred on transporting messages over a central message bus, where all monitoring and governance functionalities reside. This makes governing and managing the IoT data integrations complex when messages are not sent over a central component, if not impossible. Additionally, error handling functionalities are lost as well. Potentially, these issues could be overcome as the integration platform adopts event streaming or other functionalities that allow for decentralised governance.

2.3 Summary

Both a literature review, and three extensive interviews were conducted to provide background to this research, and to identify challenges with IoT integration that this research can address.

An IoT integration can be conceptualised as a series of data processing operations that process data from a set of data sources, and deliver output to a set of destinations. The integration will satisfy certain requirements, for instance related to what inputs it should support, what outputs it should produce, where it should run, and what the throughput should be. Several stakeholders with different roles are involved in the integration, such as integration developers, architects and the support team to maintain the integration. The integration may be running continuously, or it may be started based on a trigger, such as when data is pushed from a data-source. Each operation in an IoT integration is linked to another operation that it either uses data from or sends data to. There are several different operations and, based on the literature review, the following high-level operations have been defined:

- **Trigger**: This operation starts the integration, and may for example be a data push or a continuous process that keeps the integration alive.
- **Data pulling**: This operation is responsible for retrieving data from a data source. Data sources can be anything from which data is retrieved.
- **Data send**: This operation is responsible for sending data to the target that will consume it. Data targets can include for example applications, databases or other integrations.
- **Data transformation**: This includes any operation that changes the data itself. For instance, a translation from one data format to another, enrichment of data, filtering data, or aggregating data.
- **Data routing**: This describes how data can get from one operation to another, and which data should be sent to which operations. Depending on the implementation this can implemented using local variables that are passed from one method to another, or using more complex techniques such as streaming databases.

Finished integrations can be deployed to the execution environment (IoT processing infrastructure). Each integration may be deployed to a different environment, and a single integration can be deployed to different environments as well for scalability. Figure 2.4 provides a visualisation of how IoT integrations can be conceptualised.

An IoT integration is concerned with data transportation and transformation tasks. From a data-mining perspective, this means that data preparation and pre-processing tasks, such as cleaning, enrichment, generalisation and reduction are covered by the integration. However the data modelling and complex event processing tasks, such as data mining and analytics are not part of the integration, but could rather be conceptualised as applications of IoT.

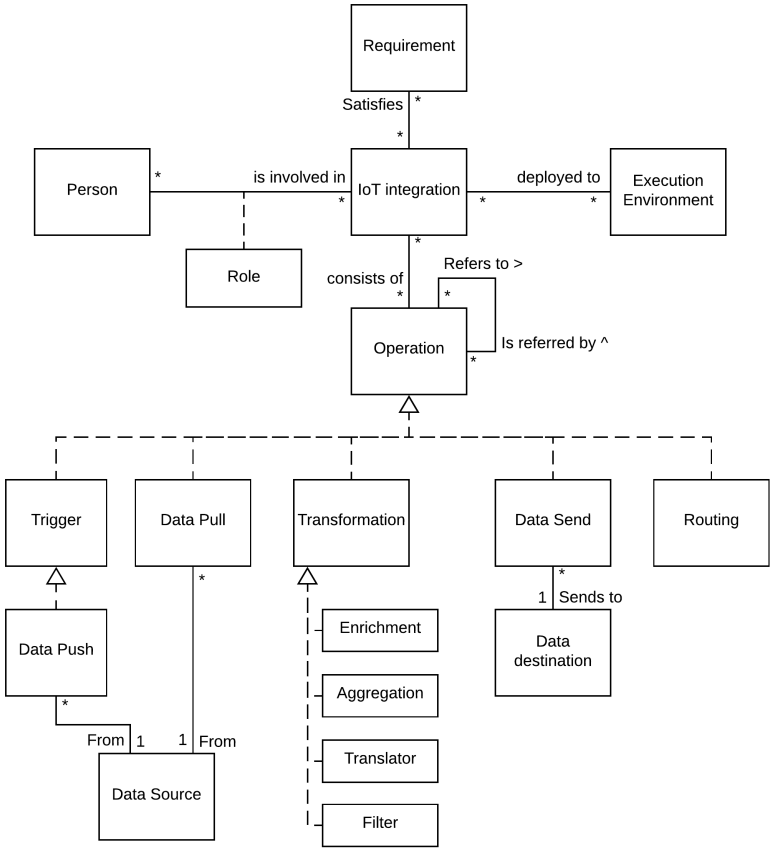


Figure 2.4: IoT Integration Model

This process of specifying, developing, running and managing the IoT integrations is complex. In the survey and the literature review, several challenges related to IoT integration have been identified. Table 2.2 below overviews these challenges and maps them to the relevant interviews and review results. All but two challenges could be mapped to both literature and interviews, which suggests that both the interview and the literature review were comprehensive.

Challenge	Literature	Interview
Challenge 1: Semantics IoT sensor data is highly contextual. Hence, before the data can be integrated, the semantics (meaning) of the data must be understood. Oftentimes, semantics are not explicitly specified and are only implicitly known among the group of users responsible for collecting it. This results in a steep learning curve before IoT data can be used for integration.	2.1.2.6, 2.1.2.2	2.2.2.1, 2.2.2.2
Challenge 2: Heterogeneity Each data source uses different data formats, data structures, syntax and protocols. Supporting each of these data sources, and overcoming the gap between the source data formats and the target data format is a time intensive and complex process.	2.1.2.7, 2.1.2.4, 2.1.2.5	2.2.2.1, 2.2.2.2, 2.2.2.3
Challenge 3: Finding a business case IoT has countless applications and use cases, however, monetizing IoT them and finding a business case can be a challenge. While this challenge is not immediately an integration challenge, the higher integration costs supported by the other challenges do affect the business case.	2.1.2.2	2.2.2.2
Challenge 4: Developing IoT infrastructure IoT Infrastructures are different from traditional IT infrastructures. Hence, developing IoT infrastructures, and consolidating both infrastructures is a complex task with many unknowns, for which expertise is required.	2.1.2.3	2.2.2.1
Challenge 5: Preprocessing Extracting relevant and workable data from raw sensor data is fundamental to use IoT. Literature shows that closing this 'abstraction gap' between primitive data and more use-full data is a complex task that oftentimes requires coding experts to develop custom data processing applications. Preprocessing can be any process that happens before the actual processing of data can happen. For instance, the collection of data, but especially process that increase the informative value of an event, such as enrichment or aggregation. Therefore, preprocessing also covers Challenge 6 and 2, which also contribute to the informative value of an event, by respectively addressing the quality and heterogeneity aspects.	2.1.2.7	2.2.2.1, 2.2.2.3
Challenge 6: Quality & Reliability Individual IoT sensors are prone to errors, they can for example lose connectivity or power. And even when sensors are working as expected, they only provide a snapshot of the reality, which is not always representative. Dealing with such invalid or missing data is a significant challenge.		2.2.2.1, 2.2.2.2
Challenge 7: Dispersed IoT integrations There is a lack of generic platforms and methodologies to design, develop and implement IoT integrations. Instead, IoT integrations require a large variety of different approaches and custom developed problem-specific solutions, resulting in low re-usability and large overhead.	2.1.2.7	2.2.2.1

Challenge 8: Central management and governance There is a lack of a centralised approach to manage and govern deployed IoT integrations. Due to the dispersed nature of IoT integrations, log data and error management are dispersed as well. This is problematic, as when applications break, this is hard to detect and debug as each involved integration has to be reviewed. Additionally, since there is no central overview of the IoT infrastructure and the associated responsibilities, changes remain undetected and common understanding about the infrastructure and the responsibilities breaks down.	2.1.2.7	2.2.2.2
Challenge 9: IoT stream processing support for integration platforms Current integration platforms, as documented in literature, are unsuitable for building and managing IoT integrations. These messaging-based platforms, do not provide provide support for statefull, high-throughput, low-latency event stream processing, which is required for working with IoT data. For instance, common tasks used in IoT data preparation like enrichment, storage, and aggregation over a window of time are not, or poorly, supported. Efficiently supporting such operations is vital to develop cost-effective IoT integrations. Yet, current literature does not describe designs for stream-processing based integration platforms, creating a challenge for anyone seeking implement or use such a platform.	2.1.2.9	2.2.2.3

Table 2.2: Challenges

While each challenge is self-contained, weak relations between challenges can be identified. For instance, the challenge that integration platforms do not provide proper support for building IoT integrations contributes to the challenge that IoT integration for pre-processing have to be custom developed using scripts. Figure 2.5 visualises the relations among challenges. All challenges contribute to the challenge that business cases for IoT are hard to make, as all challenges increase integration costs, hence affecting the business case.

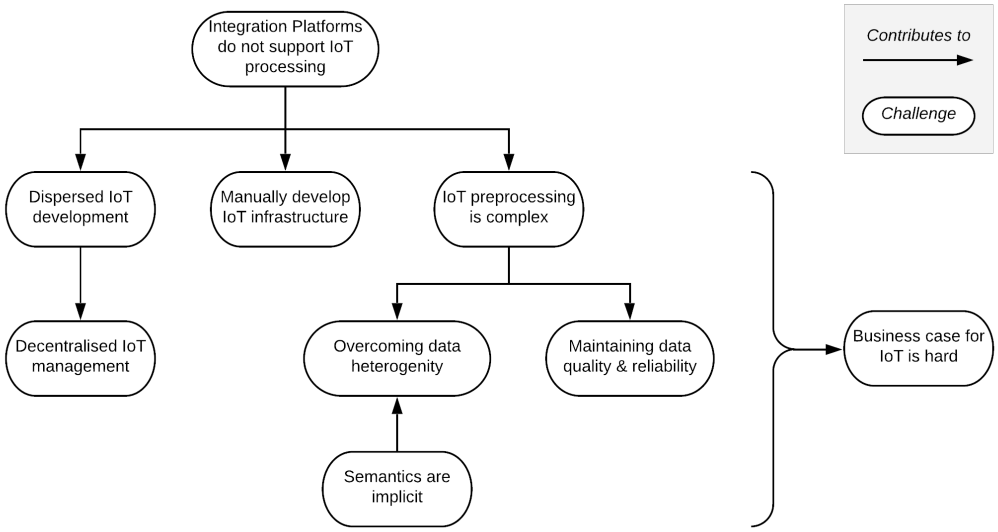


Figure 2.5: Mapping of challenges

The mapping suggests that Challenge 9, the lack of IoT support on integration platforms, is a key challenge. However, since the relations are weak, resolving the challenges higher in the hierarchy does not guarantee the resolution of all challenges lower in the hierarchy. Therefore, when addressing a challenge, it is important to ensure that the sub-challenges are addressed as well. For instance when addressing the challenge of IoT support for integration platforms, one should ensure IoT integrations are centralised, that one can abstract away from infrastructure, and that there should be a workflow for developing IoT processing integrations.

The literature review shows that there is not yet a holistic approach for addressing the aforementioned challenges. That is, literature does not provide an approach to overcome the challenges involved transporting and transforming data from IoT data sources to IoT applications. In practice, this gap in knowledge means that users that want to build applications on IoT, need to build point-to-point integrations with the IoT platforms from the ground up. This includes addressing complex tasks like developing infrastructure, data transformations, and management and transportation. This makes the development of IoT integrations a tedious and repetitive task, and the resulting integrations are hard to maintain. The impact of this on developers and business is tremendous, as it sets a high threshold for business to integrate IoT in their applications. Additionally, the vendor lock-in caused by the point-to-point integrations prevents businesses to horizontally scale out IoT applications on different IoT infrastructures.

2.4 Goal

The objective of this thesis is to improve IoT integration by addressing the aforementioned IoT integration challenges. As suggested by the challenge mapping, addressing Challenge 9 could be the first step towards addressing all challenges. The development of an integration platform for IoT integrations would satisfy the need for a reusable and flexible way to develop IoT integrations, as identified in Section 2.1.2.7. Furthermore, integration platforms have successfully addressed integration challenges in other domains than IoT. For instance in enterprise integration, where messaging integrations and event processors are used to integrate enterprise systems (Section 2.1.2.8). The hypothesis is that an integration platform for IoT could address the integration challenges and could therefore increase the efficiency of IoT integration development to reduce integration costs. However, to the best of our knowledge, no research into the design of an integration platform for IoT has been conducted.

Based on the hypothesis, it is possible to formulate a more concrete objective: *to provide a novel approach for designing a model-driven IoT stream processing platform*. The objective contains three definitions:

- The qualifier 'model-driven' represents use of modelling in integration platforms to define processes and data-flows rather than programming. In fact, as also described in Section 2.1.2.9, a key attribute of integration platforms is to to allow users to develop, deploy, and manage integrations through an easy to use graphical interface to reduce complexity.
- The term 'platform' reflects that it is a centralised multi-tenant environment that provides services for all phases in the integration life-cycle.
- The term 'stream processing' follows from Section 2.1.2.9, since this is the integration pattern that is most suitable for processing IoT.

This objective forms the foundation for the research goals formulated in Section 1.3. Based on the goal and problem statement, the next chapter identifies the requirements for the design.

Chapter 3

Requirements

To answer research question 2 this chapter discusses the requirements for the design. To establish the requirements, first the stakeholders are identified. This is, because stakeholders play a central role in setting requirements and constraints for the platform. According to Wieringa, stakeholders are those who are effected by treating the problem, and hence they form the source of the goals and constraints of the project [12]. Next, the stakeholder goals are identified and related to the challenges from Chapter 2. These stakeholder goals form the source of the requirements for the design [12].

3.1 Stakeholders

As with any platform based solution there are two key parties involved, the platform consumer and the platform provider. The consumers are the integration platform clients and the provider is the integration platform vendor. It can be that the consumer and the provider are within the same organisation, this would for instance be the case for an internal integration platform. However typically the provider is an external organisation that provides the platform to different clients. Both parties have stakeholders with goals for the project, which must be identified. Alexander et al. provide a taxonomy for identifying possible stakeholders in system development based on an onion model with several slots that can be occupied by stakeholders [77]. Depending on the system to be implemented, only a subset of the slots is relevant.

The platform client full-fills several key slots. First. the platform clients form the normal operators, or end users, of the platform. Several different normal operators can be identified:

- The architects, that define the requirements for integrations and that design the architecture for the integrations.
- The developers, that actually implement the integrations as per the requirements. These developers are often not programmers, but rather integration domain experts or consultants. Oftentimes, the architect and the developer are the same person, however, this may vary per platform client.
- The support team, that will maintain and manage existing integrations.

Especially in smaller organisations, the two or more of the last three stakeholder roles may be full-filled by a single end user.

The platform client also has stakeholders that full-fill the slot of functional beneficiary. These beneficiaries do not directly use the platform, but they may for example benefit from the IoT integrations by using applications powered by the integration platform. For instance, a building manager that has access to a dashboard where he can see the temperature of all rooms in a building. The management of the platform client full-fills the slot of purchaser, as this management is in charge of whether the platform should be adopted by the organisation or not. However, during development, when there are no definitive purchasers, the vendor's product manager should assume this role on behalf of potential purchasers [12].

The vendor also fills several slots. First, the slots of maintenance operators and developers, which are occupied by the platform vendor development team. Next the operational support slot, which is full-filled by the platform vendor's support team. The slot of interfacing system is also relevant, as the proposed stream processing integration platform should be capable of being integrated into the architecture of an existing integration platforms. This allows vendors to provide a unified integration service across integration patterns. The management of the vendor full-fills the role of project sponsor, as they initiate the project provide funding for the development.

In addition to the vendor and the clients of the vendor, external stakeholders can be identified. The slot of threat agents may be formed by hackers, that could want to breach the system. Negative stakeholders include competitors, or vendors of substituting products. The slot of regulator is assumed by the government, which poses regulations on data privacy and security.

Table 3.1 overviews all stakeholder slots and the stakeholders full-filling these slots. Additionally, this table shows how close the given slot is to center of the onion model, as discussed by Alexander et al. [77], with high involvement representing the most inner circle and low involvement representing the outer circle.

Slot	Stakeholder(s)	Involvement
Normal Operator	Client's integration architects, Client's integration developers, Client's support team	High
Operational Support	Vendor's support team	High
Maintenance Operator	Vendor's development team	High
Functional Beneficiary	Client's IoT application users	Medium
Purchaser	Client's management	Medium
Interfacing System	Vendor's enterprise integration platform	Medium
Developer	Vendor's development team	Low
Sponsor	Vendor's management	Low
Threat Agent	Hackers	Low
Negative Stakeholders	Competitors, Substitutes	Low
Regulator	Government	Low

Table 3.1: Stakeholder

3.2 Goals

The aim of this design research is to improve the problem context, by achieving stakeholder goals. In general, the goals of the platform vendor and the platform users are aligned, as it is the goal of the vendor to increase the use of the platform by satisfying the goals of the user. Both parties want to maximise their profits, the user does this by lowering integration costs through the use of the platform, the vendor does this by trying to attract users to their platform, at the lowest possible cost.

The stakeholder goals for all stakeholders with medium to high involvement in the project have been listed in Table 3.2, since these are most relevant for the high-level design. As becomes apparent from Table 3.2, the IoT integration developers have to face the most challenges to reach their goals. Next are the IoT integration architects. And finally, the other stakeholders.

#	Goal	Stakeholder	Challenge(s)
1	To reduce IoT integration costs through accelerating development and increasing governance	Client's management	3
2	To elicit requirements for, and design architecture of, IoT integrations	Client's architect	1, 7
3	To design an IoT infrastructure (for integrations to be deployed on)	Client's architect	4, 7
4	To efficiently and easily model, test and implement IoT integrations as per the requirements	Client's developers	2, 5, 6, 7, 9
5	To use reliable, high quality, applications to support their work	Client's IoT application users	6
6	To efficiently manage all deployed IoT integrations from a single location	Client's support	8
7	To provide users with an integrated and coherent integration platform	Vendor's enterprise integration platform	9
8	To resolve, and reduce the number of, operational incidents	Vendor's support team	9
9	To develop the integration platform to deliver the most value to its users	Vendor's development team	9

Table 3.2: Stakeholder goals

3.3 Requirements

This section describes the artifact requirements. To justify these requirements, they are supported by contribution arguments as described by Wieringa et al. [12]. These arguments are predictions about how artifacts that satisfy the requirements contribute to reaching stakeholder goals.

Requirements are classified as either functional or non-functional. If possible, requirements are operationalized by defining measurable indicators and a norm value for these indicators. For instance, in the requirement "the platform should be able to allow users to model data transformation X", the variable to be measured is the presence of transformation X, which can either be present, or not. In some instances however, it may not be possible to operationalize requirements.

Only those arguments relevant for a high-level design are identified. A range of other (non-)functional requirements could be defined, for instance requirements for security, reliability, and usability. However, these requirements will have limited impact on the design of the architecture as they are implementation specific.

The requirements have been validated with two experts within the problem context. In particular with a data scientist from InfraCorp, and with the product manager from the vendor from the case, who assumes the role of purchaser to represent the clients desires during development.

3.3.1 Functional requirements

R1 - The platform should support modelling to allow users to develop IoT stream processing integrations

This requirement is expected to contribute to goal 4. By developing integrations using model driven

engineering, rather than code, it is possible to lower the barrier for application development, allowing applications to be developed even by non-programmers [78], [79]. Additionally, it is expected that integrations become more understandable, and faster to develop by providing repetitive and complex operations as ready-to-use model elements [79], [80]. Integrations are modelled using operations, which represent the model elements. All operations described in Appendix B should be available as model elements.

R2 - The platform should allow the user to manage IoT integrations
This requirement is expected to contribute to goal 6 and 4. The management features for IoT integrations should match the management features currently provided by integration platforms (Section: 2.1.2.9):

- The user should be able to deploy integrations to the runtime environment
- The user should be able to scale integrations over runtimes.
- The user should be able to manage the versions of integrations.
- The user should be able to collect metrics about the integration, such as metrics about the data volume running through operations in the integration.
- The user should be able to collect log data from the operations in the integration.
- The user should be able to view errors in the integration.

In addition to the management functions above, that also apply to messaging, a new management function specific to IoT processing should be introduced for managing the retention policy for the data and metadata stored by the platform.

R3 - The platform should support stream processing, that is, the processing of high-volume, distributed, persistent, real-time data
This requirement is expected to contribute to Goals 7 and 9. As described in 2.1.2.9 an IoT integration platform should support event stream processing functionality to meet the requirements for use cases with high-throughput, statefull, processing, such as IoT. Other not streaming based processing patterns, such as messaging, offer lower processing throughput and do not support statefull processing. The interview with the consultancy firm confirmed the previous, stressing the limitations of messaging when processing IoT data and the need for change. To obtain the aforementioned benefits of stream processing, the platform should natively support stream processors. That is, it should not only convert the streams to messages for processing and vice versa, but it should be process the data stream directly from the streaming broker to allow for the real-time, scaleable, statefull, continuous processing of data.

R4 - The stream processing functionality should be embeddable on a runtime
This requirement is expected to contribute to Goals 7 and 9. Embeddability reflects the ability of the stream processing integrations to run anywhere in a dedicated, standalone, single-tenant, fashion. This is a key requirement for integration platforms, which gives the user full control as to where the integration should be deployed. This allows satisfying integration requirements regarding to data localization and throughput, as integrations can be deployed on the same location as the data to reduce data transfer.

The embeddability requirement is in strong contrast to common architectures of multi-tenant stream processing solutions, like Spark and Flink, that provide a single execution environment where all users can execute their processing applications. These solutions then oversee the complete application lifecycle, and create and manage the processes needed to execute the application. These solutions reduce the burden on the developer to manage applications at the cost of flexibility. However, flexibility and control are crucial for integration platforms to meet user requirements, for example to allow users to define deployment locations or to control scaling. Additionally, embedded stream processing solutions contribute to simplicity in the architecture, and benefits from the advantages of single-tenancy such as increased security and reliability. Aside from management, embedded and non-embedded stream processing solutions deliver similar performance which is discussed in more detail in Section 4.2.1.1.

R5 - The frontend should be accessible using a browser
This requirement is expected to contribute to Goal 7 since as it allows users to develop and manage integrations at any time, on any device, using a web-based interface. A native frontend introduce overhead (of installing, updating, running software) and impose restrictions on the devices that the platform can be used on.

R6 - The platform should be able to convert integration models to executable integrations
This requirement is expected to contribute to Goal 4 as the developed integration models should be executable. Therefore, all model elements should be convertible to executable instructions. These executable instructions (code) should be packaged in executables together with the runtime parameters, such that they can be deployed to a (remote) runtime environment without further configuration.

R7 - The platform should provide a runtime environment that can execute integrations
This requirement is expected to contribute to Goals 4 and 3. The runtime is twofold:

- The runtime environment should provide the software stack that integrations need to be executed, for instance, a dependency manager and a Java runtime. The runtime environment should be agnostic of the integrations executed on it, that is, all integrations should be able to run on all runtime environments deployed at any location in the infrastructure. This reduces development efforts, as integration architects re-use runtime environments for deployment.
- The software runtime environment is hosted on the computational infrastructure. The platform should allow the user to instantiate the necessary infrastructure as needed. For instance, by requesting the needed infrastructures (such as the cloud for hosting the runtime, and the event streaming broker) and cloud infrastructure from an infrastructure as a service provider and installing the runtime environment on it. By allowing the creation and management of the infrastructure through the platform, users can quickly deploy integrations, without first needing to design, create and prepare the required infrastructure manually.

As the software of the runtime environment is separate from the infrastructure, users do not need to use the managed infrastructure provided by the platform, and should also be able to bring their own infrastructure and install the runtime environment on.

R8 - The platform should support to use of schemas to manage data
This requirement is expected to contribute to Goals 4 and 5. Schemas are an important concept in integration, as it describes how data should look, allowing modelling on a meta level rather than having to write transformations for raw data streams. Schemas can also help to improve data quality and reduce errors, as any data not conforming to the schema is filtered out and can be processed accordingly. All schema compliant data can then easily be transformed, queried and filtered. The choice for Schemas for validation, rather than ontologies is based on the conclusions from Section 2.1.2.6. At least the following schema related functionalities should be supported:

- The user should be able to import, create and edit schemas.
- The user should be able to assign schemas to data sources
- The platform should be able to compute the new output schema of input data, when it is used in an operation.
- The user should be able to use the input data schemas of operations to formulate schema-based queries, filters and transformations

R9 - The platform should be able to validate integration models
This requirement is expected to contribute to Goals 4 and 9. During design-time the platform should be able to check the validity of an integration model. A model is valid, when it fully confirms to the integration meta model. For instance, whether all inputs and configuration options provided for an operation are valid and supported. The validation also ensures that the type and input schema of an operation (as per R8) is supported. This validation allows developers to quickly identify issues in their integrations without the need to completely review

or test an integration. Additionally, validation contributes to runtime stability and reducing operational incidents since syntactically invalid integrations cannot be deployed. This makes it less likely that edge cases cause an incident during execution.

R10 - The platform should be user extendable
This requirement is expected to Goal 7. Some complex and specific integration use cases may not be covered by the modelling operations in Appendix B. Therefore, the platform should allow the user to use custom operations in integration models, for instance using custom scripts or plugins. This also contributes to Goal 6, allowing all integrations, even custom ones, to be managed from a single platform.

3.3.2 Non-functional requirements

R11 - The platform should be extensibility
This requirement should contribute to Goal 9. The platform should be designed with extensibility in mind, such that more integration operations can be added by the vendor's development team as user requirements change. One developer should be able to add an additional transformation or connector to the platform within one two-week sprint.

R12 - The platform should be modular
This requirement should contribute to Goal 9. Modularity contributes to reducing complexity of further development by increasing testability, extensible, reusability, stability, scalability and other benefits. A system is modular when end-user functionalities (such as operations, or management functions) are implemented as interchangeable components.

R13 - The platform should be scalable
This requirement should contribute to Goal 8 and Goal 5. The runtime, backend and infrastructure should be scalable to support the concurrent development, execution and management of a large number of integrations. Furthermore, the runtime should be horizontally scaleable, to allow the instantiation of multiple runtime applications to increase throughput capacity linearly as needed.

Chapter 4

Review of Existing Solutions

In this section, research question 3 is addressed using a literature review to surveys existing solutions and approaches for developing an artefact that satisfies the requirements established in Chapter 3.

4.1 Methodology

The primary purpose of this literature review is to answer Research Question 3, that seeks existing designs for stream processing and graphical programming of stream processing integrations. This question is two-fold, as it refers to stream processing itself as well as the visual model-based development of stream processing integrations. These two components together can form an integrated platform that allows both the design and execution of stream processing integrations. Therefore, to design the integrated platform, the designs of these key components must be understood.

To this end, this review has two goals. First, to understand how embedded stream processing frameworks work by evaluating and comparing the different available frameworks and options for stream processing. Second, to understand how model-driven development tools for stream processing work by reviewing platforms and tools that allow user to graphically design, or model, stream processing integrations. To understand how these solutions work, the solutions are evaluated and compared to each other to discover the different designs, concepts and mechanisms used in these tools, as well as their effects.

These two goals yield the following literature research questions:

- LRQ 1: What are the available methods for event stream processing?
- LRQ 2: What are existing designs for graphical model-driven development of stream processing integrations?

The systematic literature review methodology proposed by Kitchenham et al. [15] is used to guide this literature review. This methodology is selected since a systematic approach is the most suitable approach when surveying existing treatments to identify all available alternatives [13], as is the case for this review.

4.1.1 Keywords

The keywords are used to search literature by metadata, specifically the title, abstract and keywords, and the are based on a decomposition of the research question.

LRQ 1: kafka AND stream AND processing
For LRQ 1 all literature that discusses stream processing on Kafka will be considered. The choice is made to search explicitly for solutions that support Kafka as the broker to narrow the search, as the terms 'stream' and 'processing' alone are not sufficiently specific. One could argue that searching for Kafka would reject stream

processing solutions that use other streaming message brokers, however, at the time of writing the concept of stream processing virtually implies the use of Kafka. The current consensus is that Kafka is the standard broker for streaming messages [81], [82], as per the definition in 3. Additionally, Yongou et al. surveyed message brokers and found only one alternative to Kafka as a streaming broker [83], which has not even been evaluated in the context of stream processing. Furthermore, the search is not explicitly restricted to papers that reflect upon the embedded aspect of a stream processing solution (requirement 4) since not all papers may discuss this particular aspect while still discussing the solution itself.

LRQ 2: (((graphical OR visual OR "model-driven") AND (stream OR "data analytics") AND (processing OR programming) AND (flow OR query OR etl)))
For LRQ 2 the keywords are more complex to determine. Tools for visual programming of stream processing integrations can be found under many different names, for instance under 'graphical' and 'model based'. And rather than 'stream processing' one may also refer to 'big data pipelines', 'flow-based applications', or 'programming' plus the name of a stream programming framework. As every possible combination of these keywords yields a search query that is too broad, specific combinations of these keywords have been selected as a trade-off of finding the most relevant results with the most specific query.

These keywords are used to search literature by metadata, specifically the title, abstract and keywords.

4.1.2 Inclusion and exclusion criteria

A paper is included if it is relevant for the research question at hand, that is if it matches one or more of the following criteria:

- The paper describes a framework, or compares frameworks, for embedded stream processing compatible with Apache Kafka.
- The paper describes a visual tool, or compares visual tools, for programming an integration.
- The paper describes a methodology for instantiating integration models into executable integrations.

Additionally, the paper should meet all of the following search criteria:

- The paper is available through the catalogue of the University of Twente or is otherwise publicly accessible.
- The paper is written in English.
- The paper is published between 2015 to 2020.
- The paper is published in the subject area of computer science or engineering
- The paper is published in a scientific journal, magazine, or conference proceedings.

However, not all results are immediately relevant. Therefore, several criteria are introduced to increase relevance. For instance, should present domain-independent findings and should be focused on the topic at hand. Therefore, a paper is excluded if it matches one or more of the following criteria:

- The paper only describes the application of stream processing, or the application of a visual tool, for a specific domain (i.e. Health).
- The paper only describes stream processing in a multi-tenant setting.
- The paper only describes a visual tool for non-programming related tasks associated with integrations, such as management, troubleshooting, etc.
- The paper only describes a stream processing platform that allows the composition of existing stream processing applications (such as Spring Cloud Data Flow), rather than the definition of new event stream processing applications.
- The paper is a duplicate of another paper.

4.1.3 Review protocol

First, the aforementioned keywords are entered into Scopus and the search criteria are supplied as parameters, such as the date range. Scopus is used since it provides the largest abstract database of (peer-reviewed) literature. According to Cavacini et al., Scopus provides the highest coverage of computer science journal articles, while maintaining high-quality [84].

Next, all the found papers are reviewed by their title and abstract based on the in-and-exclusion criteria. All papers that are irrelevant based on the title and/or abstract are discarded, and the remaining papers are added to the database.

During a full-text reading, all papers in the database are read. Papers were discarded from the database if they did not actually fulfil the inclusion criteria. Next, all references of the remaining primary studies are analysed, first based on their title and then based on a full read, and were subsequently added to the database if they met the criteria.

Finally, for data extraction a concept-centric approach, as proposed by Webster et al. [85], is adopted. This approach is used to extract all concepts from the relevant papers, and map them in a matrix of the respective literature.

Figure 4.1 provides an overview of the literature review protocol and the number of papers included/excluded in each phase. The included papers are discussed in Section 4.2.

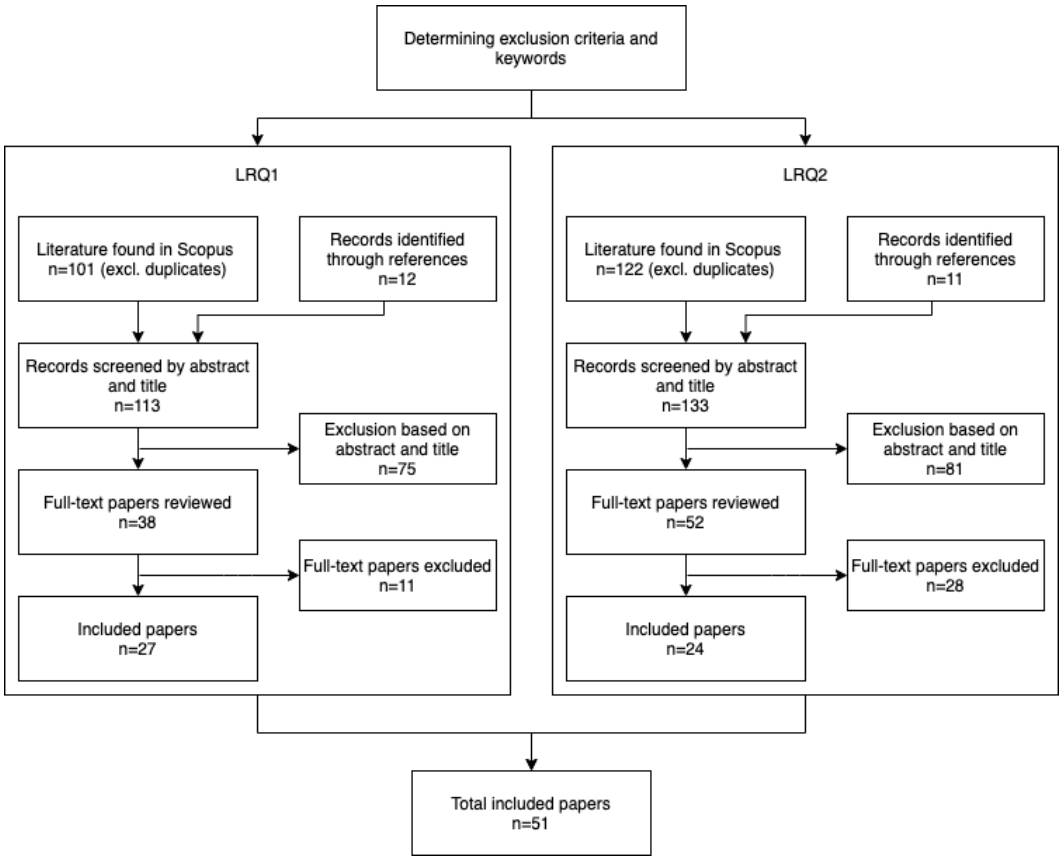


Figure 4.1: Literature review approach

4.2 Results

4.2.1 Kafka stream processing

There is a wide body of knowledge on stream processing frameworks. However, embedded stream processing (in the context of Kafka) has received less attention. After excluding papers per the exclusion criteria, for instance, papers focused solely on stream processing with clustered solutions such as Apache Spark or Flink, 15 papers remained. Through backward reference searching, 12 additional papers were added resulting in a total of 27 papers.

17 papers provide a survey or benchmark of stream processing solutions. Of the remaining papers, 7 discuss a single stream processing solution and 3 papers discuss a DSL or abstraction layer for stream processing. First, each stream processing solution is described individually and next all solutions are compared.

4.2.1.1 Overview

Before discussing each individual stream processing solution. It's important to note the difference between Kafka, and stream processing on top of Kafka.

Kafka is an implementation of an event streaming platform, as introduced in Section 2.1.2.9. It is a distributed platform that allows the publishing of, and subscribing to, streams of records to enable real-time stream processing. Each record in the stream consists of a key, a value and a timestamp, and these records are categorised into topics. These records are persisted for a fixed amount of time, regardless of whether the message has been consumed. Only the last offset of the consumer is stored, allowing consumers to freely consume records in any order they want. This also allows the addition and removal of consumers without impacting other consumers. Kafka is run as a cluster that can run on one or more servers and optionally across data-centres. For a detailed description of Kafka, please refer to Wang et al. or Shree et al. [86], [87]. While Kafka is focused on storing and transporting records, stream processors focus on creating, consuming and transforming records. Such processing, including consuming and producing data, always happens outside the Kafka cluster, as seen in Figure 4.2 Instead, the processing can be done on a separate processing cluster, or as discussed in this research, in any existing application (embedded). The tasks of connecting to the cluster and processing the records can be supported using libraries provided by the Kafka project, such as Kafka Streams and Kafka Connect, as well as any other Kafka compatible frameworks such as Flink, Samza and others. The remainder of this section will focus on these frameworks.

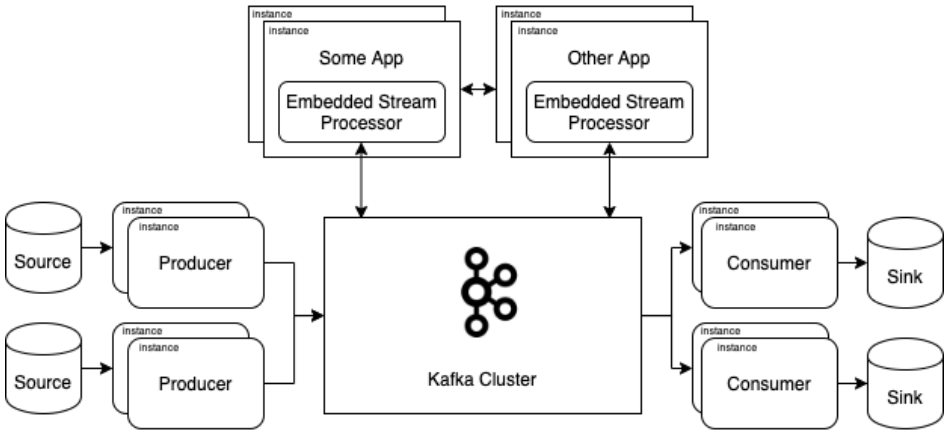


Figure 4.2: Kafka Processing Architecture

Kafka Streams The Kafka project provides two Java APIS for stream processing on top of Kafka, the Kafka Streams DSL and the Processor API. The processor API facilitates the connection between the source and

destination topics, and provides a low-level API with direct access to the tuples to be processed, allowing both stateless and stateful processing. The latter is achieved by automatically creating Kafka topics that manage the state. The Kafka Streams DSL is an abstraction on top of the Processor API which provides common high-level data processing operations, such as flatmap, aggregate and group. Both Alaasam et al. and Fernandez-Rodriguez et al. present a stream processing architecture based on the Kafka Streams framework [88], [89]. The Streams DSL is based on a dual streaming model of streams and tables [90]. This is because a typical stream processing task, uses both streams and databases (tables), representing the stream of changes and then a view of the current state respectively. Consider for example an IoT temperature sensor; one may want to process both the stream of temperature updates, for example, to send an alert when a certain threshold is met, and store a table with the latest known temperature for every room, such that the user can request the current temperatures. Kafka Streams is currently managed and developed by Confluent as an Apache project. Confluent also offers a set of Kafka-related projects outside the Kafka Project itself such as Connect, for building producers and consumers, and ksqlDB. KsqlDB is an abstraction layer on top of Kafka Streams allowing stream processing with SQL statements. While initially, KSQL supported embedded deployments [91] the newer versions of KSQL (ksqlDB) are only designed to be deployed as a cluster[1].

