UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Information Collection Platform for Smart Nudging

A Microservice-Based Approach

Raymon Skjørten Hansen

INF-3990 Master's Thesis in Computer Science, June 2020

To Me.

This time, I will finish!

“If debugging is the process of removing software bugs, then programming
must be the process of putting them in.”
–Edsger Dijkstra

“Premature optimisation is the root of all evil.”
–Donald Knuth

Abstract

This thesis aims to explore the problem of integrating heterogeneous data sources into the Smart Nudge system. The Smart Nudge system is a system that produces personalised nudges that are contextually relevant to each user. The system relies on access to live data that could be constructed and presented in specific ways to influence users behaviour towards an agreed-upon goal. The goal is to ascertain the suitability of a microservice-based approach to designing the component that is responsible for integrating various data sources. A small prototype of two microservices provided a practical look at integrating real-world sources, namely a Norwegian weather service and a bus tracking service in Chicago. The proposed architecture is analysed using a set of requirements derived from a theoretical examination of the Smart Nudge system and a general theoretical look at decomposition techniques used to evaluate microservice architectures. Evaluating the prototype revealed that the Smart Nudge system is highly dependant on augmenting data sources with additional meta-data to produce personalised nudges. The analysis indicates that a data-driven microservice-based architecture seems well suited to resolving some of the problems and requirements that are somewhat unique to the Smart Nudge system setting.

Acknowledgements

There are a number of people that I would like to thank for their support during the writing of this thesis. Three people, in particular, stand out. Remove any one of these, and I would not have been able to complete my Masters!

To my supervisor, Prof. Randi Karlsen, for your constant dedication to this project. For your uncanny ability to jump into my frame of mind and offer comprehensive guidance. For helping me avoid dead-ends without ever taking the wheel from me. For answering emails about the contents of a particular paragraph or section on a Saturday night while you were on holiday! You are an example to follow.

To Mads, for your enthusiasm for everything related to architecture (and computer science for that matter). You have been an inspiration to me since that first lesson you implemented Bresenham's algorithm while the rest of us struggled to put pixels on the screen. You not only offered some of the best advice and thoughts on the problems I struggled with, but you opened up your home to give me a place to keep writing when the Coronavirus closed campus. I am lucky to count you among my best friends, and I'm looking forward to us turning DIPS into a strongly distributed, loosely cohesive, independently coupled conglomerate of something remotely related to microservices!

Most importantly, to Tine, my love. You are not just responsible for me being able to write this thesis; you also gave me the time and opportunity to complete a five-year Computer Science program. Handling two kids is a challenge at the best of times, but you managed to carve time out for me to spend late nights and weekends on campus. You even manage to listen to me talk about how data integration is linked to architecture without falling asleep; that's no small feat. You are always there when I need you, and even when I doubt myself the strongest, you somehow manage to make me feel like it will sort itself out in the end. I love you!

To Thale and Kasper, for providing the best breaks.

Thank you also to Michael and Kine for proofreading. Thank you to anyone else who might, in some small way, have helped, whether you knew it or not.

Finally, thank you to me. I did it this time!

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BCSS	Behavior Change Support System
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
GDPR	General Data Protection Regulation
HCI	Human-Computer Interaction
HTTP	HyperText Transfer Protocol
ICC	Information Collection Component
IoT	Internet of Things
JSON	JavaScript Object Notation
NATO	North Atlantic Treaty Organization
ODS	Open Distributed Systems
QoS	Quality of Service
REST	REpresentational State Transfer
RMI	Remote Method Invocation
RS	Recommender Systems

RTT Round Trip Time

SOA Service-oriented Architecture

UiT The Arctic University of Tromsø

XML Extensible Markup Language

Glossary

Choice Architecture The structure and presentation of the available information used to make a decision.

nudge To deliberately structure information in a way that seeks to influence peoples behaviour and choices.

Recommender System A system designed to identify items that a user will likely find useful or interesting. Usually based on the users previous activities and preferences.

Smart Nudging The idea of using tailored nudges where the information and presentation is specific to a particular user.

/1 Introduction

Nudging stems from economic and political theory for influencing decisions and behaviour using suggestions, positive reinforcement and other non-coercive means, so as to achieve socially desirable outcomes. The term *nudge* was defined by Thaler & Sunstein [1], where it was defined as:

> *. . . any aspect of the choice architecture that alters people's behaviour in a predictable way without forbidding any options or significantly changing their economic incentives[1].*

A choice architecture refers to the "environment in which individuals make choices". An important note is also:

> *To count as a mere nudge, the intervention must be easy and cheap to avoid. Nudges are not mandates. Putting the fruit at eye level counts as a nudge. Banning junk food does not[1].*

Nudges aim to influence people's behaviour towards decisions that are beneficial for society, but usually also in the individual's long-term interest [1]. Nudges can, for example, encourage a healthier or more environmentally friendly behaviour. A nudge should both inform and motivate the user to choose the suggested activity or option.

Digital nudging is described as a "[...] subtle form of using design, information and interaction elements to guide user behaviour in digital environments, without restricting the individual’s freedom of choice.”[2]. Smart nudging differs from traditional nudges in that it attempts to match the information to the current situation of the user. Smart nudging is “digital nudging, where guidance of user behaviour is tailored, to be relevant to the current situation of each individual user.”[3].

A system for generating Smart Nudges is presented in Karlsen & Andersen [4] and Andersen et al. [3]. An early stage in the nudging process is to collect information that is used for creating the nudge. This includes both practical and motivating information about the nudged activity, such as bus routes and departure time, road and footpath conditions, distance and time to destination, cost of travelling, weather conditions and much more. The collected information is combined, analysed and processed to produce the right set of information to motivate and inform about the selected activity.

1.1 Problem Definition

A vital task in the information collection process, is interfacing with a large number of different devices, public Application Programming Interface (API) and data sources. The functionality can be likened to that of data warehouse solutions whose aim is to collect, process, canonicalize and provide easy access to multidimensional data from heterogeneous information sources[5].

The smart nudge setting will influence the requirements placed on this component of the information retrieval process. In order for a nudge to be useful, it needs to be adapted to not only the specific users preferences, but the context of the user. Since a nudge relies heavily on presenting information in particular ways to induce a user to do a specific action, the information it is based on must be accurate, correct and valid. A user who is prompted to walk to work because it is sunny and it only takes 10 minutes, will not trust the system after being drenched and finding it took 30 minutes to walk. These aspects often requires that the provided data has both temporal and geographical information attached to it.

In addition to the above mentioned requirements, the component needs to be adaptive. This means that adding new data-sources at a later stage and incorporate the data they provide does not require rewriting the system. Lastly, the component itself should be agnostic compared to the goal of the nudges. So if the goal changes from promoting environmentally friendly transportation to living a healthier life, this should not influence the functionality or design of the Information Collection Component (ICC). This is the component responsible for integrating various heterogeneous data sources and present them to the

Smart Nudge system as a repository of all available information.

1.2 Goals

The main goal of this thesis is to establish the main requirements of the ICC of the Smart Nudge system and evaluate if a microservice based architecture is uniquely qualified to meet these requirements. How can information collection for the Smart Nudge system be organised? How can the ICC be designed to be adaptive in terms of adding new data sources? How can it be designed to be agnostic in relation to the overall nudge-goal of the Smart Nudge system? Does a microservice based approach to designing the information collection part of the system offer any particular advantages?

Collecting information that is useful in constructing nudges, might present unique challenges and requirements. The goal is to identify these through analysing a proposed Smart Nudge system architecture. Next, I will design and implement a microservice based prototype incorporating the insights gained from the analysis with particular regard to the unique setting the smart nudge system presents. The thesis will establish a set of requirements for the information collection part of the system and evaluate how well a microservice based architecture manages to meet these requirements and help answer the research questions above.

1.3 Approach

One part of the research is to identify aspects and features of the Smart Nudge information collection process which might provide a basis for finding a set of requirements for the ICC. Identifying these is important, not only for the continued development and understanding of the system as a whole, but to establish a base set of criteria which can facilitate the design and implementation of an experimental prototype.

Using the above criteria as a guide, an experimental prototype will be implemented, again emphasising the identified unique characteristics derived from the smart nudge setting.

The analysis of the prototype will be an analytical study, comparing it to alternative solutions. This is to attempt to ascertain how well it solves the earlier criteria and requirements and hopefully establish a solid base-line that might aid and inform future design-related decisions about the information collection process of the smart nudge system.

1.4 Contributions

This thesis makes the following contributions:

- A prototype of how microservices that demonstrate the architectural approach to integrating sources. The sources they integrate should preferably demonstrate the versatility of the approach, while also revealing interesting aspects of data integration into the Smart Nudge system.

- An analysis which might help establish a base line of requirements and demands that need to be met in order to solve the complex information collection process of the Smart Nudge system.

1.5 Context

The context of this thesis is the Open Distributed Systems (ODS)[1] group at The Arctic University of Tromsø (UiT). The group focuses on middle-ware which aids in constructing distributed applications with a particular focus on information exchange, analysis of data and combination of data from multiple sources. Among others, the group focuses on various aspects of Smart Nudge Systems, including but not limited to; component based design, adaptability, personalisation, security, privacy, consistency and reliability.

1.6 Outline

The remainder of the thesis is structured as follows:

Chapter 2 - Technical Background Presents theoretical information about nudging i general. It also describes digital nudging and the Smart Nudge system that is the basis for the whole thesis. Middleware and Microservices are also presented.

Chapter 3 - Method Presents the research methods for this project.

1. http://site.uit.no/ods/

Chapter 4 - Design Presents the proposed design of the Smart Nudge architecture and the design of the proposed architecture for the Information Collection component together with changes made to the Smart Nudge system architecture.