Samza Samza is a stream processing library similar to Kafka Streams and was developed at LinkedIn, which also incubated and open-sourced Kafka. Compared to Kafka Streams, Samza is more mature and also supports batch processing. Both Noghabi et al. and Kleppman et al. discuss the design of Samza [92], [93]. Samza, like Kafka Streams, is tightly coupled with Kafka and both Samza and Kafka Streams can be used as lightweight libraries embedded in microservices. However, Samza also offers support for YARN allowing optional clustered deployments. Similar to Kafka, Samza provides two Java APIs; a low-level API with direct data access, as well as the Samza Streams DSL that provides a set of built-in operators for common stream processing tasks. Compared to the Kafka Streams API, the Samza Streams DSL provides still requires the definition of a dataflow graph (topology of processors) while Kafka Streams completely abstracts away from access to the graph. The Samza Streams DSL does conceptualise both streams and tables, and on top of the Java APIs, Samza offers SamzaSQL for stream processing with SQL statements [94].

Esper Esper is a stream processing solution from EsperTech. The solution was developed before Kafka was, however, it has since then evolved to support Kafka and rely on Kafka for horizontal scaling. The primary way to process in events in Esper is using EPL, a domain-specific language extending from SQL. EPL is very expressive, with support for many features required for complex event processing. Moreover, Java (MVEL) code and Javascript can be used in EPL. Like Samza and Kafka Streams, Esper conceptualises both streams and tables. İnçki et al. demonstrate how Esper can be used to create an architecture for processing streaming data from Kafka [95].

Akka Streams The Akka Streams API is part of the Akka, which is a runtime and toolkit for the development of concurrent applications. Akka programs are designed according to the actor model, which involves programming message-driven microservices called 'actors' that communicate with each other. Akka Streams is an abstraction of Akka actors for stream processing, rather than message-based event processing. Akka Streams provides both a high-level DSL implementing the Java Reactive Streams API, with support for the most common stream processing operations, as well as the low-level GraphStage API. Akka does not rely on Kafka, however, Akka provides a catalogue of adapters using the Alpakka project that allows reading from and writing to different data sources. This way, Akka can be used to consume from, and publish to, Kafka. Lv et al. [96] show how Akka can be used, together with Kafka to support stream processing of IoT data.

HazelCast Jet Similar to Akka, HazelCast Jet is a stream processor independent of Kafka, but able to consume from and publish to Kafka. HazelCast Jet offers a low-level Core API, and the Pipeline API DSL on top of the Core API. In addition to supporting embedded deployment, a client-server clustered topology is also supported.

[1] `https://github.com/confluentinc/ksql/issues/734` consulted 23-06-2020

Stream Processing Framework	Survey(s)
Kafka Streams	[99]–[103]
Samza	[104]–[113]
HazelCast Jet	[101]
Spring	[111]
Esper	[114]
Akka Streams	-
Beam	-
Open-Source Clustered (either Flink or Spark)	[98]–[114]
Commercial Clustered (i.e. Azure, AWS)	[103], [108], [110]

Table 4.1: Surveys

Spring XD / Stream Spring Cloud Stream, formally Spring XD, provides a library for stream processing on top of either Kafka or Kafka Streams. The latter being recommended for stream processing applications, such that the 'stream' and 'table' abstractions can be used. Spring Cloud Stream has advantages compared to Kafka Streams, as it has less boilerplate code and provides support for programming models from other Spring projects such as Spring Integration. Spring Stream exposes whatever operations are used by the binder in Spring, for instance with the Kafka Streams binder, it will expose Kafka Streams operations. Spring also provides a collection of default stream applications for input, processing, and output named Cloud Stream App Starters [2]. Moreover, Spring offers Spring Cloud Data Flow to develop topologies of Spring Stream applications and Spring App Starters.

Beam Apache Beam is a unified DSL that can execute on multiple processing platforms. From the list above, Beam applications can run on Hazelcast Jet and Samza. Beam provides a high-level Java DSL, that supports both bounded and unbounded collections (the equivalent of tables and streams in Kafka Streams). For input and output, Beam provides a catalogue of adapters called Pipeline I/O to read/write from data sources and message formats. Finally, Beam also provides another abstraction layer called BeamSQL that allows stream processing using SQL statements in two dialects, including Calcite which also forms the foundation of SamzaSQL.

Other Two other frameworks were identified in this literature review. Meehan et al. propose S-Store for streaming transaction-oriented processing (OLTP) on top of Kafka [97]. Balduini et al. [98] benchmark Natron, a stream processor specifically focused on RDF web stream processing. Compared to the other frameworks, these two frameworks remain mostly conceptual and are not offered as product-ready software. Additionally, both frameworks are less suitable for generic stream processing tasks but instead focus on OLTP and RDF processing. Therefore, these two frameworks will not be covered in more detail.

4.2.1.2 Survey

Table 4.1 maps each frameworks to the survey(s) it is evaluated by. The embedded stream processing frameworks are compared based on these surveys, guided by the taxonomy for comparing stream processing frameworks proposed by Zhao et al. [111]. Interesting to note is that on many aspects identified by Zhao; fault tolerance, data processing and data streaming, the frameworks share many attributes:

- They executed in the JVM, and provide Java APIs for developing stream processing applications.
- They provide support for both stateless and stateful processing, for instance, processing over sliding and tumbling windows.
- They provide exactly-once processing guarantees.
- They support parallel processing and can be scaled horizontally, by launching multiple instances.

[2] `https://cloud.spring.io/spring-cloud-stream-app-starters/` consulted 01-06-2020

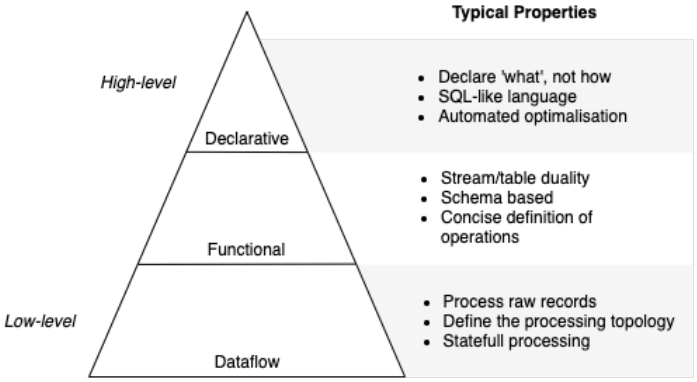


Figure 4.3: Abstractions for stream processing

- They provide a high-level DSL with support for common stream processing operations, such as filtering, mapping, and aggregating.
- They provide an API for developing custom connectors, such that developers can create interfaces with any external data sinks or sources, regardless of the protocol or format used.

However, there are differences especially concerning data ingestion, data execution and non-functional aspects identified by Zhao. Table 4.2 summarises these aspects in a comparison based on the following attributes:

- **Dual Streaming Model**: Whether or not the dual streaming model of streams and tables is supported [90].
- **Batch**: Whether or not processing data in batches is supported in addition to stream processing.
- **Native Clustering**: While all data processing frameworks can be scaled up manually, some frameworks support clustering natively, such that jobs can be allocated automatically to available workers.
- **Connectors**: This value represents the number of built-in connectors that are available to connect to a data source, examples are MQTT, JMS and Twitter.
- **APIs**: The APIs for programming on several levels of abstraction, this is visualised in Figure 4.3. Carbone et al. identify three levels of abstractions for stream processing [115]:
  - Dataflow: The presence of a low-level API for direct interaction with records and control over the data flow graph.
  - Functional: The presence of a high-level API that provides operations for transformations on data streams.
  - Declarative: The presence of a SQL-like language for defining integration models.
- **DSL Expressiveness**: Overall expressiveness of functional and declarative APIs. That is, to what degree the API supports complex event processing operations. As per [116].
- **DSL Simplicity**: Overall simplicity of declarative APIs, as per [116].
- **Overall ease of use**: The overall judgement about the complexity of the system, specifically how easy it is to deploy the system, develop applications and to troubleshoot. Ease of use is subjective to the specific requirements, therefore the table maps to the surveys for further consultation about the context of the judgements.

In addition to the differences covered by Table 4.2, stream processing platforms also have different underlying data transportation and serialisation mechanisms. First a common communication protocol is needed to process data. Kafka Streams, Samza and Spring Stream Kafka use Kafka as underlying communication layer, the remaining solutions use a proprietary communication layer. In addition, to provide users with higher level APIs, data must conform to a certain data structure such that the processor can interpret the data to invoke

	Kafka Streams	Samza	HazelCast Jet	Esper	Akka Streams	Beam	Spring Stream Kafka
Dual Streaming Model	Yes	Yes	No	Yes	No	Yes	Yes
Batch	No	Yes	Yes	Yes	Yes	Yes	Yes
Native Clustering	No	Yes	Yes	Yes	Enterprise	N/A	N/A
Connectors	50+	5+	20+	5+	50+	50+	50+
Dataflow API	Processor API	Task API	Core API	-	GraphStage API	-	Kafka Binder
Functional API	Kafka Streams API	Streams API	Pipeline API	-	Reactive Streams	Beam DSL	Kafka Streams Binder
Declarative API	KSQL (Clustered)	SamzaSQL (Calcite)	-	EPL	-	ZetaSQL, CalciteSQL	-
SQL Expressiveness	Low	Medium	N/A	High	N/A	Medium	N/A
SQL Simplicity	High	Medium	N/A	Low	N/A	Medium	N/A
Overall ease of use	High [101]	High [113]	High [101]	-	High [96]	High [117]	-

Table 4.2: Comparing embedded stream processing solutions

transformations on it. Incoming data should first be deserialised or 'parsed' to such a data structure before it can be used, and after processing the outgoing data must be serialised again for transportation. This is visualised in Figure 4.4.

Therefore, when writing a connector, the developer must ensure that the serialised data can be transported by connecting to the data store using the specific communication protocol supported by the store, for instance, JMS or MQTT. Next, the developer should ensure that the data can be serialised/deserialised (SerDes). If the format of the data is compatible with the default SerDes of the processing platform, no action is needed. For other formats, the developer needs to define custom SerDes to support the specific data format. The default SerDes that are supported vary per platform and are listed below.

- Samza provides support for Java primitive types, Java Serializables and JSON[3].
- Kafka Streams and Spring Stream Kafka support JSON, Avro and Protobuf[4].
- HazelCast Jet supports only Java Serializables[5].
- Esper supports XML, Java POJO, and Avro[6],
- Akka Streams supports JSON and Avro[7].
- Apache Beam, provides a different approach to and will not use fixed SerDes but SerDes associated with a collection[8].
- Spring Stream Kafka natively supports JSON and Java Serializables [9].

In practice, users will seldom need to define custom connectors as they can choose from a library of pre-built connectors that handles all protocol conversion and serialisation tasks. The exceptions being Samza and Esper which provide a rather small ecosystem of connectors.

[3] https://samza.apache.org/learn/documentation/latest/api/high-level-api.html
[4] https://www.confluent.io/blog/kafka-connect-deep-dive-converters-serialization-explained/
[5] https://jet-start.sh/docs/next/api/serialization
[6] http://esper.espertech.com/release-6.0.1/esper-reference/html/event_representation.html
[7] https://doc.akka.io/docs/alpakka-kafka/current/serialization.html
[8] https://beam.apache.org/documentation/programming-guide/
[9] https://docs.spring.io/spring-cloud-stream/docs/Brooklyn.RELEASE/reference/html/contenttypemanagement.html

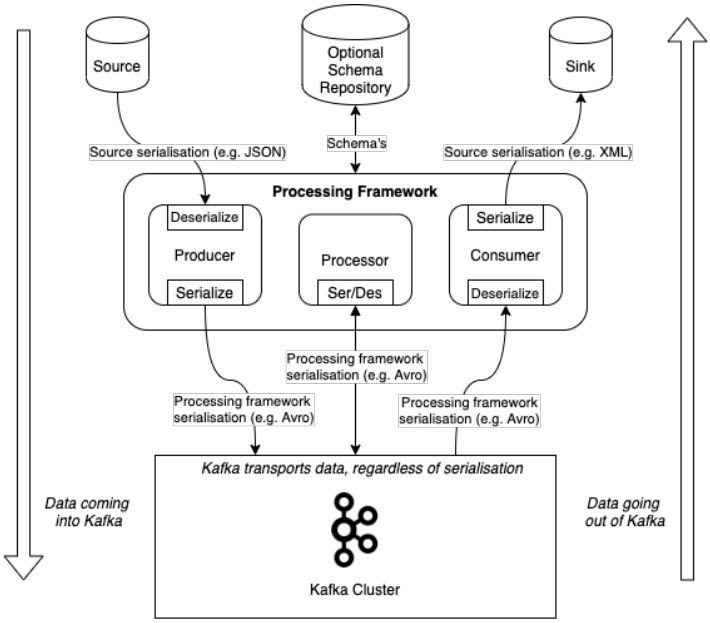


Figure 4.4: Serialisation for Kafka Stream processing

	Kafka Streams	**Samza**	**HazelCast Jet**
Latency	Low [99]–[102]	Low [105], [106]	High [101]
Throughput	Medium [99], [100], [102]	High [106]	-
Efficiency	Medium-High [100], [101]	High [92]	High [101]

Table 4.3: Performance of stream processing solutions

Several stream processors have also been benchmarked for performance. Specifically, Kafka Streams, Samza and HazelCast Jet. Table 4.3 provides an overview of benchmarks for these solutions. The performance evaluations are relative to open source clustered stream processing solutions, specifically Flink and Spark, and overall the surveys show similar performance evaluations for both clustered and embedded stream processing. Zhao et al. argue that performance across stream processing solutions is comparable and that the differences in performance are only significant for specific demanding use cases [111]. Additionally, all stream processing performance ultimately depends on the infrastructure, which is scalable, allowing users to scale-up throughput when needed. In addition to performance, studies evaluating the energy efficiency of stream processing platforms found embedded solutions to be equally as cost-effective as distributed solutions [98], [114]. These studies, however, only reflect on smaller frameworks, such as Natron and Esper, therefore future further research in this area is needed.

4.2.1.3 Conclusion

The surveys provides an overview of the key properties of embedded stream processing solutions, as well as a taxonomy of the attributes on which the solutions differ the most. All solutions have great overall performance and they all provide key features such as statefull processing and exactly-once processing guarantees. When evaluating the different options it becomes apparent that there is no single best solution for stream processing. Rather, the selection of a stream processing solution will depend on the functional and non-functional requirements for the specific use-case and context. If one is already invested in actor-based programming using Akka, or if one would be interested in adopting Akka for actor-based processing, Akka Streams would be noteworthy option. Users seeking a solution that is deploy-able across different stream processing frameworks may opt for Apache Beam. Data scientists, researchers and other users seeking an expressive query language could opt for Esper. For users heavily invested in the Spring ecosystem, or for users seeking to use Spring

Cloud Data Flow to orchestrate stream processing applications, Spring Stream would be the preferred solution. Overall Kafka Streams and Apache Samza are the most complete and mature stream processing solutions. Both provide great performance, and a large ecosystem of abstractions and community support. Compared to Kafka Streams, Samza provides embedded SQL, native support for batch processing, and more control over data processing graphs even when using higher level APIs. Kafka however provides commercial support, easier to use Java APIs, and a larger collection of connectors.

4.2.2 Graphical stream programming

While stream processing is an increasingly popular topic, research into graphical or model-driven programming for stream processing is still sparse. After excluding papers per the exclusion criteria, for instance, papers on programming for non-streaming data, 14 papers remained. Through backward reference searching, 11 additional papers were added resulting in a total of 24 papers. These 24 papers discuss a total of 13 different solutions.

4.2.2.1 Overview

Gokalp Gokalp et al. present a stream processing platform for complex event processing [118]. Unique is the use of Node-Red, which is a popular solution for visually programming, and executing event-driven applications. However, only the visual programming component of Node-Red is used, the visual models are then parsed, validated by an application manager module after which they are converted to Storm typologies that form the distributed execution engine.

OptiqueVQS OptiqueVQS is an ontology-based approach to data stream processing, funded by the European Union [119]–[123]. OptiqueVQS includes a visual interface for developing queries named StreamVQS. This visual interface allows users to visually compose queries over heterogeneous data streams using ontologies. These visual queries generate a StarQL (an ontology-based query language) query which is then translated, using the ontologies and mappings, to several data-specific SQL queries. These SQL queries are then executed over data streams using the ExaStream, a distributed streaming extension of the SQLite database manager.

RheemStudio RheemStudio is a visual IDE on top of the cross-platform analytics platform Rheem [124]. RheemStudio allows visual, interactive programming and monitoring of data analytics tasks. The visual models are translated to RheemLatin (SQL-like) queries. RheemStudio also allows users to view the physical execution plan and to define custom operators using a high-level Java DSL, without leaving the web interface. Once jobs are defined in RheemStudio, they can be executed using Rheem, which is a platform-independent stream processing platform similar to Apache Beam. It can automatically allocate tasks to the most efficient underlying processing framework for that task, for instance, Spark or Hadoop. Rheem was however excluded from the earlier survey on stream processing frameworks since it does not provide support for Kafka, and since the stream processing capability of Rheem is still under development [125]. RheemStudio is built using the MEAN stack and is not yet made open-source.

aFlux aFlux is a visual node-based programming tool for IoT stream processing [126]–[130]. aFlux consist of a frontend, similar to Node-RED with a visual designer, and a backend that can translate these visual models to executable Akka Stream integrations. Additionally, aFlux offers extensions that allow code generation for Flink, Spark, Pig and Hive execution. aFlux backend runs in the JVM, and the javascript-based frontend runs in the browser.

StreamLoader StreamLoader is a streaming ETL tool [131]. The frontend allows the user to design ETL integrations, and the backend will execute these integrations. The backend supports worker nodes that can be used to execute the ETL integrations to allow for scalability.

Lemonade Lemonade is a platform for visual programming of data analytics and ETL tasks [132], [133] (Surveyed by: [127]). Lemonade itself is developed using a micro-service architecture. The Citron microservice serves the user interface that allows the visual programming of the integration model. Citron exports the visual model as JSON to a backend component called Juicer, which is responsible for transforming the JSON definition to Spark code. In addition to generating Python Spark code, Juicer can also deploy the job to a Spark cluster, and measure the resources used during execution. Since the initial paper was published, Juicer has been extended to SciKit-Learn and other Python frameworks. All backend micro-services are written in Python while the frontend is written in VueJS.

QryGraph QryGraph provides a graphical, visual interface to big data processing on Hadoop [134] (Surveyed by: [127]). the QryGraph interface allows users to visually create Pig Latin queries, which are SQL-like queries for the Pig data processing platform that runs on top of Hadoop. Pig itself was not covered in the previous survey, as it does not provide Kafka support. The interface ensures that the Pig queries are valid, and also provides options for executing and managing Pig queries. QryGraph itself is built with the Play framework, supported by the actor-oriented Akka library. QryGraph is a web application, and the backend runs in the JVM.

Sydow et al. Sydow et al. present a prototype of a native graph editor that allows users to visually design stream properties, routing and tasks [135]. These visual designs can then automatically be converted to Rust code for distributed stream processing. The tool only provides a way to model the integration on a high-level, and the actual processing task needs to be defined textually.

Flision Flision is a prototype of a graphical user interface for designing integration models, generating Flink source code and automated deployment of the generated Flink applications to the Flink cluster [136] (Surveyed by [118]). Flision is designed to be general-purpose and provides users with the option to create custom operators from within the user interface.

MEdit4CEP MEdit4CEP is a graphical editor for defining complex event processing (CEP) applications and architecture [137], [138]. MEdit4CEP supports ModeL4CEP, which consists of a domain-specific modelling language, and a graphical modelling language for defining complex event processing integrations. Therefore, the purpose of MEdit4CEP is also two-fold, first to allow domain experts to define CEP domain models and second to allow end-users to define CEP applications using these domain models. Once designed, the models can be transformed into executable code for Esper, StreamSQL and CCL. MEdit4Cep is an Eclipse-based native editor.

QualiMaster-IConf QualiMaster is another EU funded data processing infrastructure [139]. (Surveyed by: [127]). The QualiMaster IConf is a native tool to graphically model big data streaming applications and to generate Apache Storm based streaming applications based on these models. To determine the model elements for this tool, different big data streaming applications were analysed. QualiMaster-Iconf is a Java application built upon Eclipse.

ClowdFlows The EU funded CloudFlows platform supports the construction and execution of big data processing and mining workflows [140]. In contrary to the previous solutions, CloudFlows attempts to be a platform and community for sharing and executing workflows. The CloudFlows user interface is accessible using the web browser and is backed by the CloudFlows server. This server is written in Python and contains the data model of the available workflows and widgets (operations). Finally, there are work nodes that run a headless version of the CloudFlows server which are designed to execute workflows. Consequently, CloudFlows does not generate any code. Instead, workers directly parse and execute Workflows. CloudFlows is also available as a SaaS application at `cloudflows.org`.

	StreamPipes	ClowdFlows	QryGraph	Lemonade	aFlux	Optique	Flision
Code export	No	No	SQL	Python	Java	SQL	Java
Live generation	N/A	N/A	Yes	Yes	No	Yes	Yes
Code deployment	Yes	Yes	Yes	Yes	No	Yes	No
Stream Processors	Flink, Spark, Esper, Java	Prop.	Pig	Spark	Flink, Spark	Prop.	Flink
Validation	Ontology	Type	Schema	Schema	Order	Ontology	None
Live validation	Yes	Yes	Yes	Yes	No	Yes	N/A
Setup	Easy	Easy	Medium	Complex	Complex	Medium	Complex
Custom operations	Plugin	Plugin	Script	Script	Plugin	Script	Script
Native operations	50+	25+	10+	25+	15+	5	7
Windowing	Yes	Yes	No	Yes	Yes	Yes	No
CEP	Yes	Yes	No	Yes	No	No	No

Table 4.4: Comparing graphical editors

StreamPipes StreamPipes started as a EU funded semantic stream processing platform [141]. Since 2015 (when StreamPipes was been proposed) the project has been under active development and is now an incubating Apache software project. The focus of StreamPipes has shifted from only ontology-based processing to also support lightweight schema-based semantics. StreamPipes provides an intuitive visual editor for developing integrations or 'pipelines'. Users can model these pipelines using pipeline element containers, which represent an operation, source or sink. Each pipeline element container requires a certain input and output format, defines a certain execution logic and is implemented using a wrapper that represents an underlying processing framework. Currently supported frameworks are Flink, Esper and running directluy on the JVM [10]. This also implies that each element in StreamPipes can only run on the frameworks for which a wrapper is provided. After the user has developed an integration, StreamPipes will automatically invoke each pipeline element container, this container will then submit the program to the underlying processing framework using the wrapper. StreamPipes will arrange the messaging between the processing frameworks using a suitable message broker, such as Kafka or JMS.

4.2.2.2 Survey

While the graphical stream programming editors discussed above share the same fundamental concept, the ability to graphically program stream processing applications, they are each fundamentally different in their functionality, implementation and maturity. When considering maturity, there is a striking contrast with the stream processing platforms discussed earlier. While almost all of the discussed processing platforms were adopted by the industry, this is seldom the case for graphical stream processing solutions. Some of the editors discussed are only conceptual and have no public prototypes, this includes [118], [124], [131], [135]. Most tools do have a public prototype but do not offer a stable release or a website targeted at end-users, nor are they used outside an academic context. These tools include QualiMaster-IConf, Flision, QryGraph, aFlux and OptiqueVQS. Only three tools do offer a website targeted at end-users, Lemonade, ClowdFlows and StreamPipes. The last being the only framework that shows actual signs of adoption in practice with over 150 stars on Github and currently incubating as an Apache project. Overall, the literature suggests that graphical stream programming, while actively being researched, is still an upcoming technology with low adoption rates in practice.

For the remainder of this survey, the focus will be on the web-based editors with a public prototype or a public release. This, since a public prototype is needed to properly evaluate the editor. These editors are StreamPipes, ClowdFlows, QryGraph, Lemonade, aFlux, OptiqueVQS and Flision. To compare graphical editors, a concept-centric approach is used to extract the concepts discussed for each solution [85]. Overall, the extracted concepts

[10] https://streampipes.apache.org/docs/docs/dev-guide-architecture/

can be classified under different themes; code generation, extensibility, validation, and functionality. Some of these themes are reviewed in Table 4.4 by comparing the attributes below, and all themes are discussed in detail in the paragraphs beneath.

- **Code Generation**
  - **Code export**: The source code / queries for deployment on a processing infrastructure that is generated by the editor. In some instances, the editor does not generate code, and integration models are directly interpreted by the stream processing framework.
  - **Live generation**: When the code is generated live during design time or not live, for instance at build time. Live code generation is preferable, such that the user can instantly review the generated code.
  - **Code deployment**: Whether the tool has a built-in mechanism to deploy the integration to the stream processing infrastructure.
  - **Stream Processors**: The stream processing framework(s) supported for executing the integration models.
- **Validation**
  - **Type checking**: Whether primitive type checking, order based type checking, schema-based type checking (for complex types), or ontology-based type checking is supported. Validation is explained in more detail in Paragraph 4.2.2.2
  - **Live validation**: Whether validation is live during design time or during build time. Live design time validations are preferable, as they provide instant feedback to the user regarding the validity of the integration model.
- **Functionality**
  - **Setup**: Complexity of setting up the editor before an integration can be created. 'Easy' when a ready to use executable or a SaaS service is provided. 'Medium' when a Docker container or build script was provided to run the editor. 'Complex' when multiple manual commands and configurations were needed to set up the tool.
  - **Custom operations**: Whether the custom operations can be defined as custom script nodes in the editor or as plugins. Plugin based custom operations allow for easier re-use and sharing among users, however, custom scripts are easier for quickly implementing custom operations.
  - **Native Operations**: The number of operations built-in, such as join, reduce, aggregate, etc. This number does not include sink and source operations and CEP operations.
  - **Windowing**: Whether the framework supports windowed operations
  - **CEP**: Whether the framework supports complex event processing or data mining. For instance, classification, regression, clustering, etc.

Code Generation The different editors provide different architectures for generating source code. Overall, three different approaches can be identified, as depicted in Figure 4.5. This paragraph will discuss all three approaches in detail.

Lemonade, aFlux, and Flision have a frontend that exports the visual models as JSON models, which are then parsed and validated on the backend. The backend then uses these internal logical models to generate executable code. Each node in the logical model provides a function that allows it to generate code for its instantiation. How code is generated varies per implementation. Flision generates Java code directly from hardcoded strings, as does Lemonade for Python, while aFlux relies on JavaPoet [11] which is a Java API for generating java source files. The code generator traverses all nodes in the model to compose the executable

[11] `https://github.com/square/javapoet`

code.

QryGraph and Optique use a single model for both the backend and the frontend, thus avoiding the need to maintain two different models and converting between them. The model in the frontend and backend are kept in sync using an underlying framework, such as Akka.JS. For QryGraph the actual code is generated on the backend, while for Optique the frontend generates the SQL and sends it to the backend. In both instances, each model node provides hardcoded instructions for generating executable SQL code and the model is traversed to generate the code. For both GryGraph and Optique, the SQL is also available in the user interface for users to view and manipulate, the benefit of this approach is that visual models can also be generated from SQL queries.

StreamPipes and ClowdFlows follow two different approaches to avoid the generation of deployable code. ClowdFlows sends the integration models directly to the executors, which can understand and execute these models without translation or code generation. StreamPipes provide so-called 'pipeline element containers' that provide the execution logic to the underlying processing framework based on the visual model once the pipeline has started. This execution logic is embedded in the container, using the APIs provided by the stream processing framework.

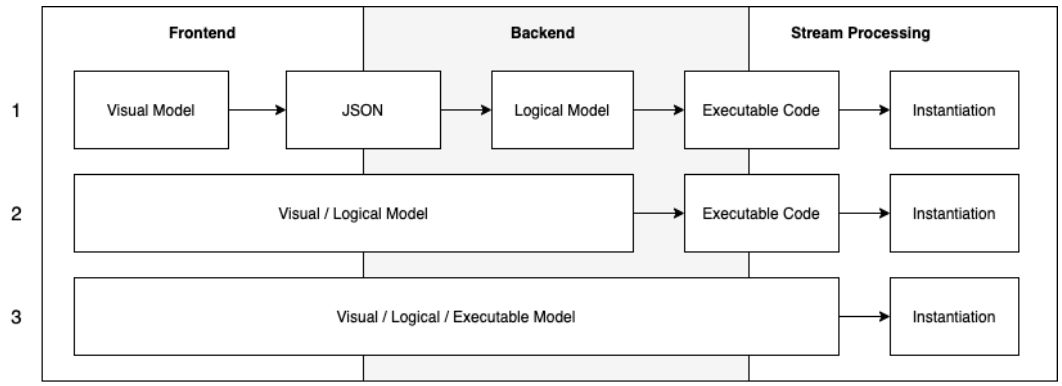


Figure 4.5: Approaches for code generation

Extensibility While inherent features, such as the stream processing platform used or the type of validation supported are hard to modify and change, all frameworks are extensible in the operators they support. As Table 4.4 shows, frameworks either allow the definition of custom operations through plugins or runtime scripts. The built-in operators are also designed in a modular fashion such that new operations can be added when needed. For built-in operations and plugins, the frameworks provide templates that one can use to define a custom operation. The actual effort needed to implement a custom operator will vary per framework. In case of StreamPipes, for example, creating a plugin is a complex process which first requires defining the ontology or schema both on a static level and runtime level, as well as separate definitions for the model, binder, controller and execution logic, including the execution instructions for each of the supported processing frameworks [12]. In QryGraph or aFlux custom operations are small single classes that define the execution logic and in-and-outputs in a compact format, mapping operations to the high-level DSL. In lemonade, the definition of an operation is more dispersed as each operation is defined in two locations, in the 'Tahiti' module which tracks the metadata of operation, such as the required in-and-outputs and the configuration options, and in 'Juicer' which performs the code generation for an operation[13].

Validation Next, the editors can be classified based on their support for validatio. Optique and StreamPipes provide the most extensive support for validation through ontology-based semantics. StreamPipes supports

[12] https://streampipes.apache.org/docs/docs/dev-guide-tutorial-processors/
[13] https://docs.lemonade.org.br/en/architecture.html

domain-specific data processors that can be specified using RDF (implemented using JSON-LD)[14]. These specifications are used on the backend to validate whether the in and output of two connected operations are compatible at design time. During runtime, only lightweight schemas (such as JSON) are used to reduce overhead. In addition to domain-specific data processors, generic processors with weaker validation are also supported. These generic processors can validate complex types against schema-based type definitions. Consider for example source of location data, of the type Latitude Longitude. This would be a domain-specific data stream, which could be processed by both a domain-specific processor accepting only longitude and latitude, as well as a generic processor accepting complex data types consisting of two integers. Optique supports only domain-specific nodes, that can be specified using OWL. Compared to StreamPipes, Optique allows querying complex data structures and relations but provides limited to no support for generic operators for data manipulation. As with StreamPipes, the ontologies are only used during design time, and actual data processing operations are validated against regular schemas.

Schema-based validation for complex type checking are not only supported in Optique and StreamPipes, but also in QryGraph and Lemonade. In QryGraph and Lemonade, type checking is instant and happens during design type, this allows the user to quickly identify mismatches. Also, since schema information is available during design time, the schemas can assist users in configuring operations. For instance, when configuring an 'average' operation in QryGraph or Lemonade, and selecting a complex data type as input (i.e. a temperature event with a temperature, time and location) the editor will recognise this schema and allow the user to select the attribute that should be averaged. QryGraph provides type validation by instantly compiling Pig queries and requesting the Pig server to return the schema output schema and any possible type errors. In StreamPipes a different approach is used. Each operation has specific input and output interfaces, when connecting two operations the backend verifies whether the two interfaces are compatible. Input interfaces are always static but can be defined at any level of abstraction. For instance, they may only require specific ontology elements, or they may accept any primitive type. For output interfaces, StreamPipes allows the output interface to change depending on the connected input and the configuration of the operation during runtime. StreamPipes provides several alternatives for this:

- The output interface will mirror the interface of the input data (i.e. when inputting an order object, the output interface will be of type 'order').
- The output interface can be fixed, for example, as a static schema. An average function may for example always output a single float.
- The output interface may append to the input data interface. For instance, it may add a new attribute.
- The output interface may use a subset of the input data interface. For instance, it may remove a few attributes.
- The output interface may be a transformation of the input data interface. For instance, it may always rename certain attributes.
- The output interface follows other custom logic that is defined during runtime.

In Lemonade, the editor (Citron) requests metadata, such as the interface, for each operator from the metadata storage (Tahiti) and then verifies whether all connected operators use compatible interfaces for their in and outputs. Lemonade's documentation does not reveal how such interfaces are implemented, yet based on the described functionality it presumably uses a similar mechanism as StreamPipes.

The editors with the weakest support for validation are ClowdFlows, aFlux and Flision. ClowsFlows provides primitive type checking only, but does not support complex types. For instance, it can ensure that an 'average' operator receives an int input, but it cannot validate that a 'customer' object transformer, in fact, receives an object of the type 'customer' as input. aFlux performs order based type checking, for instance, a node can specify that it must be used before or after a certain another type of node. This may, for example, be useful to ensure that a location filter can only be used with a location data source, however, it limits flexibility as it only

[14] https://streampipes.apache.org/docs/docs/dev-guide-architecture/

enforces specific combinations of operators. Flision provides no support for validation at all, and will only detect issues at runtime.

Functionality For an initial impression of the functionality of an editor, one can consider the user interface, Each editor provides a unique different user interface, shown in Appendix C. Nonetheless, all editor UIs share a few key components:

- A palette, or catalogue, of available operators. Usually, these operators are grouped. Most editors group operators based on whether it is a sink, a source, or a processor. However, ClowdFlows and Lemonade also group processors based on the kind of processing (i.e. Table-based, CEP, or primitive). aFlux and QryGraph group operators based on the underlying framework that provides the operators, such as Flink or Spark.
- A canvas where users can drop operators from the palette, and connect them by dragging lines between them.
- A toolbar with actions that affect the integration as a whole, i.e. the ability to save the model, undo actions or to deploy the model.
- An options pane that allows the configuration of an operator. In Lemonade, QryGraph, Optique and aFlux, the option pane becomes immediately visible when selecting an operator, while in the other editors the options are only revealed when double-clicking or right-clicking an operator in a separate window.

All editors also provide an interface for viewing output. This can vary from a simple console which shows the log with runtime errors (Flision, aFlux, ClowdFlows) or an advanced UI with widget-based dashboards for monitoring the integration (StreamPipes, ClowdFlows and Lemonade).

Some tools provide only a single screen from which all the editor features can be used, such as aFlux, Optique, and Flision. The other editors are multi-screen and provide other screens for management and configuration. For example, screens to manage the created integrations, to configure deployment settings or to manage custom operators and data sources/sinks.

The amount of screens depends on the scope of the application. The scope varies from basic tools to design integration models to full platforms for managing the integrations. The platforms, StreamPipes, CloudFlows and Lemonade, are designed as standalone end-products, that aim to support the complete life-cycle of a stream processing application from creation, to deployment and management, and these tools provide a rich catalogue of operations to model with. The remainder of the editors are primarily tools to support the user in the design phase. Since these tools do not provide management functionalities they allow for more control and freedom, but they require more expertise from the user. Both kinds of editors, therefore, target different users; the platforms target end-users with specific use cases, primarily big data analysis and preparation. The tools focus more on assisting professional developers in creating multi-purpose data integrations, providing the essential design functionalities while leaving developers with the freedom to manage the integration and the and define specific operators as they choose.

4.2.2.3 Conclusion

As indicated at the beginning of this survey, visual stream programming is a domain that is still actively researched, and little to no mature solutions are available. StreamPipes is the most mature solution of all alternatives, yet it is still an incubating Apache project. Additionally, it is not a suitable solution if one is seeking to export code for self-management and deployment. This is especially problematic if one needs to cover a wide number of use-cases that are not all covered by StreamPipes. In these situations, custom integrations can only be managed outside StreamPipes, while StreamPipes integrations can only be managed within StreamPipes, scattering the management of integrations. While the other solutions do provide code export functionality, none provide Java code generation along with schema-based validation to ensure the validity of the generated integration, which would be a key property to satisfy the requirements. Additionally, none of the

solutions supported embedded processing except for StreamPipes, which provides a proprietary embeddable Java executor.

While none of the surveyed solutions satisfies the requirements, this review provides a comprehensive overview of the mechanisms and concepts used in the surveyed solutions which can be re-used in the design of a novel solution. For instance, the survey shows that there are several approaches to code generation, from using different models for the visual models and the logical models together with node-based code generation, to using a single shared model that can directly be executed by the processing framework. There are also various approaches towards validation, from simple fixed order and type based validation to dynamic ontology and schema based validation with dependencies between the input and the output of an operation. The survey shows that these design decisions with regard to the different approaches for code generation and validation significantly impact the extendability of a solution. Finally, a difference in scope between the editors is observed, from solutions that target end-users by providing full life-cycle management and a rich catalogue of operations, to tools that target developers. These tools focus on the design-stage, stimulating the development of custom operations and self-management. The identified concepts and design choices can be used to support the development of new editors, for instance to discover and evaluate design alternatives.

Chapter 5

Design

This chapter addresses research question 4 by proposing a new design for a platform that allows users to model and manage IoT event streams processing integrations. The goal of these integrations is to connect IoT data sources with IoT applications (Chapter 2). Figure 5.1 overviews the design from an end-user perspective. This first section discusses the methodology used to guide the design. The remaining sections discuss the design itself.

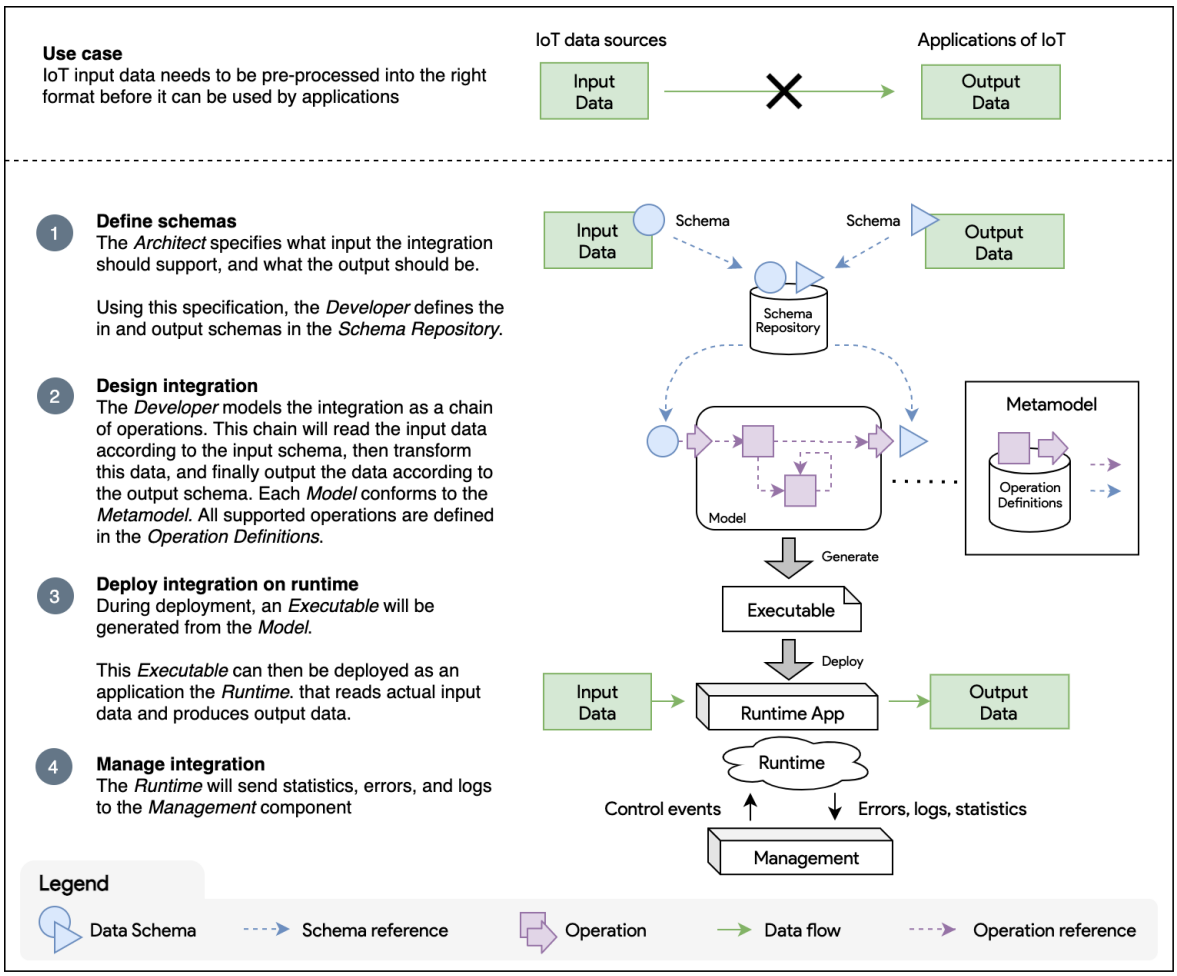


Figure 5.1: Design overview

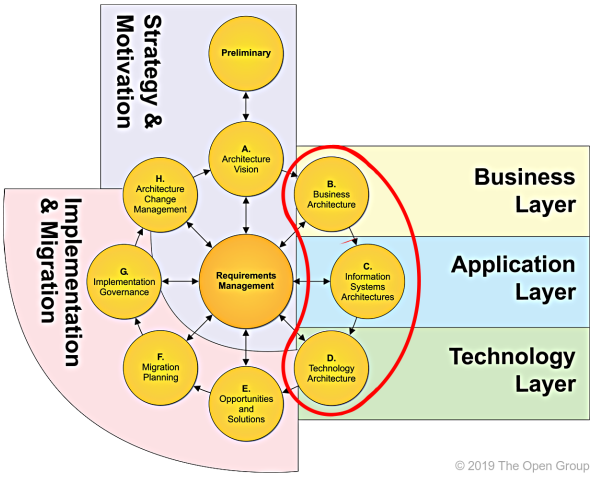


Figure 5.2: Design phases of the TOGAF ADM[1]

5.1 Methodology

The TOGAF Architecture Development Method (ADM) as discussed by Iacob et al. is used to guide the design [142]. The ADM provides an approach towards the design, planning, implementation and governance of enterprise architecture. According to Gils et al. such an architecture is "a formal description of a system, or a detailed plan of the system at component level to guide its implementation. [143]. The term 'enterprise' reflects the broad scope of the ADM, as it is suited to guide the architecture design of virtually any IT-system used in an enterprise [143]. The ADM provides three phases, the strategy and motivation phase, the design phase and the implementation and migration phase. In this chapter, only the design phase is used, as the strategy and motivation phase is already covered by the DSM in Chapter 3. The implementation and migration phase is covered by Chapter 6 and 7 as the design is applied to the problem context.
The ADM guides the design for four primary domains, the business domain, the data domain, the application domain and the technology domain. The business domain defines the business process that are supported by the architecture. The data domain describes the structure of the organisations logical data. The application domain provides provides an overview of all the systems, their interactions, and how they are exposed to the business domain. The technology domain describes the technological infrastructure that supports the deployment of integrations. An architecture can be described from the perspective of any of these domains using the ArchiMate specification.

ArchiMate is a modelling language for architectures that is integrated with TOGAF. ArchiMate provides a layered view on architecture, that maps onto the domains of TOGAF. The business domain is represented by the business layer, the application and data domains are represented by the application layer and the technology domain is represented by the technology layer. The higher layers rely on the services and interfaces provided by lower layers. The design phase of TOGAF, with the corresponding layers in ArchiMate, is highlighted in Figure 5.2.

The ArchiMate 3.1 notation will be used to visualise designs when possible, and other tools will be used throughout the design phases of the ADM to provide more detail when necessary. For instance textual descriptions, UML diagrams, and interface designs will be used to provide further clarification. This is, since ArchiMate is designed to provide a holistic overview of an architecture, while other notations may be more suitable to describe low-level designs [143]. While the design of an architecture is always involves a creative process, key decisions will be supported by the requirements and the literature review to ensure traceability. Table 5.1 traces all requirements to the architectural elements that realise them.

[1]Adapted from ArchiMate 3.1 spec, retrieved 23/07/2020 from `https://pubs.opengroup.org/architecture/archimate3-doc/apdxd.html`

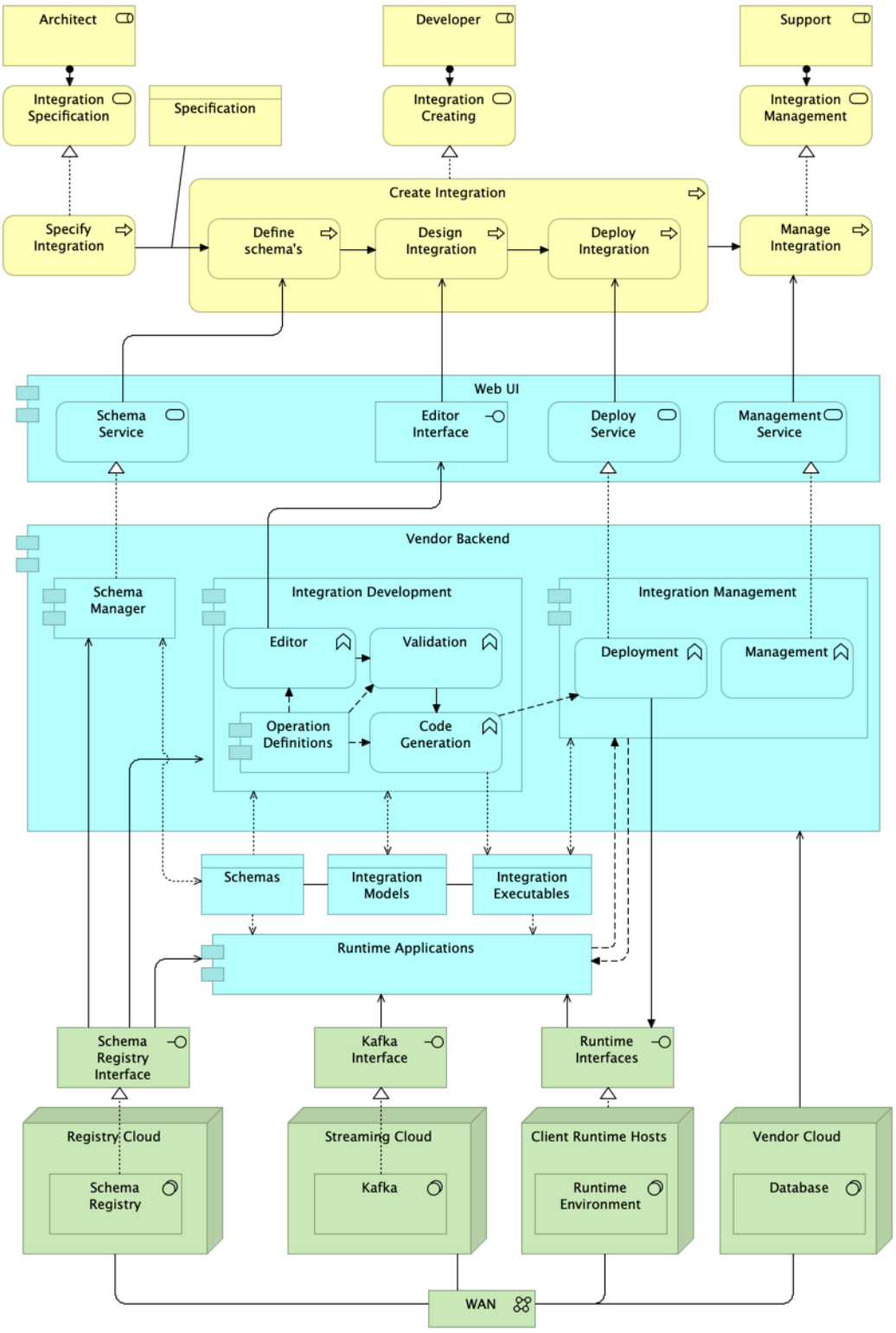


Figure 5.3: Layered Architecture

Requirement	Architectural element(s)
R1	5.3.3.2, 5.2
R2	5.3.5, 5.2
R3	5.4.3
R4	5.4.2, 5.3.7
R5	5.3.1
R6	5.3.3.4, 5.3.3.1
R7	5.3.7
R8	5.3.6
R9	5.3.3.3
R10	5.3.3.1
R11	5.3.3.1
R12	5.3
R13	5.4, 5.3.7

Table 5.1: Requirement tracing

5.2 Business layer

Figure 5.3 overviews all layers of the design on a high-level. The yellow top layer represents the business layer. The middle blue layer represents the application layer, and the bottom green layer represents the infrastructure layer. Figure 5.4 shows the business layer in more detail, as well as its interaction with the application layer interfaces and services. The three top swimlanes represent the three normal operators of the system (Section 3.1); the *Architect*, the *Developer* and the *Support* team.

During the *Specify Integration* process, the *Architect* will identify the need for the integration, and create a *Specification* for the integration that describes all the information needed by the *Developer* to actually design the integration. For instance, it includes the schemas of the inputs and outputs of the integration, as well as any information needed to understand these schemas, for instance the semantic value of ambiguous fields. Additionally, the *Specification* includes all the information needed to connect to the data sources and sinks. The *Specification* also includes a deployment description of the integration, such as the processors that should be designed, and the location to which they must be deployed as well as the expected throughput they will be facing. The *Specify Integration* process can (but does not have to) be supported by the system, depending on the implementation. Therefore, this is the only business process that is not depicted as being realized by the application layer. The *Specification* that the *Architect* produces is passed on to the *Developer* for implementation.

The *Developer* will actually design and deploy the integration during the *Develop Integration* process. This process consists of three sub-processes. The first is the *Enter Schemas* process during which the *Developer* will enter the schemas of the data sources and sinks into the system. This process of defining and manage the schemas is realised by the *Web UI*, through which the *Developer* interacts with the *Schema Manager*. Once the schemas have been entered, they will be passed on to the *Design Integration* process such that they can be used with operations and transformations. This *Design integration* process is a complex activity in which the *Developer* will define the connection to the data sources and sinks and will define the operations to be performed on the data stream. The design process is supported by the *Editor Interface*, which is embedded in the *Web UI*. All changes of the *Integration Model* are processed by the *Integration Development* component which also generates the *Integration Executables* of the *Integration Model*. Once the design is complete, the *Integration Model* is passed on to the *Deploy Integration* process. During deployment, the *Developer* defines the deployment parameters such as the version of the integration, after which the *Integration Executables* are deployed to the runtime. The *Develop Integration* process is iterative and the *Developer* may return to any of

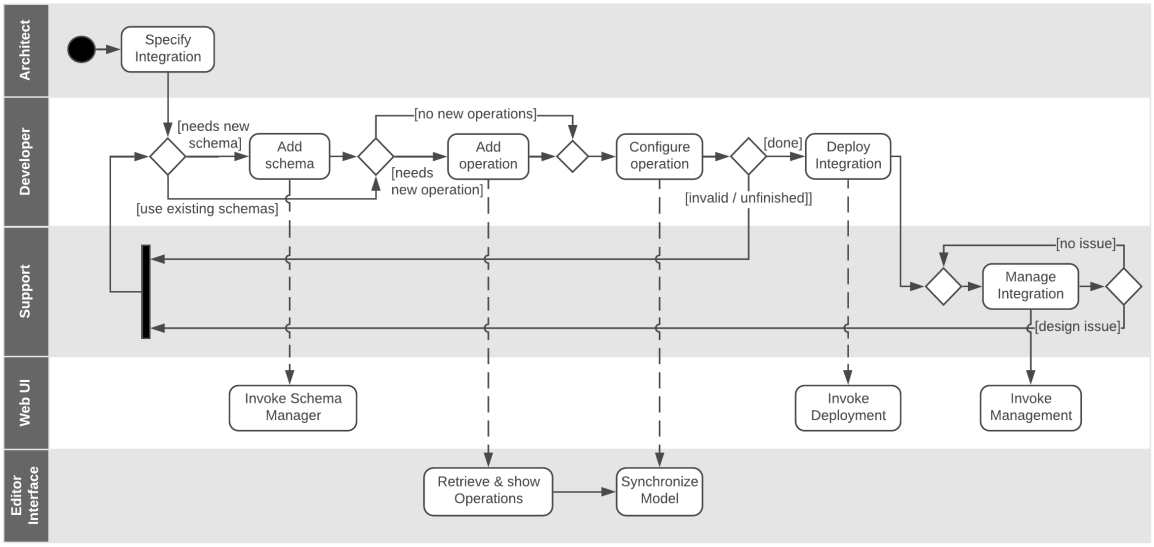


Figure 5.4: Business Activity Diagram

the previous activities to iterate upon the integration and re-deploy it.

After the integration is deployed, the *Support* team will start the *Manage Integration* process. Management is primarily concerned with monitoring the deployed integration, for instance, monitoring errors, resource usage, and throughput statistics to find any anomalies. To ensure separation of concerns, the *Support* team will not interfere with the integration when any anomalies are found but rather contact the *Developer* to make any changes to ensure that any modifications are properly designed and tested. Only basic operations, such as restarting the integration or making more resources available, can be executed within the *Manage Integration* process.

5.3 Application layer

The application layer has been designed in a modular fashion, and has three key components. The *Web UI*, which is a frontend component, the *Vendor Backend* which supports the frontend, and the *Runtime Applications*, that represent the running integrations developed and deployed by the *Integration Developers*. The *Vendor Backend* is the most complex component, which has three modules; the *Integration Development* component, the *Integration Management* component and the *Schema Manager*. The sections below will describe the design choices for each of these components in detail, as well as the data model of the application layer.

Figure 5.5 provides an overview of all models discussed in this section. Central are the *Integration Models*, which are the models representing the integration, created by the *Developers* using the *Editor*. The metamodel of these *Integration Models*, that describes what an *Integration Model* should look like, is formed by the *Integration Meta-Model* (part of the *Data Model*) and the *Operation Definitions*. The *Integration Meta-Model* is general metamodel for the integration, while the *Operation Definitions* complements this metamodel by describing all possible operations that can be used in the *Integration Model*. The *Operation Definitions* are developed by the vendor, extending from the *Operation Parent Model*. During code generation, the *Integration Model* is translated to *Integration Executables* as is explained in section 5.3.3.4. This executable integration can be deployed (instantiated) as *Runtime Applications* that can process incoming data. All of the models are also discussed in more detail in the sections below.

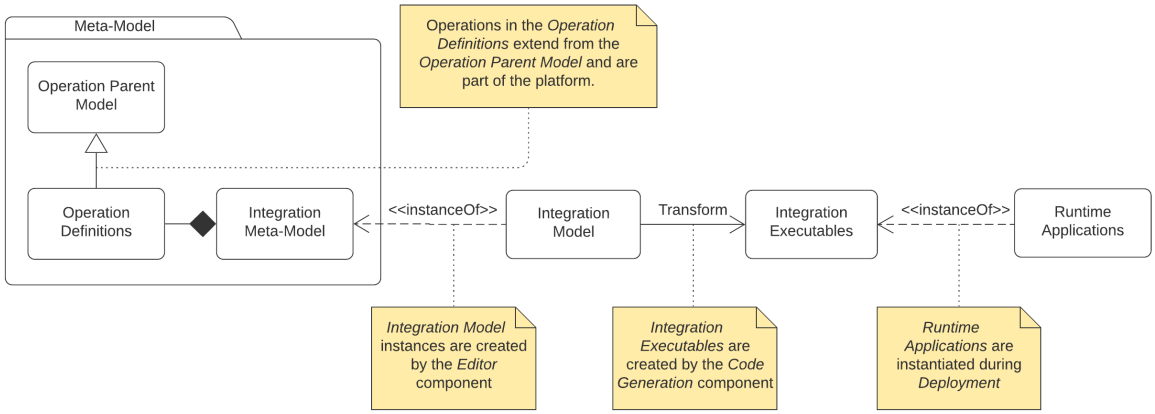


Figure 5.5: Model overview

5.3.1 Web UI

The users always interacts with the system through the *Web UI* application component. The *Web UI* component acts as a user interface for the deployment, management and schema services. For instance, consider the deployment function. The *Web UI* may show a web page with a 'deploy' button, as the user presses this button, the UI will call the Deploy service and adjust the UI based on its response.

Additionally, the *Web UI* hosts the *Editor Interface*. The *Editor Interface* is a standalone user interface that is tightly integrated with the *Editor* backend, this interface for instance includes the model editor and the catalogue of operations that the user can select and configure.

5.3.2 Data Model

The *Data Model* of the system describes the data objects used at the application level. It contains the *Integration Meta-Model*, as well as the integration executable (*ExecutableIntegration* object) and the used schemas (*Schema* object). A detailed version of the data model is shown in Figure 5.6. The *Integration Meta-Model* is based on the conceptual integration model proposed in Chapter 2 in Figure 2.4. Central in this model is the *Integration* object. Each *Integration* has a set of *Operation*s, which will be instances of vendor-defined *Operation*s as defined in the *Operation Definitions* (Section 5.3.3.1).

Each *Operation* has a location on the canvas and a name. Moreover, each *Operation* has one or more *Connection*s to other operations. Additionally, each *Operation* has a set of *Configuration Options*, which may be re-used across operations. Next, each *Operation* can access 0 or more schemas for use in the the in and/or output format. These *Schema* objects are stored in the *Schema Registry*, and consist of the schema definition or a type. Finally, each *Operation* can optionally use one or more *ConfigurationValue*s. Such a values for example store, or refer to, runtime variables to be used across operations.

Each *Integration* also has an *ExecutableIntegration*, that is the generated code to execute the integration. Actual implementations of this data model can have more classes, for instance for analytics, management, versioning and deployment, however, since these classes are implementation specific they have not been included in Figure 5.6.

5.3.3 Integration Development

The integration development module consists of three functions, *Editor*, *Validation* and *Code Generation* as well as the *Operation Definitions* component that defines all the operations that can be used in the *Editor*.

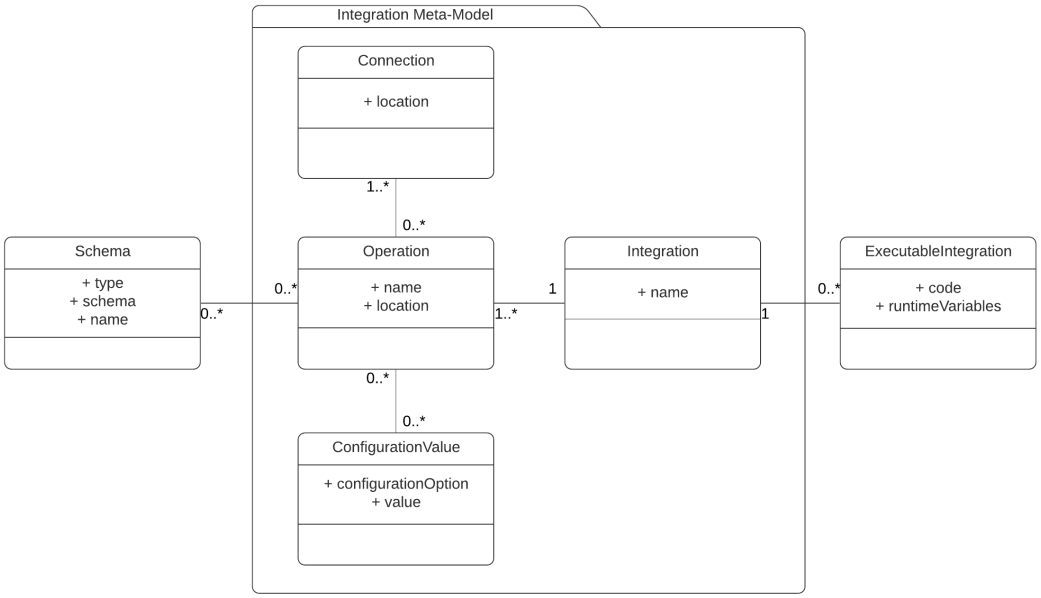


Figure 5.6: Data Model

5.3.3.1 Operation Definitions

The *Operation Definitions* component is an application component that contains all the operations that are supported by the *Editor*. The component is shared by all functions. For each *Operation*, the component defines 1) the behavior and options in the *Editor*, 2) the in-and-output for each *Operation* and 3) the code to be generated. The use of a common shared model containing all operations is a design pattern discussed in Paragraph 4.2.2.2 and promotes extendability and simplicity, as a single definition is used rather than definitions that are dispersed throughout the system.

Each operation in the definition should extend from an abstract *Operation* in the *Operation Parent Model* depicted in Figure 5.7. Central in the diagram are the abstract *Operation* classes (in *italic*). Each *Operation* in the *Operation Definitions* will extend from such an abstract operation. All operations have a name, and they have should implement a method that returns the code for the *Operation* based on its inputs and configuration options. The *Operation Parent Model* distinguishes three kinds of operations; data retrieval operations ('Source' either pull or push based), data output operations ('Sink'), and transforming data ('InOutOperation'). These three operations are based on the IoT integration model defined Chapter 2 (Figure 2.4). An *Operation* can inherit from either one of these operations.

Each of these operations can implement input and/or output requirements. They may specify a certain kind of in/output, such as a table or a stream. For instance, an average operation may require a stream input, and a table output type. The *IOType* variations are based on the dual Stream / Table model discussed in Section 4.2.1.2. In addition to a type, they can specify the format of the input and output. The *InputFormat* is always fixed, based on a schema that defines which inputs are supported, while the output format of an operation can dynamically depend on the input that is provided during design time. Each operation that produces outputs should be able to compute its output given its inputs and configuration options, or return null when the inputs are not supported. There are several patterns defined for computing the output type. For instance, the *AppendToInput* pattern describes how an output format mirrors the input, and adds a new attribute to it. All the alternatives have been adopted from StreamPipes, and are discussed in 4.2.2.2.

Additionally, each *Operation* can define a set of *ConfigurationOption*s. Several different kinds of options are identified such as a string input, float input and boolean input, as well as a dropdown value or an option that

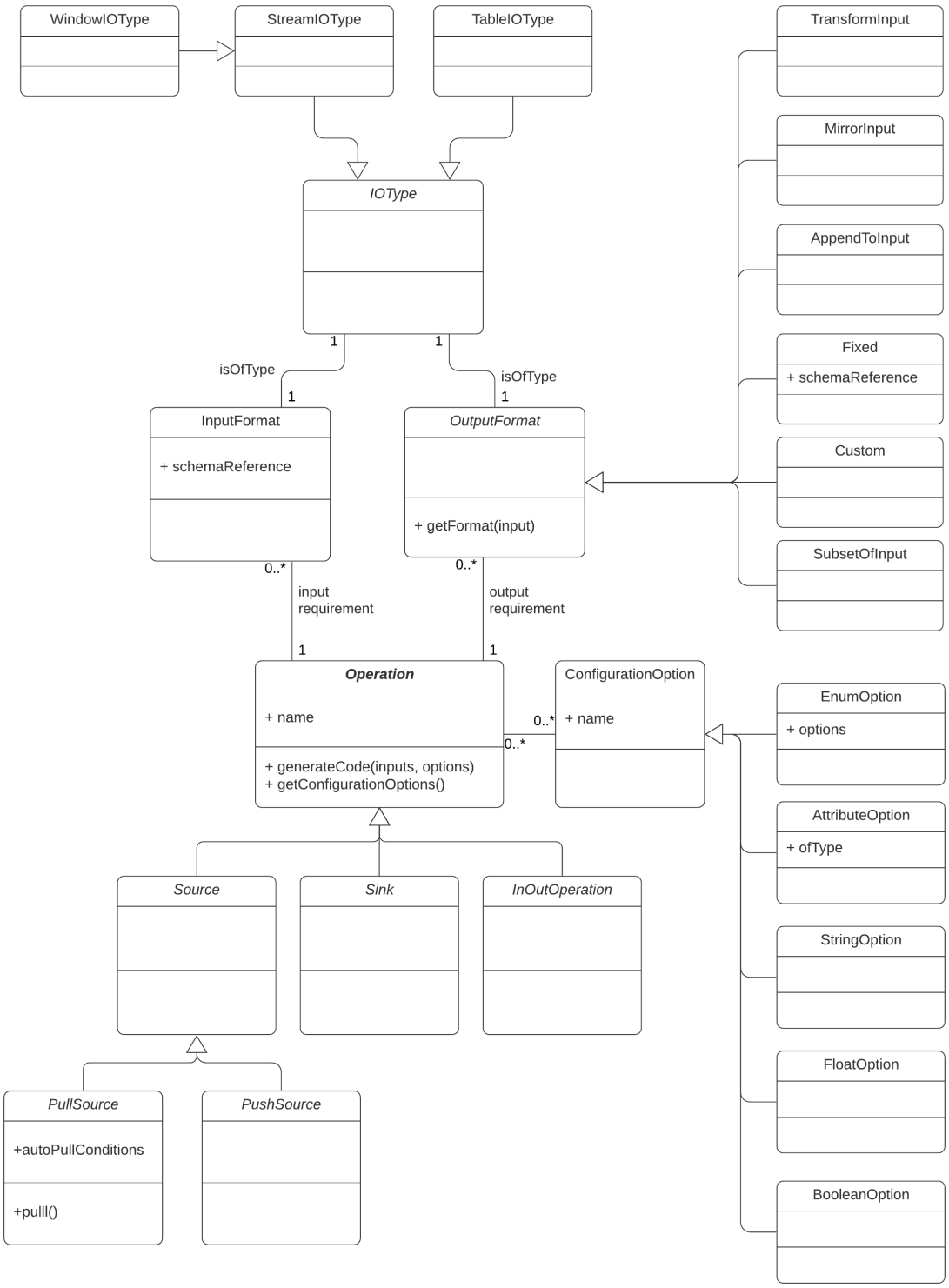


Figure 5.7: Operation Parent Model

allows the *Integration Developer* to select an attribute of the input data.

To ensure the validity of the model, it has been ensured that the model is able to represent all operations defined in Appendix B. Consider for example the 'aggregate' operation which can perform an aggregation over a window of time, such as the average value of a certain attribute, and store the latest aggregated value for each key as a variable in a table. This operation would be defined in the *Operation Definitions* component, and would conform to the provided parent class diagram. The operation would inherit from '*InOutOperation*' as it requires an input and produces an output, the *ConfigurationOption*s would be the attribute that should be aggregated and an enum for the kind of aggregation (for instance 'sum' or 'max'). The *generateCode* function would return the code that would be able to perform an aggregation, based on the configuration as per the configuration options. The operation would also have an *InputFormat* of type *WindowIOType*, as an aggregation requires a windowed stream over which is to aggregate. The *InputFormat* should be of type float, such that any schema containing a float attribute is allowed. The *OutputFormat* would be type Table. The *OutputFormat* would be fixed, since the operation always outputs a float.

The design does not specify which operations should be included in the *Operation Definitions*. It is recommended that at least the operations from Appendix B are considered. Additionally, for the sink and source nodes, it is recommended to implement connectors compatible with the typical IoT interfaces defined in Section 2.1.2. In some instances, the *Integration Developer* may want to perform a custom operation that is not included in the operation definitions. To account for this, the vendor should also add a generic *Operation* to the *Integration Definitions* which accepts any input, and any output. As *ConfigurationOption* the operation could have a text input that allows the user to enter custom code. The *generateCode()* method of this generic *Operation* could simply return this custom code. This is just one example, the vendor could also allow the user to define operations in a custom DSL, and then translate that DSL to executable Java code in the *generateCode()* function.

5.3.3.2 Editor

*Integration Developer*s can *Design Integration*s using the *Editor Interface*. The *Editor Interface* is a user interface for visually editing the integration model, this user interface can be implemented using a web-based diagramming tool. The nodes available for modelling, that is, the operations and configurations options that the user can use, are retrieved from the *Operation Definitions* component by the Editor function which serves as a backend for the *Editor Interface*. The *Integration Model* itself is synchronised between the *Editor Interface* and the Editor function. The model is first loaded from the *Editor* into the *Editor Interface* after initialisation. Any edits made to the model in the *Editor Interface* are synchronised to the *Editor* backend for validation, where the model is persisted. As discussed in the review (Paragraph 4.2.2.2) one can either adopt a visual model for the frontend and a separate logical model for the backend and translate between the two, or one can use a single unified model. The proposed design implements the latter using a serialiseable model that can be synchronised between the backend and the frontend, which simplifies the architecture and promotes extendability.

After committing the changes, the *Editor* will trigger the *Validation* and *Code Generation* functions respectively. These components are described in more detail in the sections below. After these functions are finished, the results are passed on by the *Editor* to the *Editor Interface* such that the *Integration Developer*s can see the validity of their designs, and get input-based configuration recommendations.

The *Editor* sequence diagram in Figure 5.8 visualises how the *Editor Interface* integrates with functions in the *Integration Development Component* through the *Editor* backend.

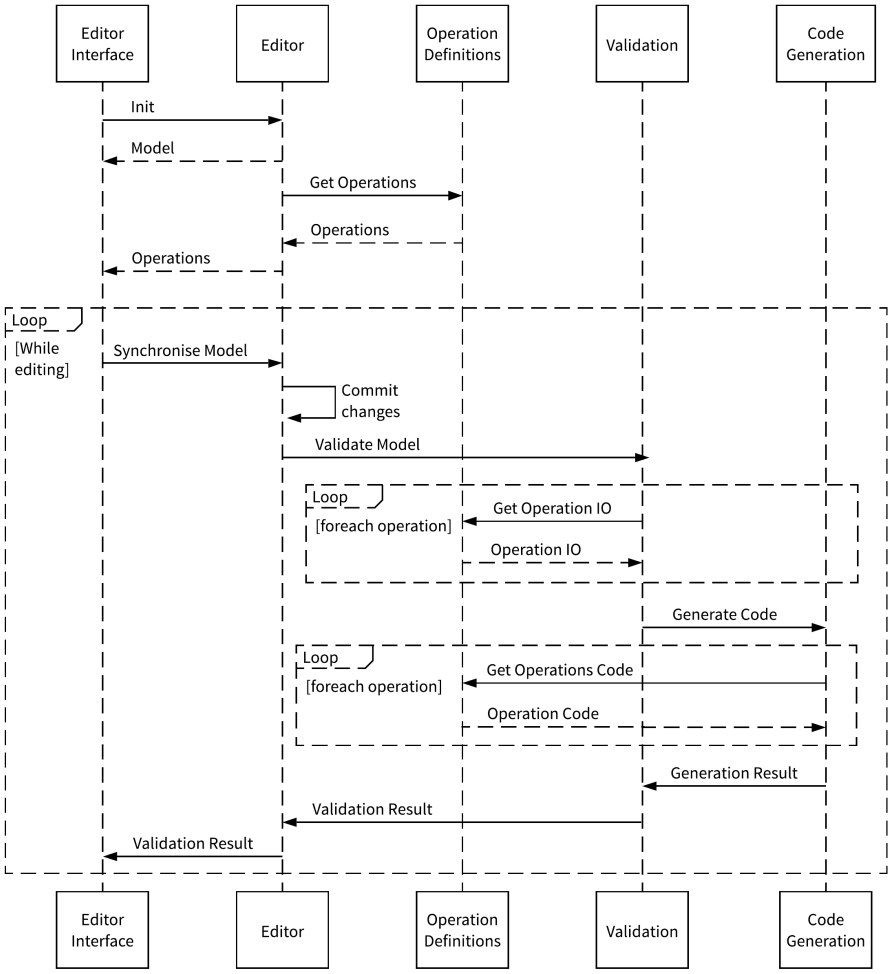


Figure 5.8: Editor sequence diagram

5.3.3.3 Validation

Any edits made in the *Editor* trigger the *Validation* function. As discussed in the review, three approaches exist towards validation to ensure validity of the model; primitive type checking, complex (schema based) type checking and and full ontology based semantics. The reference design will cover both primitive and complex type checking. Ontology based semantics are not included, however the design does not prohibit this so during implementations one can opt to include this. The choice not to include ontology based semantics is based on the findings in Section 2.1.2.6. Two key aspects form this decision, the domain-specificness of ontologies and the low maturity of ontologies in IoT. Ontology based interoperability has the highest payoff in predictable, domain specific environments, such that semantic operations can be developed and used to increase ease of use and validity of the operations. The proposed design is, however, to be used for data preparation across domains. The cost of identifying ontologies and developing domain-specific processors would most likely not out-weight the benefits. Furthermore, the use of ontologies for specifying IoT semantics is still in its infancy, is complex and difficult to use, and is predominantly applied in research contexts only.

The validation design is based the design discussed in Section 4.2.2.2; The *Validation* function will traverse all the *Operation* nodes in the *Integration Model*, starting with the data source node that marks the start of the graph. The *Validation* will compute the output of each node, based on its input and the output-format transformation of the node. When the output format has been determined, based on the input, the *Validation*

function will ensure that all inputs connected to this output are compatible. The *Validation* function will then proceed to the next node, which can now be validated as its input has been determined, until all nodes are validated. The *Validation* function will pass any validation errors back to the *Editor*, such that *Integration Developer*s will receive design-time feedback on the validity of the integration. Additionally, the derived inputs of each operator connected an output to an will be passed on to the *Editor* for such that recommendations for the configuration can be made depending on the input. For instance, when connecting a 'Average' operator to a temperate data source, this allows the *Integration Developer* to select which attribute of the input to aggregate.

5.3.3.4 Code Generation

The approach for code generation is heavily influenced by the API for which code is generated. As discussed in the review (Section 4.1) one can generally identify three API levels, a low level data flow API, a functional API and a declarative API. The survey shows that the declarative APIs, while easy to use, generally offer low expressively rendering them insufficient to cover all data integration use-cases. More suitable are the data flow APIs and the functional APIs. The functional APIs expose comprehensive, high-level, operations for data processing, while the low-level APIs provide direct access to the data and do not provide high-level operations. Therefore, the functional APIs are the most suitable for code generation, as operations in the model can be directly mapped to the method of the functional API.

After the model has been validated, the *Code Generation* function is triggered. Several approaches towards code generation exist. For instance, using languages facilitating model to code transformations such as Eclipse QVT and ATL (Atlas Transformation Language) [144]. However, none of the approaches towards code generation surveyed in Section 4.2.2 adopted such a language for model to code transformations. Presumably because, according to David et al. [145] such approaches are targeted at generating structural code rather than logical code for stream processing. Additionally, defining a single model for code transformation is complex, because this will depend on the specific frameworks and languages used during implementation. Instead, most graphical editors rely on a distributed node-based approach towards code generation (Paragraph 4.2.2.2). With such approaches, an code generation component traverses all nodes and requests each node to generate its own code. A similar approach is adopted for the *Code Generation* function. It first generates the boilerplate code common for all applications, such as the code to boot the *Runtime Application* and to connect to a *Kafka* cluster. Next the *Code Generation* function will traverse all nodes, and request each node in the integration model to return its executable code based on its configuration. Therefore, most of the logic with regard to code generation is stored per-node in the *Operation Definitions*. Oftentimes, a very basic approach is used for this, where static code is filled with the operations configuration variables and. When the code for all nodes has been collected, the *Code Generation* function embeds the operations code into the boilerplate code and ensures its validity. An example of such an approach is given with the prototype in Chapter 6.