Chapter 5 - Implementation Overview of the prototype.

Chapter 6 - Evaluation An two-part analysis of the prototype. The first is based on the established requirements and criteria. The second is a general architectural analysis of the proposed design, both drawing on the experience gathered while implementing the prototype and integrating real world data-sources.

Chapter 7 - Conclusion Concludes the thesis and outlines future work.

2 Technical Background

This chapter gives a thorough theoretical overview of the concepts that are relevant to the development and evaluation of the ICC in the Smart Nudge system. Section 2.1 Nudging gives a brief overview of nudging. Section 2.2 Digital Nudging places nudging in a digital context, while Section 2.3 Smart Nudging explains the concepts of smart nudging. Section 2.4 The Ethics of Nudging discusses some of the key ethical concerns to be considered in any system that attempts to change peoples behavior. Section 2.5 Smart Nudge System Architecture outlines the smart nudge system which is the main architectural context for this thesis, as presented by Karlsen et. al[4]. Of particular importance is the Information Collection Component(Section 2.6 Information Collection Component which can be seen as a middleware(Section 2.7 Middleware) between data sources and the rest of the Smart Nudge system. Finally, Section 2.8 Microservices outlines the architectural design used to build a prototype of the ICC.

2.1 Nudging

To nudge is to deliberately structure information in a way that aims to influence people's behavior and choices. Nudges work because people's choices depend on how the choices are presented[6]. The structure or presentation of a choice is influential because human beings are not entirely rational when it comes to making decisions[7]. People's capacity for calculating the correct probability of an outcome, given uncertain information, is minimal. Therefore we tend to rely on a set of simplified heuristics, which in turn leads to biases and errors in judgment[8]. For example, the measurement of distance (without tools) is often based on how sharp an object appears. How sharp an object appears is a somewhat valid heuristic, but it leads people to overestimate distances in foggy weather because objects naturally appear less clear[8].

Choice Architecture describes the structure and presentation of the available information used to make a decision[1]. The structure of contextual information affects choices, regardless of whether the structuring is intentional or not. Intentional structuring is a nudge. A nudge aims to predictably alter people's behaviour, with a particular goal in mind. It is important to note that nudges differ from simple mandates in that they do not remove specific options from the choice architecture; it merely alters the way relevant information is presented[1].

Nudging is closely related to Recommender Systems (RS)[9] employed by search engines for years. RS seeks to find items that match a user's interests, often based on that user's previous activities and interests[9]. RS are used in information filtering and e-commerce to help users find items of interest that they might not otherwise have found[10]. But nudging, and other persuasive technologies go beyond simply supplying necessary information. These systems aim toward behavioural change in relation to a specific goal[11]. Oinas-Kukkonen[12] introduces the term, *Behavior Change Support System (BCSS)* to describe;

> A behavior change support system (BCSS) is a socio-technical information system with psychological and behavioral outcomes designed to form, alter or reinforce attitudes, behaviors or an act of complying without using coercion or deception[12].

One technique used in nudging, is about presenting options in a way that is more appealing and thereby improving the probability of choosing the improved options[1, 13]. Recommender System can help find items or options that are more appealing because they are of interest to us and suits our preferences. Nudging can employ a variety of techniques and strategies to actualise the desired behavioral changes. Research has, to some extent, established a set of techniques and psychological effects that appear fundamental to nudges. **Table 2.1** presents an overview of these principles.

Table 2.1: The main principles and psychological effects used in nudging with descriptions[1, 13].

Principle	**Description**
Incentive	Make incentives more obvious and pronounced and thereby increasing their effectiveness. This normally involves focusing on positive attributes of an option or item.
Mapping	Focuses on mapping difficult to understand information onto familiar and easy to understand scenarios.
Defaults	This principle rests on the fact that people tend to trust information providers, this in turn legitimises pre-selected options done by the provider.
Feedback	Providing positive or negative feedback depending on the actions of the user. Heavily used in various gamification solutions.
Expecting errors	Accepting and designing for the fact that humans are error prone beings. A good system foresees possible errors people can make and ideally designs a system that does not allow them. An example is the ATM not giving money out before the credit card has been removed from the machine, thus making it impossible to forget the credit card after grabbing your money.
Structure Complexity	Is about gathering all available information concerning a choice or set of choices in order to make it easier to evaluate them.

2.2 Digital Nudging

While a lot of the theory and research on nudging[1, 13] focuses on the psychological and conceptual aspects, the theories can be applied in a digital setting as well [14, 15]. Digital nudging is conceptually the same; it is "[...] the use of user-interface design elements to guide people's behaviour in digital choice environments."[15] A simpler definition is that *digital nudging* is a nudge facilitated by information technology[16]. Many of the same concepts and principles (See Table 2.1), transfers easily to a digital setting, while others do not. Online users are more willing to disclose personal information but are more wary of default choices[14]. Digital nudging relies on two core mechanics. One is to alter the content, or *what* is presented[13], the other is to alter how the content is presented[6]. Interface design is the primary way the presentation is altered in an online setting[14].

Online environments can refer to the digital context, where people make choices. User interfaces on screens present the digital choice architectures. These online environments offer several advantages that could help increase the effectiveness of nudging. These include the opportunity to modify visual aspects of a choice, analysis of user behaviour to help evaluate various nudge strategies, and the opportunity to personalise the content of a choice[14]. Understanding digital nudging is essential because, as people make an increasing number of their daily decisions in a digital environment, the impact of the choice architectures that guide these decisions will increase proportionately[14].

2.3 Smart Nudging

As presented by Karlsen & Andersen[4], Smart Nudging is the idea of producing tailored nudges to users. Nudges are tailored by incorporating relevant contextual information specific to that user. It is a context-sensitive recommender system that aims to change the behaviour of the user as specified by the nudge goal. The nudge goal is a common good goal such as *living healthier* using *environmentally friendly transportation* or *eating less meat*. Their contextual and situational relevance defines smart nudges. Nudges will be generated based on a stated goal. The goal is something a user wants to achieve, but it constitutes a change that is too difficult to facilitate alone. The user knows the goal and consents to let nudges help bring about desired behavioural changes. The nudges are designed by combining information from a wide variety of sources. These sources include, but are not limited to, bus tables, boat, flight arrivals, and weather forecasts, sensor data like actual weather, traffic jams and road conditions, and user data collected from calendar plans, interests. The system combines the information gathered from the various sources, combines them with relevant contextual information such as time of day, season, and geographical position and produces a nudge, tailored to a specific user.

2.4 The Ethics of Nudging

Any system that aims to influence peoples behaviour in any way raises ethical concerns. It is important to highlight these concerns and briefly explore the main arguments of the discussion about the morality of nudges. This section groups the discussion in three main issues regarding nudges; 2.4.1 *Autonomy*, 2.4.2 *Transparency* and 2.4.3 *Nudge goals*. Both sides of the argument are presented when relevant. 2.4.4 *Smart nudge ethics* considers some issues that are particularly relevant in the smart nudge setting. For a more comprehensive overview of the main arguments both for and against nudging, see Renaud & Zimmermann[17].

2.4.1 Autonomy

The main argument for the use of nudging is presented in Thaler & Sunstein[18], which is that the *choice architecture* is present, whether by design or by circumstance[1]. When presented with the information needed to make a choice, this information will inevitably have some form of structure to it. Therefore, using this structure to achieve a 'common good' goal, is not unethical [1]. The arguments against this is that it still uses people as a means to an end, which is fundamentally wrong and devalues them as humans. The other is that by manipulating which choices people make, they are robbed of their autonomy[19].

Several core mechanics of nudging rely on the fact that humans are not entirely rational decision-makers. Biased heuristics[7] often govern our decisions. Some types of nudges, such as the *'Default'* See **Table 2.1**, work because people are unaware of their own biases towards keeping the status quo. Nudges violate our autonomy by operating out of sight. Because of this, they unethical[19]. Exploiting imperfections in human decision-making to achieve a goal could be said to be inherently manipulative and immoral[20].

An important corner stone of nudging is liberty and freedom of choice. A nudge does not remove options, but improves the chances of us picking some of them[1]. This notion of freedom has been challenged as being too simple. It only looks at the presence of choice. Proper ethical consideration should be given to the quality of the choices. This renders nudges abusive because they work by setting up barriers to limit our choices. These barriers might be subtle, but they still render nudging coercive[20].

Finally, Goodwin[20] argues that since people's capacity to make choices varies, any attempt to influence these choices will rob people of the opportunity to make choices on the same basis. This individual capacity is largely arbitrary, and the argument against it, is that this unfairness is present, regardless of a wilfully structured choice architecture or not[1]. Wilkinson[21] claims that if a person chose differently after being nudged, that person was manipulated. In short, peoples' differing capacity to consider their choice architectures makes them unfair if manipulated. Rawls' concept of fairness could be utilised to construct an argument against this. Complete freedom of choice in an ideal society might not lead to freedom in practice[22]. One could argue that a complex set of options is more unfair for those less equipped to make sense of it, than a sense of robbed freedom of choice is to those better equipped to handle it. A pro-social nudge in some cases might actually be more fair.

2.4.2 Transparency

The issue of transparency has to do with whether or not people know that their choice architectures have been altered. Do they know for what reason? Some argue that nudges "[...] work better in the dark."[19]. If people are made aware of how a choice architecture has been rearranged to induce them to a specific course of action, the are likely to resist and the nudge will not work[19]. If people resist nudges when they are informed about them, it implies that people see nudges as unethical[23]. One could further argue that non-transparency is a defining characteristic of nudges. This view has been challenged by several studies that provide evidence that transparency does not necessarily make nudges less effective[24, 25]. Sunstein[23] has argued that both these studies (Loewestein et al. and Bruns et al.[25, 24]), base the data on simple 'default option' experiments that might hide a complexity that remain untested. Depending on the goal and the extent to which a choice architecture is modified, people might indeed resist nudges if the nudge was presented with complete disclosure[23]. The general consensus seems to be that transparency is an important ethical requirement in the use of nudging[1, 23, 24, 25, 26, 27].

2.4.3 Goals

The issue of nudge goals, sits slightly outside the rest of the discussion which is more focused on the various moral and philosophical aspects of nudging. It is equally important to consider the implications of how nudging is used as a tool. To what end are people being nudged? Who are the choice architects?

Whether or not nudging in itself is immoral or not, several have pointed to the dangers and concerns about illicit nudge goals[1, 27]. The fundamental principles and mechanics behind nudging are not complicated, so they could easily be used for evil purposes[18]. Choice architects are flawed as well. They succumb to biases and prejudices and are as imperfect as those being nudged. An example of this is when a company offers a free or cheap starting period after which you are quietly enrolled in the payment plan at a higher rate. This is not done to make it easier for the customer to sign up, it is further it's own products and services[1]. Another example of this is when search engines offer companies the opportunity to pay to be put higher on a list of results from a users search[28]. As an argument against this, is that nudges with intentionally 'good' goals, can counter-act nudges for more illicit purposes.

Calo[29] points to the problem of the aforementioned 'common good' goal. Who decides what this shall be? And how do we know that this is a good goal? There are examples where people were nudged away from a particular kind of product thought to be unhealthy. This turned out to be incorrect[17]. In line with this argument is the risk of using nudges to perpetuate the moral failings of society. Similar problems exist in other fields where technology directly interferes in peoples' lives like Artificial Intelligence (AI)[30] that have been shown to often echo the inequalities of the system that produced it and "[...] perpetuate existing forms of structural inequalities even when working as intended."[30].

2.4.4 Smart Nudge Ethics?

Yeung[28] points to concerns about Big Data driven decision guidance systems. As large scale data collection and processing becomes more efficient and fine grained in terms of individual preferences and information, its potential for generating far more efficient nudges increases. Nudging has generally been employed in a broad scope and indeed they rely on the fact that people are generally biased and reliant on flawed heuristics. There is an increasing tendency in e-commerce to target individuals specifically with nudges[28]. Parallel to this, are concerns about such systems' ability to evaluate and refine nudges on an individual level, thus making nudges ever more effective[28]. Sætra[31] points to three factors that increase the efficiency of nudges. More data with which to tailor nudges. Better understanding of how people can be affected by nudges. And, better opportunities to deliver and target specific individuals.

These concerns are important to consider because they tie directly into the Smart Nudge setting of this thesis. Smart nudging is about presenting information designed to increase the probability of an agreed upon goal being reached.

The system will use various sources of personal and external data do accomplish this. The system must rest on a solid ethical platform that ensures that the user is not exploited in any way. The system must support full transparency. Smart nudge works toward a goal that the user has agreed with. The reasoning behind each nudge must be visible. In short, the user must be able to trace the complete provenance of each nudge. Finally the system must be able to disclose which nudging techniques that are being utilised in each nudge. As the nudge design component is likely to involve AI and machine learning models, providing accurate provenance might become a problem[32], but falls outside the scope and theme of this thesis.

2.5 Smart Nudge System Architecture

This section will present the smart nudge system architecture. 2.5.1 *Smart Nudge Design Process* outlines the workflow presented by Schneider et al.[14] and Karlsen et al.[4]. 2.5.2 *System Architecture* presents a high level overview of a distributed system based on the workflows established in 2.5.1.

2.5.1 Smart Nudge Design Process

Schneider et al.[14] proposes a four step design process for designing digital nudges (See Figure 2.1 Four step for designing digital nudges). Step one is to define the goal. Step two is to understand the users and how heuristics and biases play into their decision making process. Step three is to design an appropriate nudge using techniques and principles based on the previous steps results. Step four evaluates the effectiveness of the nudge. Information from step four will feed into each of the other steps and improve the overall quality of the nudge design process[14].

The smart nudge design process presented by Karlsen et al. [4] (see Figure 2.2 Flowchart outlining the steps to designing a smart nudge), adds a few steps to the process presented by [14](see Figure 2.1 Four step for designing digital nudges). The additional steps focus on the context needed to construct a nudge geared toward a specific user in a specific situation, thus making a nudge based on more relevant and fine-grained information. The process starts similarly by defining a nudge goal.

The system needs to understand users in a general and broad way based on the behavioural psychology of decision making. This is necessary to generate the nudge that is most likely to succeed. The system will also attempt to understand the specific preferences of individual users to generate even better tailored

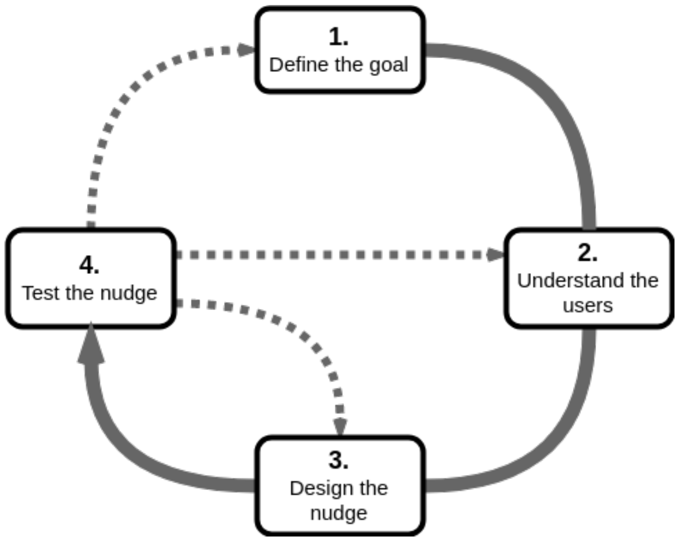

Figure 2.1: Four steps for designing digital nudges as proposed by Schneider et al.[14].

nudges. Understanding the situation of a user ensures the nudge is relevant to the user by not suggesting things that are currently impossible. Next, a target activity is chosen based on primarily on the nudge goal. Based on the target activity, the next step will gather relevant information from external sources. This information will then feed into designing the smart nudge before it is presented and evaluated. The nudge will not be effective if it is not context relevant, therefore this needs to be taken into consideration[4].

As in the four step process (see Figure 2.1 Four step for designing digital nudges), each step will generate information that can help improve on each of the separate steps of the smart nudge design process. This is to ensure that the system is able to react to changes in user behaviour as a result of the generated nudges. Understanding the user, the user situation and selecting relevant information require external information.

2.5.2 System Architecture

Based on the workflow and steps in Figure 2.2 Flowchart outlining the steps to designing a smart nudge, the high level main architecture is a distributed system of components all working to generate a personalised smart nudge. See Figure 2.3 Smart Nudge system architecture The Nudging goal component provides a sense of relevance to the system. The nudging goal will determine what types of information is needed, what is relevant contextual data and what is not. It will also determine what activities to suggest.

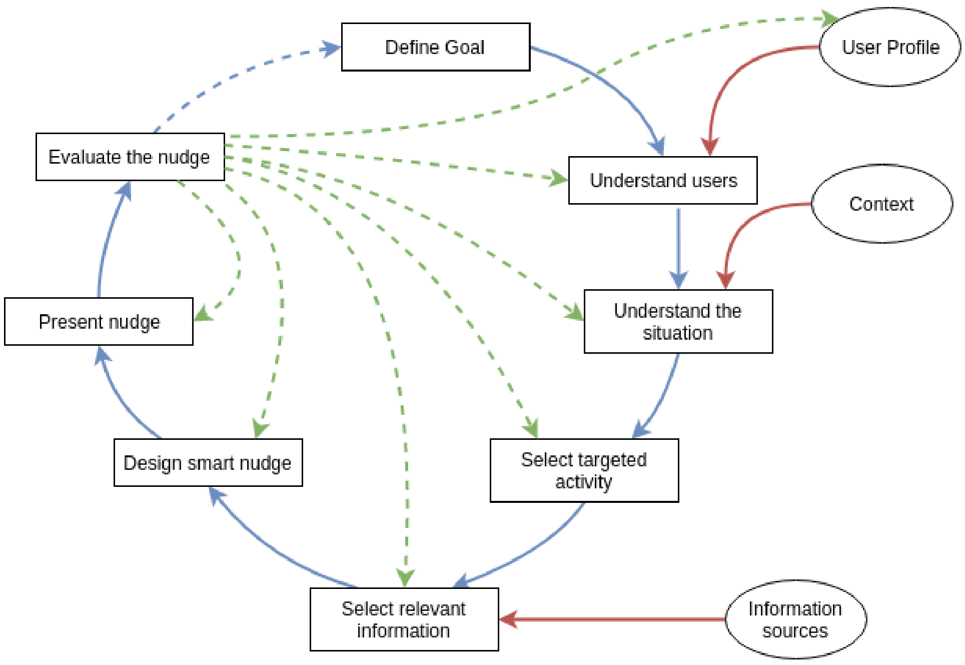

Figure 2.2: A flowchart outlining the steps needed to design a smart nudge[4].

The Profile Learner provides a way for the system to respond dynamically to changes in user data. It will update the user profile, which in turn feeds into how a nudge is designed to best suit a specific user. The users reaction to nudges will feed into the profile learner so as to avoid generating the same nudge over and over if it clearly is not working for this particular user. The Profile Learner will rely on both static data collected from the user such as interests, behaviour and capabilities[4]. The Profile Learner will ensure the User Profile gets updated and is able to provide relevant data to the Nudge Design component.