5.3.4 Deployment

The *Deployment* function is accessed through the *Deploy Service*. Before any integration can be deployed, the *Runtime Environment* must be started on a *Client Runtime Host* after which it will connect to the *Vendor Backend* through the *Runtime Interface*. If a *Runtime Environment* is running, deployment instructions can be sent from the *Deployment* function to the *Runtime Environment* where integration should be deployed. The deployment instructions should contain all the information that the runtime needs to run the integration, such as the version, runtime parameters and the resources required. Based on the deployment instructions, the *Runtime Environment* will connect to the backend to retrieve the *Integration Executable* from the *Code Generation* component. This is a continuous process, such that at any point in time the runtime can retrieve the latest *Integration Executable* based on its deployment instructions, for example to automatically receive bugfixes, without the need to re-deploy the integration. Deployed integrations (*Runtime Applications*) will send metrics to the *Management*. Overall, the deployment process is visualised in Figure 5.9. Depending on the implementation, the *Deployment* function could also take on additional responsibilities such as version

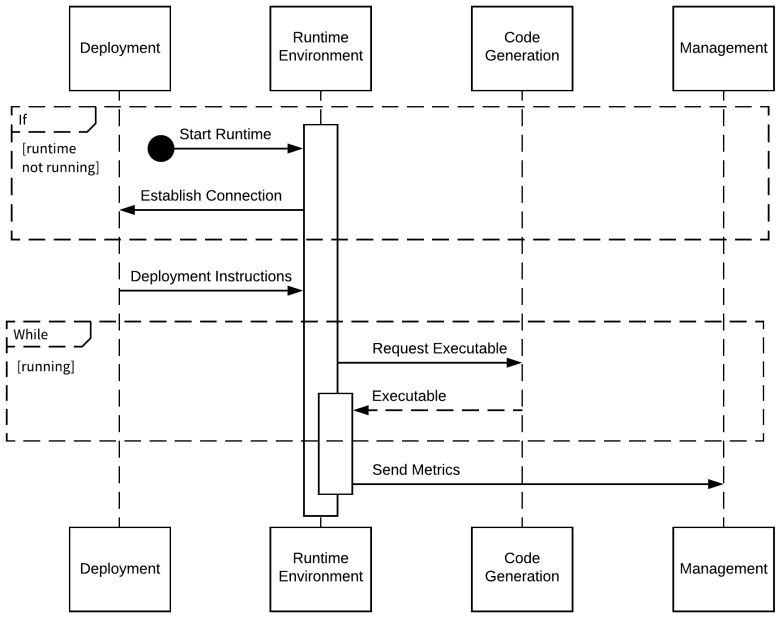


Figure 5.9: Deployment sequence diagram

management and migrations. The architecture for the *Deployment* and *Management* functions, as well as the runtime are based on the architecture of integration platforms as discussed in Section 2.1.2.8.

5.3.5 Management

The *Management* component is concerned with supporting management tasks, such as monitoring the deployed integration by collecting metrics. The scope of the metrics collected will vary per implementation, as described in Paragraph 4.2.2.2. Examples of metrics include; (error) logs, runtime metrics and data metrics. Logs are warnings and outputs produced by the infrastructure and processors. Runtime metrics include statistics about the runtime applications, such as throughput, resource usage and processing time. Logging and runtime metrics are key metrics that should be implemented to allow the *Support* team to review the integration for any unhanded errors and unexpected events, and to ensure that the integration is properly scaled. Finally, data metrics are less essential, and are more analytical in nature, giving insights in the data that is passed through the integration. For instance, to observe what kind of data is most frequently processed. In addition to collecting metrics, the *Management* function allows the *Support* team to perform basic interventions, such as the ability to restart or redeploy integrations.

5.3.6 Schema Manager

The *Schema Manager* allows the *Integration Developer* to maintain a register of all data formats used by data sources, data sinks and data operations. These data formats are used to deserialise serialised data into data structures that can be understood by the processor, as described in detail in Section 4.2.1.2. *Developers* will store the data schemas in the registry, and during design, the *Developers* can reference to a schema stored in the registry for the in/output of an operation. Behind the scenes, the *Schema Manager* stores all schemas in the *Schema Registry*, and shares only a reference of the schema to the *Editor* such that the integration can obtain the schema from the registry during runtime. The *Schema Registry* is discussed in more detail in Section 5.4.

The exact functionality of the *Schema Manager* can vary per implementation. Existing integration platforms

often already provide a schema management functionality. Such vendors may opt to automatically generate Kafka-compatible schemas from their existing schema manager and store them in the registry.

5.3.7 Runtime Applications

The *Runtime Application* is a running instance of the *Integration Model*, which is instantiated during deployment. Therefore, there are many *Runtime Applications* running at the same time, one for each running integration on the integration platform. Additionally, there may be multiple instances per *Runtime Application* to provide scaling. All stream processing frameworks discussed in the review support horizontal scaling, simply by launching multiple instances of the *Runtime Application*.

The *Runtime Application* may run as a standalone application, however, it may also run embedded in another application. Most integration platform vendors already have an existing runtime applications in charge of executing message integration flows, and the stream processing applications can run embedded as part of the existing runtime applications.

5.4 Infrastructure layer

5.4.1 Vendor Cloud

The *Vendor Cloud* hosts the *Vendor Backend* applications. Additionally, it provides a database for storing persistent data of the vendor backend, such as integration models, metrics, and user information, and other metadata. Typically, the vendor cloud is hosted by an Infrastructure as a Service (IaaS) provider, which allows dynamic scaling of the infrastructure depending on the number of integrations that need to be supported.

5.4.2 Client Runtime Hosts

The *Client Runtime Hosts* support the execution of *Runtime Applications*. Each client can have multiple hosts, such as on-premise hosts and cloud based hosts, and each host serves a *Runtime Environment* to which *Runtime Applications* can be deployed. This *Runtime Environment* provides:

- An execution environment that allows the execution of *Runtime Applications* regardless of the underlying infrastructure. This allows on-premise deployments, as well as cloud-based deployments and hybrid combinations of both.
- A mechanism for remotely managing the runtime, including the ability to submit integrations to the runtime, start them, and collect metrics. The layered architecture depicts this as the *Runtime Interface*.
- A dependency manager, that can fetch any dependencies needed and facilitate the re-use of dependencies among *Runtime Applications*.

How the *Runtime Environment* is implemented will vary per implementation. An example of this is OSGI, which is a standard for running software modules on Java runtimes, which can be used to re-use services among *Runtime Applications* and to remotely control the deployment of *Runtime Applications* within the OSGI execution environment [1].

The design describes the use of a single runtime per host, which runs all *Runtime Applications*. This choice was made to allow the sharing of resources and to reduce overhead compared to containerised approaches such as Docker [146]. Since all *Runtime Applications* require identical runtime environments and share resources, full containerisation would bring little advantage. Additionally, most of the benefits of containerisation approaches, such as dependency management and clustering support, are available also with non-containerised runtime solutions such as OSGI frameworks.

Like *Runtime Applications*, the *Runtime Environment* can easily be scaled. By adding more *Runtime Hosts* and

launching the *Runtime Environment* on these hosts, or by vertically scaling the capacity of a single *Runtime Hosts*, more *Runtime Applications* can be deployed to facilitate the horizontal scaling of *Runtime Applications*.

5.4.3 Streaming Cloud

The *Streaming Cloud* hosts the streaming infrastructure. Therefore, this cloud provides the *Kafka* cluster used by all *Runtime Applications*. The cluster consists of various brokers, and the cluster can be scaled indefinitely by adding or removing brokers. The vendor may host the cluster, but one may also opt to use a hosted Kafka as a service provider. Regardless of how *Kafka* is hosted, clients can connect to any broker in the cluster and use the Kafka discovery protocol to obtain metadata of the cluster to find out to which broker to connect for which topic. Consumers and producers can then open a connection to this broker to exchange data. This process of discovering and connecting to a broker is usually handled by the stream processing framework, all of the processing frameworks discussed in Section 4.2.1.1 support this. Note that the only applications that must exchange data with *Kafka* are the *Runtime Applications*. Optionally, the backend could also connect to *Kafka*, for instance to create / manage topics, to read topics for configuration suggestions on design-time, or to collect metrics about the cluster in *Management*. This is however optional and depends on the implementation, as alternative approaches, such as runtime topic creation, are available.

5.4.4 Registry Cloud

The *Registry Cloud* hosts the *Schema Registry*. The use of a *Schema Registry* is common with Kafka, but not mandatory. An alternative to the use of a registry is to distribute the schemas with the executables. At first sight, this may seem like an attractive approach since changing schemas will require changes to the processors and consumers as well. However, as schemas change, this would requiring the schemas used by all producers and consumers at the same time. A *Schema Registry* allows multiple versions of a record to be in flight, and can deserialise each record with the right schema while ensuring compatibility between the schema of the recipient and the sender. Other benefits of a *Schema Registry* include automated translation from schema formats and decoupling the code bases and the schema management.

Chapter 6

Prototype

In this chapter, the development of the prototype in the problem context is described. This prototype is used to validate the design with end-users in Chapter 7 and therefore the problem context is that of the vendor with whom the design is validated described in Section 1.5. To this end, the design is applied to the architecture, requirements and technologies of the vendor. The resulting prototype is a novel model-driven interface for generating Java Kafka Streams stream processing integrations.

6.1 Methodology

Wieringa describes the use of a prototype for single-case mechanism experiments, as are used in this research for validation [12]. However, Wieringa does not describe how such a prototype should be developed, therefore, a separate methodology is used to guide this phase. Specifically, an Agile methodology is used to guide the development of the prototype.

For the development of the prototype, the Scrum methodology is used. Scrum is an Agile methodology to effectively and flexibly develop artifacts, most often applied in software engineering. Core to Scrum is the Scrum Team, which consists of the Product Owner, the Development Team and the Scrum Master. The Product Owner represents the stakeholders and specifies and prioritises the requirements in the form of user stories on a list named 'the backlog'. The Development Team is responsible for developing the artifact in iterations. Each iteration is named a 'Sprint' and the Development Team plans items from the backlog into Sprints. At the end of each Sprint, there should be a working increment of the artifact. The Scrum master ensures that the Scrum rules are being followed, and is responsible for supporting the team in doing so.

Guiding the development of the prototype using this Agile methodology allows for continuous feedback on the design choices made through constant validation. This reduces the cost of change, and increases the chance of success as any possible design flaws can be established early on in the project. Other benefits include an increase in transparency by continuously managing expectations, a reduction of risk by identifying and responding to risks early on in the project and flexibility in development allowing the requirements to change when needed [147]. Overall, the use of the Agile methodology ensures that the resulting product meets the stakeholder goals.

For this project a lightweight version of Scrum is used. In the first phase, the Discovery phase, the design is translated to a high level system architecture. The results of this phase are discussed in Section 6.2. Next, in Sprint zero the system architecture is divided up into stories and designs, and these stories are assigned to Sprints based on their priority for the prototype. The stories are validated by the Product Owner from the vendor. This results in the minimal scope for the prototype, limiting the prototype to certain architecture components and requirements of the overall design. Items from the backlog are implemented in 1 week Sprints, and at the end of each Sprint the results are presented to the Product Owner and to the vendor's

Development Team. The results of Sprint zero are discussed in section 6.3.

6.2 System Architecture

The first step towards building the prototype is a applying the design to the problem context. That is, showing how the design could be implemented in the context of the vendor, by applying the design to the vendors eMagiz iPaaS platform.

Some architectural components of the design, such as the *Vendor Backend* and the *Integration Management*, already exist in the current architecture of the vendor to support the messaging integrations and would need only minor adjustments support stream processing functionalities. Most architectural components, however, would need significant changes, or do not exist in the current architecture. These components are marked in green in Figure 6.1. Additionally, Figure 6.1 shows the specific technologies used to realise some of the components.

6.2.1 Business layer

The Business layer has the same number of process steps regardless of the type of integration that is built, be it a messaging integration or a stream processing integration. The differences are within the process steps themselves. During the specification phase, the *Architect* needs to determine whether stream processing or messaging technology is more suitable to support the integration. Next, during the *Design Integration* phase, the *Developer* needs to be familiar with the stream processing patterns, such as tables, streams, topics and aggregation, to design the integration. Also during *Manage Integration* and *Deploy Integration*, knowledge about stream processing is needed to estimate the required deployment parameters and to recognise stream processing specific errors and problems. Therefore, proper training is needed to allow existing end users to built stream processing integrations. This is in addition to a proper user interface which supports users throughout the development and management of an integration.

6.2.2 Application layer

As discussed in Chapter 5, the end users interact with the system through the *Web UI*. With the exception of the *Stream Editor Interface*, the user interface is highly similar for both stream processing and messaging. The user can use the existing schema management functionality, the CDM, to manage schemas for stream processing. Behind the scenes, the *Vendor Backend* can then convert the schemas for stream processing to the required format and submit them to the *Schema Registry*. For *Deployment*, only the deployment parameters will be different, for instance to specify Kafka related parameters, while the remainder of the UI will be similar. The same holds for *Integration Management*, while the reported information will be different as it may contain Kafka specific metrics and logs, the UI will otherwise be similar. Therefore, both *Deployment* and *Management* will be handled by the vendor's existing components for this.

The most significant changes in the Application layer are made in the *Editor Interface* and the underlying *Integration Development* component. The current *Editor Interface* and its backend are designed by the vendor for the development of messaging integrations. Although there are similarities between stream processing and messaging, the current UI and backend do not support all stream processing specific concepts such as table-stream duality, aggregations, joins, topics and more. Therefore, either a new *Stream Editor Interface* is needed or the existing *Editor Interface* needs to be extended to support the development of stream processing integrations. Such extension would include the ability to model nodes with multiple in/outputs, to model the difference between streams and tables, and to extend configuration options with UIs for specifying stream specific operations such as joins and aggregations. The *Stream Editor Interface* is embedded or integrated into the vendors *Web UI*. Next, a backend component is needed that can support the UI, and trigger *Validation* and *Code Generation*. Both *Code Generation* and *Validation* need to be re-implemented as the vendor's current

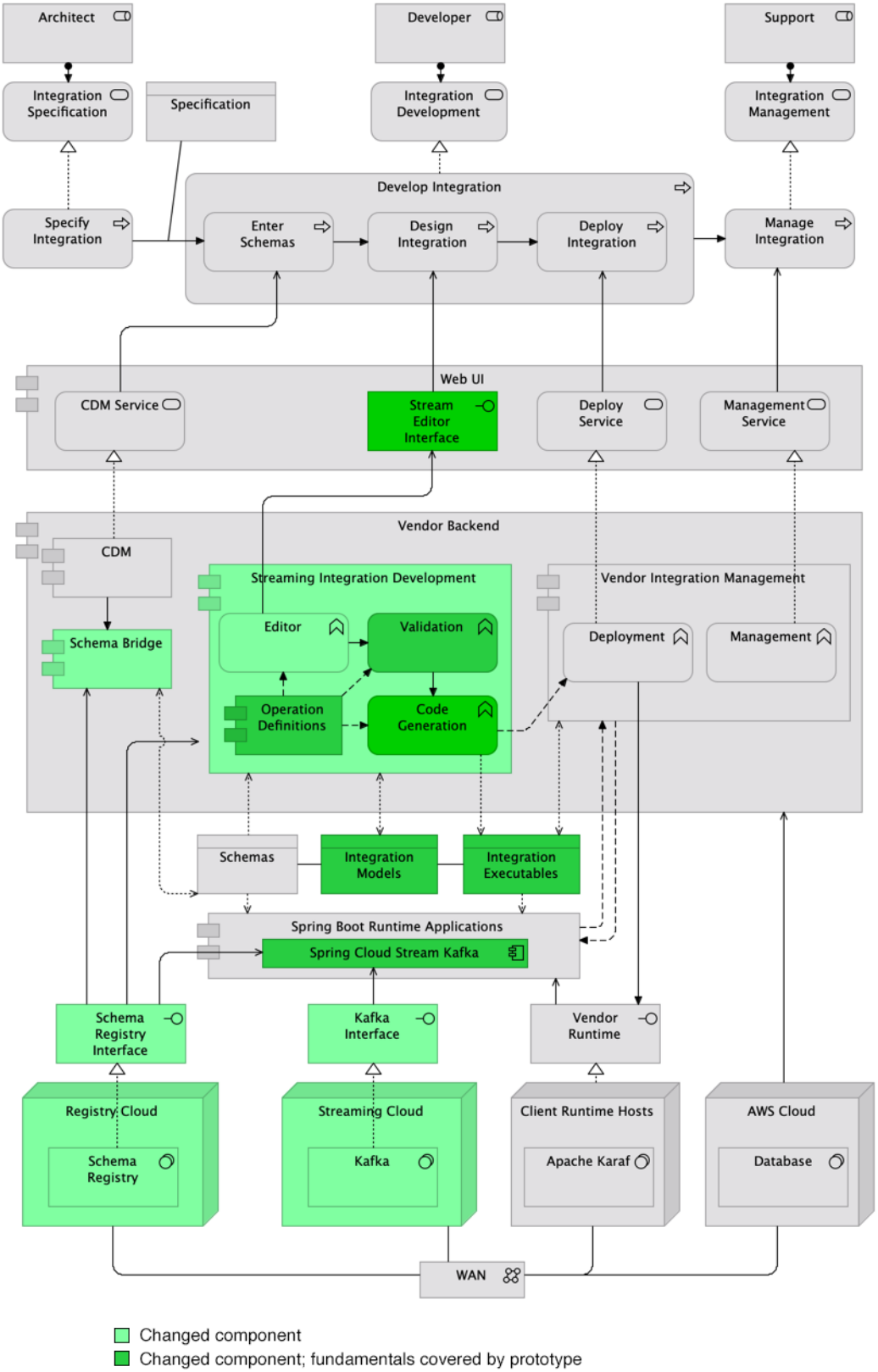


Figure 6.1: Design applied to the vendor's current architecture

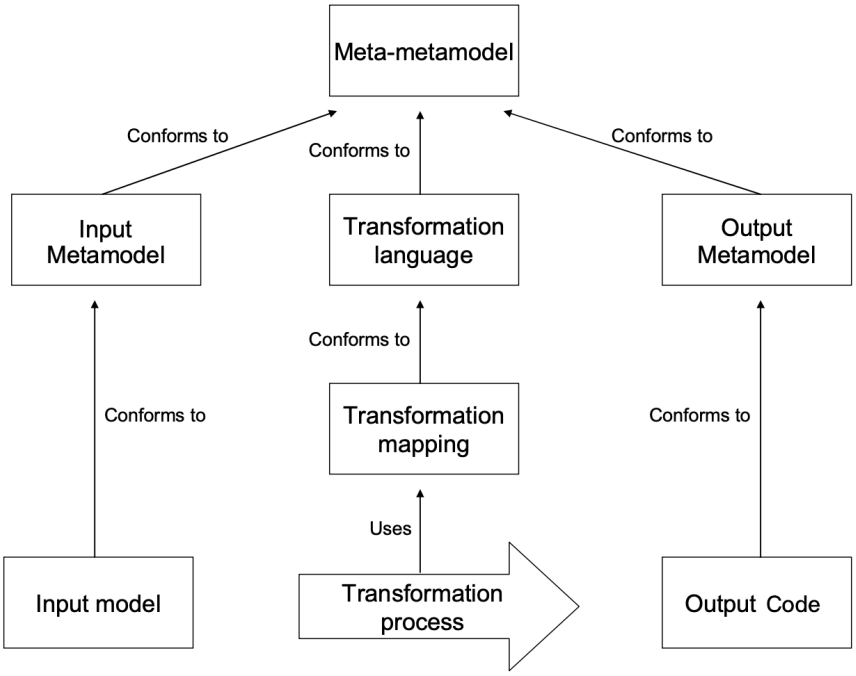


Figure 6.2: Typical Model Driven Architecture

architecture does not support the generation of Java code and complex type checking. Additionally, a new shared *Operation Definition* is needed to define stream processing operations and the validation logic and code generation instructions per operation. To determine the operations that should be included in the definition, the operations from Appendix B can be considered, as well as the findings of Section 2.1.2 which describes the specific protocols and formats to support in IO nodes.

Finally a stream processing framework needs to be adopted within the *Runtime Applications*. For this *Spring Cloud Stream Kafka Streams* is used. This decision is based on the results from Section 4.2.1.2 as the vendor currently relies heavily on the Spring ecosystem, using Spring Boot and Spring Integration to power all their runtime applications. As Spring Cloud Stream Kafka Streams relies on Kafka Streams, this framework combines the reliability and functionalities of Kafka Streams with the ecosystem of Spring.

Consider Figure 6.2 to get an overview of what decisions in the transformation are guided by the design, and which decisions have been made specifically for this prototype. The design provides guidelines, but no concrete implementation, for the transformation process, and the input metamodel and the transformation mapping. The output metamodel is not described by the design, and depends on the stream processing framework that is used in the implementation. For this prototype, the framework is Kafka Streams and therefore the output metamodel is the Kafka Streams Java DSL. The input metamodel and the transformation mapping are both covered by the operation definition, as is described in Section 6.4.2. This is conform the guidelines of the design (Section 5.3.3.1) that specifies that the definitions should provide both the meta-model as well as the transformation definition (code generation logic). The transformation language is Typescript, since this is the language used for developing the prototype and defining the input metamodel.

6.2.3 Infrastructure layer

In the Infrastructure layer, two key components should be added to the existing architecture. A *Schema Registry*, to host the schemas for the runtime stream processing applications and a *Kafka* cluster to provide the streaming infrastructure that the runtime applications need to communicate. Additionally, the System Architecture shows

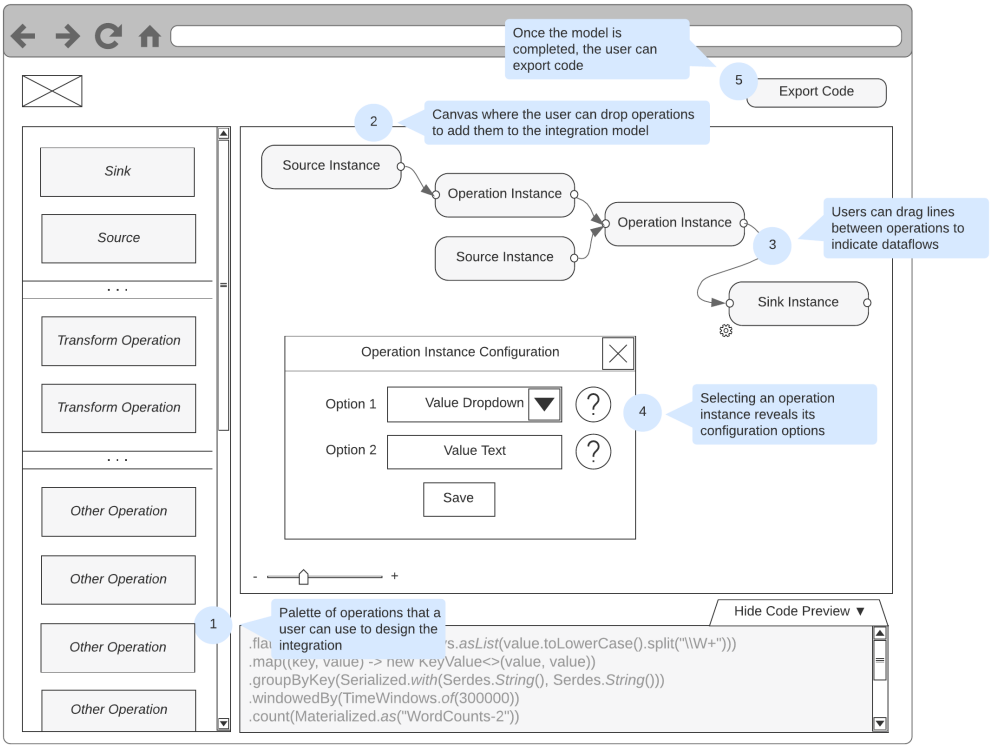


Figure 6.3: Prototype Wireframe

that the vendor's client runtime is implemented using *Apache Karaf*, which is an OSGI implementation that can be used to host Java applications, such as stream processing applications. The backend itself is hosted using Amazon Web Services.

6.3 Plan

Scope Given the timeframe and scope of this research, it is not feasible to cover the whole design with the prototype. Therefore, a minimum viable prototype (MVP) is developed that is able to showcase key aspects of the design for validation and demonstration purposes. It has been decided together with the product owner that the prototype should cover at least R1 and R6 to demonstrate the designs ability to facility the graphical design of stream processing integrations. Requirements R3, R4 and R5 are also covered since the prototype will generate code for consuming from Kafka using an embedded processor, and since the editor runs in the browser. The remainder of the requirements are not, or only partially, covered by the prototype. For instance, the prototype assumes that the data conforms to some schema that is familiar to the user, but it does not provide any explicit frontend or backend support for schemas as per R8. Overall, only the *Stream Editor Interface*, *Operation Definitions*, *Validation* and *Code Generation* elements of the design are implemented in the prototype. Additionally, the prototype will feature *Integration Models*, *Integration Executables* (the generated code) and a component in the runtime to support the execution of *Integration Executables*. All components covered by the prototype are highlighted in dark green in Figure 6.1. Again, given the scope, only the essential functionalities are implemented for each component. For instance, the *Operation Definitions* in the prototype covers only a subset of the operations and options that would be present in a production implementation, the *Validation* component will only perform type validation, and the *Code Generation* will only produce executable statements rather than executable applications.

User stories A full breakdown of all functionalities supported by the prototype is provided in Appendix D in the form of user stories. These user stories are created based on the design, the scope of the prototype, and on the (semi)fictional use cases listed below. All user stories have been assigned an expected workload and a sprint based on their dependencies. The total workload of all user stories amounts to 183 hours. Since the recommended time for the development of a prototype is set at one month by the external supervisors of this thesis, the expected workload matches the recommended workload. The UI related user stories are shown in Figure 6.3, which provides a mock-up of the prototype UI. This wireframe is based on the findings in Paragraph 4.2.2.2.

Use cases To provide context to the user stories, two use-cases, one fictional and one semi-fictional, are defined. The semi-fictional use case is based on a use case from the consultancy firm in Interview A.4. Both use cases describes the skills and goals of the persona of the Integration Developer used in the user stories. Additionally, both use cases provides tangible examples and context to the operations defined in the user stories.

Use case 1: An integration developer seeks to develop a stream processing integration for stream of temperature measurement data. This stream of measurements is produced by a set of temperature sensors with different locations that continuously produce measurements containing the measured temperature, and the ID of the current location of the sensor. The goal of the integration to be developed is to process this raw stream of incoming events, and to create an alert when the average temperature over a period of time exceeds a threshold value, for a certain location. These alerts are picked up by a dashboard application from an output topic. To achieve this goal, the developer would like to read the data from the topic, transform it into a generic data format (such that support for other input formats could be added in the future). Then, he would like to group the measurements by location, and compute the average temperature for each location for a certain period of time. Next, he would like to enrich the data by replacing the location ID with the actual location information data, such as the name of the location, stored on a lookup topic. Finally, he would like to send a stream of alerts to an output topic for each average measurement that exceeds a threshold value. The developer creating the integration is familiar with the concepts of stream processing and integration in general, such as schema based data transformation, however, he has no knowledge of Java or other functional programming languages. Therefore, he would like to design the integration visually using a model-driven interface to generate the executable code that he can embed in a runtime application.

Use case 2: An integration developer seeks to develop a stream processing integration for a stream of RFID sensor measurements. These RFID sensors are located in package delivery vans that contain packages with RFID tags. When the driver of the van reaches a delivery address, he will open the door and remove the package from the van. Each time a door is opened or closed the RFID sensor will scan all the tags inside the van and send their IDs, as well as the latest GPS location, as a stream of updates. Using this stream of information, one can determine which packages (RFID tags) are removed from the van at which locations by tracking the last known location of each RFID tag. However, since the sensors are not 100% reliable, it can happen that a tag is not scanned by the sensor while it is still in the van. Therefore, the goal of the integration is not only to track the last known location for each of the tags, but also track the update history and a list of all location updates to allow applications to determine the reliability of the measurements. To achieve this goal, the developer would like to read the data from the topic, and store the location update event time in data storage, then he would like to flat map the location update into a new record for each RFID tag. Next, he would like to group the updates by the RFID tag, and track both the last location for each tag, as well as aggregate a list of all location update times for each tag. Finally, he would like to join both aggregates into a single record for each RFID tag and write it to a data storage. The developer has the same experience and skills as the developer from use case 1.

After prototype completion, the integrations for both of the use-cases were successfully implemented using prototype. Mock data producers were used to simulate the production of IoT data as described in the use case.

The deployed integrations then consumed the input data, processed it, and produced output data as described in the use case.

Technologies The prototype is designed as a standalone product. That is, it is not integrated into the vendors currently architecture. This choice was made, as the vendors current architecture is complex, and extending it would be beyond the scope of the prototype. Therefore, the prototype is not bound to the vendor's technology stack and any technology can be used to implement the prototype. The only restriction is that, as per the requirements, the prototype should generate executable code compatible with Spring Stream Kafka Streams. The Web UI is based on Spring Flo[1], which is a framework by Spring to develop model-driven web-applications. This framework provides a UI for dragging nodes onto a canvas, and connecting them, to create a model. Additionally, Spring Flo provides APIs for validating and parsing the model and it provides an API for defining the supported nodes. For the runtime processing, Spring Stream Kafka Streams is used, which essentially means that Kafka Streams statements are generated, which can be embedded in a Spring Stream Kafka Streams application to bind to the cluster. As discussed in Section 4.2.1.2 (de)serialisation needs to be supported to perform operations on the records data structures. Typically, in Kafka Streams this is implemented by (de)serialising from/to Java Object classes. However, for this prototype such an approach is not only infeasible, since the schemas of the data are unknown, but it would also be cumbersome since generating the code to transform JSON Objects is significantly less complex than generating the source and target Java Object classes, and then generating logic to transform between these objects. Therefore, this prototype will query JSON objects directly using JsonPath[2]. JsonPath can query JSON Serialized data structures directly, without the need for parsing JSON into native data structures. The queries to read attributes from the data can be inferred by the user based on a schema of the data. In addition to querying data from JSON, JsonPath also allows for basic data manipulation for instance by renaming, adding or removing attributes. For advanced JSON transformations, JSLT is used [3] which allows users to define complex transformations from and to any JSON schema.

The development environment consists of the Intellij IDEA integrated development environment, which is used to develop and deploy the Editor, and the Confluent platform. The Editor, as it based on Spring Flo is written in Javascript and deployed using NPM (Node Package Manager) as a web application. The runtime applications are written in Java and use Spring Boot to be executed. Finally, the Kafka cluster that the runtime applications connect to is based on the Confluent Platform. Confluent Platform was selected, as it is an easy to deploy local Kafka cluster with a built-in interface for managing statistics, logs, topics and more. There is no actual dependency on the Confluent Platform, and the runtime applications would also work with any other Kafka installation.

6.4 Results

6.4.1 Application design

The prototype is designed as an Angular.js application. Angular.js is a popular web application development framework to build client-side JavaScript applications. Most Angular applications, such as this prototype, are developed TypeScript, which is compiled to JavaScript at build-time. The application relies on the Angular Spring Flo directives for all graphing functionalities. Spring Flo in turn uses Joint.js for rendering the graphs, and provides a wrapper that can be used to easily embed the graphical editor, define model elements, and retrieve model instantiations in an application.

The main function of the prototype is to embed the Spring Flo editor. This editor provides the UI for the palette and the canvas to drag operations on, and connect them. Beyond the UI of the Spring Flo diagram editor, the

[1] `https://github.com/spring-projects/spring-flo` consulted 07-07-2020
[2] `https://github.com/json-path/JsonPath` consulted 07-07-2020
[3] `https://github.com/schibsted/jslt` consulted 07-07-2020

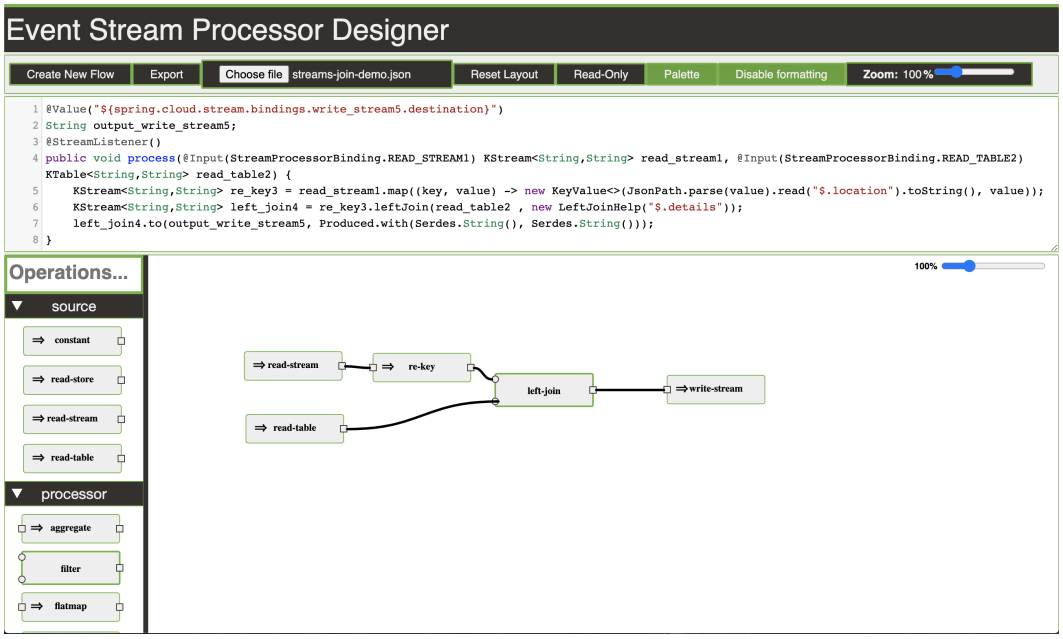


Figure 6.4: Stream Editor Interface

UI contains logic for showing a configuration dialog to set the operation properties based on the operation properties. It also contains a code viewer based on CodeMirror, which shows the generated code with highlighting. Additionally, the UI contains logic for importing and exporting integration models. The export functionality exports all operations and links on the canvas to a JSON file, and upon importing the import functionality programmatically iterates over all objects in the JSON file to re-create the graph using the Spring Flo Render service. The prototype extends Spring Flo using a Metamodel extension for the operation definitions and code generation, and an Editor extension for the validation. Each of these extensions are discussed in detail in the remaining sections. These extension functions are triggered directly from the UI, since there is no Editor backend component in the prototype implementation as there is no need to commit the integration model to a server.

6.4.2 Operations

The operations available to the user are defined in the Metamodel definition, which represent the 'Operation Definitions' application component. This allows operations the be defined in a centralised location, and in a modular fashion.

Each operation has the following attributes:

- A name, which allows users to identify the operation.
- A group, which defines the kind of operation. For instance, whether it is a source, a sink, or an input-output operation.
- A list of configuration options. Each configuration option has:
  - A title, which allows users to identify a configuration option.
  - A description, which allows users to understand what the option does.
  - A default value, which is used if the user does not provide a value.
  - A requirement, which indicates if the option is required or not.
  - A value type, which is either String, Number, Enum, Code, or Boolean.

- An output type, which is a logical definition that allows the operation to compute its output, depending on its inputs and its configuration options. The operation will return null, if it can not provide an output based on its inputs or properties.
- A literal, which is the code to execute the operation. Similar to the output type, this is based on a logical definition that generates the code based on the inputs and the properties.

This single definition is used throughout the application. The editor UI uses this to the operations in the palette, and to dynamically generate a UI for the configuration options, see for instance Figure 6.5. The code generation uses these definitions to generate code, and the validation uses the output types to validate the connections between operations. Appendix E provides an overview of all the operations included in the prototype.

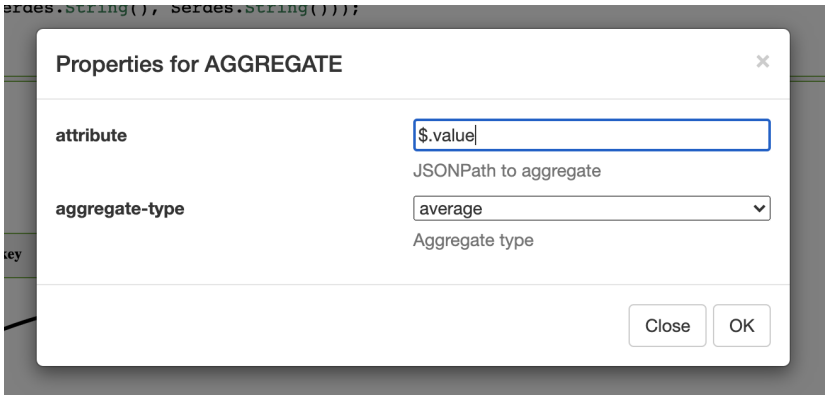


Figure 6.5: Example of operation configuration options

6.4.3 Validation

The first task of the validation component is to retrieve a sorted list of all instantiated operations. This sorted list is based on the dependency graph between operations, and each operation on the list is at a lower position that any of its inputs. To obtain this list, a small algorithm is developed. First, all the source nodes (operations without input) are added to the the list. Next, for all non-source nodes, the algorithm will select a random node of which all inputs are already on the sorted list and add it to the end of the list. This is repeated until all nodes are added to the sorted list.

Now that the sorted list has been created, the validation component will iterate over the list and compute the output for each operation based on its inputs and configuration, using the logical definition provided by the operation definition. The validation component will then push the computed output back to the UI, if the output can not be computed a validation error is pushed to the UI instead. Figure 6.6 shows how the output types and validation errors are shown to the user, if a *KTable* (read-table) is provided to the group operation, a *KGroupedTable* is the output, if a *KStream* (read-stream) is provided, *KGroupedStream* is the output, if a *KGroupedStream* is provided to the group operation, an error is given since this type is not supported as input for the group operation.

In addition to type validation, the validation component also validates whether all required fields have been met, and whether an appropriate number of inputs is used with the operation. All operations require exactly 1 input, with the exception of multi-operations (such as left-join) which support two inputs and sources which support zero inputs. Import to remark is that there is no schema or semantic based validation, due to the limited scope of the prototype. However, such a functionality could be implemented similar to the type validation.