The Nudge Design component drives the whole process and is responsible for choosing the activity (that the user will be nudged towards), what information to use from the Information and Collection Analysis component and the User Profile component. It then needs to decide both when and how to present the nudge.

The User Reaction Component is responsible for detecting how a nudge was received, or rather, how effective the nudge was. This a vital part of making the system dynamic and able to react to changes in user behaviour. The user shouldn't be told how easy it is to ride a bike, if biking has already become the users primary means of transport. The User Reaction component might over time provide information about what type of nudges that seem to work on certain groups of people, thus making the nudges more effective[4].

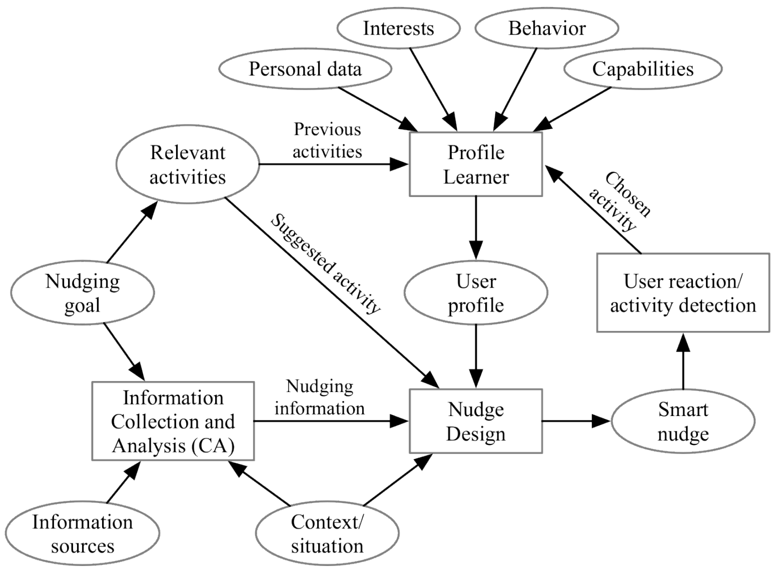

Figure 2.3: Architecture of a Smart Nudge system as proposed in [4].

2.6 Information Collection Component

An important component in the proposed system architecture is the Information Collection and Analysis Component. It is responsible for collecting data from a wide range of sources all with varying degrees of relevance depending on the goal, the situation and the user. It also collects relevant contextual and situational information such as geographical information, time of day, weather and lighting conditions. Establishing the requirements of the Information Collection and Analysis Component is important for the design of the component. Andersen et al.[3] identifies four main categories of data that an information collection component needs to support. These categories are all differentiated by aspects which are associated with time. In other words, past, present and planned/future occurrences respectively. **Historical** data deals with sensor data and past events. **Current** data provides current sensor data, travel conditions, pollution levels, current location of expected bus. **Plans** focuses on events that are either scheduled to take place such as meetings, bus and train times and public maintenance whereas **predictions** aims to represent events that are likely to happen, such as weather predictions, probabilities of traffic jams and bus schedule deviation.

The above categories gives a preliminary basis which can be used to establish a set of requirements for the Information Collection and Analysis Component.

The main goal of the component, however is to provide a layer of abstraction that hides the complexities presented by the host of various information sources. This functionality is commonly referred to as middleware.

2.7 Middleware

Middleware can generally be defined as software that hides lower level complexity of network or operating systems and presents clearly defined interfaces to developers and users[33]. It is also defined as software that combines software components together in some way[34]. The term 'middleware' stems from a 1969 North Atlantic Treaty Organization (NATO) Software Engineering Conference[1] and was used to describe software that facilitated access to, and communication with, more complicated lower level systems. Some have pointed out that the term is somewhat overused and has been applied to many different types of software[2]. In a distributed systems setting, middleware software is mainly an abstraction layer which aims to provide *ease of use*, *location transparency*, *data integrity* and *data delivery*[35]. It masks the heterogeneity of the underlying hardware and networking layers. In a more general sense, middleware can be seen as abstraction layers where each layer makes use of services offered by the layer beneath[36].

2.7.1 Internet of Things and other services

An increasing number of internet-connected, small, wireless devices make up what is commonly referred to as the Internet of Things (IOT)[37, 38]. The main challenge with IOT is its extremely large scale ubiquitous nature. A large number of events can be generated simultaneously by a network composed of many heterogeneous small devices. Due to the large scale, it becomes impractical, if not impossible to impose standards to ensure interoperability between devices in terms of protocols and data formats[38].

2.7.2 Middleware Categories

Middleware software is normally categorized based on their design or architectures. The most common middleware architectures include, but are not limited to the following:

1. `http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF`
2. `https://www.capitalware.com/dl/docs/MQOverview.pdf`

Message Oriented Middleware are indirect communication systems designed to facilitate integration between components of a distributed system. These are typically implemented as variations of message queues and are further divided into categories based on how messages are consumed. A producing process sends a message to a particular queue. A receiving process can block, while waiting for a message, it can check for a message and carry on with other tasks while waiting, or the message queue can notify all processes that subscribe to this particular event[36, 37, 35].

Distributed Objects attempts to hide the complexities presented by a distributed system by using distributed objects. Objects typically communicate through Remote Method Invocation (RMI). The main idea behind distributed object middleware is to use the encapsulation already present in an object-based solution to hide distribution[36, 35].

Service Oriented Middleware builds distributed system components based on services. These can be inter system provided or external services. This approach focuses on loose coupling, technological neutrality and service composability[37, 35]. Another key feature is dynamic discovery of services[36].

2.8 Microservices

Microservices is an architectural style where the aim is to develop a single application as a set of smaller services. Newman[39] defines them as small autonomous services that work together. According to Fowler & Lewis[40], the term *'microservices'* originates from a workshop on software architecture outside Venice in 2012. An important idea behind microservices as an architectural approach is to adhere to the single responsibility principle.

Robert C. Martin[41] outlines this in an article called *Principles of Object Oriented Design* where an important design principle is that an object or module should only change for a single reason. It should only have one responsibility. This is by now common knowledge, but microservice architectures aims to employ this on a larger scale. The reason is that when done correctly, it makes for a system that is more robust and better able to handle changes in requirements, technologies, data formats or communication protocols. Microservices tries to achieve a strong degree of decoupling which gives several benefits that are often highly valued in a large scale commercial setting, such as easier deployment, scaling, testing and technological choices per service[42].

Each service is modelled around actual business boundaries. As an example, a service might handle orders, another payment and a third authentication.

The idea is to foster decoupling by making it illogical to grow out of its business bounds. It makes no sense for the authentication service to also handle orders[39, 43].

In the microservice architecture, each service is independently scale-able and will typically run several instances of the core service. A load balancer will typically provide a single point of entry for the microservice and will manage mapping to the running instances to share load, kill instances that are not working or non-responsive as well as starting new instances as the need arises[39, 43].

2.8.1 Service-oriented Architecture

Isn't the microservice based architecture just the same as the good old Service-oriented Architecture (SOA)? According to some definitions, SOA is;

> *...a set of design principles whereby distributed systems are developed using sets of loosely coupled services that can be dynamically discovered and then communicate with each other or are coordinated through choreography to provide enhanced services[36].*

SOA is described as offering most of the advantages given by microservices such as interoperability between differing services, loose coupling and so on[36]. The approach was however criticised for being too loosely defined and tending to hide complexity behind the Enterprise Service Bus (ESB) part of the system which is responsible for managing communication between the various services[44]. Sam Newman argues that microservices is just a different approach to SOA and that SOA simply failed to sensibly outline specifically how a system should be split into different services[39]. Cerny et al[45] emphasises centralised coordination of services as integral to SOA and the lack of centralised coordination as integral for microservices. Many of the ideas and principles behind SOA are indeed the same as those for microservices.

2.9 Related Work

This section presents an overview of related work.

Tsilingiris et al.[46] showcases the Diligent system which is a platform for collecting and analysing data from online social networks. The Diligent system is divided into five distinct layers. 1 - Data Gathering which is a collection of microservices that perform gather data from various sources. 2 - Raw database stage consisting of multiple databases matching the sources. 3 - Data Integration stage is a new collection of microservices performing various data integration tasks and storing into; 4 - Integrated databases that serve the actual querying of the system. 5 - A final collection of microservices that handle the querying to the integrated database layer. The approach used by Tsilingiris et al.[46] is similar to the one taken in this project in that it seems (unclear) to dedicate specifically designed microservices to handle a corresponding data source. The approach differs from the design proposed here in that it uses multiple layers of storage and additional layers of microservices to further separate responsibilities.

Trinh et al.[47] presents a web based platforms that exhibits interesting similarities to the microservice-based architecture proposed in this thesis. Trinh et al.[47] presents a web based platform for integrating heterogeneous data sources through 'Linked Widgets'. These widgets are responsible for integrating data sources and expose them to end users who might then use these to build new applications. The approach taken is similar to a service based approach. Each widget is technically an HTML *iframe* which presents an interface with which to access the data. The platform and the widgets is web based and aimed directly at end users without any programming experience. The approach taken in this thesis is remarkably similar in that each microservice is responsible for integrating a single data source and present an interface that allows it to be integrated into the Smart Nudge system. The microservice based approach taken in this thesis is aimed at internal use in the Smart Nudge system, and thus needs to solve additional needs and requirements from the the Smart Nudge system.