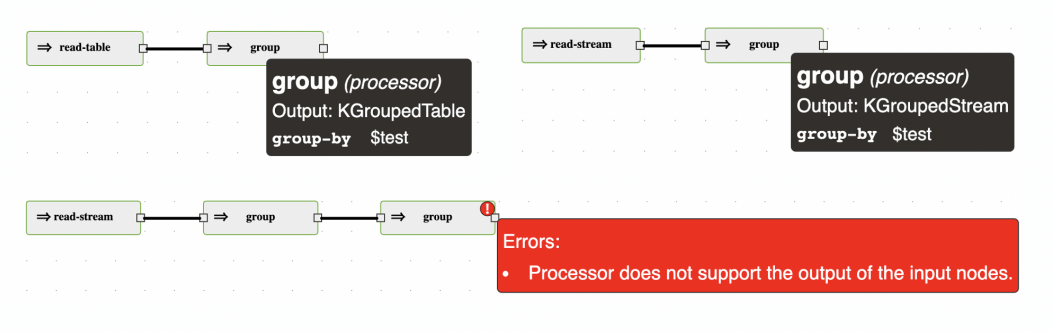


Figure 6.6: Type validation

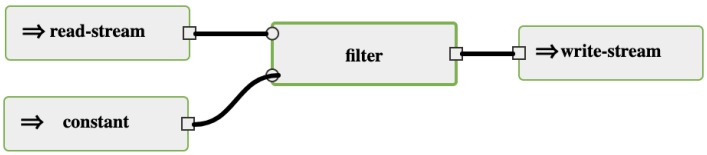
6.4.4 Code generation

The code generation component will re-use the methods from Validation used to created a sorted list of operations, and to compute the output type for each operation.

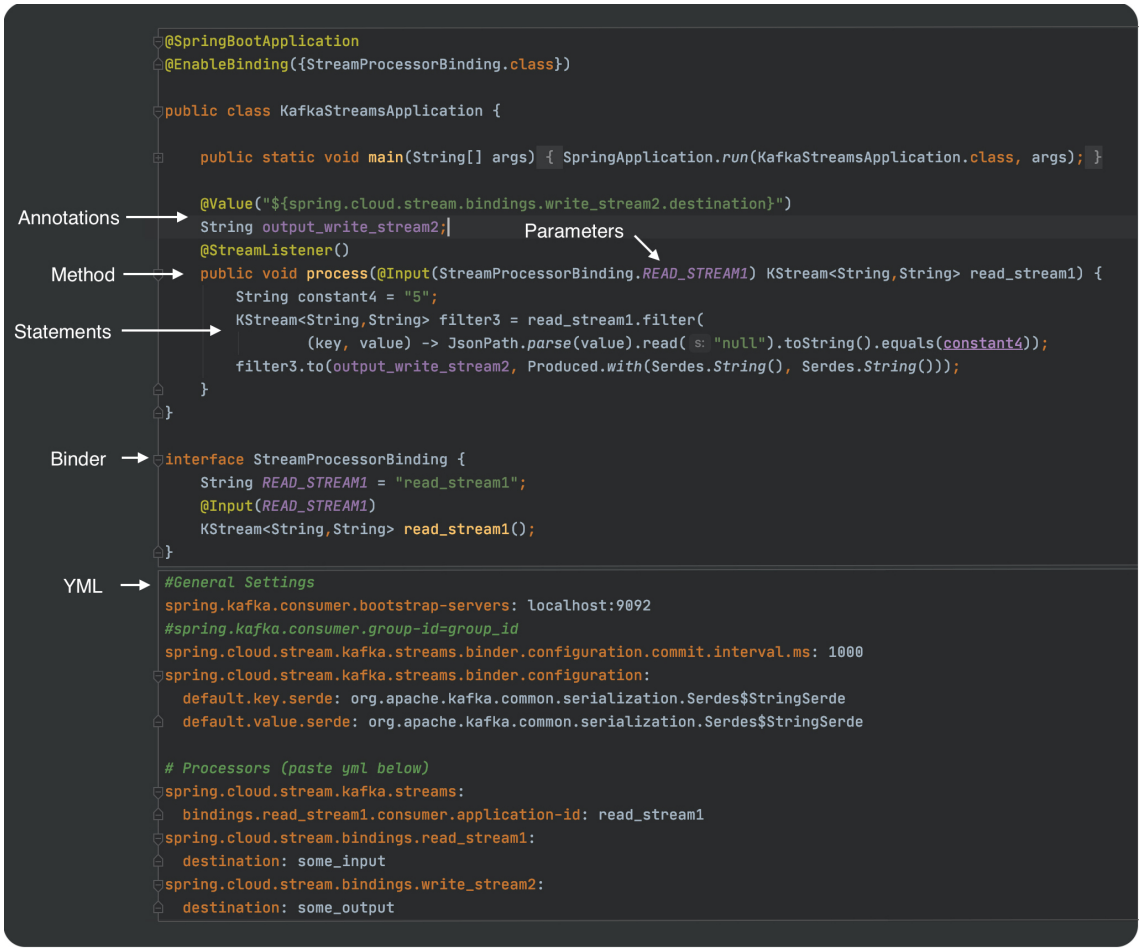
The first step of the code generation is to generate the processing function itself. This function does is described in more detail in subsection 6.4.5. In this step, the read-table and read-stream nodes are translated to input parameters of the processing function. Additionally, Annotations are added to the method for write-stream operations. After the function has been created, statements will be generated for each processing node. First, the variable declaration is generated, based on the output type for the operation. Next, the variable literal is generated based on the logic for generating literals embedded in the processor. Finally, the full statement is added to the method body.

After all the statements have been generated and added to the body, the application will continue to generate the Annotations, the Binder, the YML configuration file. For each write-stream operation, an annotation is created to link the output topic to the YML file. Next, the Binder is generated for each read-table and read-stream operation, and finally, the YML file is generated for all read-stream, read-table and write-stream. Finally, if there are any read-store or write-store operations, a StateStore binder is generated to connect with the store.

Code generation is live and instantaneous, this means that as soon as any edit to the integration model is made this is immediately reflected in the code which is shown in the *Stream Editor Interface*. Figure 6.7 provides an annotated example of the code that is generated for a simple integration that reads a stream, checks whether the record matches a certain criterion ('value' field is higher than 5) and then outputs the filtered stream to an output topic.



(a) Model



(b) Generated code

Figure 6.7: Code generation example

6.4.5 Generated code

6.4.5.1 Kafka Streams

Kafka Streams is introduced from a high-level perspective Section 4.2.1.1. In this section, the technical details of Kafka Streams will be discussed. Overall, Kafka Streams is a API for processing incoming streams of key-value records. Core to Kafka Streams is the concept of a KStream, a topic can be read as a KStream and then operations can be applied to create a new, modified, KStream from a source KStream. An example of such a high-level operation is *filter()*, when calling *filter()* on a KStream with a specific condition, Kafka Streams will create a new KStream will only those records that match the filter condition. Similar to *filter()* there are many other stateless operations, such as *map()* that allows to transform one record into another record, that work on a per-record basis.

In addition to stateless operations, Kafka Streams also offers State-full operations, for instance one can create a KTable from a KStream, that tracks the latest value for each key. Such a KTable can then for instance be used for joining KStreams, or for detecting changes. Essentially, a KTable could be conceptualised as an indefinite aggregation of a KStream. In addition to KTable aggregations, Kafka also offers other built-in aggregations, such as *count()* that simply counts the number of records for a key, but also custom aggregations that allow the user to specify how an aggregate value should be computed based on a current state, and a new incoming object. Each of these statefull operations uses a state store. Such a state store is created automatically for some operations, such as count and KTable creations, but can be created and managed manually as well.

Contrary to a KTable, that is only a view a of a KStream, a state store is a true database-like service, that allows storing and retrieving values based on their keys. While state stores are always used for local processing, they can be persisted in Kafka using a changelog stream for fault tolerance, and state stores can also be made public such that the entire application state can be queried store[4]. KTable's for instance are by default internal, and therefore they can not be accessed as a key-value store. While this is not an issue for native KTable operations such as joins due to grouping of keys, it may be problematic when one needs to retrieve individual keys from a KTable. To address this, one can explicitly materialize a KTable to make it accessible as a read-only, key-value store.

Kafka Streams uses various subtypes of KStream and KTable, to reflect certain attributes or restrictions over the object. For instance a GroupedKStream, or a WindowedKStream, to represent groupings and windows of time. Such subtypes may be required for certain operations, such as rolling aggregations or joins, and APIs are provided to instantiate these sub-types, such as the *.window()* and *groupByKey()*.

In addition to the data container, such as a KStream, one should provide the key and the value type. This is on a per-operation basis, and holds also for the state stores. As discussed in Section 4.2.1.2, Kafka itself only stores raw bytes. However, these raw bytes are unsuitable for processing and should be translated to more meaningful data-types such as Integers, Strings, and Java Objects. When reading a stream, the user should define the key and value types, as well as the SerDes (serializers and deserializers) for those types. During processing, Kafka Streams will infer the new key and value types based on the operation. When writing back to a topic, or when using public statestores, the user should provide the SerDes for the key and value types.

With typical Kafka Streams use-cases, users will obtain Java Objects from a stream, manipulate these objects, and then write Java Objects back to Kafka. This approach requires an initial upfront investment of creating the Java Object classes, the schemas and SerDes for these objects, but it results in compatibility guarantees and ease of development, as it is easier for developers to write code to manipulate typed Java Objects rather than un-typed JSON objects. However, as discussed in Paragraph 6.3 the prototype will instead work directly with JSON Objects, as schema information is not available and generating Java Objects would result in significant overhead.

6.4.5.2 Spring Stream Kafka Streams

Developing a Kafka Streams application requires significant boilerplate code to initialize streams and tables from topics, and to set up state stores and interactive query services. Additionally, as with any Java application, Kafka Streams applications are complex to run and configure. The Spring framework offers a wrapper around Kafka Streams to address these concerns, and facilitate the development of Kafka Streams applications.

Spring Stream Kafka Streams conceptualises the development of Kafka Streams processors as developing simple Java functions. Each processor function can have one or more inputs, for instance a KStream or KTable. Spring Stream will bind the input variables to actual Kafka topics using given properties such as the SerDes. Developers can configure the binder properties in a separate configuration file, rather than having to hardcode these properties in their code. In the body of this function, users can use any of the Kafka Streams APIs to implement stream processing functions. In addition to binding input Streams and Tables, Spring Stream can also bind to state stores and query services. And finally, Spring Stream Kafka Streams also provides benefits of Spring Boot, allowing users to easily run their stream processing applications.

[4] `https://docs.confluent.io/current/streams/developer-guide/interactive-queries.html` consulted 01-08-2020

6.4.5.3 Runtime Scalibility

A key requirement for stream processing integrations is that they should be scaleable, to support large amounts of throughput. Horizontal scalability is guaranteed through the use of Kafka Streams [5]. Given that the Kafka Streams DSL is used as documented, Kafka Streams guarantees fault-tolerance and scalability for the integration for any stream processing application using the framework. This means that processor instances can be added or removed at any time to change the stream processing capacity without any data loss or downtime. Kafka will automatically distribute incoming records across all instances and for statefull operations, processors work together using shared state environments to maintain a collaborative, fault-tolerant state using intermediate topics for statefull operations.

Behind the scenes, Kafka uses partitioning to ensure an even distribution of data and to efficiently execute statefull operations. By default, partitions are randomised, however, some operations require coordinated partitioning. For instance aggregations require partitioning based on the key, and it is up to the developer to ensure that the keys are properly distributed to allow for efficient scaling. Additionally, scaling requires the proper use of state stores to and partitioning in custom operations, this can however be enforced through proper code generation following the recommended coding practices of the Kafka Project. Throughout the prototype, it has been ensured that only valid DSL and Processor code is generated, such that the scalability features of Kafka Streams can be utilised. However, for certain operations it is up to the end user to use them properly ensure scalability. For instance, with the grouping operation, the user should to group by attributes that are evenly distributed such that the records can be evenly distributed among all the processor nodes, rather than all records being processed by one node [6].

[5] `https://www.confluent.io/blog/elastic-scaling-in-kafka-streams/` consulted 03-08-2020
[6] `https://blog.newrelic.com/engineering/effective-strategies-kafka-topic-partitioning/`

Chapter 7

Validation

This chapter describes the validation process for this research. During the validation, models of the artifact are validated in a model of the problem context to produce predictions about what the effects of the artifact would be when it would be implemented in the actual problem context. This chapter first overviews a methodology for validation, then the observed effects and finally concludes with the mechanisms and conclusions with regard to the effects of the validation. Together with the discussion in Chapter 8 this chapter addresses research question 5.

7.1 Methodology

7.1.1 Dimensions

A key part of validation is to determine the dimensions on which the design should be validated. For conceptual models, such as the design presented in this thesis, many dimensions can be identified for evaluation. For instance, Nelson et al. identified over 20 different quality attributes for conceptual models [148], such as maintainability, usefulness, semantic quality, and more. Naturally, it is impossible to identify and target all dimensions. Overall, two key different aspects should be addressed; does the proposed platform really work? And can the design help organisations to build such a platform?

To evaluate the first aspect, the prototype is evaluated using the Technology Acceptance Model (TAM) from Davis [149]. The TAM is a widely used framework to evaluate new information systems from the perspective of end-users. The TAM provides insight into three key questions: Would end users actually use the system? How usefull do end-users perceive the system to be? And do end-users consider the system to be easy to use? Therefore, applying the TAM will help to assess whether the concepts demonstrated in the design provide value to end-users when they are designing IoT integrations.

To validate the second aspect, the architecture is validated using architectural quality attributes. Timm et al. [150] conducted a structured literature review to survey the quality principles of Enterprise Architecture and applied it to a case study using ArchiMate. They found *validity, relevance, economic efficiency, clarity, systematic model structure*, and *comparability* to be the most important principles. However, Timm notes that the use of Archimate ensures that the quality principles for *systematic model structure* and *comparability* are properly supported. Therefore, given that the architecture is valid, these principles do not need validation.

The dimension of *validity* is defined by Timm as the degree to which the model matches the reality [150]. This principle, especially from the perspective of semantic correctness, is complex to evaluate for external experts since they must trace the validity of the model to the concepts presented in the thesis, rather than to an existing implementation of an architecture. Instead, semantic correctness of the design has been ensured by providing traceability throughout the design chapter. Syntactical correctness is also not considered explicitly, since the

model has been designed according to the latest ArchiMate 3.1 specification using the ArchiMate tool which automatically checks for syntax errors [1]. Therefore, the dimension of validity will not be considered explicitly. However, during the validation, the experts can still provide feedback on any possible syntax or semantic issues if they emerge when reviewing the design for clarity and usability.

The dimension of *relevance* is best evaluated by practitioners. That is, it is best evaluated by the architects within the problem context that are implementing the architecture. With *relevance*, the validation focuses especially on *usefullness* and *completeness vs. conciseness*. *Usefulness* reflects that the model is relevant and beneficial for the participant, while *completeness v.s. conciseness* evaluates whether the model contains all sufficient information while also not providing too much information.

The dimension of *clarity* reflects the ability of stakeholders to comprehend the model. Comprehensibility is key since practitioners need to be able to understand the architecture to adapt, implement and review it.

The dimension of *economic efficiency* primarily reflects the cost-effectiveness of the modelling process itself and Timm states that the modelling should have a clear purpose and aim to ensure its cost-effectiveness. The quality attribute of this dimension that best aligns with the research goal is *flexibility*. Flexibility reflects the ability of the model to adapt to environmental changes. This includes for example the ability to apply to model in different, but similar contexts, such as in different organisations with IoT integration needs. Therefore, this attribute reflects how flexible the model is, to be re-used in different contents. This is also referred to as the external validity of the model.

7.1.2 Techniques

To obtain the value for each of the aforementioned dimensions, two different validation methods are used: single-case mechanism experiment and expert opinion. Figure 7.1 provides an overview of the dimensions, and the methods, artifacts and end users used to validate the dimension. Both methods evaluate the effects that the artifacts produce in the problem context.

For validating the dimensions from the TAM, single-case mechanism experiments are used. According to Wieringa, such experimental case studies can be used in validation research to evaluate the cause and effect behaviour of the object of study, or more simply put, for testing the prototype [12]. To evaluate architectural quality dimensions from Timm et al., the expert opinion validation method, as described by Wieringa, is used [12]. During expert opinion validation, the experts will imagine how the artifact will interact with the problem context. For instance, experts are asked to predict the benefits and disadvantages of the design for end-users. For some architectural quality attributes the interaction is less relevant and these attributes are measured directly. For instance, experts are asked to evaluate the comprehensibility of the design directly, rather than evaluating the effect of the comprehensibility of the design.

Validation methods that warrant a direct comparison to the effects caused by alternative solutions are not employed during this research. For the expert opinion, a comparison to other reference architecture for model-driven stream processing tools is unfeasible, since these are non-existent (Section 8.1). The same holds for the single-case-mechanism experiment since the prototype is designed to enable new use-cases that participants cannot, or hardly, complete using their existing methods and tools. Future research could employ statistical difference-making experiments to evaluate the performance of instantiations of the design (such as the prototype) to similar tools, such as identified during the literature review. However, such a performance comparison is beyond the scope of the current research.

The remainder of this chapter will detail how the validation methods are applied in this research. For both methods, a semi-structured interview is used as an instrument to measure the effects. To reduce confounds,

[1] Archi User Guide retrieved 17-09-2020 from archimatetool.com

Quality attribute	Method	Artifact validated	Participants
Intention to Use [149]	Single-case mechanism experiment	Prototype	End-users
Perceived Ease of Use [149]			
Perceived Usefulness [149]			
Usefulness (Relevance) [150]	Expert opinion	Architecture	EA Experts
Completeness vs. Conciseness [150]			
Comprehensibility [150]			
Flexibility [150]			

Table 7.1: Evaluation method per quality attribute

each dimension is observed using multiple questions, both open and closed. To account for socially desirable remarks or inability of the expert to attribute the effects to the mechanisms, measures are put in place to explicitly survey for disadvantages, points for improvements and for the mechanisms causing the effects.

7.1.2.1 Single-case mechanism experiment

During the single-case mechanism experiment, the prototype is validated with three participants that full-fill the slot of a normal operator. This validation consists of a hands-on session with the prototype, during which each participant will implement three integration use-cases using the prototype.

All participants (Male) have experience with developing messaging-based system integrations. Two participants have over 5 years of experience building integrations, and one participant has 2 years of experience. Additionally, all participants have some familiarity with the concepts IoT and event streaming, the participants had no previous experience event stream processing, or building actual stream processing integrations. Some familiarity with these concepts is important since the normal operators of the system will also need some familiarity with these concepts to understand the use cases and implement them. Validating the prototype with users that are not familiar with IoT use cases or the concept of event streaming would lead to sub-optimal results as the users may not be able to relate to, and understand, the use cases.

The validation protocol for the prototype validation is as follows:

- **Introduction (10 minutes)**: During the introduction, the participant is introduced with the concepts of event stream processing. This introduction compares event stream processing to messaging, and then introduces some of the operations ('blocks') available to the participant. For each of these operations, it is explained what the operation does, what inputs it supports, and what output it will produce.
- **Example demo (10 minutes)**: During this activity, the prototype is explained and demoed. This demo ensures that the participant knows how to use the prototype and that he has a basic understanding of what the prototype is capable of.
- **Hands-on with use cases (30-60 minutes)**: During this hands-on session, the user uses the prototype on his laptop to address an event stream processing use case. Each use case consists of a context description, describing why processing is needed, as well as a description of the input and expected output formats of the stream. The following use cases are implemented by the participants:
  - Enrichment - During this IoT use case, the participant is asked to enrich a stream of IoT data by joining the stream with a table on a specific ID.
  - Aggregation - During this IoT use case, the participant is asked to perform an aggregation over a stream of data to retrieve the average value of the stream of IoT data grouped by a certain attribute, over a window of time.

  - Transform & Filter - During this IoT use case, the participant is asked to clean a stream of IoT data by filtering irrelevant data, and transforming from the input data format into a stream of alerts with a fixed schema. This use case could be implemented with both stream processing and messaging and could therefore be used to isolate the effects of the editor from the effects of the streaming pattern.

- **Feedback session (20 minutes)**: During the feedback, the experience of the participant with the prototype is obtained using a structured interview, as per the protocol and questionnaire in Appendix F.1. As was explained in Chapter 6, the prototype covers the requirements of one of the key stakeholders; the integration developer. Therefore, this validation focuses on confirming whether the requirements and goals for this stakeholder have been met. More specifically, R1 and Goal 4. The interview questionnaire has several different questions to assess whether or not these requirements are met. First, the questionnaire assesses whether the participant was able to use the prototype to implement the use cases, and whether or not this would have been possible using alternative approaches such as messaging and manually coding the integrations. Next, the questionnaire assesses the three key dimensions of the Technology Acceptance Model; the intention to use, the perceived usefulness and the perceived ease of use of the proposed design. Moreover, the questionnaire evaluates how these ratings are supported by the design, by assessing why the participant would recommend the prototype, and why the participant believes that the prototype would, or would not, contribute to IoT use-cases. Finally, the questionnaire provides room for general thoughts on the prototype.

7.1.2.2 Expert opinion

During the expert validation, the design is validated with three different experts. One expert (Male) is a CTO of an integration platform, seeking to extend the platform with IoT stream processing support. This CTO has over 20 years of experience in software architecture within the problem context. The second expert (Male) is an associate professor, with over 25 years of experience in architecture for distributed systems and model-driven engineering. And the third expert (Female) is a professor with expertise in Enterprise Architecture, and a contributor to the Archimate specification with more than 15 years of experience in the field of enterprise architecture. This validation consists of an introduction to the problem context, a presentation of the design, and a feedback session. The full validation protocol for the prototype validation is as follows:

- **Introduction (10 minutes)**: During the introduction, the expert is introduced to the problem context and the requirements of the design. The problem context describes the problem context as observed in Chapter 2. That is the challenges that practitioners encounter when manually developing integrations. This is to ensure the stakeholder can imagine the use of the design in the proper context.
- **Design (20 minutes)**: During this activity, the design is shown to the expert. To this end, all the designs of Chapter 5 are shown as slides alongside a verbal explanation of the design, based on the explanations in the Chapter.
- **Feedback session (20 minutes)**: During the feedback session, the expert is asked to imagine the prototype being applied in the problem context. Some experts can apply this to their own problem context, and other experts may use the problem context sketched in the introduction. The feedback is obtained using a structured interview, as per the questionnaire described in Appendix F.2. Three generic questions are used to determine the expertise and background of the expert. Next, the overall anticipated effects of the artifact in the problem context are evaluated. The questionnaire addresses positive effects, negative effects and asks the expert what he or she would add, remove or change in the design. Finally, all of the architectural quality dimensions are evaluated. Each of the dimensions is validated using at least one scalar question, and one open question for elaboration.

7.2 Results

7.2.1 Single-case mechanism experiment

All the results are discussed in Appendix F.2. Table 7.2 provides some of the overall findings per participant.

	Participant 1	**Participant 2**	**Participant 3**
Cases completed	3/3	3/3	3/3
Ease of use	4/5	3.67/5	4/5
Usefulness	5/5	4/5	4.25/5
Challenges	• Understanding the operations	• Understanding the operations • Understanding streaming concepts • Syntax issues	• Understanding the operations • Understanding streaming concepts
Feedback	+ Instant type validation + Categorisation of operations in input, output, etc + Input quantity of operations is visualised + Operation options are visualised + Overall intuitive and easy to use - Learning curve for operation dependencies (I/O compatibility) - Learning curve for what operations do and what output they produce - Formulating queries and transformations is complex - Blocks could be suggested to the user based on current flow	+ Editor works fluently and the interface is clear + Applications and options appear to be endless + All operations relevant for stream processing appear to be there - Look and feel could be improved	+ Editor is clear + Editor is in line with how consultants current model messaging based integrations + Simple operations that allow you to build complete integrations + Complete as-is - Frequent combinations of operations could be made available to the user as composite operations
Would recommend	5/5	4/5	4/5

Table 7.2: Key results per participant

All use cases could be completed by the participants in the given time-span, using the available information. With respect to the completion of the use-cases in the prototype, this section will first discuss the challenges faced, then how the participants evaluate prototype compared to alternative solutions, then the evaluations from the TAM and finally perceived ability of the prototype to support IoT use-cases.

Overall, the three kinds of challenges were identified while implementing the use cases:

- Understanding the operations: Understanding what an operation requires as an input, what it does with that input, how to configure it, and what it outputs.
- Understanding streaming concepts: Understanding that processing happens over a stream of key-value pairs, and that therefore processing can be statefull and either over individual records as well as groups of records.

- Syntax issues: Schema-related issues related to querying data, or ensuring the correct output format.

The first challenge was the most common and was faced by all three participants during the first two use cases. One participant also noted this challenge for the third (messaging) use case, while the remaining participant did not experience any challenges here. The second challenge was noted by two of the participants, during the first use case, and for one of these participants also in the second use case. The third challenge was noted by one participant, who noted that querying data was complex since mistakes could be made easily as the editor did not provide any validation towards the schema.

When asked whether the use-cases could also be completed using any other methods or tools, all participants indicated that they would have been able to build all use cases in messaging. However, they all agreed that for stream processing use-cases (case 1 and 2) it would be significantly more complex to do this in messaging and that workarounds would be needed, such as writing all data to a database. One participant quantified that it would take between 5 and 10 times longer to complete these use cases in messaging-based tooling compared to the prototype. Another participant also noted in addition to the integration taking longer to be built, he would also expect poor performance for messaging due to the overhead and mediocre scalability. Overall, all participants agreed that the messaging pattern was simply not suitable for the first two use cases, except for one participant in one use case. All of the participants did recognise that use case 3 would have been equally possible with messaging, as with stream. processing. One participant also noted that it would be technically possible to build the integrations as a standalone application, using no/low-code development platform Mendix, however, he also stated that this would involve workarounds as well and that this would be a sub-optimal solution with quite some drawbacks.

All participants agreed that the prototype was easy to use, with an average rating of 4 out of 5. Usefulness was rated at least 4 out of 5 by the participant, with one participant even rating it a full 5 out 5. As expected per the TAM, they also showed a high-intention to use the framework during their work, as well as recommending it to colleagues. In their motivation for using it or to recommend it to colleagues, users pointed out that they would prefer to use the prototype due to the better support for use-cases where streaming patterns are preferred over messaging patterns. For instance with high-throughput processing (participant 1 & 3), use-cases involving retention (participant 1), use-cases for publish/subscribe (participant 1 & 3), use cases for aggregation (participant 1) or when reading/writing from streams (participant 2).

All of the participants agreed that the system would increase the ability of their organisation to support IoT use-cases. Participant 3 called IoT "a textbook example" of stream processing, and that the features showed in the prototype definitely have value for supporting IoT. Participant 1 said that the prototype would make it significantly easier to "unlock customer data and put it to use" because the stream processing functionalities offered by the prototype are much more suitable for dealing with such integrations than the current messaging toolset.

Finally, the survey concludes with the overall positive feedback, as well as points for improvement. These results are listed per participant in Table 7.2. Another finding of both participants 1 and 3 was is that they appreciated how easy it was to build an integration using simple operations, compared to the overhead that is currently involved with developing messaging-based integration. While they still would prefer the existing messaging-based platform for complex messaging use cases, they would prefer and recommend the prototype for simple messaging operations (such as use case 3) due to how quick it was to use.

7.2.2 Expert opinion

All the results are discussed in Appendix F.2. Table 7.3 provides some of the overall findings per participant.

	Participant 1	**Participant 2**	**Participant 3**
Expertise	Model-driven engineering	Integration Platforms	Enterprise Architecture
Comprehensive	4/5	4/5	5/5
Conciseness	3/5	5/5	4/5
Usefulness	5/5	4.67/5	4.33/5
Comprehensible	4/5	5/5	4/5
Flexibility	4/5	4/5	4/5
Feedback	+ Offers integration development at the level of the user + Meets specific user requirements + Design is complete and generally applicable + Right choices in functionality, methodology and generalisation + Comprehensive for practitioners - Archimate design not separated from meta-modelling design	+ Can be used as a supplement to implement event-stream processing in existing architecture + Properly applies the concept of MDE + Provides architectural directions to support architectural decisions + Re-usable among practitioners seeking to implement a solution for building integrations - Dependency of integration on deployment and runtime could be reduced.	+ Editor is clear + Facilitates data management + Facilitates operationalisation of business processes based on IoT + Facilitates IoT integrations in an architecture + Flexible, allowing adaptable integrations + Architecture is complete as-is - Requires expertise in data integration from end-users - Prototype does not yet cover the full design - Code generation design is complex for non-technical people
Remarks	• Only practice will show whether all design choices are correct • Problem domain is complex, which inevitably impacts overall conciseness	• As a consequence of conciseness, work remains for actual implementation	• Prototype could be made more accessible for non-experts. • The design could use further validation with real-life implementations
Would recommend	5/5	4/5	4/5

Table 7.3: Key results per participant

With the advantages, the CTO, EA, and MDE expert stressed the proper application of MDE onto the domain of stream processing as one of the major benefits of the design. That is, the design properly applies this concept to allow domain-users to build integrations, without the need to concern for how integrations are coded, built and deployed. The EA expert extends on this by comparing the integration model to a business process model, where each operation in the model represents the business rules in the process. This allows users to actively implement business processes to act upon real-time data streams. Both the EA expert and the MDE expert list the flexibility of the design as a benefit. The MDE expert states that the development of the prototype inevitably involved some design decisions that exclude specific requirements, the architecture itself is generic and can be adjusted to the specific use cases of the practitioners. Additionally, the CTO stresses the ability of the design to

identify architectural components that need to be added or changed, by breaking down the total solution of event stream processing into sub-problems that can be analysed and compared to an existing platform. The CTO also states a high degree of alignment between their current architecture and the proposed design. He argues that the design can be used as a supplement to the current architecture, to assist him in adding event stream processing support to the platform. Also the EA expert and MDE expert acknowledge the value of the design to help practitioners add event stream processing support to their platform.

Concerning the disadvantages, the MDE expert states that they will show during validation in practice with more real-life use cases. He states that making design decisions, either in the design or the prototype, inevitably impact the requirements that you can satisfy and that only validation across a broad range of use cases can determine what the limitations are. Also, the EA expert stresses the need to implement the design in the problem context. The CTO reported the design deploying integrations as a disadvantage. The initial design described how integrations are only pushed to the runtime on deployment, while the CTO suggested that only deployment instructions should be pushed, allowing the runtime to fetch the actual integration as needed. The CTO argued that this change reduces the dependency of the integration on the deployment process, this allows the runtime to keep its integrations up to date without the need for re-deployment or model changes. The EA expert also states the required expertise of the end-user as a limitation. The prototype currently targets integration experts, excluding novice users without technical experience (i.e. a Farmer that uses IoT), resulting in a learning curve.

With regard to suggested changes, none of the experts had any substantial suggestions. The MDE expert states that the right choices appear to have been made in the functionality, methodology and generalisation and that no changes appear to be needed. The CTO also did not propose any changes. The EA expert also did not propose any changes to the design but did suggest to adapt the prototype to increase usability towards non-technical people as the prototype matures.

All participants rated comprehensiveness high, with a 4 or 5, and had no further remarks on how to improve comprehensiveness. The CTO stated that there was an excellent balance between comprehensiveness and conciseness, rating conciseness with a 5 out of 5. The EA expert and the MDE expert rated conciseness lower with a 4 and a 3 respectively. Both argued that, for a software development project, the conciseness was as expected but that the overall conciseness is inevitably sub-optimal due to the complexity of the problem domain. According to the MDE expert, the problem domain requires an exhaustive design with many facets, for instance covering all phases in the lifecycle as well as all the layers in the architecture.

Usefulness was rated with a 4 or higher by all participants. The CTO stated that, unless someone would provide a very good reason not too, he would certainly implement the design. As a suggestion to improve usefulness, the EA expert would recommend expanding the scope of the prototype to fully reflect the design, since such a prototype would be able to better demonstrate and sell the design to practitioners. The CTO noted that implementing the design would still take some effort due to the high-level nature of the design, but also acknowledged that this was an unavoidable side-effect of conciseness.

The CTO rated comprehensibility with a 5 out of 5, while the EA expert and MDE experts rated it a 4 out of 5. The EA expert stated that the code generation component of the prototype was complex to understand due to the non-technical nature of the EA domain the expert was operating in, but that the design itself was very comprehensible. The MDE expert stated that the Archimate model could be separated from the code-generation / metamodelling parts of the design. This, since the typical projects the expert was involved in only featured the code generation approach with a clear distinction between the metamodels, the mapping and other MDE elements of the design. However, the expert acknowledged that it would be hard to present this design as such since Archimate is selected as a primary view and since the combination of both views is tough.

Finally, all experts rated the flexibility of the design with a 4 out of 5. All agreed that the design would be

re-usable among those practitioners seeking to implement a platform for building integrations, making the design less suitable for practitioners seeking to build just a few integrations. The CTO noted that the concepts presented in the design are re-usable, independent to their specific product and platform. The EA expert even concluded that this design would be suitable for any business seeking to implement a large range of integrations.

7.3 Conclusion

This section first discusses the conclusions from the end-user perspective, then from the expert perspective and finally overview is presented based on both perspectives.

End-users The single-case mechanism experiments show a high intention to use, a high perceived ease of use, and a high perceived usefulness of the design. Users attributed this to the intuitive and simple model-driven design of the prototype, that allowed them to build complex integrations in less time. This efficiency increase holds especially for stream processing integrations. Participant 1 quantified even that it would up to 10 times longer to build the same integration using a messaging-based solution. And two participants even indicated they would even prefer the design for messaging-based integrations. Additionally, completeness of the set of operations and the type validation were also reported as mechanisms contributing to the high intention to use.

Some minor challenges were identified during the use-cases. All participants found it initially challenging to understand what the operation exactly did and what input they required. This is as expected since the prototype does not provide any extensive help tooling or documentation about the operations other than the provided introduction. In production implementations, this challenge could be alleviated using proper documentation for each of the operations, as well as a built-in help tool that shows the expect in- and outputs. In addition to understanding the operations, two users initially found it challenging to understand some of the concepts involved in stream processing. Again, this would be as expected for users without previous experience in stream processing, and this could be addressed using proper training. Moreover, both challenges could also be addressed through experience, as all users were eventually able to complete all use-cases within 20 minutes. Suggestions primarily included low-level feedback, such as improvements to the overall look and feel, adding documentation for operations, adding composite/recommended operations, and including schema-based validation. Fortunately, these suggestions do not expose any faulty design assumptions in the prototype but are rather valid suggestions to further refine the prototype or to take into account for production implementations. This holds especially for the suggestion on schema-based validation. This attribute is covered extensively design to allow for easier schema-based queries, validation, and transformations, but could not be included in the prototype due to the set scope and time-frame.

Experts The validation indicates that the design has a high architectural quality, which can be attributed to the high ratings for key attributes of the design such as usefulness, flexibility and comprehensibility and the good balance of comprehensiveness v.s. conciseness. The non-practitioners noted a slightly lower rating for comprehensibility and conciseness than the practitioner, which can be attributed to the high complexity of the problem domain and the contrast with the experts' typical research projects. Nevertheless, all of the experts found the overall comprehensibility and conciseness of the design to be more than sufficient. The high usefulness can be primarily attributed to the application of MDE. All participants acknowledged the value of MDE in the problem domain, and all found that MDE was properly used within the design to address the user requirements. Furthermore, participants valued the designs for its ability to assist practitioners in implementing such a model-driven platform for stream processing. Participants attributed this to the high level of architectural guidance and support provided by the design. The CTO stated that the design allows the identification of components that need to be changed, by breaking down to the total solution into sub-problems that can be analysed and compared to a current platform. Finally, flexibility is also rated high, with all experts concluding high-re-usability for any practitioner seeking to build a platform for stream processing integrations.

There was no substantive feedback related to the design choices, with one exception. The initial design described deployment in one step, sending the integration executable directly to the runtime. However, CTO suggested a two-step approach towards deployment to reduce dependency on the deployment process for runtime updates. With this approach only deployment instructions should be sent to the runtime, allowing the runtime should retrieve the integration executable from the backend. Based on this feedback, the initial deployment design has been adapted into the current design described in Section 5.3.4. The remaining feedback by the EA and MDE experts concerned the maturity of the prototype (to cover a larger part of the design and to be more accessible) and further validation of the design by implementing it in the problem context. Both are valid suggestions for future research, however, they are beyond the scope of this thesis.

Overview Overall, the results from the single-case mechanism experiment and the expert opinion are positive as would be expected by the concise use of proven methodologies and the close involvement of stakeholders throughout the design and development process. Furthermore, the validation results were consistent among participants, demonstrating low variability. The list below overviews the key findings from both validation methods:

- The design is rated high for perceived usefulness. End-users found the platform to be very usefull to build stream processing integrations due to its complete set of stream processing operations and easy to use model-driven interface. Experts found the design to be very useful in supporting practitioners to integrate such a platform in their architecture, as a result of the architectural directions, and detailed architectural breakdowns of the platform components provided by the design.
- As a consequence of this high usefulness, end-users showed a high intention to use the design during their work and to recommend the design to colleagues. Experts and practitioners showed a high intention to adopt the design or to recommend the design to practitioners.
- Key architectural quality attributes, such as the comprehensibility, flexibility and conciseness of the design are rated high by experts, indicating good architectural quality of the design, and good re-usability among practitioners.
- No key issues in the design or prototype were identified, indicating that no fundamentally incorrect design choices were made. However, one substantive suggestion for the deployment process was made, after which the specific design for this process was improved.
- Both the experts and the practitioners noted a modest learning curve for end-users, which was attributed to the low maturity of the prototype. While the learning curve was small the experts did note opportunities to decrease it. For instance, through increase accessibility and adding documentation or help tooling in future revisions of the prototype.
- The experts agreed that further validation of the design, through implementing the design in practice, would be a crucial next step. This includes expanding the scope of the prototype to cover the complete design, and increasing the overall maturity of the prototype. This would allow for a better demonstration of the design towards practitioners, validating the design itself, and for supporting practitioners in low-level design choices through the examples in the prototype implementation.