Salvadori et al. (2017)[48] presents a framework for integrating heterogeneous data sources by using semantic web technologies. The authors describe the architecture of an Alignator which consists of four microservices that together handles the semantic integration of data from the microservices that handle the sources. This helps resolve problems of heterogeneous ontologies, or in other words how a source describes a field. The main research problem is that two data sources might describe the same entity in different ways, how does one resolve the problem of detecting this equivalence? The study does not go into detail about the architecture and management of the microservices that handle data sources. The issue of aligning ontologies is not explicitly handled in this thesis. This thesis assumes a global schema mapping is sufficient. Salvadori et al. (2019)[49] discusses Semantic data-driven microservices in greater detail.

A final point worth mentioning is a branch of nudging research that focus on employing nudging techniques in various online settings. Horne et al.[50] proposes an algorithm for producing nudges that help combat the spread of misinformation by information systems that are better served by not presenting nuanced sources of information. Wang et al.[51] explores how nudge and decision theory can be used in online environments (FaceBook) to limit disclosure of private information online. Other studies have also focused on improving privacy in online social media settings[52, 53]. The approach in these studies are mostly nudging through a browser extension or plug-in and the architecture of the systems used are often not the main concern of the studies. They do however speak to the effect and usability of automatic nudge systems, such as the Smart Nudge system.

3 Method

This chapter outlines the research method for this thesis. We will briefly revisit the problem statement as a background for understanding the various methods and methodologies described in this chapter. The chapter consists of the following sections: *Section 3.1 Research Problem and Hypothesis* gives a brief overview of the research problem and the hypothesis the project aims to solve or answer. *Section 3.2 Research Methods* Explains the various methods used in this project/thesis. *Subsection 3.2.1 Evaluation Methods* Looks at how an architectural approach might be evaluated against a non-existent software system.

3.1 Research Problem and Hypothesis

Before outlining the research methodologies used in this thesis, it pays to briefly revisit the problem statement and goals of this thesis. The problem arises from a novel and not yet implemented distributed computer system intended to produce personalized nudges to induce people to change according to an agreed upon goal. The production of these nudges relies on the system having access to information from a large number of vastly different sources of data. How can these heterogeneous information sources be integrated seamlessly into the system? How can this integration be organized in a way that remains agnostic to the aforementioned nudge goal? This requires a thorough understanding of the system itself. This is a challenge since the system does not yet exist. The goal is to evaluate a particular architectural approach to building the component that enables the integration of all the various information sources.

The hypothesis of this thesis is that a microservice based architecture applied to the ICC of the Smart Nudge system, will fulfil many of the requirements. The inherent modularity of a microservice based design will ensure the ICC is able to insulate the system from the heterogeneity of the data sources.

In order to evaluate an architectural approach, a set of requirements are needed. When the requirements and specifications have been established, a prototype of the component may be built and thus evaluated using the requirements. The prototype will hopefully allow for an evaluation of the architectural approach. It will likely also reveal other aspects or problems not covered by the requirements provided by analysing the proposed architectural design of the whole system. Simply implementing and iterating the design of a prototype is likely to reveal unforeseen problems and hurdles that needs to be overcome in order for the system to meet the requirements. Indeed the process might also uncover new requirements, not stated or established by analysing the proposed experimental system architecture.

3.2 Research Methods

Denning et al.[54] differentiates between *computer science* and *computer engineering* and establishes *computing as a discipline* to encompass both aspects of computer science. This duality puts computer science in an odd middle ground between a purely mathematical/analytical realm and the engineering side of things[55].

A somewhat crude characterisation could say that the mathematically founded side is mainly focused on the algorithmic aspects and involves theoretical proofs and formally provable solutions to very specific problems. The engineering side focuses on all problems that arise when building complex systems. Problems might focus on how a system can best be built, or how a design most efficiently solves a given problem. One could say there is a broader view which encompasses not just the purely algorithmic solutions, but attempts to take into account available technology, infrastructure, economic resources and developer resources[55].

Denning et al. further differentiates our discipline into three main paradigms, namely *theory*, *abstraction* and *design*. The engineering side of computer science is further divided into abstraction and design. Abstraction is rooted in experimental scientific method of forming a hypothesis, construct a model and make a prediction before doing experiments, collecting data and analysing the data[54]. The design paradigm is more strongly rooted in engineering and consists of the following steps:

1. **State requirements** - This is to establish a bar with which to measure how well a system lives up to its expectations, or how well it fulfils its purpose.

2. **State specifications** - Outlines how the requirements can be achieved.

3. **Design and implement the system** - The system is designed based on the requirements, and implemented based on specifications.

4. **Test the system** - The system is tested to see how well the initial problem is solved.

This thesis falls within the design but might also borrow from the other paradigms. The fourth point makes a little more sense when seen in light of the experimental scientific method. The design might more easily be tested against an initial hypothesis. This would be; *A microservice-based design solves the problems brought on by needing to integrate many heterogeneous information sources in this proposed Smart Nudge system.*

The project will mostly use a quantitative approach in the applied research methods. From a scientific philosophy standpoint, the project leans towards a *realism* point of view. Testing a prototype correctly should yield observations to provide credible facts about a proposed architecture. The main research method of the project is *applied research*[56]. The goal is to answer a specific question which arises from a given system or architecture. The project relies heavily on existing research, for providing the context in which the main problem can be understood.

The research approach is mainly inductive, but centres on two strategies. The projects can be seen as an experimental study in which a proposed design is implemented as a prototype to answer an initial hypothesis. One could also choose to see each source of information as a case study. It is the hope that integrating data from real world information sources will provide a more rigorous test of the design and thus strengthen any conclusions drawn from the analysis.

3.2.1 Evaluation Methods

Parnas[57] emphasises the importance of knowing the criteria for decomposing a system into modules, or in our case, microservices. Parnas[57] focuses on knowing the decisions that lead to changes in the one or more module. Changing formats, changing storage solutions, processing capabilities(in a REST based microservice architecture, this might mean adding an aggregation endpoint.). The fewer modules that need to change on account of these, the more efficient the modularisation.

It is worth pointing out, that the microservice architecture is often applied on existing monolithic systems. As such looking at how the process of moving from a monolithic based architecture to a microservice based one, might offer insight into how we might go about evaluating the decomposition of the proposed architecture of this thesis. Carvalho et al.[58] identifies the following criteria to employ when migrating to a microservice based architecture (See 3.1)

In the study[58], cohesion, coupling and software requirements were considered the three most useful criteria by which to extract microservices. Cohesion and coupling is to a certain extent tied together. A strongly cohesive component of a larger system is less likely to be heavily coupled with other components. Despite being derived from a study the process of migrating from a monolithic architecture to a microservice based architecture, the hope is that the most useful criteria used in this process might be useful in evaluating a microservice based architecture that didn't necessarily arrive from a migration process.

The third most useful criteria from the study was *requirements impact*, or how extracting a sub-component of a system might impact stated functional or non-functional requirements of the system[58]. This supports using an established system requirements as a means to evaluate a prototype. A less important criteria in the study is communication overhead. This becomes a far more important criteria in this particular setting because each source presents a subset of a global schema, the time to fulfil this needs to be kept small in order to complete a query/request within a reasonable amount of time. As each data source is wrapped by a microservice, each source will have the added communication overhead of the round-trip time from the API-gateway to the service and back. If one service is slow to respond, it impacts the quality of the overall response, and might even render the query response unusable in a nudge. Another aspect to be considered, regarding communication overhead is the amount of data wrangling, parsing and processing each microservice might need to do before being able respond with the requested data.

Another important criteria to be used when evaluating the Information Col-

Table 3.1: Criteria for Microservice extraction[58]

Criteria	**Description**
Coupling	Denotes the manner and degree of inter-dependency between software modules and functionalities.
Cohesion	The manner and degree to which inner elements such as data and functionalities are related to each other.
Communication Overhead	The additional amount of time not spent on the users needs due to the fact that microservices spend more time communicating with each other. In a monolithic approach, this would be local function calls.
Potential Reuse	The potential for a single service to be used in more than one place, thus reducing the total amount of code and/or improving the comprehensibility of the system.
Requirements Impact	Functional or Non-functional requirements that may impact what can and can not be extracted into a separate microservice.
Visual Models	Used to construct an understanding of an existing system, and then re-engineer it based on identified logical business functionalities.

lection component, is how well the information presented fits the needs of the Nudge Design component. Presenting a unified view of heterogeneous data sources, needs to fit the client. The only client in our case is the Nudge Design component, therefore the unified view needs to be judged based on how well it suits the needs of this only client. Dalecke[59] provides an in-depth theoretical study of the production of personalized nudges. Dalecke defines a nudge as being based on up to five sub-components. These sub-components define distinct aspects of a nudge. They are; *goal, content, incentives, effect and presentation*[59].

The content, incentives and effects are of interest in terms of defining the optimal view of the available data from all the sources. Content is pure factual information and from a transportation point of view this might include weather conditions, bus arrival times, road congestion, road or path conditions or travel times. This is information which is able to stand on it's own. Incentives are needed to make a nudge more attractive. They might include information on health benefits, environmental impact or time and money saved. This category somewhat bleeds over into the previous one, but generally works in comparison to other data. Taking the bus is x amount cheaper than taking... If the user typically walks, this makes no sense. If the user typically drives, it makes more sense. Effects refers to the type of psychological effect that is being utilised in the nudge. This depends on the user, the content and the incentives used[59]. While an in-depth description of all available psychological effects would fall outside the scope of this thesis, a few examples serves to define how this part of a nudge might define a particular need the Nudge Design component has.

Dolan et al.[13] describes several of the psychological effects used in Daleckes'[59] *effects* part of a nudge. Caraban et al.[60] offers a comprehensive overview of the use of nudging in Human-Computer Interaction (HCI).

Framing refers to the way information is presented. Framing fuel consumption in terms of fuel per distance, rather than distance per fuel will greatly effect how the information is perceived[61].

Defaults is the effect of people more readily choosing a pre-selected option. Egebark & Ekström[62] found a prevailing 15% drop in paper consumption by setting the default printing option to double-sided.

Loss Aversion is the effect of valuing a potential loss higher than a potential gain[61]. Gouveia et al.[63] used this effect in designing a smart watch interface to only show the last hour of activity to the user. They found that reducing the time frame prompted users to check the watch more often, and lead to an increase in physical activity[63].

Social Norms is an effect based on our desire to conform to social norms because we want to avoid ridicule. Nudging can use this effect in multiple ways. By raising the visibility of a users action to promote an action which is approved by society. Another option is to enable comparisons with others to exploit the fact that we tend to look to the actions of others for confirmation that what we ourselves are currently doing is 'correct', or at least within a perceived social average[60, 64].

Priming uses some sort of stimuli (visual, auditory, ...) to change the mood of the user and increase the likelihood of a particular behaviour[65].

Dalecke[59] offers less information regarding the particular type of data needed to produce the *effect* part of the generated nudge.

3.3 Summary

This section offers a brief summary of the main methods used to evaluate the proposed/prototyped Information Collection component.

Established Requirements The Nudge Design component is our client. Analysing the proposed Smart Nudge system architecture will be used to establish a set of functional and non-functional requirements as, seen from the Nudge Design component. Establishing these requirements, relies on theoretical research to make assumptions about a non-existing system. See chapter 3 - Design

Architectural analysis Established system design theory provides the basis for evaluating the architectural approach used in the prototype.

The combined results of these two approaches will be stronger because it not only evaluates the design with respect to the particularities of the Smart Nudge system, but with respect to distributed systems design in general.

4 Design

This chapter outlines the design of the ICC. *Section 4.1 Smart Nudge Architecture Revisited* briefly revisits the main architecture of the Smart Nudge System (as proposed by Karlsen & Andersen[4]) as a whole. It also explains some important changes made to the architecture. Section 4.2 Assumptions and Requirements Discusses some of the assumptions that needs to made regarding the Smart Nudge system as a whole. This section also outlines the main requirements of the ICC including some requirements that have not been regarded for the purposes of this thesis. They are no less important, and warrant research in the future, particularly scalability and privacy. Section 4.3 Information Collection as Microservices outlines the microservice-based architectural approach and design of the ICC is then detailed and discussed with the goal of establishing a set of requirements. Since the Smart Nudge system is not yet implemented, any attempt at designing a component of the system, must necessarily rely heavily on a few assumptions about how the system might operate. These assumptions will impact decisions about communication protocols, privacy, scalability requirements and over-all architecture. Assumptions are detailed in section 3.2.

4.1 Smart Nudge Architecture Revisited

This section briefly revisits the Smart Nudge system architecture to outline some important changes, before detailing the design of the ICC as a set of micro-services.

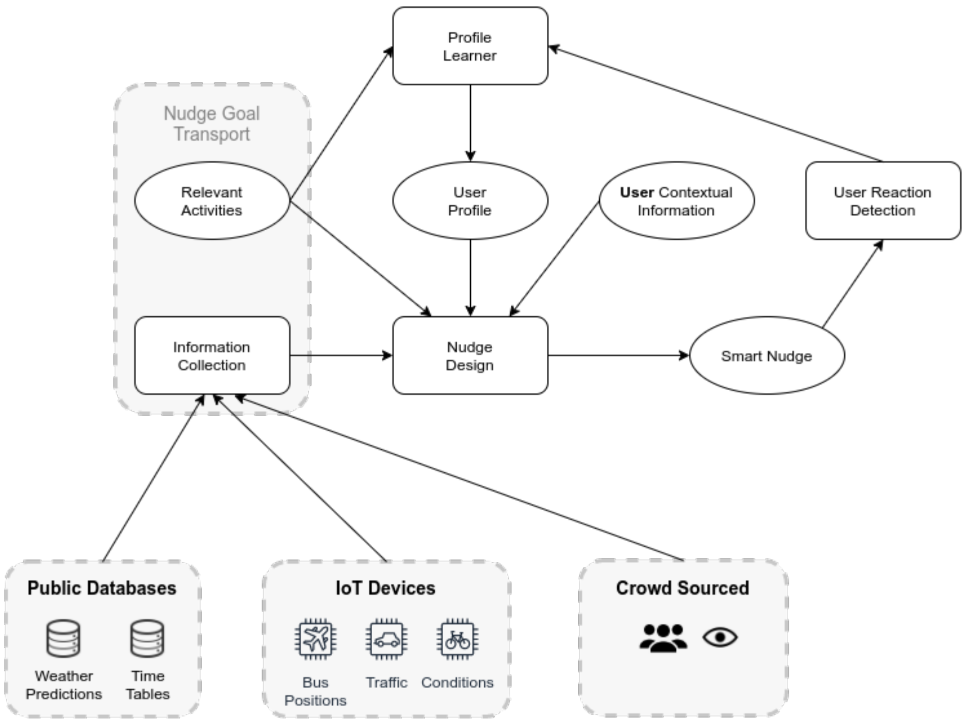

Figure 4.1: Architecture of the Smart Nudge system revisited to highlight changes made to the overall architecture.

The architecture seen in Figure 4.1, is a somewhat simplified and modified version of Figure 2.3 Smart Nudge system architecture by Karlsen & Andersen[4]. The *Nudge Goal* component is gone as the goal itself does not suddenly change if it has not yet been achieved or aborted. The components that need to know about it have the nudge goal information embedded into it upon system start. The ICC needs to work as a middleware that combines an IoT-device network, and public services and databases. These changes are reflected in Figure 4.1 Smart Nudge System architecture revisited.

Another more significant change is to separate the previous *Context* component into two parts to reflect the duality present in the contextual information the system will utilise. Contextual information is necessary to produce a personalised nudge. This contextual information is divided into two main categories, namely the meta-data and the user context.

Meta-data is information about data. What area does this forecast cover? How long is it valid? The other his potentially more intrusive from a users perspective and has to do with a user's current activities. Is the user en-route to work? Is the user sick? Geographical location and address? Medical history? These two main categories are now reflected in the architecture. A dedicated component will handle all queries to contextual data tied to a specific user. Moving this responsibility to a separate dedicated component also makes sense from a privacy perspective.

Meta-data is contextual information tied to pieces of data and is essentially spatial and temporal metadata. As mentioned above, this is an important aspect, especially in an IoT setting where data might be challenging to interpret and understand without contextual information. This will be (if possible to obtain), tied to each piece of data supplied by the ICC.

The main reason for dividing the context information into two different parts, is to ensure that the ICC remains a pure information repository, separated from user-specific data. The ICC should not need to know anything about a user to be able to supply useful information to the system.

4.2 Assumptions and Requirements

The primary assumption about the Smart Nudge system is that the production of a nudge is not caused by external events, nor queried by the user. It is important to establish the driving factors behind the creation of a nudge and the demands it places on the system. The main concept behind the Smart Nudge system is to move a user towards a goal they would otherwise not have reached. Giving control over when nudges appear, seems counter to this unwillingness or inability to move towards the stated goal.

Another critical assumption is that the production of a nudge is not caused by external events or factors such as a cancelled train, a closed road or a traffic jam. Ideally, the Smart Nudge system should be able to react to such external events and generate a nudge that might say something like; "*Hey, your bus needs to make a big detour today because of a traffic jam. Walking is actually faster, and the weather is nice.*" This would require the system to monitor information sources actively and continuously sift through events to determine which users might be impacted by them. This falls outside the scope of this thesis.

The nature of the system also alleviates some requirements that are often considered very important in an IoT setting, such as autonomous discoverability of new information sources[37, 66]. This type of discovery is not to be confused with the *Service Discovery* referred to when discussing microservice architectures. Service discovery in that regard is about routing a request to a microservice, to a particular instance of the service. This routing is typically done by a load-balancer[43]. Razzaque et al.[37], underscores the importance of the autonomous discovery of devices in an IoT-environment because a global, deterministic knowledge of the availability of a resource is simply infeasible in such a dynamic network of heterogeneous devices. This is less true in the Smart Nudge setting because the type of information needed to produce a particular nudge is known in advance. This makes it possible to have a centralised service with knowledge of all available resources and their availability.

4.2.1 Requirements

This section outlines the various requirements as highlighted by the revised architecture and assumptions described above. Most of these are functional requirements related to how information is gathered, how it is queried and processed by the ICC. The primary purpose of the ICC is to provide a layer of abstraction to hide the heterogeneity of the underlying network of information sources. This network is mainly built up of public services, but also any other information source that might conceivably be added at a later stage. Razzaque et al. .[37] and DaCruz et al. .[66] provides valuable insights into the IoT specific requirements that are also important to the smart nudge system. It is worth noting that the ICC is not meant to be a dedicated IOT platform. It is not however inconceivable that a collection of stand-alone IOT-devices could be wrapped by a microservice to form a custom source of information.

Functional Requirements

Insulate from heterogeneity The ICC should insulate the Smart Nudge system from the heterogeneous data sources that it uses. Queries for information should be possible to do in a uniform and well-defined way. Responses should have a similarly uniform nature. As a mostly industry-driven technology sector, data integration in the IOT environment is complicated. This complexity stems from significant variation in protocols and data-formats, which again stems from a large number of producers[67]. The ICC should be able to integrate the information by converting to a data model and format to make the information useable for the rest of the system.