Chapter 8

Discussion

This chapter evaluates the contributions of the design by comparing the design to alternative solutions and by relating the design to the challenges identified during the problem investigation.

8.1 Comparison to alternatives

This section provides a comparison of the design presented in this thesis and existing solutions for stream processing platform development. In particular, the prototype and the design are compared to StreamPipes, aFlux and QryGraph which are alternative solutions found during the literature review. StreamPipes was found to be the overall best solution for stream processing currently available. aFlux was found to be the best of all solutions for generating Java code. Lemonade was found to have the best approach for generating code while also providing schema-level validation. Section 4.2.2.2 details how StreamPipes, aFlux and Lemonade compare to all identified alternatives.

Tool	**Deploy & manage**	**Validation support**	**Code generation**	**Embeddable**	**Framework**
StreamPipes	Yes	Ontology	No	Yes[1]	Flink, Spark, JVM
Lemonade	Yes	Schema	Yes, build-time	No	Spark
aFlux	No	Order	Yes, build-time	No	Flink, Spark
Prototype	No	Type	Yes, instantly	Yes	Kafka Streams
Proposed Design	Yes	Schema	Yes	Yes	Any

Table 8.1: Comparison

The most striking difference between the proposed approach and the available alternative solutions is the level of abstraction. While the proposed design can be used as a reference to build a stream processing platform, such as the prototype, current research only provides concrete implementations. None of the surveyed approaches offer a reference design, which is essential for complex use-cases. For instance, given that it meets your requirements, StreamPipes offers a ready to use, integration product. However, if you seek to implement an integrated stream processing solution, an off the shelf product like StreamPipes is hard to adapt or integrate. The proposed design targets platform architects seeking support for designing an integration platform. The design allows these practitioners to design a platform for their specific needs, and integrate it within their enterprise architecture. For instance, to add stream processing capabilities to an existing integration platform to allow for centralised management of all integrations patterns. The lack of reference designs in the domain of integration in general, and the need for further research in this area, was established earlier in Section 2.1.2.9. Solutions like aFlux do allow for further integration and customisation since these solutions only cover the design phase and allow for freedom regarding the implementation of the management

[1] Scaleability in JVM mode is unknown

and deployment of integrations. However, since no reference design is provided on how to implement these processes, this still leaves the associated challenges for practitioners unaddressed.

Table 8.1 provides a comparison between the proposed design and the aforementioned alternatives. However, a comparison between the alternatives identified and the conceptual design does not always hold since concrete implementations inevitably involve design choices imposing restraints. Therefore, the table overviews both the proposed design as well as the prototype, as an implementation of the design in the problem context, to allow for a fairer comparison to alternative solutions. The table compares the designs based on the following attributes:

- **Validation**: The proposed design provides a detailed approach towards supporting schema-level validation as well as type-level validation. The prototype demonstrates the feasibility of this approach, by showing how types can be computed dynamically in a dependency tree, which can also be applied to schemas. While other tools do provide semantic support, such as Lemonade and StreamPipes, semantic support is still rare among the solutions and even absent for tools that generate Java code and/or embeddable processors. Ontology level semantics, such as found in StreamPipes, are not required by the design, because the costs for implementing ontology-level semantics are not expected to outweigh the benefits anticipated in the problem context (Section 5.3.3.3).
- **Embeddability**: Virtually all of the solutions generate code to be executed on a large multi-tenant cluster and do not support the development of embeddable integrations (as per R4). While the difference between generating embeddable code and non-embeddable code may seem trivial, this has a vast impact on the life-cycle of the integration since embeddability implies that the platform needs different tools for managing, scaling and deploying integrations.
- **Deploy & Manage**: Integration deployment and management is covered only by the more mature platforms, such as StreamPipes and Lemonade. Since deployment and management were identified as key challenges in Chapter 2, the presence of these functionalities is an important requirement to consider when comparing alternative solutions. However, most of the tools identified focus only on code generation, such as aFlux, and do not cover the deployment or the management.
- **Code Generation**: Most platforms offer code-generation features, with the exception of StreamPipes. Streampipes instead produces logical models that are interpreted during runtime. This makes the architecture of Streampipes significantly more complex, due to the need for runtime wrappers that need to support all operations in the logical model as discussed in Section 5.3.3.4. Of the alternatives, most approaches offer code generation on build-time. The prototype offers live code generation and live model validation, to allow users to instantly review the validity of the model, as well as the code produced.

Overall, the proposed design offers the first architectural reference for practitioners seeking to implement an integration platform for stream processing. This design describes all the key functionalities found across other platforms such as deployment, management, schema-level validation, code generation, embeddability and live code-generation. All of which are crucial requirements for integration providers, as well as practitioners seeking to implement single-tenant integration solutions. The prototype demonstrates the ability of the design to be implemented within a problem context. Even with its limited scope, the prototype is able to provide a novel combination of features not found in current solutions, with code generation for embeddable stream processing applications in combination with type level validation. Additionally, it is the first model-driven tool for developing Kafka Streams applications.

8.2 Relating to the challenges & goals

In Chapter 2 and 3, the challenges and the goals stakeholders face with regard to IoT integration were identified. These challenges represent the issues in the problem context that makes it more difficult for stakeholders to reach their goals. Based on these challenges and goals, as set of requirements was established for the design which, to mitigate the challenges and contribute to stakeholder goals. Chapter 5 showed that the design meets these requirements, by tracing all the requirements to architectural elements. In

this section it is shown that, by addressing these requirements, the design contributes to resolving the challenges faced by stakeholders.

Table 8.2 provides an overview of all challenges identified and explains how the thesis contributes, either directly or indirectly, to mitigate these challenges. Furthermore, table provides references to the sections that explain in more detail how this challenge is addressed. Finally table shows the goals that are contributed to by resolving the challenges.

Challenge	How it is addressed in the proposed design	Reference	Goal(s)
Challenge 1 Semantics	*Indirectly addressed*: To account for semantics, a specification process is described in the business layer of the design. Yet, full ontology-level semantics are not covered due to the low maturity and high costs of ontologies within the problem context (Paragraph 4.2.2.2). The literature review provides a comprehensive body of literature on the current state of IoT semantics to support future research on this topic.	5.3.3.3, 5.2	2
Challenge 2 Heterogeneity	*Directly addressed*: The design documents the use of schemas to manage the data formats of different data sources and the use of processing operations to overcome these schema differences. The validation shows that end-users are able to use the prototype to create data mappings to overcome heterogeneity.	5.3.3.1, 7.2.1	4
Challenge 3 Finding a business case	*Indirectly addressed*: The design contributes to IoT business cases through lowering development and management costs of IoT integrations. During validation all participants agreed that the prototype made it easier and faster to develop IoT stream processing integrations. Furthermore, experts noted that the design could also allow for new business cases as it allows companies to easily operationalize their business processes as integration models and unlock data throughout the organisation to act on this data in real-time data. Furthermore, unlike traditional implementations, these models can easily be changed as requirement change.	7.2.2, 7.2.1	1
Challenge 4 Developing IoT infrastructure	*Directly addressed*: During validation, experts agreed that the design provides great architectural support in practitioners seeking to setup a platform for IoT infrastructure. Additionally, experts agreed that the model-driven approach allows end-users to deploy integrations without the need to concern for implementation level details, such as the infrastructure	5.3.7, 7.2.2	3
Challenge 5 Pre-processing	*Directly addressed*: End-users appreciated the ability of the prototype to build complex stream processing integrations using simple operations. None of the participants involved in the validation were programmers or had previous experience in building IoT stream processing integrations, nevertheless, they were able to use the prototype for complex processing integrations including enrichment, aggregation and filtering.	7.2.1	4

Challenge 6 Quality & Reliability	*Indirectly addressed*: The design describes the use of re-usable data processing operations to overcome quality and reliability issues. This is demonstrated in the prototype, which provides a wide range of operations to check for and overcome, data quality issues to ensure reliability. For instance, filters to filter for errors or data format mismatches and aggregation operations to obtain average windowed measurements or detect outliers and anomalies in data streams. Both example use-cases in Section 6.3 show how the prototype can be used to handle low-quality or unreliable measurements from IoT devices.	5.3.3.1, 6.3	4, 5
Challenge 7 Dispersed IoT integrations	*Directly addressed*: All integrations developed using model-driven development interface are represented in the same language, and rely on a common software stack and infrastructure for execution. The operations available to the user can be extended by the platform vendor as needed to support any kind of integration. Both the expert validation, as well as the validation with end-users confirm the high re-usability of the platform for different use-cases and integration scenarios across contexts.	7.3	2, 3, 4
Challenge 8 Central management and governance	*Directly addressed*: The design features a unique combination of embedded event stream processing combined with centralised management. This allows flexible deployment to any infrastructure, while also providing centralised management of all integrations. Furthermore, the design can be used to extend existing integration (management) platforms with event stream processing support to allow for centralised management across integration patterns.	5.3.5	6
Challenge 9 IoT stream processing support for integration platforms	*Directly addressed*: The design provides guidance to practitioners on implementing a model-driven stream processing platform. Validation with architectural experts confirms the usefullness of the design for providing IoT stream processing support on integration platforms. Also, end-users perceived that the design would contribute to their ability to model IoT integrations on an integration platform	7.2.2, 7.2.1	4, 7, 8, 9

Table 8.2: Challenges

The table shows that the design contributes to addressing all key challenges in IoT integration. Six challenges are directly addressed by this thesis, as is confirmed through validation with both architectural experts and end-users. Some challenges are indirectly addressed, the semantic challenges, the business challenges and the data quality challenges. While the challenges are not fully mitigated, the design does provide a first step towards addressing these three remaining challenges by identifying them, and by providing practitioners with an overview of the tools, designs and methodologies available to mitigate them.

Using the validation sessions and the contributions listed in Table 8.2 is possible to qualitatively support that stakeholder goals are addressed. Based on the number of challenges listed for a goal, taking into account the impact of the challenge and whether it is directly addressed, it is possible to determine to which degree the goal has been addressed. Optimally supported are the the goals of the platforms' end-users, which are the clients developers, architects and support teams, since the proposed design directly mitigates multiple challenges related to IoT integration development, infrastructure development, deployment and management (Goals 3, 4, 6). The contributions towards these goals have also been validated in Chapter 7. The integration

specification goal (Goal 2), is supported through mitigating Challenge 7 by providing a uniform approach for developing integrations. However semantic challenges during the specification process are only indirectly addressed (Challenge 1). The goal of the platform clients' management team to reduce integration costs and increasing efficiency (Goal 1) is indirectly supported through the mitigation of the other integration challenges. For instance by making it easier to develop integrations and setup infrastructure, development time and the skills needed for development are reduced, which in turn contributes to cost reductions. Further research, for instance comparative experiments with alternate solutions could be employed in future research to quantify the anticipated cost reductions and efficiency increase. The same holds for the overall reliability and quality of IoT integrations (Goal 5) which is indirectly addressed through Challenge 6. Finally, key goals include those of the platform vendor (Goal 7, 8, 9). The vendor is directly supported in their goals of delivering customer value and offering a complete and integrated platform, by addressing the challenge associated with implementing a stream processing based integration platform. This is also shown in Chapter 7 during validation. The goal of reducing operational incidents is indirectly addressed, through the increase of integration quality and reliability, and as typical streaming integrations (currently implemented using other patterns) can be migrated to stream processing to reduce integration complexity.

Overall, this section shows how key integration challenges are addressed in this thesis and how, by resolving these challenges, the design contributes to achieving stakeholder goals.

Chapter 9

Conclusion

This chapter addresses the findings of this research. The first section discusses the answers to each of the research questions. Then, the limitations of this research are discussed, as well as opportunities for future research. And finally, the contributions of this research are discussed.

9.1 Research questions

The goal of this research is to address the following main research question:

How to *design a model-driven integration platform*
that *allows the development and management of stream processing integrations*
so that *developers can preprocess data streams more efficiently*
in *IoT integration*?

To answer this question several sub-questions have been formulated and answered. This section provides a concise summary is for each of these questions.

1: What are the challenges that are faced by organisations during IoT integration?
An IoT integration is conceptualised as a series of data processing operations that process data from a set of data sources, and that deliver output to a set of destinations. Based on a literature review as well as expert interviews, nine challenges with IoT integration were identified. These include challenges related to the nature of the IoT data such as high heterogeneity, low data quality, and ambiguous semantic attributes of the data. Next are development related challenges, such as a the complexity of programming processors, setting up the processing infrastructure, and dispersion in the technologies and methods used for development. Finally, there are high-level challenges such as with defining the business case for IoT, the decentralised management for IoT integrations, and the lack of integration platform support for streaming IoT. Overall, the challenges indicate the need for a reusable unified approach towards IoT integration, which research currently does not provide. This gap in knowledge means that the development of integrations remains to be a complex, tedious and repetitive task. A model-driven integration platform for IoT stream processing is hypothesised as a solution to this problem. *Chapter 2 overviews the problem context in detail.*

2: What is the minimal set of requirements for a model-driven programming platform to allow the effortless integration of event-streams?
The minimal requirements for the platform should address the challenges for IoT integration previously identified. Additionally, the requirements should support stakeholder goals. Stakeholders include the platform vendors' development and support teams, which are implementing and supporting the platform, and the management of (potential) customers of the platform, that should be convinced that the platform provides value. The most important stakeholders are the end-users of the platform or 'normal operators'. The platform should allow these end-users to develop, deploy and manage the integration without the need for any

programming, and without the need to know the technical details such as the underlying architecture. Furthermore, the platform should provide stream processing functionalities, which supports the statefull, high-throughput, low-latency processing capabilities needed for processing IoT data. These stream processing integrations should be embeddable such that they can be deployed to any infrastructure. To ensure the quality of the integrations, the platform should support schema-based validation of the integrations. Finally, requirements are presented concerning extensibility, modularity, and scalability to ensure that the platform can be easily adapted and integrated according to the specific needs of the client and the vendor and to ensure that the platform supports scalable integrations for processing high-volumes of realtime IoT data. *Chapter 3 overviews the stakeholders, goals and requirements in detail.*

3: What are the existing designs for stream processing, and graphical programming platforms for stream processing?
Through a systematic literature review, a comprehensive overview of all solutions for embedded stream processing and graphical programming platforms, and their designs, is provided. For embedded stream processing solutions, the study first identifies a taxonomy of the key attributes by which alternatives can be evaluated. Designs vary with respect to data ingestion, data execution and non-functional aspects. When comparing the identified frameworks, Kafka Streams and Apache Samza are found to be the most complete and mature stream processing solutions. Both provide great performance, and offer a complete set of functionalities and declarative APIs. The survey also provides a taxonomy of attributes for evaluating graphical stream processing platforms. The designs vary with respect to code generation models, validation and life-cycle support. Solutions can support code exported based on a logic model derived from the integration model, or the integration models can be directly interpreted by the runtime. In case of the former, the solutions differ in the underlying stream processing frameworks that they support. For validation, different designs include primitive type checking, order based type checking, schema-based type checking (for complex types), and ontology-based type checking. Additionally, validation can be static, or dynamic based on dependencies between the input and the output of an operation. When comparing the identified graphical editors, the results show that the overall maturity of the alternatives is low and that none of the solutions match the requirements identified earlier. Of all identified solutions, the incubating Apache StreamPipes project is the most complete. *Chapter 4 discusses the results in detail.*

4: What combination of the alternative solutions would form the most optimal design for a model-driven programming platform for distributed event stream processing?
A novel design for a model-driven stream processing platform is proposed, based on the identified existing designs. The integrated design describes the platform from three perspectives. First, from the business perspective, the design shows how end-users interact with the platform, from specifying an integration, to developing it, deploying it and managing it. Then, from the application perspective, the design describes the key application components such as the model-driven editor, the model validation and code generation functions, management and deployment components, and the data models used. Finally, from the infrastructure perspective, the design describes the infrastructure required for running the platform and the integrations. For the latter a dynamic, embeddable runtime is described that can run scalable, stateful stream processing integrations on-premise or in the cloud. *Chapter 5 provides a complete architectural breakdown of the design.*

5: To what extent does the proposed design combination contribute to stakeholder goals?
The design contributes to stakeholder goals by supporting platform architects in developing a platform that mitigates the integration challenges end-users face. From an architectural perspective, evaluation with enterprise architecture experts shows that the design provides high architectural quality and that it supports platform architects in developing an IoT integration platform architecture. Experts attributed this to the high comprehensibility, flexibility, conciseness and comprehensiveness of the design. From a practical perspective, the validation shows that the implemented design supports end-users in the problem context with developing integrations. A prototype of the platform was developed for a Dutch integration platform vendor and this

prototype was successfully used by end-users to implement IoT stream processing use cases. The validation shows that the end-users have a high intention to use and recommend the prototype for developing IoT stream processing integrations, which they attributed to the easy to use model-driven interface and the complete set event-stream processing operations. *Chapter 7 provides a detailed description of the validation results*. Overall, all but three of the integration challenges identified through research question 1 are directly mitigated by the design and the remaining challenges, with respect to integration costs, integration quality and semantics, are indirectly addressed. By resolving these challenges, the findings contribute to achieving stakeholder goals, including cost reductions, increased integration reliability, and reduced integration development and management efforts. *Chapter 8 discusses how each of the integration challenges is addressed in the design and how this contributes to stakeholder goals.*

Concluding remarks
Through the research outlined above, the main research question can be answered. This thesis shows how to design a model-driven integration platform that allows the development and management of stream processing integrations so that developers can preprocess data streams more efficiently in IoT integration. This is done through two main findings:

First, to show that the design allows for efficient IoT integration development, this thesis describes what the IoT integration challenges are, and how these challenges are addressed by implementing the proposed architecture. Specifically, the proposed design enables a reduction of IoT integration efforts, and ultimately integration costs, through an easy to use model-driven interface and a re-usable, embeddable infrastructure for scaleable, statefull stream processing. The design has been validated through the development of a prototype as an instantiation of the design in the problem context. Validation with end-users shows that the design successfully supports users with developing IoT stream processing integrations, without programming skills or prior knowledge about stream processing.

Second, to assist platform architects with designing and implementing the platform, a detailed architectural design and a breakdown of the components to be implemented is provided. The architectural quality of the design has been confirmed through validation sessions with enterprise architecture experts. Additionally, this thesis provides system-level implementation support through an exhaustive review of relevant technologies, frameworks, and architectures in the IoT domain and designs for stream processing and graphical code generation.

9.2 Limitations

Naturally, this thesis is subject to several limitations. Three dimensions for limitation are identified, namely the lack of existing research on this topic, validation limitations due to time constraints, and validation limitations and due to the sample.

First, concerning the lack of existing research. As was identified in Chapter 2 existing research on integration platforms is sparse, resulting in several gaps of knowledge. Consequently, the identification the integration platform architecture had to be made based on limited information, which may not reflect the actual diversity of the architectures in practice, limiting the external validity of this research. This lack of cross-referencing and literature on integration platform architecture hinders research in this area and without a common ground to rely on divergence in the respective field is likely. The same limitation holds for architectures of graphical model-driven stream programming platforms. While plentifully research is available on low-level components and implementations of model-driven stream programming solutions, research overviewing the architecture of such solutions is limited.

Second are the limitations in the time available for validation. Since time is limiting factor for validation in any research, compromises are unavoidable to accommodate the available time-frame. Compromises in this

research include limiting the number of dimensions that are validated, performing qualitative validation rather than quantitative validation, and limiting the number of participants during validation, and performing the validation in an artificial setting rather than implementing the design in practice. These compromises are discussed in more detail in Section 7.1. An effort has been made to validate the design and the prototype to the best possible extent given the scope of this research, which includes the development of a prototype and validation with three different experts and three end-users. With regard to the prototype validation, one limitation is that the prototype only covers a part of the design. This implies that only the design process (which is the most important process) could be validated with end-users, while deployment and management scenarios could not be covered by end-users and were only validated through experts opinion validation. The motivation for the scope of the prototype is discussed in Section 6.3. Overall, further validation with a more mature prototype, on a wider range of (quantitative) dimensions would increase the internal validity of this research, strengthening the inferences of the effects.

Finally, there is a bias with respect to the sample. Specifically, in participant selection and sample size, which relate to the external validity of this research. With respect to the participant selection, the prototype has been validated exclusively with participants from the same organisation that have experience with the same tools for modelling integrations. However, the sample within the organisation was diverse, since each of the participants worked on vastly different projects for different clients. The sample bias also applies to the expert validation sample. This sample includes one expert from the problem context and two external experts from the University of Twente that are familiar with the problem context. Ideally, however, the sample should contain more experts from different problem contexts. This is however difficult to achieve, considering the low availability of such experts. For instance, there is only 1 major integration platform provider in the Netherlands. With regard to the sample size, The total sample size for the validation is intentionally set at 6 participants (three experts, and three end-users). This is because each validation session is very time consuming and cannot be automated using, for example, surveys. Rather, intensive one-on-one sessions are required for both expert validation as well as end-user validation with developers. Since experts and developer time is very valuable, and since validation sessions are time-consuming, only a limited number of validation sessions can be held. As holds for all research, more extensive validation sessions, with more experts and end-users across a broader range of contexts would increase the external validity of this research. The selection bias only applies to the validation, the interview sessions conducted in Chapter 2 were conducted with a diverse sample to ensure a proper representation of the problem context.

9.3 Future research

This research brings several new opportunities for future research on IoT integration. First, future research could increase the internal and external validity of the findings through additional validations. Several research opportunities for further validation have been identified:

- The experts agreed that further validation of the design by implementing the design in practice would be a crucial next step (Section 7.2.2). This includes expanding the scope of the prototype to cover the complete design, and increasing the overall maturity of the prototype. This would not only allow for increased validation of the design, but it would also allow for a better demonstration of the capabilities of the design,
- The prototype, as well as the design, could be validated on more dimensions, such as all the quality attributes identified by Nelson et al. [148]. Furthermore, these validation sessions could be held with a broader range of end-users and architectural experts from different organisations to obtain further insights into the applicability and generalisability of the findings.
- The design could be experimentally evaluated to obtain quantitative insights about the efficiency, the costs, and the performance of the design when implemented, as suggested in Section 7.1. The current research does not employ quantitative comparisons, as it would be out of scope due to the time-consuming nature of such experiments. However, these methods could be used in future research to substantiate

and quantify the anticipated cost reductions and performance increase. When comparing the design to other stream processing based integration tools, one challenge would be the identification of alternative designs, considering the novelty of this approach and the lack of alternative architectures to compare with (Section 7.1, Section 8.1).

Second, the findings allow for future research on integration platforms in a general sense. Chapter 2 explains the need for further research into integration platforms, to avoid divergence in the field and consolidate knowledge. Research on integration platforms is sparse, and the few literature that is available focuses on messaging-based integrations. Future research could strengthen the body of knowledge on integration platforms by providing a unified design of an integration platform across integration patterns. Such 'hybrid' integration platforms are already available in practice, however, they have yet to be documented in research (Section 2.1.2.8). The current research provides the first step towards a hybrid description of integration platforms, by providing the first description of the architecture from a streaming perspective to cover IoT use cases. Similarly, the findings can also support future research into the architectures of model-driven programming platforms. While implementations for such platforms are extensively described in literature, there is a lack of research from an architectural perspective (Chapter 4).

In conclusion, as a result of this research, several novel research opportunities arise that could further strengthen the body of knowledge on integration platforms and model-driven programming platforms for stream processing to support IoT integration.

9.4 Contributions

This thesis presents the first architecture of a model-driven IoT stream processing platform to facilitate IoT integration. This section discusses both the scientific and practical contributions of the findings.

9.4.1 Scientific contributions

First, this research contributes an exhaustive analysis of the problem context (Chapter 2. Surveys address both architectures of IoT from a high-level viewpoint, as well as IoT platforms and middleware implementations. This provides researchers with an overview of the different architectural patterns and components in IoT architecture from both a practical and a theoretical viewpoint. Additionally, IoT integration is covered through a survey of the protocols, semantics and integration practices of IoT, contributing to a comprehensive view on the IoT integration domain. Furthermore, integration is addressed in general by providing an overview of different approaches towards enterprise integration and the architecture of integration platforms. Overall, the analysis of the problem context forms a solid foundation for any researcher in the domain of IoT integration, as this research provides a thorough definition of IoT integration, and a description of IoT integration challenges to be addressed.

Based on the results of the problem investigation, a gap of knowledge regarding model-driven approaches towards the integration of IoT with applications has been identified, which prevents flexible and dynamic integrations. This research addresses this gap by proposing a design for a model-driven platform for IoT stream processing. Existing model-driven platforms for (IoT) stream processing are oftentimes immature and incomplete (Section 4.2.2). Moreover, existing literature does not describe integration platforms from an architectural point of view. This leads to isolated knowledge and divergence in the field of model-driven development of IoT integrations. The proposed research proposes a comprehensive architecture of an IoT integration platform (Chapter 5), as well as an initial prototype (Chapter 6). This design provides a foundation for further research into IoT integration platforms, as well as research into hybrid integration platforms that provide integration capabilities across integration patterns (such as IoT). Researchers could, for example, research the applicability of the design in different contexts, or adapt it to different patterns, to evaluate and extend the ability of the design to function across context and domains (Section 9.3).

9.4.2 Practical contributions

Currently, developers have to overcome many challenges when developing IoT stream processing integrations (Section 2.3). These integrations are complex and time-intensive to develop, deploy and maintain. Research into model-driven platforms for IoT integration development that could address these issues is still immature and the few platforms that are available do not live up to the requirements for a true integration platform (Section 4.2.2). Organisations seeking to implement an integration platform for IoT, or that want to extend their integration platform with IoT stream processing support will find that no academic guidelines nor reference architectures have been presented to assist them with this.

The proposed design addresses these concerns, and assists organisations in implementing an integration platform for IoT. First, the design provides platform architects in the organisation with the requirements for the platform. Next, the design provides architectural directions on how to implement the platform and a full architectural breakdown of the components to be implemented (Chapter 5). In addition to providing architectural support, this thesis also provides guidance for implementation. First, through an extensive description of the problem context that allows platform architects to understand the protocols, semantics and middleware involved in IoT (Chapter 2). Second, through a survey of IoT stream processing frameworks and model-driven programming platforms (Chapter 4) which supports the platform architects in evaluating implementation-level mechanisms and frameworks. And third, through the description of a prototype which provides a tangible example of the platform that architects can refer to for practical guidelines on how to implement the design (Chapter 6).

Once the design is implemented, the resulting platform allows users to easily model IoT stream processing integrations and deploy them on a re-usable infrastructure. The platform assists users in overcoming key challenges in IoT data processing through a model-driven interface that allows users to rapidly create complex IoT integrations using a set of simple operations. Compared to manually developed integrations, integrations created and deployed through the platform are also easier to maintain and manage, hence contributing to a decrease in maintenance and management efforts for IoT integrations [10], [11]. Through this, the design fosters innovation by lowering the bar for IoT integration. For instance, by enabling organisations to easily unlock IoT data to support their business processes or offer novel smart products, applications or services.

Bibliography

[1] V. Gazis, M. Gortz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger, and E. Vasilomanolakis, "A survey of technologies for the internet of things," in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*. Dubrovnik, Croatia: IEEE, Aug. 2015, pp. 1090–1095. [Online]. Available: http://ieeexplore.ieee.org/document/7289234/

[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context Aware Computing for The Internet of Things: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014, arXiv: 1305.0982. [Online]. Available: http://arxiv.org/abs/1305.0982

[3] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference on - DAC '10*. Anaheim, California: ACM Press, 2010, p. 731. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1837274.1837461

[4] N. Jazdi, "Cyber physical systems in the context of Industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*. Cluj-Napoca, Romania: IEEE, May 2014, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/6857843/

[5] P. J. Mosterman and J. Zander, "Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems," *Software & Systems Modeling*, vol. 15, no. 1, pp. 5–16, Feb. 2016. [Online]. Available: https://doi.org/10.1007/s10270-015-0469-x

[6] F. Wortmann and K. Flüchter, "Internet of Things - Technology and Value Added," *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221–224, Jun. 2015. [Online]. Available: https://doi.org/10.1007/s12599-015-0383-3

[7] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6714496/

[8] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7498684/

[9] A. Chowdhury and S. A Raut, "Benefits, Challenges, and Opportunities in Adoption of Industrial IoT," *International Journal of Computational Intelligence & IoT*, vol. 2, no. 4, p. 7, Mar. 2019. [Online]. Available: https://papers.ssrn.com/abstract=3361586

[10] N. Ebert, K. Weber, and S. Koruna, "Integration Platform as a Service," *Business & Information Systems Engineering*, vol. 59, no. 5, pp. 375–379, Oct. 2017. [Online]. Available: http://link.springer.com/10.1007/s12599-017-0486-0

[11] R. C. Pathak and P. Khandelwal, "A Model for Hybrid Cloud Integration: With a Case Study for IT Service Management (ITSM)," in *2017 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. Bangalore, India: IEEE, Nov. 2017, pp. 113–118. [Online]. Available: http://ieeexplore.ieee.org/document/8332564/

[12] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. [Online]. Available: http://link.springer.com/10.1007/978-3-662-43839-8

[13] H. Snyder, "Literature review as a research methodology: An overview and guidelines," *Journal of Business Research*, vol. 104, pp. 333–339, Nov. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0148296319304564

[14] S. R. Bader, M. Maleshkova, and S. Lohmann, "Structuring Reference Architectures for the Industrial Internet of Things," *Future Internet*, vol. 11, no. 7, p. 151, Jul. 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/7/151

[15] B. A. Kitchenham, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Software Engineering Group Keele University, Keele, UK, Tech. Rep., 2007, event-place: Lund, Sweden.

[16] O. Zimmermann, C. Pautasso, G. Hohpe, and B. Woolf, "A Decade of Enterprise Integration Patterns: A Conversation with the Authors," *IEEE Software*, vol. 33, no. 1, pp. 13–19, Jan. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7368007/

[17] S. H. Shah and I. Yaqoob, "A survey: Internet of Things (IOT) technologies, applications and challenges," in *2016 IEEE Smart Energy Grid Engineering (SEGE)*. Oshawa, ON, Canada: IEEE, Aug. 2016, pp. 381–385. [Online]. Available: http://ieeexplore.ieee.org/document/7589556/

[18] P. Sethi and S. R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," *Journal of Electrical and Computer Engineering*, vol. 2017, pp. 1–25, 2017. [Online]. Available: https://www.hindawi.com/journals/jece/2017/9324035/

[19] T. Ara, P. Gajkumar Shah, and M. Prabhakar, "Internet of Things Architecture and Applications: A Survey," *Indian Journal of Science and Technology*, vol. 9, no. 45, Dec. 2016. [Online]. Available: http://www.indjst.org/index.php/indjst/article/view/106507

[20] P. Asghari, A. M. Rahmani, and H. H. S. Javadi, "Internet of Things applications: A systematic review," *Computer Networks*, vol. 148, pp. 241–261, Jan. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128618305127

[21] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of things journal*, vol. 3, no. 1, pp. 70–95, 2015.

[22] A. Alreshidi and A. Ahmad, "Architecting Software for the Internet of Thing Based Systems," *Future Internet*, vol. 11, no. 7, p. 153, Jul. 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/7/153

[23] D. Kutzias, J. Falkner, and H. Kett, "On the Complexity of Cloud and IoT Integration: Architectures, Challenges and Solution Approaches:," in *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security*. Heraklion, Crete, Greece: SCITEPRESS - Science and Technology Publications, 2019, pp. 376–384. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0007750403760384

[24] Y. Cao, Y. Chen, and B. Jiang, "A Study on Self-adaptive Heterogeneous Data Integration Systems," in *Research and Practical Issues of Enterprise Information Systems II Volume 1*, L. D. Xu, A. M. Tjoa, and S. S. Chaudhry, Eds. Boston, MA: Springer US, 2008, vol. 254, pp. 65–74. [Online]. Available: http://link.springer.com/10.1007/978-0-387-75902-9_7

[25] B. N. Silva, M. Khan, and K. Han, "Internet of Things: A Comprehensive Review of Enabling Technologies, Architecture, and Challenges," *IETE Technical Review*, vol. 35, no. 2, pp. 205–220, Mar. 2018. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/02564602.2016.1276416

[26] M. T. Moghaddam and H. Muccini, "IoT Architectural Styles: A Systematic Mapping Study," in *Software Architecture*, C. E. Cuesta, D. Garlan, and J. Pérez, Eds. Cham: Springer International Publishing, 2018, vol. 11048, pp. 68–85. [Online]. Available: http://link.springer.com/10.1007/978-3-030-00761-4_5

[27] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," in *2016 Cloudification of the Internet of Things (CIoT)*. Paris, France: IEEE, Nov. 2016, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7872918/

[28] T. Bradicich, "Exploring the Four Stages of an Industrial IoT Solution," Nov. 2016. [Online]. Available: https://community.hpe.com/t5/IoT-at-the-Edge/Exploring-the-Four-Stages-of-an-Industrial-IoT-Solution/ba-p/6917607

[29] A. Tiwary, M. Mahato, A. Chidar, M. K. Chandrol, M. Shrivastava, and M. Tripathi, "Internet of Things (IoT): Research, architectures and applications," *International Journal on Future Revolution in Computer Science & Communication Engineering*, vol. 4, no. 3, pp. 23–27, 2018.

[30] P. Tuwanut and S. Kraijak, "A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends," in *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015)*. Shanghai, China: Institution of Engineering and Technology, 2015, pp. 6 .–6 . [Online]. Available: https://digital-library.theiet.org/content/conferences/10.1049/cp.2015.0714

[31] M. Burhan, R. Rehman, B. Khan, and B.-S. Kim, "IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey," *Sensors*, vol. 18, no. 9, p. 2796, Aug. 2018. [Online]. Available: http://www.mdpi.com/1424-8220/18/9/2796

[32] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "IOT Gateway: BridgingWireless Sensor Networks into Internet of Things," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. Hong Kong, China: IEEE, Dec. 2010, pp. 347–352. [Online]. Available: http://ieeexplore.ieee.org/document/5703542/

[33] E. G. Petrakis, S. Sotiriadis, T. Soultanopoulos, P. T. Renta, R. Buyya, and N. Bessis, "Internet of Things as a Service (iTaaS): Challenges and solutions for management of sensor data on the cloud and the fog," *Internet of Things*, vol. 3-4, pp. 156–174, Oct. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2542660518300350

[34] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, "Survey of platforms for massive IoT," in *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*. Eger: IEEE, Jan. 2018, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/8325598/

[35] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta, "Role Of Middleware For Internet Of Things: A Study," *International Journal of Computer Science & Engineering Survey*, vol. 2, no. 3, pp. 94–105, Aug. 2011. [Online]. Available: http://www.airccse.org/journal/ijcses/papers/0811cses07.pdf

[36] D. Navani, S. Jain, and M. S. Nehra, "The Internet of Things (IoT): A Study of Architectural Elements," in *2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. Jaipur, India: IEEE, Dec. 2017, pp. 473–478. [Online]. Available: http://ieeexplore.ieee.org/document/8334789/

[37] Y.-H. Lee and S. Nair, "A Smart Gateway Framework for IOT Services," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. Chengdu, China: IEEE, Dec. 2016, pp. 107–114. [Online]. Available: http://ieeexplore.ieee.org/document/7917072/

[38] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in Internet of Things: Taxonomies and Open Challenges," *Mobile Networks and Applications*, vol. 24, no. 3, pp. 796–809, Jun. 2019. [Online]. Available: http://link.springer.com/10.1007/s11036-018-1089-9

[39] A. Farahzadi, P. Shams, J. Rezazadeh, and R. Farahbakhsh, "Middleware technologies for cloud of things: a survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 176–188, Aug. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2352864817301268

[40] M. Zdravković, M. Trajanović, J. Sarraipa, R. Jardim-Gonçalves, M. Lezoche, A. Aubry, and H. Panetto, "Survey of Internet-of-Things platforms," in *ICIST*. Kopaonik, Serbia: HAL, Feb. 2016, p. 6.

[41] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Computer Communications*, vol. 89-90, pp. 5–16, Sep. 2016, arXiv: 1502.01181. [Online]. Available: http://arxiv.org/abs/1502.01181

[42] C. Badii, P. Bellini, A. Difino, P. Nesi, G. Pantaleo, and M. Paolucci, "MicroServices Suite for Smart City Applications," *Sensors*, vol. 19, no. 21, p. 4798, Nov. 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/21/4798

[43] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of Cloud computing and Internet of Things: A survey," *Future Generation Computer Systems*, vol. 56, pp. 684–700, Mar. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X15003015

[44] K. J. Singh and D. S. Kapoor, "Create Your Own Internet of Things: A survey of IoT platforms." *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 57–68, Apr. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7879392/

[45] A. Triantafyllou, P. Sarigiannidis, and T. D. Lagkas, "Network Protocols, Schemes, and Mechanisms for Internet of Things (IoT): Features, Open Challenges, and Trends," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–24, Sep. 2018. [Online]. Available: https://www.hindawi.com/journals/wcmc/2018/5349894/

[46] Y. Li, "An Integrated Platform for the Internet of Things Based on an Open Source Ecosystem," *Future Internet*, vol. 10, no. 11, p. 105, Oct. 2018. [Online]. Available: http://www.mdpi.com/1999-5903/10/11/105

[47] G. Fortino, C. Savaglio, C. E. Palau, J. S. de Puga, M. Ganzha, M. Paprzycki, M. Montesinos, A. Liotta, and M. Llop, "Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach," in *Integration, Interconnection, and Interoperability of IoT Systems*, R. Gravina, C. E. Palau, M. Manso, A. Liotta, and G. Fortino, Eds. Cham: Springer International Publishing, 2018, pp. 199–232. [Online]. Available: http://link.springer.com/10.1007/978-3-319-61300-0_10

[48] M. Díaz, C. Martín, and B. Rubio, "State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing," *Journal of Network and Computer applications*, vol. 67, pp. 99–117, 2016.

[49] J. Dizdarevic, F. Carpio, A. Jukan, and X. Masip-Bruin, "Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–29, Jan. 2019, arXiv: 1804.01747. [Online]. Available: http://arxiv.org/abs/1804.01747

[50] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among IoT services," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 72–79, 2015.