The ICC should present a sensible way to query the available data. Since the prototype is a set of microservices, this is likely to be URI-based rules for querying different types of data. Similar to how a REpresentational State Transfer (REST)-API exposes various endpoints for different services. The component should also support querying the same data in different ways. For example, weather data might be queried for a larger area, or as an aggregate of predicted weather for the coming week, or specific conditions in a particular spot.

Context-awareness Contextual awareness is a crucial factor in both IoT and the nudge system[37, 66, 4]. Sensed data, will need to be accompanied by contextual information for it to be useful at all. This type of meta-data is crucial in the nudge design process, particularly concerning temporal and spatial information. A lot of information is simply useless, without temporal meta-data. "*The bus leaves at 17:00*" makes little sense if we don't know from which stop.

Data management The collected data is the central resource the ICC should provide. This data will come from sensed data from various IOT devices, but also publicly available services. The component needs to provide for the acquisition of the data, preprocessing of data such as filtering and aggregation and data storage[37, 66]. Further demands might be put on this particular aspect of the ICC as the nudge system relies heavily on the ability to access both current, historical, planned and predicted data[3].

Data integration The ICC should ensure that the data collected and made available follows a schema which is suitable and useful for the Nudge Design component. Where applicable, data should have contextual data tied to it. A request for weather data should have the location and temporal data attached to it to ensure the system knows how to use this correctly for a particular user. Another aspect of the data integration requirement is data size. The ICC should attempt to insulate the rest of the system from an information source that is a bad fit. An example of this might be an API that only exposes one endpoint and any request results in massive amounts of data. Of which only some might be useful to the Nudge Design component.

Non-Functional Requirements

Lack of infrastructure and availability The data resource provided by devices should be available or appear available at all times[37, 66]. IoT devices form a large network of often wirelessly connected things with limited resources. It is important that the ICC insulate the system from an information source environment largely based on devices that are prone to disconnect and become non-responsive at arbitrary times. From a nudge-design perspective this might mean that if the exact location of the next bus is unavailable, the system could fallback on the time-table as a less accurate but available option.

4.2.2 Disregarded Requirements

Scalability

Several requirements have largely been disregarded due to the limited scope of this thesis. That does not imply that these requirements are considered any less important for the overall system. This section discusses some of these requirements and possible future approaches to realising them.

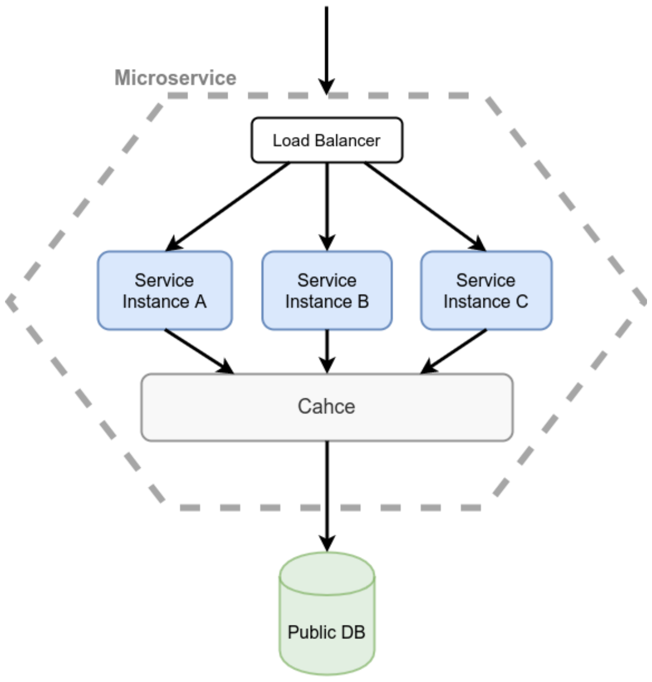

Figure 4.2: Scaling microservices.

The goal of the thesis is to examine a particular architectural choice to solve the problem of integrating highly differing sources of information, both in terms of data formats, communication protocols, hardware and availability. These are the main problems to solve. Microservices enables more dynamic scaling of a system. Rather than having to upscale a complete monolithic system and all of the underlying infrastructure and storage solutions, we now only need to scale the specific services that are experiencing a spike in traffic. Figure 4.2 Scaling microservices shows a simple view of how a microservice might do scaling. Because microservices in this project are used as wrappers for actual information sources, it means that a microservice does not entirely own its data. It might also mean that requirements might come from the information source. For example, a public database might expose a REST API but might limit the number of requests allowed to avoid overloading its servers. Limitations like request-caps must be handled responsibly by each microservice. Figure 4.2 Scaling microservices shows how a load balancer might deploy three instances of the service to manage a request spike. Several architectural patterns might enable caching, but they boil down to three main approaches: Client-side, proxies, and server-side caching. Client-side caching would only help if the Nudge Design component were a centralised component servicing many users. The most straight forward solution would likely be some form of server-side caching[39]. This approach might use embedded caches like *redis*[1] or *memcached*[2]. Caching responses is most advantageous in situations where the data is queried often. The more often we can avoid having to ask a source for a response, the better. Long-lived data also benefits from caching. Data which is unlikely to change will ensure that the cached version will be valid for a longer period of time, thus avoiding more calls to the source. Spesific caching strategies and their impact on short-lived live data is discussed further in subsubsection 6.2.3 Caching.

Automated Discovery of Data Sources

Automated integration of data sources is a complicated problem. One of these problems is schema-mapping between overlapping data sources. Another is how to deal with uncertainty in the data itself[68]. Is the data correct and up to date? Is the source trustworthy? Requiring the ICC to handle automated source integration is therefore not considered in this thesis because the quality of the information from a device or service needs to be audited before adding it to the Smart Nudge system.

Privacy and Security

Privacy concerns are largely ignored because the ICC only collects data from publicly available devices. This information is, however, combined with personal data from the user profile to make a very personalized nudge. That is not to say that the collected data cannot be miss-used to infer personal details about a user such as a home address, work address and working hours. The data source providers themselves might also have use-case limitations linked to privacy concerns which might require that a service handles data in a particular way. This handling might entail encrypted storage of data, only storing aggregates of data, limitations on the storage period of data and where it is stored. The General Data Protection Regulation (GDPR)[69] brings privacy

1. `redis.io`
2. `memcached.org`

concerns to the forefront of software architecture research and distributed systems design[70].

4.3 Information Collection as Microservices

The key benefits of microservices is the decoupling of a system architecture which in turn makes maintainability easier, scaling easier and it provides technological heterogeneity often sought after in distributed software systems development. When designed correctly, it also offers highly resilient systems because one service failing, is not enough to take down the entire system. Updating a service does not require recompiling the whole system. This makes adding features and general development faster[39]. With proper load balancing, service discovery and data storage solutions, the approach also offers a high degree of availability. Multiple instances of a service can be run at the same time to share load and provide backups should one instance go down.

The idea is to use some of the architectural ideas and principles behind the microservices system architecture model and apply it to the ICC in the Smart Nudge setting. The technological freedom offered by the model might help to mask the extreme heterogeneity present in the various information sources the Smart Nudge system relies on to produce nudges. Rather than using a microservice-based approach to the whole Smart Nudge system, it is only used in the part of the system responsible for gathering information. Microservices aims to build systems by modelling each service around a specific business logic or domain. This is then a separate service that operates independently. The fundamental idea is to employ the *single responsibility principle*[41] on a larger scale.

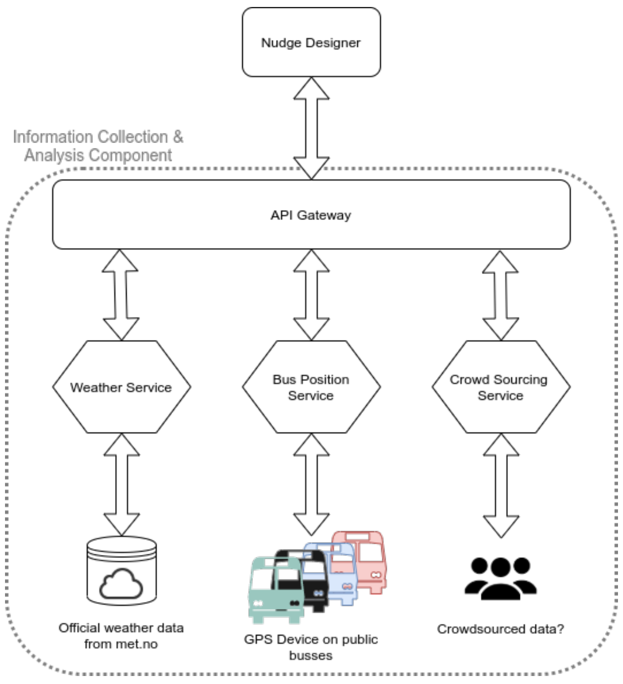
Nudge Designer
Information Collection & Analysis Component
API Gateway
Weather Service
Bus Position Service
Crowd Sourcing Service
Official weather data from met.no
GPS Device on public busses
Crowdsourced data?

Figure 4.3: Basic microservice architecture.

The API gateway provides a single point of entry for the Nudge Design component, which is the client or consumer of the services. The API gateway provides a way to query multiple of the sources of information via a single query. The API gateway is responsible for breaking down the query and forwarding specific requests to the appropriate microservices. The API gateway then collects all responses and composes a single response to the original query.

Each service is designed to do one particular thing very well. The hope is that the system will be decoupled in its design, which in turn means that it is more resilient to failure. If one service fails, it will not cascade through systemic inter-dependencies and cause the whole system to break down completely. The system might not operate at full capacity, but it will not fail. Moreover, each service can be scaled independently.