[51] A. J. Jara, A. C. Olivieri, Y. Bocchi, M. Jung, W. Kastner, and A. F. Skarmeta, "Semantic Web of Things: an analysis of the application semantics for the IoT moving towards the IoT convergence," *International Journal of Web and Grid Services*, vol. 10, no. 2/3, p. 244, 2014. [Online]. Available: http://www.inderscience.com/link.php?id=60260

[52] A. Sheth, "Internet of Things to Smart IoT Through Semantic, Cognitive, and Perceptual Computing," *IEEE Intelligent Systems*, vol. 31, no. 2, pp. 108–112, Mar. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7435181/

[53] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor, "The SSN ontology of the W3C semantic sensor network incubator group," *Journal of Web Semantics*, vol. 17, pp. 25–32, Dec. 2012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1570826812000571

[54] K. Kotis and A. Katasonov, "Semantic Interoperability on the Internet of Things: The Semantic Smart Gateway Framework," *International Journal of Distributed Systems and Technologies*, vol. 4, no. 3, pp. 47–69, Jul. 2013. [Online]. Available: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jdst.2013070104

[55] A. Venceslau, R. Andrade, V. Vidal, T. Nogueira, and V. Pequeno, "IoT Semantic Interoperability: A Systematic Mapping Study:," in *Proceedings of the 21st International Conference on Enterprise Information Systems*. Heraklion, Crete, Greece: SCITEPRESS - Science and Technology Publications, 2019, pp. 535–544. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0007732605350544

[56] I. Szilagyi and P. Wira, "Ontologies and Semantic Web for the Internet of Things - a survey," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*. Florence, Italy: IEEE, Oct. 2016, pp. 6949–6954. [Online]. Available: http://ieeexplore.ieee.org/document/7793744/

[57] H. Rahman and M. I. Hussain, "A comprehensive survey on semantic interoperability for Internet of Things: State-of-the-art and research challenges," *Transactions on Emerging Telecommunications Technologies*, Feb. 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3902

[58] C. Janiesch, A. Koschmider, M. Mecella, B. Weber, A. Burattin, C. Di Ciccio, A. Gal, U. Kannengiesser, F. Mannhardt, J. Mendling, A. Oberweis, M. Reichert, S. Rinderle-Ma, W. Song, J. Su, V. Torres, M. Weidlich, M. Weske, and L. Zhang, "The Internet-of-Things Meets Business Process Management: Mutual Benefits and Challenges," *arXiv:1709.03628 [cs]*, Sep. 2017, arXiv: 1709.03628. [Online]. Available: http://arxiv.org/abs/1709.03628

[59] Shen Bin, Liu Yuan, and Wang Xiaoyi, "Research on data mining models for the internet of things," in *2010 International Conference on Image Analysis and Signal Processing*. Zhejiang, China: IEEE, 2010, pp. 127–132. [Online]. Available: http://ieeexplore.ieee.org/document/5476146/

[60] M. Ma, P. Wang, and C.-H. Chu, "Data Management for Internet of Things: Challenges, Approaches and Opportunities," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. Beijing, China: IEEE, Aug. 2013, pp. 1144–1151. [Online]. Available: http://ieeexplore.ieee.org/document/6682212/

[61] S. Lempert and A. Pflaum, "Towards a Reference Architecture for an Integration Platform for Diverse Smart Object Technologies," in *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, vol. 6, Feb. 2011, p. 14.

[62] C.-W. Tsai, C.-F. Lai, M.-C. Chiang, and L. T. Yang, "Data Mining for Internet of Things: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 77–97, 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6674155/

[63] M. O. Gokalp, K. Kayabay, M. A. Akyol, P. E. Eren, and A. Kocyigit, "Big Data for Industry 4.0: A Conceptual Framework," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. Las Vegas, NV, USA: IEEE, Dec. 2016, pp. 431–434. [Online]. Available: http://ieeexplore.ieee.org/document/7881381/

[64] W. He and L. D. Xu, "Integration of Distributed Enterprise Applications: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 35–42, Feb. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6165353/

[65] C. Sarkar, S. N. A. U. Nambi, R. V. Prasad, and A. Rahim, "A scalable distributed architecture towards unifying IoT applications," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*. Seoul, Korea (South): IEEE, Mar. 2014, pp. 508–513. [Online]. Available: http://ieeexplore.ieee.org/document/6803220/

[66] D. Schel, C. Henkel, D. Stock, O. Meyer, G. Rauhöft, P. Einberger, M. Stöhr, M. A. Daxer, and J. Seidelmann, "Manufacturing Service Bus: An Implementation," *Procedia CIRP*, vol. 67, pp. 179–184, 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S221282711731140X

[67] N. Serrano, J. Hernantes, and G. Gallardo, "Service-Oriented Architecture and Legacy Systems," *IEEE Software*, vol. 31, no. 5, pp. 15–19, Sep. 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6898686/

[68] M. Kleeberg, H. Kirchner, and C. Zirpins, "Information Systems Integration in the Cloud: Scenarios, Challenges and Technology Trends," in *Future Business Software*, G. Brunetti, T. Feld, L. Heuser, J. Schnitter, and C. Webel, Eds., 2014, pp. 39–54. [Online]. Available: http://link.springer.com/10.1007/978-3-319-04144-5_4

[69] S. Palanimalai and I. Paramasivam, "An enterprise oriented view on the cloud integration approaches - hybrid cloud and big data," *Procedia Computer Science*, vol. 50, pp. 163–168, 2015.

[70] L. González and R. Ruggia, "A reference architecture for integration platforms supporting cross-organizational collaboration," in *Proceedings of the 17th International Conference on Information Integration and Web-based Applications &Services - iiWAS '15*. Brussels, Belgium: ACM Press, 2015, pp. 1–4. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2837185.2843854

[71] G. Hohpe, “Enterprise Integration Patterns,” in *Proceedings of the 9th Conference on Pattern Language of Programs*, Monticello, Illinois, Jul. 2002, p. 36.

[72] K. Kritikos, P. Skrzypek, and M. Różańska, “Towards an Integration Methodology for Multi-Cloud Application Management Platforms,” in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion - UCC ’19 Companion*. Auckland, New Zealand: ACM Press, 2019, pp. 21–28. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3368235.3368833

[73] P. M. Singh, M. Van Sinderen, and R. Wieringa, “Reference Architecture for Integration Platforms,” in *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. Quebec City, QC: IEEE, Oct. 2017, pp. 113–122. [Online]. Available: http://ieeexplore.ieee.org/document/8089870/

[74] A. Bakulev and M. Bakuleva, “Moving Enterprise Integration Middleware toward the Distributed Stream Processing Architecture,” in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. Budva, Montenegro: IEEE, Jun. 2019, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/8760002/

[75] P. Dobbelaere and K. S. Esmaili, “Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper,” in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems - DEBS ’17*. Barcelona, Spain: ACM Press, 2017, pp. 227–238. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3093742.3093908

[76] R. Yasrab, “Platform-as-a-Service (PaaS): The Next Hype of Cloud Computing,” *ArXiv*, vol. abs/1804.10811, p. 21, 2018.

[77] I. F. Alexander, “A Taxonomy of Stakeholders: Human Roles in System Development,” *International Journal of Technology and Human Interaction (IJTHI)*, vol. 1, no. 1, pp. 23–59, Jan. 2005. [Online]. Available: https://ideas.repec.org/a/igg/jthi00/v1y2005i1p23-59.html

[78] N. Cunniff and R. P. Taylor, “Graphical vs. Textual Representation: An Empirical Study of Novices’ Program Comprehension,” in *Empirical Studies of Programmers: Second Workshop*. USA: Ablex Publishing Corp., 1987, pp. 114–131.

[79] K. N. Whitley, L. R. Novick, and D. Fisher, “Evidence in favor of visual representation for the dataflow paradigm: An experiment testing LabVIEW’s comprehensibility,” *International Journal of Human-Computer Studies*, vol. 64, no. 4, pp. 281–303, Apr. 2006. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1071581905001163

[80] J. P. Morrison, *Flow-based programming: a new approach to application development*, 2nd ed. Unionville, Ont.: J. P. Morrison, 2011, oCLC: 816295591.

[81] B. R. Hiraman, C. Viresh M., and K. Abhijeet C., “A Study of Apache Kafka in Big Data Stream Processing,” in *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*. Pune: IEEE, Aug. 2018, pp. 1–3. [Online]. Available: https://ieeexplore.ieee.org/document/8533771/

[82] H. Wu, Z. Shang, and K. Wolter, “Performance Prediction for the Apache Kafka Messaging System,” in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. Zhangjiajie, China: IEEE, Aug. 2019, pp. 154–161. [Online]. Available: https://ieeexplore.ieee.org/document/8855525/

[83] J. Yongguo, L. Qiang, Q. Changshuai, S. Jian, and L. Qianqian, “Message-oriented Middleware: A Review,” in *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*. QingDao, China: IEEE, Aug. 2019, pp. 88–97. [Online]. Available: https://ieeexplore.ieee.org/document/8905013/

[84] A. Cavacini, “What is the best database for computer science journal articles?” *Scientometrics*, vol. 102, no. 3, pp. 2059–2071, Mar. 2015. [Online]. Available: http://link.springer.com/10.1007/s11192-014-1506-1

[85] J. Webster and R. T. Watson, “Analyzing the Past to Prepare for the Future: Writing a Literature Review,” *MIS Quarterly*, vol. 26, no. 2, pp. xiii–xxiii, 2002. [Online]. Available: http://www.jstor.org/stable/4132319

[86] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, and J. Stein, "Building a replicated logging system with apache kafka," in *Proceedings of the VLDB Endowment*, vol. 8. Seoul: Association for Computing Machinery, 2015, pp. 1654–1655, iSSN: 21508097 Issue: 12. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84953887655&doi=10.14778%2f2824032.2824063&partnerID=40&md5=2835d2557f80a1ba6f352881b039958c

[87] R. Shree, T. Choudhury, S. C. Gupta, and P. Kumar, "KAFKA: The Modern Platform for Data Management and Analysis in Big Data Domain," p. 5, 2017.

[88] A. Alaasam, G. Radchenko, and A. Tchernykh, "Stateful stream processing for digital twins: Microservice-based kafka stream dsl," in *SIBIRCON 2019 - International Multi-Conference on Engineering, Computer and Information Sciences, Proceedings*. Institute of Electrical and Electronics Engineers Inc., 2019, pp. 804–809. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85079071242&doi=10.1109%2fSIBIRCON48586.2019.8958367&partnerID=40&md5=5f0481758f430edccf7567e0415b8f25

[89] J. Fernández-Rodríguez, J. Álvarez García, J. Arias Fisteus, M. Luaces, and V. Corcoba Magaña, "Benchmarking real-time vehicle data streaming models for a smart city," *Information Systems*, vol. 72, pp. 62–76, 2017, publisher: Elsevier Ltd. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85030771090&doi=10.1016%2fj.is.2017.09.002&partnerID=40&md5=a8e8561f91eb20efdad76a7297f98266

[90] M. Sax, G. Wang, M. Weidlich, and J.-C. Freytag, "Streams and tables: Two sides of the same coin," in *ACM International Conference Proceeding Series*. Association for Computing Machinery, 2018. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85055838860&doi=10.1145%2f3242153.3242155&partnerID=40&md5=54de5d55e5ccae9e6411d0a7ae665916

[91] H. Jafarpour, R. Desai, and D. Guy, "KSQL: Streaming SQL engine for Apache Kafka," in *Advances in Database Technology - EDBT*, K. Z. Reinwald B., Binnig C., Ed., vol. 2019-March. OpenProceedings.org, 2019, pp. 524–533, iSSN: 23672005. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85064933207&doi=10.5441%2f002%2fedbt.2019.48&partnerID=40&md5=65298a191bcf6406ad06512c6cc22a9d

[92] S. Noghabi, K. Paramasivamy, Y. Pany, N. Rameshy, J. Bringhursty, I. Gupta, and R. Campbell, "Samza: Stateful scalable stream processing at linkedin," in *Proceedings of the VLDB Endowment*, S. K. Boncz P., Ed., vol. 10. Association for Computing Machinery, 2017, pp. 1634–1645, iSSN: 21508097 Issue: 12. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85036620015&partnerID=40&md5=77d785062bc8cd88d18e101ea9e96dce

[93] M. Kleppmann and J. Kreps, "Kafka, Samza and the Unix Philosophy of Distributed Data," in *IEEE Data Engineering Bulletin*, vol. 38, Dec. 2015, pp. 4–14.

[94] M. Pathirage, J. Hyde, Y. Pan, and B. Plale, "SamzaSQL: Scalable Fast Data Management with Streaming SQL," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Chicago, IL, USA: IEEE, May 2016, pp. 1627–1636. [Online]. Available: http://ieeexplore.ieee.org/document/7530060/

[95] K. İnçki and M. Aktaş, "Improving Awareness in Ambient-Assisted Living Systems: Consolidated Data Stream Processing," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 187, pp. 89–94, 2016, iSBN: 9783319512334 Publisher: Springer Verlag. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85011263979&doi=10.1007%2f978-3-319-51234-1_14&partnerID=40&md5=955cb0f94d3c581809017667885f6879

[96] H. Lv, X. Ge, H. Zhu, C. Wang, Z. Yuan, and Y. Zhu, "Design and implementation of reactive distributed internet of things platform based on actor model," in *Proceedings of 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2019*, X. B, Ed. Institute of Electrical and Electronics Engineers Inc., 2019, pp. 1993–1996. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85067870421&doi=10.1109%2fITNEC.2019.8729169&partnerID=40&md5=e62e0900449055ca7577973a2d56f51b

[97] J. Meehan, N. Tatbul, and J. Du, "Data ingestion for the connected world," in *CIDR 2017 - 8th Biennial Conference on Innovative Data Systems Research*. Conference on Innovative Data Systems Research (CIDR), 2017. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85084013925&partnerID=40&md5=c9ee0872f74d3231f14512f12f3d3483

[98] M. Balduini, S. Pasupathipillai, and E. D. Valle, "Cost-Aware Streaming Data Analysis: Distributed vs Single-Thread," in *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. Hamilton New Zealand: ACM, Jun. 2018, pp. 160–170. [Online]. Available: https://dl.acm.org/doi/10.1145/3210284.3210294

[99] R. Bansod, S. Kadarkar, R. Virk, M. Raval, R. Rashinkar, and M. Nambiar, "High Performance Distributed In-Memory Architectures for Trade Surveillance System," in *Proceedings - 17th International Symposium on Parallel and Distributed Computing, ISPDC 2018*. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 101–108. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85053865726&doi=10.1109%2fISPDC2018.2018.00023&partnerID=40&md5=cb647b3efe66c88ba2c4ec2eac2ca875

[100] C. Barba-González, A. Nebro, A. Benítez-Hidalgo, J. García-Nieto, and J. Aldana-Montes, "On the design of a framework integrating an optimization engine with streaming technologies," *Future Generation Computer Systems*, vol. 107, pp. 538–550, 2020, publisher: Elsevier B.V. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85079616944&doi=10.1016%2fj.future.2020.02.020&partnerID=40&md5=247cda66bcb18d9022b1061693bb090f

[101] E. Shahverdi, A. Awad, and S. Sakr, "Big stream processing systems: An experimental evaluation," in *Proceedings - 2019 IEEE 35th International Conference on Data Engineering Workshops, ICDEW 2019*. Institute of Electrical and Electronics Engineers Inc., 2019, pp. 53–60. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85069216125&doi=10.1109%2fICDEW.2019.00-35&partnerID=40&md5=1d4cecac6147c94f034a6f919266e193

[102] G. Van Dongen and D. Van Den Poel, "Evaluation of Stream Processing Frameworks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1845–1858, 2020, publisher: IEEE Computer Society. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082884278&doi=10.1109%2fTPDS.2020.2978480&partnerID=40&md5=26b022034b6e88f7451b2a6f5952b215

[103] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan, "A survey of distributed data stream processing frameworks," *IEEE Access*, vol. 7, pp. 154 300–154 316, 2019, publisher: Institute of Electrical and Electronics Engineers Inc. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077809554&doi=10.1109%2fACCESS.2019.2946884&partnerID=40&md5=ddaf12ada663531aec42d33dd266b8c0

[104] F. Gurcan and M. Berigel, "Real-Time Processing of Big Data Streams: Lifecycle, Tools, Tasks, and Challenges," in *ISMSIT 2018 - 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies, Proceedings*. Institute of Electrical and Electronics Engineers Inc., 2018. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85060774920&doi=10.1109%2fISMSIT.2018.8567061&partnerID=40&md5=adc075547ba2208440ed3d7528659e4e

[105] V. Gurusamy, School of IT, Madurai Kamaraj University, Madurai, India, S. Kannan, School of IT, Madurai Kamaraj University, Madurai, India, K. Nandhini, and Technical Support Engineer, Concentrix India Pvt Ltd, Chennai, India, "The Real Time Big Data Processing Framework Advantages and Limitations," *International Journal of Computer Sciences and Engineering*, vol. 5, no. 12, pp. 305–312, Dec. 2017. [Online]. Available: http://www.ijcseonline.org/full_paper_view.php?paper_id=1621

[106] G. Hesse and M. Lorenz, "Conceptual Survey on Data Stream Processing Systems," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. Melbourne, VIC: IEEE, Dec. 2015, pp. 797–802. [Online]. Available: http://ieeexplore.ieee.org/document/7384369/

[107] G. J. Chen, S. Yilmaz, J. L. Wiener, S. Iyer, A. Jaiswal, R. Lei, N. Simha, W. Wang, K. Wilfong, and T. Williamson, "Realtime Data Processing at Facebook," in *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. San Francisco, California, USA: ACM Press, 2016, pp. 1087–1098. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2882903.2904441

[108] M. Dias de Assunção, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, Feb. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804517303971

[109] M. Gehring, M. Charfuelan, and V. Markl, "A Comparison of Distributed Stream Processing Systems for Time Series Analysis," 2019, iSBN: 9783885796848 Publisher: Gesellschaft für Informatik, Bonn. [Online]. Available: http://dl.gi.de/handle/20.500.12116/21808

[110] T. Kolajo, O. Daramola, and A. Adebiyi, "Big data stream analysis: a systematic literature review," *Journal of Big Data*, vol. 6, no. 1, p. 47, Dec. 2019. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0210-7

[111] X. Zhao, S. Garg, C. Queiroz, and R. Buyya, "A Taxonomy and Survey of Stream Processing Systems," in *Software Architecture for Big Data and the Cloud*. Elsevier, 2017, pp. 183–206. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780128054673000119

[112] Q.-C. To, J. Soto, and V. Markl, "A survey of state management in big data processing systems," *The VLDB Journal*, vol. 27, no. 6, pp. 847–872, Dec. 2018. [Online]. Available: http://link.springer.com/10.1007/s00778-018-0514-9

[113] S. Qian, G. Wu, J. Huang, and T. Das, "Benchmarking modern distributed streaming platforms," in *2016 IEEE International Conference on Industrial Technology (ICIT)*. Taipei, Taiwan: IEEE, Mar. 2016, pp. 592–598. [Online]. Available: http://ieeexplore.ieee.org/document/7474816/

[114] M. Dayarathna and S. Perera, "Recent Advancements in Event Processing," *ACM Computing Surveys*, vol. 51, no. 2, pp. 1–36, Jun. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3170432

[115] P. Carbone, G. E. Gévay, G. Hermann, A. Katsifodimos, J. Soto, V. Markl, and S. Haridi, "Large-Scale Data Stream Processing Systems," in *Handbook of Big Data Technologies*, A. Y. Zomaya and S. Sakr, Eds. Cham: Springer International Publishing, 2017, pp. 219–260. [Online]. Available: https://doi.org/10.1007/978-3-319-49340-4_7

[116] R. Tommasini, S. Sakr, E. D. Valle, and H. Jafarpour, "Declarative Languages for Big Streaming Data," 2020, version Number: 1 type: dataset. [Online]. Available: https://openproceedings.org/2020/conf/edbt/paper_T1.pdf

[117] G. Hesse, C. Matthies, K. Glass, J. Huegle, and M. Uflacker, "Quantitative Impact Evaluation of an Abstraction Layer for Data Stream Processing Systems," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. Dallas, TX, USA: IEEE, Jul. 2019, pp. 1381–1392. [Online]. Available: https://ieeexplore.ieee.org/document/8884832/

[118] M. Gökalp, A. Koçyiğit, and P. Eren, "A visual programming framework for distributed Internet of Things centric complex event processing," *Computers and Electrical Engineering*, vol. 74, pp. 581–604, 2019, publisher: Elsevier Ltd. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85042022936&doi=10.1016%2fj.compeleceng.2018.02.007&partnerID=40&md5=319d8165205ae09a8a3da64ebabb580d

[119] C.-H. Jin, Z.-M. Liu, M.-H. Wu, and J. Ying, "FastFlow: Efficient Scalable Model-Driven Framework for Processing Massive Mobile Stream Data," *Mobile Information Systems*, vol. 2015, 2015, publisher: IOS Press. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84935831498&doi=10.1155%2f2015%2f818307&partnerID=40&md5=57f4e1a13f6d121cfc95c8b744d4572e

[120] A. Soylu, M. Giese, R. Schlatte, E. Jimenez-Ruiz, O. Ozcep, and S. Brandt, "Domain experts surfing on stream sensor data over ontologies," in *CEUR Workshop Proceedings*, S. T. Bikakis A., Meditskos G., Ed., vol. 1588. CEUR-WS, 2016, pp. 11–20, iSSN: 16130073. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84977530378&partnerID=40&md5=0ab8095c7ba16e8be98a97b2a7393196

[121] A. Soylu, M. Giese, E. Jimenez-Ruiz, G. Vega-Gorgojo, and I. Horrocks, "Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users," *Universal Access*

in the Information Society, vol. 15, no. 1, pp. 129–152, Mar. 2016. [Online]. Available: http://link.springer.com/10.1007/s10209-015-0404-5

[122] A. Soylu, M. Giese, E. Jimenez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks, "Ontology-based end-user visual query formulation: Why, what, who, how, and which?" *Universal Access in the Information Society*, vol. 16, no. 2, pp. 435–467, Jun. 2017. [Online]. Available: http://link.springer.com/10.1007/s10209-016-0465-0

[123] A. Soylu, M. Giese, R. Schlatte, E. Jimenez-Ruiz, E. Kharlamov, O. Ozcep, C. Neuenstadt, and S. Brandt, "Querying industrial stream-temporal data: AnÂ ontology-based visual approach," *Journal of Ambient Intelligence and Smart Environments*, vol. 9, no. 1, pp. 77–95, 2017, publisher: IOS Press. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85010078916&doi=10.3233%2fAIS-160415&partnerID=40&md5=1c0af2d037f8881d77b77207ff76ef08

[124] J. Lucas, Y. Idris, B. Contreras-Rojas, J.-A. Quiane-Ruiz, and S. Chawla, "Rheemstudio: Cross-platform data analytics made easy," in *Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018*. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 1553–1556. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85057098362&doi=10.1109%2fICDE.2018.00179&partnerID=40&md5=0d471e60979bb0bf848cf2925de1a6c2

[125] S. Chawla, B. Contreras-Rojas, Z. Kaoudi, S. Kruse, and J.-A. Quiané-Ruiz, "Building your Cross-Platform Application with RHEEM," *arXiv:1805.11723 [cs]*, May 2018, arXiv: 1805.11723. [Online]. Available: http://arxiv.org/abs/1805.11723

[126] T. Mahapatra and C. Prehofer, "aFlux: Graphical flow-based data analytics," *Software Impacts*, vol. 2, p. 100007, Nov. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2665963819300077

[127] T. Mahapatra, I. Gerostathopoulos, C. Prehofer, and S. Gore, "Graphical spark programming in IoT mashup tools," in *2018 5th International Conference on Internet of Things: Systems, Management and Security, IoTSMS 2018*. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 163–170. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85059982001&doi=10.1109%2fIoTSMS.2018.8554665&partnerID=40&md5=ca819ab090aa8a7a4261a934d0e9c35e

[128] T. Mahapatra and C. Prehofer, "Graphical Flow-based Spark Programming," *Journal of Big Data*, vol. 7, no. 1, 2020, publisher: Springer. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077578119&doi=10.1186%2fs40537-019-0273-5&partnerID=40&md5=64a2e11f4e8d79d2eed1ae95b3444b3b

[129] T. Mahapatra, C. Prehofer, I. Gerostathopoulos, and I. Varsamidakis, "Stream analytics in IoT mashup tools," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, E. G. Kelleher C., Ed., vol. 2018-October. IEEE Computer Society, 2018, pp. 227–231, iSSN: 19436092. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056899799&doi=10.1109%2fVLHCC.2018.8506548&partnerID=40&md5=760a12d2abe6b8111cfae436c4dd6fb7

[130] T. Mahapatra, I. Gerostathopoulos, F. A. Fernández Moreno, and C. Prehofer, "Designing flink pipelines in IoT mashup tools," in *4th Norwegian Big Data Symposium, NOBIDS 2018*, ser. CEUR Workshop Proceedings, vol. 2316. CEUR Workshop Proceedings, 2018, pp. 41–53.

[131] M. Mesiti, L. Ferrari, S. Valtolina, G. Licari, G. Galliani, M. Dao, and K. Zettsu, "StreamLoader: An event-driven ETL system for the on-line processing of heterogeneous sensor data," in *Advances in Database Technology - EDBT*, P. E. Manolescu I., Ed., vol. 2016-March. OpenProceedings.org, 2016, pp. 628–631, iSSN: 23672005. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85013442005&doi=10.5441%2f002%2fedbt.2016.65&partnerID=40&md5=581cd8c6e29d936cce2f8fbaf86ed4b2

[132] W. Santos, L. Carvalho, G. Avelar, A. Silva, L. Ponce, D. Guedes, and W. Meira, "Lemonade: A scalable and efficient spark-based platform for data analytics," in *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*. Institute of Electrical and Electronics Engineers Inc., 2017, pp. 745–748. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85027453978&doi=10.1109%2fCCGRID.2017.142&partnerID=40&md5=940d984388927acec75a0e94bb195143

[133] W. d. Santos, G. P. Avelar, M. H. Ribeiro, D. Guedes, and W. Meira, "Scalable and efficient data analytics and mining with lemonade," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2070–2073, Aug. 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3229863.3275599

[134] S. Schmid, I. Gerostathopoulos, and C. Prehofer, "QryGraph: A graphical tool for Big Data analytics," in *2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016 - Conference Proceedings*. Institute of Electrical and Electronics Engineers Inc., Oct. 2016, pp. 4028–4033. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85015796998&doi=10.1109%2fSMC.2016.7844863&partnerID=40&md5=97f02052980593e1da07217b62ab336d

[135] S. Sydow, M. Nabelsee, H. Parzyjegla, and P. Herber, "A Safe and User-Friendly Graphical Programming Model for Parallel Stream Processing," in *Proceedings - 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2018*, L. P. Kotenko I., Merelli I., Ed. Institute of Electrical and Electronics Engineers Inc., 2018, pp. 239–243. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85048795324&doi=10.1109%2fPDP2018.2018.00040&partnerID=40&md5=21be5dd0f00bf398e0e5c5a767458fe9

[136] L. Thamsen, T. Renner, M. Byfeld, M. Paeschke, D. Schroder, and F. Bohm, "Visually programming dataflows for distributed data analytics," in *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, K. G. Ak R., Ed. Institute of Electrical and Electronics Engineers Inc., 2016, pp. 2276–2285. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85015155718&doi=10.1109%2fBigData.2016.7840860&partnerID=40&md5=51061eac2a61a6478764ac56bb706a20

[137] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0," *Knowledge-Based Systems*, vol. 89, pp. 97–112, Nov. 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950705115002397

[138] ——, "ModeL4CEP: Graphical domain-specific modeling languages for CEP domains and event patterns," *Expert Systems with Applications*, vol. 42, no. 21, pp. 8095–8110, Nov. 2015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0957417415004479

[139] H. Eichelberger, C. Qin, and K. Schmid, "Experiences with the Model-based Generation of Big Data Pipelines," in *Datenbanksysteme für Business, Technologie und Web (BTW 2017)*, Stuttgart, Germany, Mar. 2017, p. 8.

[140] J. Kranjc, R. Orač, V. Podpečan, N. Lavrač, and M. Robnik-Šikonja, "ClowdFlows: Online workflows for distributed big data mining," *Future Generation Computer Systems*, vol. 68, pp. 38–58, Mar. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X16302709

[141] D. Riemer, F. Kaulfersch, R. Hutmacher, and L. Stojanovic, "StreamPipes: solving the challenge with semantic stream processing pipelines," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems - DEBS '15*. Oslo, Norway: ACM Press, 2015, pp. 330–331. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2675743.2776765

[142] M. E. Iacob, H. Jonkers, D. Quartel, H. Franken, and H. van den Berg, *Delivering Enterprise Architecture with TOGAF\circledR and ARCHIMATE\circledR*. BIZZdesign, 2012.

[143] B. Gils and S. Dijks, *The practice of enterprise architecture.* s.l: BiZZdesign Academy, 2015, oCLC: 1141024623.

[144] T. Górski, "Model-Driven Development in implementing integration flows," *Applied Computer Science*, vol. 9, pp. 66–82, 2015.

[145] I. Dávid, I. Ráth, and D. Varró, "Foundations for Streaming Model Transformations by Complex Event Processing," *Software & Systems Modeling*, vol. 17, no. 1, pp. 135–162, Feb. 2018. [Online]. Available: http://link.springer.com/10.1007/s10270-016-0533-1

[146] F. Houacine, S. Bouzefrane, and A. Adjaz, "Service architecture for multi-environment mobile cloud services," *International Journal of High Performance Computing and Networking*, vol. 9, no. 4, p. 342, 2016. [Online]. Available: http://www.inderscience.com/link.php?id=77830

[147] K. Schwaber, “Scrum development process,” in *Business object design and implementation*. Springer, 1997, pp. 117–134.

[148] H. J. Nelson, G. Poels, M. Genero, and M. Piattini, “A conceptual modeling quality framework,” *Software Quality Journal*, vol. 20, no. 1, pp. 201–228, Mar. 2012. [Online]. Available: http://link.springer.com/10.1007/s11219-011-9136-9

[149] F. D. Davis, “Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology,” *MIS Quarterly*, vol. 13, no. 3, p. 319, Sep. 1989. [Online]. Available: https://www.jstor.org/stable/249008?origin=crossref

[150] F. Timm, S. Hacks, F. Thiede, and D. Hintzpeter, “Towards a Quality Framework for Enterprise Architecture Models,” *EMISA Forum*, vol. 38, pp. 31–32, 2017.

[151] C. Boyce and P. Neale, “Conducting in-depth interviews: A guide for designing and conducting in-depth interviews for evaluation input. Pathfinder International,” *Watertown, MA: Pathfinder International*, 2006.

[152] V. Charpenay, S. Kabisch, D. Anicic, and H. Kosch, “An ontology design pattern for IoT device tagging systems,” in *2015 5th International Conference on the Internet of Things (IOT)*. Seoul, South Korea: IEEE, Oct. 2015, pp. 138–145. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7356558

[153] F. F. Reichheld, “The One Number You Need to Grow,” *Harvard business review*, vol. 81, pp. 46–54, 124, Jun. 2003.

Appendix A

IoT case descriptions

A.1 Protocol

The interview protocol has been adopted from Boyce et al. [151]. At the start of the interview, the interviewer introduces himself and states the purpose of the interview, which is to obtain an insight into the challenges companies face when integrating IoT, and to use this information for research into improving IoT integration. Next, it is stated that the interview will take 1 hour, and that all responses are kept confidential such that any information included in the report will not identify the respondent. Once the respondent has agreed to this, and has no further questions, the interview is started.

First, two questions are asked to provide context to the answers of the respondent:

- Could you briefly describe what your organisation does?
- Could you describe your role in the organisation?

Second, the main questions are discussed. Since the interview is semi-structured, the interviewer is free to follow up with new questions not listed below, or to go through the list in a different order depending on the progress of the interview.

- What are your organisations current use cases for IoT?
- What is your vision for future use cases of IoT in your organisation?
- What is the process for retrieving, processing and using IoT data?
  - How is IoT data ingested?
  - How and when is IoT data stored?
  - How is IoT data processed? Processing includes for instance the cleaning, enrichment or harmonization of data.
  - How is the processed IoT data used?
  - How is the IoT integration, from integration to the usage, implemented?
  - What infrastructure is used for this?
  - What tools and software are used in this processs?
- From a software perspective, what challenges does your organisation face when implementing IoT use cases?

After the interview has been completed, the interviewee is provided with a brief overview of his answers and is asked whether he/she has anything to add or clarify, after which the interview is concluded.

A.2 Infrastructure Construction Company

The primary applications of IoT for InfraCorp are decision support and predictive maintenance for infrastructural maintenance. An example includes the analysis of a water pump to see whether it pumped the appropriate amount of water in a certain period. If any deviations are found compared to the expectation, faults can be predicted and/or detected, and an issue can be created. The vision of InfraCorp is to further increase the use of IoT, to do more predictive maintenance, and hence reduce maintenance costs and increase customer satisfaction.

Most IoT projects within InfraCorp follow a three-step process; first, the raw data is collected and stored, then it is transformed to structured and harmonised data, and finally, the data is used to power a dashboard.

In the first step, data from technical systems, such as SCADA systems is collected. Oftentimes, this data is provided by partners who are responsible for collecting this data. This data is very heterogeneous and sparse and external tables with metadata, such as a mapping from device IDs to device properties, are provided to make more sense from the data. Data collection is a challenge by itself, as it may be provided over various channels such as FTP, cloud push or even physically on a hard drive. The volume of incoming data is very high, with up to 100.000 data points per day for a single application, which all have to be processed. In most instances, scripts are used to assess the incoming data to decide whether it should be processed further or whether it should be rejected.

Next, this raw data is to be translated into more useful data. This includes operations like aggregation over various time spans and enriching the data based on the metadata tables, as well as external sources such as weather data. The operations and the complexity of these operations vary per project. For all applications, a uniform table is established based on common properties of the raw data, and all data is to be translated to meet this format. All operations are defined in custom scripts, that are created on a per-project basis. All data is finally stored in an SQL database.

Finally, the data is used in applications. For typical projects, this is a PowerBI based dashboard, that shows visualisations based on queries on the database from the previous step. In the past, Mendix was also used as it allowed for immediate feedback, for example reacting on an incoming event immediately, which is not possible in PowerBI. However, instead of using Mendix, PowerBI is now the preferred option and complemented by event handling logic in Azure that can send emails/alerts.

There are however a few projects that do not follow the above process steps. For example, on one project, the IoT data is rated within the SCADA system, and when anomalies are detected the SCADA system will send out a ticket with a priority, location and a measurement value. An iPaas platform will accept these tickets and will have (partial) access to the SCADA system database. Then, the iPaaS will direct the tickets to an asset management system through an ETL pipeline. In another project, the data needed was not available, and InfraCorp had place sensors and put in place a full IoT infrastructure, which brought challenges regarding implementing and managing IoT data, as well as challenges related to the protocols to be used. Finally, an IoT related project was mentioned in which traffic cameras are to be used to recognise road damage. This brings a whole different range of challenges related to algorithm development, but also to the selection of tools for deployment and testing. For instance, the need for a scalable infrastructure powered by Kubernetes to process the large image/video files produced by the cameras was mentioned as a challenge.

A.3 Real Estate Construction Company

EstateCorp has various large IoT projects running. Three projects were highlighted during the interview, a smart facility management solution, a smart home energy solution, and a smart portal.

The smart facility management solution is mainly concerned with workspace occupation. Sensors in the building can detect which workspaces are being used, and this information can be used to efficiently use and manage the workspaces in the building. An application has been developed that can show heatmaps with workspace occupation, as well as historical data on workplace usage. The smart facility management solution is to be included in the smart portal. This smart portal will not only show workspace occupation but also temperatures, energy uses and other comfort data. All this data is collected by the local facility management solution, this data is retrieved by a cloud-based solution named SkySpark. SkySpark is an analytics and data management application for the domain of facility management. It is an implementation of the HayStack API which provides simplified semantics in the domain of building automation, using semantic labels or 'tags' [152]. SkySpark is able to support the derivation of meaning from data, by labelling data with tags. This data is extracted from the SkySpark API and exposed in an internal digital data hub (DDH) built in Azure. This DDH has various layers, from raw data to customer data, and provides a common data model. APIs and applications that EstateCorp uses, also non-IoT applications are connected to this DDH. Once an app is connected to the DDH, such as the smart portal it can access all data available in the digital data hub. The smart portal is a low-code application built with the Mendix platform.

The last project is the smart home energy solution. This is a housing project in which new apartments are realised with IoT sensors to measure water and electricity use. The goal of these sensors is to give tenants more insight into their energy and water usage, such that EstateCorp can deliver more value to its customers and so that it can gain experience with IoT. The sensor data is collected and owned by an external IoT supplier, and EstateCorp gets access to this data. Using a REST API, the data is retrieved from the IoT supplier and stored in an Azure data warehouse using Azure Data Factory. For some data sources, such as for water monitoring, custom connectors had to be built to support the decoding of data. This raw data is implicitly uniform since a single kind of sensor is used per sensor type, i.e. the same water sensor is used for all water measurements. However, while the sensor data itself is not complex, sensor metadata such as the location of a sensor and the scope of what it measures (i.e. a building, room, or an apartment) does bring complexities. Once the raw data is stored, this data is then filtered, enriched and transformed for dimensional storage, this is done using python Azure functions. Views can be created of this dimensional data, which can then be used in PowerBI for visualisations. Tenants can then see these visualisations. In addition to seeing their own usage, they can also compare their usage with similar apartments.

The smart home energy solution, while seemingly trivial, brings many challenges. One major challenge identified is related to the reliability of the sensors. Data may temporarily be missing, and in this case, data needs to be mocked up which brings challenges. A sensor may become unreliable or send error messages, and this needs to be detected and handled. And someone needs to be held responsible for monitoring the sensors and maintaining them. Another issue is vendor lock-in, EstateCorp confirms that their current infrastructure depends on the specific data formats used by the data provider, leading to vendor lock-in. If the application would need to support a different vendor or different IoT devices, a significant part of the architecture would need to be rewritten. Another challenge is related to storing metadata. Metadata, such the hierarchy of devices and the relations to what these devices measure, is complex and hard to capture, store and manage. And finally, a challenge was brought up related to the context and meaning of the data. Currently, the only the involved developers knows exactly what data points mean, and why they have that value. In other words, correctly interpreting the data requires implicit, contextual, knowledge. While this information could be documented, it is often not, and there is no central location for this information to be stored and managed.

In addition to the above applications, other applications included the use of SkySpark for zero energy housing. In such contracts, self-sustaining houses are built and EstateCorp responsible for the energy supply of the home. In the case that equipment is malfunctioning, or more energy is needed than expected, EstateCorp will be responsible for the costs. Therefore, EstateCorp uses SkySpark to predict maintenance and energy usage to pro-actively avoid such costs. The same principle is applied on other projects, as projects move to service-based contracts, companies like EstateCorp are forced to move to smarter solutions to predict

maintenance, increase their value propositions, and reduce costs. Another application of IoT during construction is the use of the gyroscope in construction workers smartphones to detect when workers fall and send alerts to support in such an event.

A.4 Consultancy firm

The consultancy firm designs tailored made IT systems for their clients using Mendix. In addition to these tailor-made IT systems, they also design IT integrations for their clients using an affiliated integration platform. These integrations are designed using the eMagiz iPaaS platform. Two IoT integration projects have been identified, one project for the Dutch ministry of infrastructure and another project designed for InfraCorp.

The project designed for the ministry of infrastructure is described as a simple integration. In this integration, data is pushed from the IoT data source of the ministry of infrastructure as JSON messages to the endpoints of the integration platform. There are three different data streams for three different kinds of data, however, they are all treated similarly. After checking whether the push is properly authenticated, the data is then forwarded as-is to the Azure Event hub using the available APIs. In Azure, the data can then be stored and analysed. Unlike traditional integrations that the firm designs, messages are handled and forwarded completely in the connector in the DMZ (the demilitarised zone that connects internal systems with the internet) and messages are not forwarded over the central message bus to reduce the message load. However, as a side-effect, single the iPaaS assumes all messages are sent over the message bus, no monitoring and error handling is available. That is, there are no statistics about how the specific integration, making this integration complex to manage and to govern.

The second project is from InfraCorp, and was briefly mentioned in the interview with InfraCorp as the project in which a SCADA system had to be integrated with an incident management system. The consulting firm is responsible for designing this integration using the integration platform, however, the project is yet to be implemented. The integration is twofold and consists of a stream of raw data that has to be sent to the Azure Event hub as well as a stream of incidents that has to be sent to the incident management system. The stream of raw data needs to be collected in order to perform analytics over the data stream to allow for predictive maintenance, rather than only capturing incidents that have already happened, as is the case for the incident stream. To access the data the connector of the integration platform is deployed on-premise near the SCADA system. This connector has partial database access to retrieve both the low-level measurement data as well as the high-level incident data. Several challenges are involved with this integration, especially for the lower-level data. First, the retrieval of large amounts of low-level data from the database is complex as traditional database queries may be insufficient to detect changes, and alternatives such as a delta service (comparing the new data-set with a cached copy) may be unfeasible or inefficient. Next, the transportation of the data poses a challenge, sending messages over the message bus as is typically done for integrations is not feasible as the terabytes of raw data would congest the message bus. Sending messages from the connector directly to Azure is the preferred alternative, however, this will pose concerns on the governance as with the project for the ministry of infrastructure. The final challenge that is faced is related to the transformation of data. The data stream will need to be aggregated, transformed and filtered to reduce the amount of data that needs to be transported. In addition to this, InfraCorp expects that the data needs to be transformed to a generic data model, such that all the data in the Azure Hub conforms to the same format. This is to ensure that the data processing applications developed on Azure can work with a single data format, and will not need the be changed for each data stream. Most of these operations, such as aggregation, transformations, and filtering are currently not available or very inefficient for large data-sets since the XML transformations available on the integration platform are relatively resource-intensive.

Appendix B

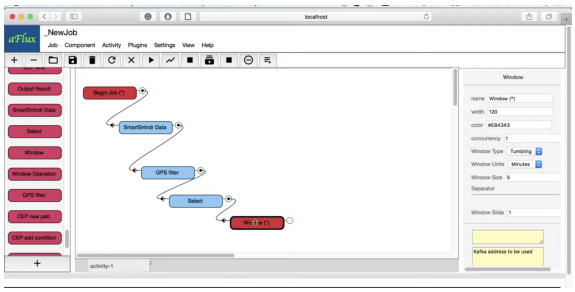
Model Elements

Element	Description
Data Object	This object can represent any data such as continuously streaming data, a set of stored data or a single data entry. Operations can consume and produce this data, and data objects are used as inputs and outputs of operations.
Trigger	A trigger can trigger an operation based on a certain event. *Example: Start an operation when a 5 minute window ends*
Input Connector	An output connector can connect to a data producer, and read data. *Example: Connect to a database and continuously stream new entries*
Output Connector	An output connector can connect to a data consumer such as an application or a database and write data to it. *Example: Stream data to a messaging system*
Filter	A filter is an operation that can check whether data conforms to certain conditions, such as a data schema, and can discard data if this is not the case. *Example: Confirm whether input data conforms to a certain schema or throw an error*
Select	A select can be used to retrieve data matching certain constraints from a set of data. *Example: Get all values higher than X*
Aggregate	Aggregation can be used to perform an operation over a set of data *Example: Get the sum, count, or average of the data*
Join	The user should be able to merge two data sources. *Example: Enrich data by joining metadata*
Translate	Translate can be used to map one data structure to another data structure. *Example: Convert from one data schema to another*
Windowing	Windowing can be used to get data over a certain period of time. *Example: Get all data from from 5 days ago until now*

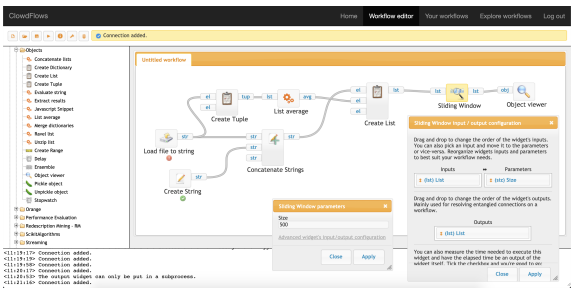
Table B.1: Model elements

Appendix C

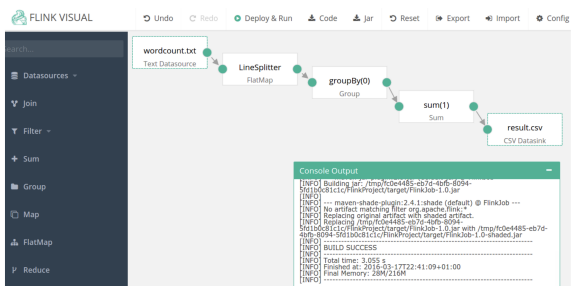
Editor UI Overview



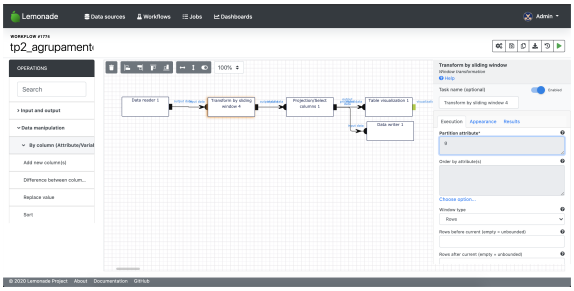
(a) aFlux



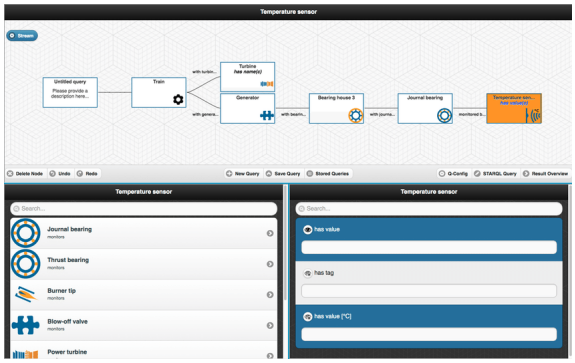
(b) ClowdFlows



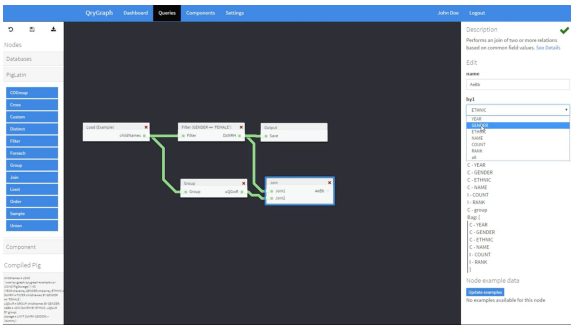
(c) Flision



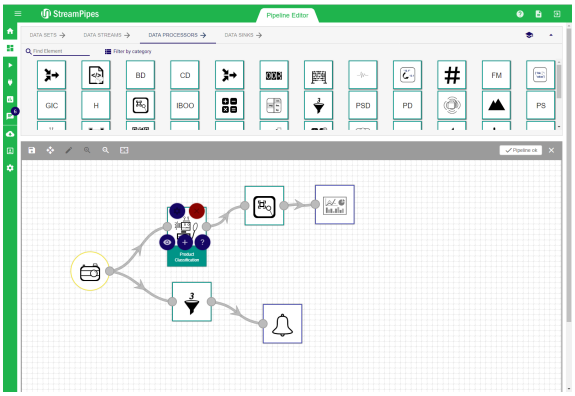
(d) Lemonade



(e) OptiqueVQS



(f) qryGraph



(g) StreamPipes

Appendix D

User Stories

Title	User Story	Story Points	Sprint
List operations	As an integration developer, I want see a list of all operations available to me so that I can easily select the operations to use.	3 Hours	2
Drag operations	As an integration developer, I want to be able to drag operations from the list to a location on the canvas so that I can easily visualise my model on the canvas and add operations to the model.	2 Hours	2
Connect operations	As an integration developer, I want be able to connect the input and the output of operations so that I am able to visualise and model the data flow between operations.	3 Hours	2
Configure operations	As an integration developer, I want to be able to view and edit the configuration options of each operation so that I am able to specify how an operation should function.	8 Hours	2
Generate code	As an integration developer, I want to be able to generate Spring Stream Kafka Streams statements of my integration so that I am able to execute the integration. *This covers the infrastructure for code generation, the time needed to create code generation logic for each specific operation is specified below*	30 Hours	1
Application template	As an integration developer, I want an application template that I can put my Spring Stream Kafka Streams statements into so that I am able to create a runtime application for my application.	10 Hours	1
Type checking	As an integration developer, I want the system to provide me with feedback about whether operations are type compatible into so that I am able to quickly correct mistakes in my integration design, for instance when I provide a stream when a table is required as input.	10 Hours	4
Import & Export	As an integration developer, I want the system to allow me to export integration models and to import them, so that I am able to save my work and resume editing at a later point in time.	8 Hours	4
	Operations		
Read	As an integration developer, I want to be able to read data streams or tables from a topic so that I can use it as an input for my integration.	16 Hours	2
Write	As an integration developer, I want to be able to write data streams to a topic so that external applications can consume the data produced by my integration.	16 Hours	3

Transform	As an integration developer, I want to be able to perform transformations from one schema to another so that I can overcome differences in data structures.	13 Hours	2
Flatmap	As an integration developer, I want to be able to perform flatmap from a schema to multiple messages so that I can split a record in multiple different.	2 Hours	4
Store variable	As an integration developer, I want to be able to read values from compute aggregates from streams and store them such that I can reference them later on.	16 Hours	4
Read variable	As an integration developer, I want to be able to read variables such that I can use them for comparisons in filters and for key based joins.	8 Hours	4
Re-key	As an integration developer, I want to be able to set keys of a record and change the value of a record based on the key, such that I can perform key-based joins, aggregations and groups.	2 Hours	4
Group	As an integration developer, I want to be able to group data streams by an attribute so that I can perform further computations, such as an aggregation, on a per group basis.	8 Hours	3
Aggregate	As an integration developer, I want be able to aggregate an attribute of a data stream over a window of time so that I can use aggregated values, such as the average value.	8 Hours	4
Filter	As an integration developer, I want filter based on certain attributes so that I can discard redundant or irrelevant records from the data stream.	2 Hours	4
Suppress	As an integration developer, I want suppress changes until a certain window of time expires so that I can use this as a trigger to perform actions only when all data in the window has been collected.	2 Hours	4
Join	As an integration developer, I want be able to join streams and tables so that I can enrich streams with additional information.	16 Hours	3
	Estimated Total	183 Hours	

Table D.1: User Stories

Appendix E

Prototype Operations

Operation	Description	Implementation
Read Stream	Read a topic as a stream.	Binder
Read Table	Read a topic as a table, the table will then reflect the latest value for each key.	Binder
Left-join	Left-join a stream or table, with a table on the right. If the table on the right contains the given key, the object for this key will be appended to the object for the key on the left.	leftJoin()
Store-join	Similar to a left-join, but joining with a key-value store rather than a table.	mapValues()
Merge	Merge two streams into a single stream that contains the records of both input streams.	merge()
Filter	Create a new stream/table that of record values that match a given condition. This implementation allows users to compare Strings or integers to a constant or to a value stored in state store.	filter()
Transform	Apply a JSLT transformation to the input values.	mapValues()
Transform-if	Apply one of two JSLT transformation to the input values depending on whether a certain condition holds.	mapValues()
Re-key	The functionality of this operation is threefold, one can set the key based on the object, add the key to the object, or set the key to null.	map() / mapValues()
Flatmap	Map an array object in a record to an array of records with one object. In Enterprise Integration Patterns [71] this is referred to as a 'splitter'.	flatMap()
Group	Groups (partitions) records based on their key, or on their value. This is a prerequisite for aggregating.	group() / groupByKey()
Window	Window a stream of records. Kafka offers a selection of windows, of which the Sliding and Hopping windows are covered by the prototype.	windowedBy()
Aggregate	Combine a set of records into a single record. The current implementation offers computing the average, count, maximum and total value of records. Aggregations can be running indefinitely onto a table, or they can produce an output stream of aggregations over a window.	aggregate(), map() / mapValues()

To-table	Converts a stream into a table, the table will then reflect the latest value for each key. Optionally, changes can be suppressed for a certain period of time, such that operations downstream are called in batches, for instance when a final value has been reached.	toTable(), suppress()
To-stream	Outputs the change-log of a table as a stream.	toStream()
Read-store	Read a value from a key-value store for a select key.	StateStore, QueryService
Write-store	Write a value to a key-value store for a select key. Also supports running aggregations, to continuously write an aggregated value to the store.	transform(), StateStore, QueryService
Write-stream	Write a stream to a topic.	to()
Write-table	Materialising a table to a state store. This continuously stores the latest values for each key in the state store, making it accessible for reading.	toStream(), toTable()
Constant	Define a string or an integer constant to be used in a condition.	String, int

Table E.1: Prototype Operations

Appendix F

Validation protocols and results

F.1 Single-case mechanism experiment

The interview protocol has been adopted from Boyce et al. [151]. At the start of the validation session, the interviewer introduces himself and states the purpose of the interview, which is to obtain feedback about the prototype of the design, and to use this information for research for evaluating the effects of this design. Next, the participant is informed that the session will take 1 hour, and that participation is completely voluntary and can be stopped at any time. Furthermore, it is stated that all responses are fully anonymous. Once the participant has agreed to this, and has no further questions, the validation session is started. First, the participant is shown a slideshow that introduces the prototype, followed by three use-cases that the participant should implement using the prototype.

Once these activities are completed, the interview is started as per the questionnaire below.

	Background & Expertise	
1	Could you briefly describe what your function is?	Open
2	Could you describe how long you have been working in this function?	Open

	Results per case study	
3	Could you complete the case within 20 minutes?	Toggle
4	If any, what challenges did you encounter while implementing the case?	Open
5	Are you aware of other methods or tools through which you could have designed this integration? How do these compare to the system?	Open

	Overall intention of use	
6	To which extent would you consider using the system during your own work?	Open
7	Would you recommend using the system to your colleagues? (Adapted from [153])	Scale
8	With respect to question 7, why / why not?	Open

	Perceived Ease of Use (Adapted from [149])	
9	I would find it easy to learn how to use the system (for developing IoT / streaming integrations)	Scale
10	I would find the system easy to use (for developing IoT / streaming integrations)	Scale
11	I would find it easy to get the prototype to do what I want it to do	Scale

	Perceived Usefulness (Adapted from [149])	

12	Using the system to build integrations would enable me to build IoT / streaming integrations much quicker	Scale
13	Using the system to build integrations would make it easier to build IoT / streaming integrations	Scale
14	Using the system to build integrations would increase my effectiveness (in creating IoT / streaming integrations)	Scale
15	I would find the system to be useful (for developing IoT / streaming integrations)	Scale
	General questions	
16	Do you expect the system to contribute to the ability to support IoT use cases within the organisation? Why?	Open
17	In your professional opinion, what do you like about the system?	Open
18	In your professional opinion, what improvements would you make to the system?	Open

Table F.1: Single-case mechanism experiment questionnaire

After the interview has been completed, the interviewee is provided with a brief overview of his answers and is asked whether he/she has anything to add or clarify, after which the interview is concluded.

F.1.1 Single-case mechanism experiment one

Question 1 Consultant Integration Developer
Question 2 5 Years
Question 3.1 Yes
Question 4.1 A hint was needed to remind that joins are key-based. Therefore, the realisation that the input stream needed to be re-keyed was the most challenging.
Question 5.1 The only alternative that comes to mind is using the eMagiz flow editor. Other methods such as coding, would not be suitable for me. Using the flow editor to build this integration would be significantly more complex, especially querying the database/table that contains the enrichment information as the requests and responses for this for each record need to be added to eMagiz using the 5 layer model. Potentially, doing this with messaging would take 2 to 5 times as long as it would be when the concept of event streaming would be used.
Question 3.2 Yes
Question 4.2 Most challenging is understanding what the output of an aggregation is. You would need to be aware of what this operation does exactly and how its output is formatted. Normally, a transformation should be used to transform the output of the aggregate to the specific format, but for this case, the output of aggregate already meets the expected output. Additionally, it was a challenge to realise that grouping the input is required for aggregation. However, the modeller indicates when an input is invalid, so this helped to figure out that another type of operation was needed to create the desired input for an aggregation.
Question 5.2 As for the first use case, it would be possible in eMagiz flow editor, but it would be significantly more complex. In this instance, because aggregation is not at all possible using the default patterns, a workaround would be needed. For instance, adding a database and then writing all values to that database only to retrieve those values later to perform an aggregate. This would take significantly longer, up to 5 or 10 times as long, to implement.
Question 3.3 Yes
Question 4.3 There were no significant challenges with this case.
Question 5.3 This can be done in messaging as well. The time needed to implement this would be comparable for the eMagiz flow editor with messaging, as for the prototype with event streaming.
Question 6 Until recently, there was only the option to use messaging during work. Therefore, the question of 'which pattern, messaging or streaming, should I use?' was not there. If the prototype were to be incorporated into eMagiz, you would be able to ask this question for every integration. Especially in use cases with high data

volume that have to be processed and used differently throughout the landscape, streaming will be relevant. Additionally, any use-case involving retention would be relevant. And finally, in use cases where producers and consumers can work independently from each other, for instance a consumer that would only need a daily summary rather than a continuous stream of data.
Question 7 5/5
Question 8 Given that event streaming is suitable for the integration. It is ideal that operations such as filtering, aggregation and transformation are much easier to use than with other patterns. This allows for quicker and more reliable integration development. This allows developers to unlock more customer data, and to use eMagiz for more use cases.
Question 9 - 11 4/5, 4/5, 4/5
Question 12 - 15 5/5, 5/5, 5/5, 5/5
Question Question 16 Definitely. First, many of our customers are in logistics and there is a wide range of use cases and opportunities for IoT there. I would expect the system to contribute to this, because it makes it significantly easier to unlock customer data and put it to use. If we look at existing methods within Cape and eMagiz, such as messaging, then its definitely more complex and less reliable to correctly build such integrations. This is, because you go beyond what messaging is designed to do. You are building a custom solution that does not fit within the pattern of messaging. As a result, you can get memory issues, configuration issues, and poor scalability. The editor (prototype) makes it possible for business consultants to model event streaming processors, which is a way better pattern for dealing with such integrations.
Question 17

- Instant errors about compatibility issues between blocks
- Categorisation of blocks in the palette, so that you know what blocks to start with, what blocks you end with, and what blocks to use in-between.
- The blocks inputs are visualised, such that it intuitively becomes apparent how many inputs to use.
- The options of each blocks are visualised, and placeholders are given, such that you know what each block is capable of.
- Editor Interface is overall easy to use and works intuitively as expected.

Question 18

- There is a learning curve to understand what blocks can be linked with each other. For instance, to understand that joining requires grouping.
- There is a learning curve to understand what a block does. For instance, what output does a block produce exactly and in what format? Documentation or a help tool could solve this.
- The notation for queries is complex. You could instead generate the query based on the schema. For instance, allow users to visually select attributes of the schema and generate the query accordingly.
- Similar to the notation for queries, the notation for transformations is complex. A visual aid for this, i.e. to drag lines between the input and output schema, would be preferable.
- To accelerate development, one could suggest blocks based on the process flow. For instance, suggests the block that the user could used based on the current output.

F.1.2 Single-case mechanism experiment two

Question 1 Integration Developer
Question 2 2 Years
Question 3.1 Yes
Question 4.1 Most challenging is understanding what the blocks do and how they work. For instance, for the Re-Key block, you must understand that the 'attribute' property is the path that you want to read from the stream and write to the key.
Question 5.1 It would have been possible to do this with the messaging pattern, but it would take significantly longer. Even if we assume that we can load the dataset to be used for enrichment into memory, we still need

to go through the full lifecycle of adding the database and mapping and transforming the data before it can be used for enrichment. For large datasets, it will be especially complex to perform this use case in eMagiz, and one will face performance issues. Since I am investigating event streaming within the organisation, I have less experience with messaging and a bias towards event streaming, however I would still assume that using the prototype would always be faster for this use case, regardless of whether the user is experienced or not.
Question 3.2 Yes
Question 4.2 As for with the first use case, the challenge is primarily in understand what the blocks do and how to use them. In this particular case, the realising that grouping was needed before aggregating was the primary challenge. A second challenge was with creating the query, the editor does not validate the query and therefore this allows you to make schema-related mistakes without the editor informing you that a mistake has been made.
Question 5.2 This could possibly be done using messaging with a custom expression. The prototype would be preferable compared to messaging.
Question 3.3 Yes
Question 4.3 As with the second and first use case, the challenge is primarily understanding which blocks are needed and how to use them.
Question 5.3 This would definitely be possible. Would be slightly faster compared to messaging due to the overhead the messaging brings, i.e. having to add all the systems, map the data formats, and go through the complete lifecycle for all systems.
Question 6 I think it's not a choice of using either the current flow editor, or the concepts of editor of the prototype. Instead, the new concepts could be brought to the existing editor to allow all use cases from within a single editor. Overall, I would use the new concepts (from the prototype) primarily when reading from, and writing to, streams.
Question 7 5/5
Question 8 While streams and messages may have a lot of overlap, they work slightly different and in most streaming cases the messaging flow editor won't do and the concepts of streaming processing are needed.
Question 9 - 11 4/5, 4/5, 3/5
Question 12 - 15 3/5, 4/5, 4/5, 5/5
Question 16 Yes, a new editor supporting the concepts of streaming will unlock many new use cases. The same holds for IoT, just as it holds for other streaming use cases.
Question 17 The editor works fluently and the interface is clear. The options and applications appear to be endless. All the operations that are relevant for streaming appear to be included.
Question 18 The editor look and feel could be improved. For instance, to access the configuration options of a block, you need to click the block, and then select the gear icon. Such actions could be smoother.
General feedback on the validation process itself: I feel like the introduction and the use cases could be better attuned to each other. For instance, to show examples that can be directly applied in the use case.

F.1.3 Single-case mechanism experiment three

Question 1 Consultant Integration Developer
Question 2 5 Years
Question 3.1 Yes
Question 4.1 Most challenging was the join object, how does it combine the two values and how do I add the object from the table to the stream. Concept of 'keys' and how they are part of a record, had to be became apparent but was quickly obvious. Overall, the case was very feasible, and once you are aware of what the blocks do it is reasonably obvious how to build the integration. This applies to all cases.
Question 5.1 This use case would be have been possible in messaging, but it would have been more complex. When the table to join on is static, a translation table could be used. However for large, dynamic, lists I would need to use a database since its the only way to be able to aggregate data from two sources. Therefore, whether I would use messaging or streaming depends on the use case. In this specific casus, I would have selected streaming because of the high load. However, for a small stream and table, or when complex transformations need to happen, I would use messaging.
Question 3.2 Yes

Question 4.2 The most complex part is realizing that you need to blocks to perform what seems like one operation. Specifically, the group and the window, which could conceptually be one block. But again, this would depend on the use case, as in this case it would be possible with one block, but in some use cases this may not be the case. However, as with the previous case, once you know what the blocks do, this is not an issue. Using multiple blocks is just a minor improvement, and combined blocks could be a part of a future revision.
Question 5.2 This case requires saving and persistent data, which is possible in messaging with workarounds but it is not something that messaging is very suitable for. You could also do this with a Mendix app, with scheduled events, data retrieval, etc. However, this is sub-optimal, since a Mendix app is not a tool for integration but rather for building standalone apps. Overall, this is not a easy use case for messaging. And this holds for all uses cases where data has to be persisted. Given that it is as-easy to do it with streaming as in this prototype, streaming would definitely be the preferred alternative. This holds for all use cases where data is to be persisted.
Question 3.3 Yes
Question 4.3 With this use case, I was more accustomed the editor and to the concept of stream processing and I had a reasonable idea of how to approach this. Possible the experience from the previous use cases helped, as I found this to be the easiest use case.
Question 5.3 This would be very easy to implement in messaging. Which pattern, messaging or streaming, would have my preference depends on the use case, and I have no overall preference. But it would be almost just as easy to do this in messaging as in the prototype.
Question 6 Again, this would very much depend on the use case. I would recommend this for typical event streaming use-cases, that is use cases with high-load, low complexity, or when you want end-systems to actively retrieve messages, such as in situations where the end-systems are often unreachable.
Question 7 4/5
Question 8 The system consists of simple components that aren't to complex. And it is a manageable number of operations that do what you would expect them to do. The way you work in the prototype is close to the way we are currently working, so I would expect each consultant to be able to quickly adopt this, given that there is some documentation to explain what each block does exactly.
Question 9 - 11 4/5, 4/5, 4/5
Question 12 - 15 4/5, 4/5, 4/5, 5/5
Question 16 IoT is textbook example of streaming, so I would say definitely. Especially because currently the platform features offered to consultants, such as messaging, are not ideal for IoT use cases. If the final implementation is as easy to use as the prototype, I would definitely see the value of this for supporting IoT.
Question 17 The prototype is clear, and in line with how consultants currently model integrations. The editor provides simple blocks, which allow you to quickly build functional integrations. Complete as-is.
Question 18 I would say that frequent combinations of blocks could be made available as composite blocks. This would make the consultants work easier and faster.

F.2 Expert validation

The interview protocol has been adopted from Boyce et al. [151]. At the start of the validation session, the interviewer introduces himself and states the purpose of the interview, which is to obtain feedback about the design quality of a novel IoT integration platform, and to use this information for improving and validating this design. Next, the participant is informed that the session will take 1 hour, and that participation is completely voluntary and can be stopped at any time. Furthermore, the participants are informed that all responses are fully anonymised. When full anonymity was not possible, for instance when experts would be traceable based on the their function and institution, participants were explicitly asked whether they would consent to be identifiable this way. Once the participant has agreed to all of the above, and has no further questions, the validation session is started with a presentation of the design.

Once the design is presented, the interview is started as per the questionnaire below.

	Background & Expertise	
1	Could you briefly describe what your function is?	Open
2	Could you describe what your expertise is in this function, and for how long you are experienced in this field?	Open
	General	
3	*Problem owners only* - Could you describe to which degree the design aligns with your current architecture?	Open
4	Could you describe, in your opinion, the explicit advantages of the design?	Open
5	Could you describe, in your opinion, any disadvantages of the design?	Open
6	Is there anything that you would add, remove, or change in the design? If so, what?	Open
	Comprehensiveness vs conciseness [150]	
7	To what extent would you consider the design to be comprehensive (complete)?	Scale
8	To what extent would you consider the design to be concise (compact)?	Scale
9	Would you make changes to the design to increase comprehensiveness or conciseness? If so, what changes would you make?	Open
	Usefulness [150]	
10	To what extent do you think the design would be usefull to practitioners to implement a platform for model-driven IoT / streaming integrations?	Scale
11	To what extent would the design make it easier for practitioners to implement a platform for model-driven IoT / streaming integrations?	Scale
12	*Problem owners only* - What is the likelihood that you would use this design in your organisation? *Non-problem owners only* - What is the likelihood that you would recommend the design to practitioners?	Scale
13	With respect to question 12, why would/wouldn't you do this?	Open
14	Do you see any opportunities for improving the usefullness of the design? If so, how?	Open
	Comprehensibility [150]	
15	To what extent were you able to understand the design?	Scale
16	Could the comprehensibility of the design be increased? If so, how?	Open
	Flexibility [150]	
17	To which degree do you think the design could be (re)used across different organisations and users?	Scale
18	Could the design be improved to increase flexibility, such that it can accommodate a wider range of contexts? If so, how?	Open

Table F.2: Expert questionnaire

After the interview has been completed, the interviewee is provided with a brief overview of his answers and is asked whether he/she has anything to add or clarify, after which the interview is concluded.

F.2.1 Expert interview MDE

Question 1 Associate professor at the University of Twente
Question 2 Over 20 years experience in architecture for distributed systems, service oriented architecture and model driven engineering.

Question 3 N/A
Question 4 With the proposed design, you offer an interface that is at the level of the user, and meets the specific requirements of the user. Therefore, end users do not need to concern about how the integration is actually build, deployed and coded. The presented design appears to be complete and seems generally applicable. However, only practice will show whether the design actually meets user requirements. The prototype already makes some design decisions to exclude specific requirements, however since the architecture is generic it can be adjusted to the specific use cases of the practitioners.
Question 5 As with the previous answers, all disadvantages are in potential and practice will show whether they are true disadvantages. Every time you make a design decision, for instance by setting a certain meta-model for the integration, it could be that this is in disagreement with the some user requirements. Therefore, the design choices made may be limitations. This is unavoidable, at a certain point in time, design choices have to be made. Even though the design is abstract, you have to make design decisions that impose limits since you are developing a prototype, even if the prototype can be translated to fit different contexts.
Question 6 Based on the designs I reviewed, I have no remarks for now. It appears that the right choices have been made in the functionality, methodology, and generalisation. However, again, only practice will show whether changes are needed.
Question 7 4/5
Question 8 3/5
Question 9 I would consider the design to be very comprehensive. On conciseness, it would depend on the perspective. From the perspective of the user (practitioner) it is concise. However, the design is inevitably not very concise in its totality, since the problem domain is complex. This requires an exhaustive design with many facets, for instance you have the three phases in the lifecycles, and the layered architecture with the three levels. Therefore, not being concise is almost unavoidable in its totality, but relative from the perspective of the user it may be concise.
Question 10 5/5
Question 11 5/5
Question 12 5/5
Question 13 This a typical problem context where MDE can be applied, so if practitioners want to offer this to their users this design would be usefull.
Question 14 No, actually not. But again, practice will show.
Question 15 4/5
Question 16 The Archimate and the metamodelling parts of the design could be separated better, with more structure. This is because I am typically involved in MDE projects that have a very clear distinction between the metamodels, the mapping and the other MDE elements of the design. However, I can related to the fact that it is hard to present the design as such, because the Archimate view is selected as the primary view. The MDE view is different, and more technical, and the combination between both views is tough.
Question 17 4/5
Question 18 Great re-usability, since this is a repetitive process that will be similar across contexts and can therefore be widely applied. I would see no immediate points for improvements, but again once this design is deployed to the problem context you will see how flexibility can be improved.
Overall, great application of MDE.

F.2.2 Expert interview CTO

Question 1 CTO of an enterprise integration platform
Question 2 Over 20 years of experience in enterprise integration, responsible for translating business requirements of the platform into technical requirements for the platforms architecture.
Question 3 The main features align with our current architecture. The design can be used as a supplement to our current architecture, to implement the event-streaming pattern. For use cases for event-stream processing, where inputs do not match outputs, we want to allow users to perform processing operations on the stream. Currently, we do this with event processors, but we would like to add stream processing to that so we can serve different use-cases. The design shown can assist us with this.

Question 4 The design fits the same model that we currently offer to users, but with a different technical implementation that allows us to serve different use cases. The design allows me to identify what components in our architecture would need to be changed. For instance, the changes needed in the runtime, and in the editor. The architecture breaks down the total solution for event-stream processing into sub-problems that can be analysed and compared to the current platform piece-by-piece. For users, the user experience remains the same. The power of our current platform is that we do not attempt to compete with technical tools. Rather, we offer a model-driven approach that is easier to use. It may be limited in some aspects, but we are able to reach different target users with this. And this is also reflected in the proposed design.
Question 5 The specific process in the design for deploying an executable integration to the runtime is different in our current platform than in the compared design. Sending an executable to the runtime creates a dependency on deployment, and the integration on the runtime. In our current platform, we only tell the runtime which integration to run, the runtime then fetches the integration and can re-fetch the integration and its dependencies at any time when we push updates to the platform so the runtime can keep itself up-to-date without the need to re-deploy.
Question 6 No, I do not not think so.
Question 7 4/5
Question 8 5/5
Question 9 Comprehensiveness: The design provides sufficient information to decide what we want and can do. Conciseness: The design is uncluttered, there is no unnecessary information. But there is also not to little information such that I'm missing something. There appears to be a good balance between comprehensiveness and conciseness.
Question 10 5/5
Question 11 4/5
Question 12 5/5
Question 13 On usefullness, the design provides a lot of support on the architectural direction to take. As a consequence of the conciseness of the design, a lot of work remains when actually implementing the architecture, however, this is unavoidable. I would use it, unless some would provide me with a good argument not to. The architecture seems to align, and it seems to do something that we want. One challenge is perhaps the business case, and finding users that want to use this.
Question 14 Questions are mainly with the implementation of the architecture. How big is the difference between implementations for stream processing and event processing (messaging). Because event-processing is not part of the design, its up to the user to discover the difference between the two. However, I would not expect such a comparison to be part of the design.
Question 15 5/5
Question 16 I got it
Question 17 4/5
Question 18 The design is a tool to build integrations with. You probably do not want to use this design if you are building just a few integrations. However, companies that are in the same business as we are could definitely use this. The design does not contain many elements specific to our product and platform, and the concepts presented are re-usable. Therefore, I'd say that the design is definitely re-usable in the 'niche' of practitioners seeking to implement a re-usable solution for building integrations.
General feedback: I liked how when you showed the design within our organisation that our people were inspired and how questions were raised about what the difference really is between event processing and stream processing, changing the way people think.

F.2.3 Expert interview EA

Question 1 Professor at the University of Twente
Question 2 15 years of experience in Enterprise Architecture. And additionally in business-process management, and model-driven software development.
Question 3 N/A
Question 4 We can consider both functional advantages, such as advantages of using the design for the user

and the organisation, and non-functional advantages. One functional advantage in any case data management. The platform allows for data management by easily allowing filtering, and other operations. These operations represent business rules, that originate from the operational business process. The design essentially allows allows the instantiating of this business process, creating an actionable process upon the data flow. This allows the active, dynamic, application of the business process, rather than retroactively applying it on a full data set. Since there are currently a lot of data flows, it is attractive (from a performance perspective) to act upon real-time data streams. From a non-functional perspective, the design allows an easy way to facilitate IoT integrations in your architecture. I think, in its core, that the design is also flexible, and allow you to easily adapt to different IoT devices and data streams.

Question 5 The design still requires expertise in data integration to be used. It is not like any novice inexperienced user would be able to use this. There is a learning curve to learn how to transform streaming data using the design. In conclusion, it is not for novices or people that have no technical experience.

Question 6 Not necessarily, however in a more mature and professional prototype I would suggest changes to the user interface to make it more accessible. The platform is currently focused towards more technical experts, such as developers and model-driven consultants. While I would be able to build an integration using the prototype, while I am not able to write code, you are not explicitly targeting business process developers.

Question 7 5/5

Question 8 4/5

Question 9 Conciseness is what I would expect from a software development project. There are many details, such as the sequence diagrams. This is nice to have, but I would prefer this in an appendix. But this may also be due to my background.

Question 10 5/5

Question 11 4/5

Question 12 4/5

Question 13 I would rate it a 4/5 because it is not a finished, since the design is not fully reflected in the prototype. One you have a full-fledged prototype, you are better able to demonstrate and sell the design to practitioners.

Question 14 Consider the technology readiness levels, that describe the level of readiness of a product from an idea to a commercial product on a scale of 1 to 5. The design is currently at a level 3 to 4. You are designing the platform, but you are still in the process of validating the design in practice and taking it into product. I would say the next key step is to validate the design with real-life data streams such that the performance of the design can be observed in reality to make the step towards an actual product.

Question 15 4/5

Question 16 I could not understand a few details, because I am not very technical. However, I was able to understand the design in general. The code generation section (of the prototype) was the most complex, but I was able to fully understand the remainder of the design.

Question 17 4/5

Question 18 These tools are very suitable for organisations already involved in smart-industry and technical innovation. Consider for instance any business involved in smart-logistics, and hospitals using smart/IoT devices. For businesses that do not use such devices, this design would be less use-full. Therefore, the only limitation I see for flexibility is the attributes of the organisation using it. Any innovative company seeking to actively seeking to innovate could adapt this.