The architecture of the ICC divides each raw information source into a separate microservice. The main idea is to apply the microservice architecture on the level of information collection to see if this might solve the problem of heterogeneity in the information sources. The three most important features that a microservice-based architectural approach typically offers are deployability, maintainability and durability/resilience.

The microservice architecture aims to model each service as a separate module to build a system that is robust, independently scalable and offers several development-related advantages over a classical monolithic design [42, 39, 71, 43]. The microservice-based architecture is not a 'solve all' solution to the distributed systems architecture and comes with its own set of pitfalls. One of these is matching each service to a specific business domain or core logic of a business or system. More simply put, the problem is how to correctly identify a separate domain around which a microservice can be built that are not revealed to contain hidden dependencies at a later stage.

There are several approaches to modelling microservices. One is to divide along business logic such *order, checkout, payment* and *delivery*. Another module boundary might be based on technical concerns. If only part of the order-microservice mentioned above gets overloaded during certain times, while the rest of it does not, it might be clever to separate that particular part of a service into a separate microservice to better support scaling[42]. Salvadori et al. [48] offers a third view on how to model microservices, a data-driven approach. Data-driven microservices refers to an approach where each service is exclusively responsible for managing the lifecycle of an entity[48].

Each service is responsible for handling the direct communication with the information source. The service then does necessary aggregation of the data (if needed), caching, as well as unifying to a common format and adding meta-data. Finally, the microservice should present an endpoint with which the API gateway other services can issue queriesFigure 4.4 Detailed example of the architecture of a single service.. The API gateway ensures that the component only presents one endpoint to the client. The client in this scenario is the Nudge Design component(Figure 4.1 Smart Nudge System architecture revisited) which communicates with the Information Collection component. The Information Collection component parses a query and furthers the request to the API-Gateway which forwards smaller requests to the correct services, receives all the responses, ensure correctness and sends the results back as a single response.

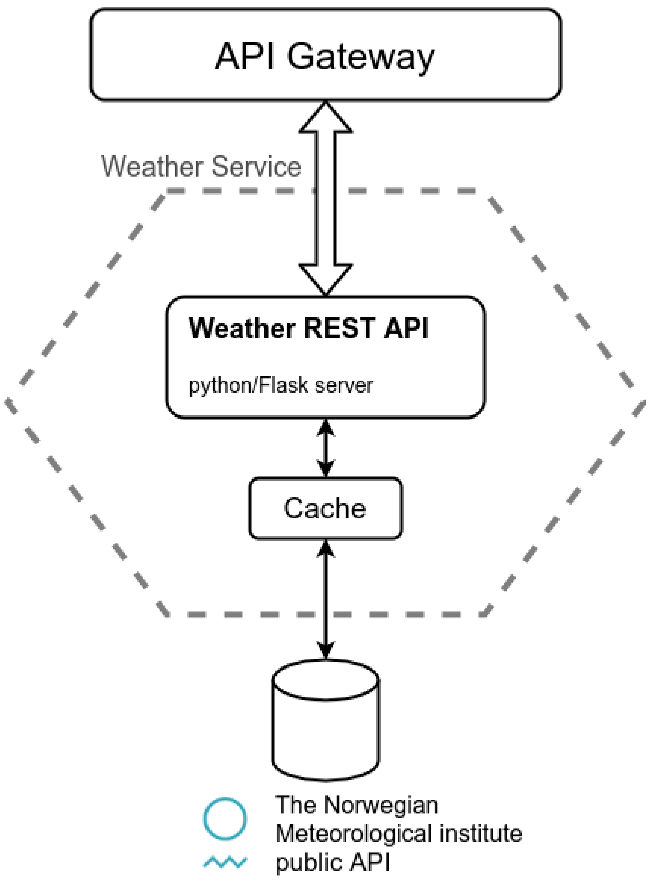

Figure 4.4: Detailed example of the architecture of a single service.

Before discussing the design and architecture further it is important to examine some of the main requirements of the Information Collection component.

4.3.1 API-Design and Data Flow

One of the shortcomings of a microservice based approach, is that services communicate over network, typically using http[42, 39]. This introduces much communication latency than a monolithic approach with direct access to each information source and no need to orchestrate with an API gateway(or other services for that matter). This also increases the chance that communication simply fails, implementing systems that, at least partially, mitigates this problem is complicated and requires even more orchestration between all available services[42]. The typical solution to designing for faulty communication is to have each service send heartbeats to a central component(often the load balancer) which then removes nodes from a service if it does not report its status for a given time-period[39]. Yet another drawback is the network traffic. More services means more data being sent over a limited bandwidth network. This means that reducing the amount of data included in each response is important.

Microservices typically communicate using REST APIs. REST is a commonly used architectural style which provides interoperability between web services that was first introduced in 2000[72]. REST APIs is an architectural design centred on exposing access to resources rather than remote processes[73]. While REST interfaces is a good fit for microservices they can lead to increased latency for example when dealing with lots of different data from different end-points[74]. GraphQL[3] is designed to mitigate this problem. GraphQL was designed by Facebook and released in 2015[75] When integrating data from heterogeneous sources, it is important that communication solutions allow the interoperability between the individual data providers to remain strong. This ensures that the services can remain independent[76].

Since the size, shape and form of the data to be connected to the Information Collection component in the future is unknown, it is important to incorporate flexibility regarding all of these aspects. This will be done by mimicking GraphQL to allow for only requesting specific parts of a large data-structure and have the service be able to process the query, thus minimising the size of the request and network latency.

3. graphql.org

5 Implementation

This chapter outlines the implementation details of the prototype ICC. The prototype is used to explore how well an architecture based on microservices fits the needs and requirements of the Smart Nudge System. Section *5.1* gives a brief overview of the prototype along with the data-sources used to test the architecture. Section *5.2* outlines the data integration approaches used in the prototype.

Preliminary approaches to the Smart Nudge system[3] outlines two main approaches to categorising information sources. One approach divides data based on its relation to time. This approach would divide information into *historical data, current data, plans* and *predictions*. The approach explored here is to divide the data based on its source, as suggested by Andersen et. al.[3], but more fine grained than to define one source to be *IoT sensors, crowd-sensing, aggregated data, crowd-sourcing* and *static data*.

5.1 Prototype Overview

The prototype is written in *python3*[1]. It features/integrates two data sources, namely weather information from *The Meteorological Institute of Norway*[2] and

1. `python.org`
2. `met.no`

live bus data from *Chicago Transit Authority*[3], see Section 5.4. Each data source is wrapped by a separate service and presents its available data to the Information Collection component via an API gateway. Communication with the service is via a query-based REST API over HTTP. The implementation is essentially two Flask servers that each wrap a different data source and present the information via a query based API.

5.1.1 Microservice Architecture

Each microservice consists of several parts, as illustrated in Figure 5.1. The controller is responsible for handling communication with the API Gateway and presents the various endpoints. The controller is implemented as a server using the Flask[4] web framework. The Repository presents an interface by which the controller can access information. It is responsible for caching if this is needed, as well as communication with the data source itself. Note that in a deployed system, any persistent storage or caching would need to be shared between all instances/nodes of the same microservice though this is beyond the scope of this thesis.

The actual data is captured in the Source Data Model. This component is a representation of the source data, along with domain-specific logic. This model needs to mirror the actual data as closely as possible to avoid making assumptions about how the data might be consumed. It is also important to avoid assuming domain-specific knowledge about how to handle the data. For example, the weather service data model should not average the temperature over several hours, to save space or processing time for later as it would necessarily break the data and potentially render it useless. Modelling the source as closely as possible also results in a more robust system, better able to adapt to client-side changes in data requirements at a later stage.

3. `transitchicago.com`
4. `https://flask.palletsprojects.com/en/1.1.x/`

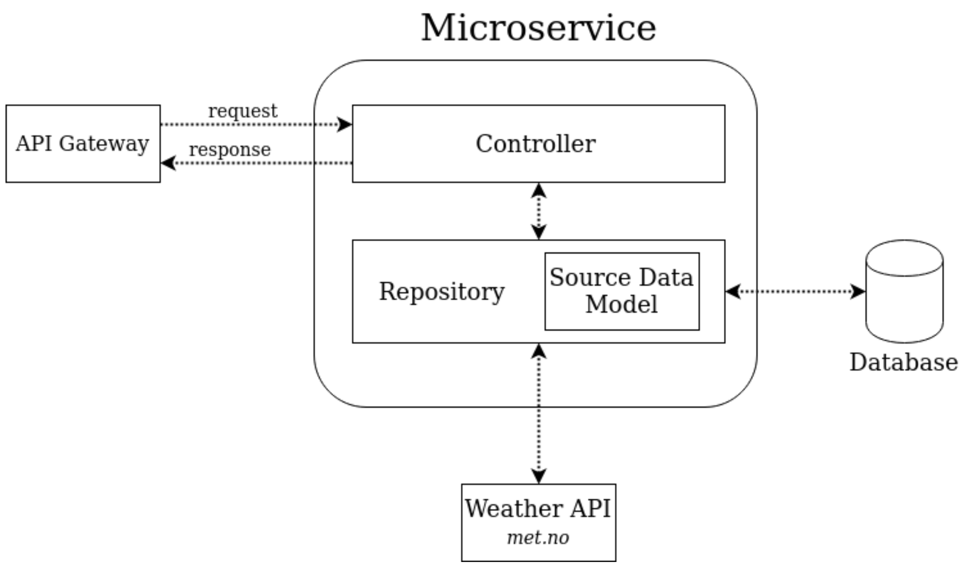

Figure 5.1: Simplified internal composition of a single microservice as used in the prototype.

5.2 Data Integration Approaches

Each service consists of a repository component which handles the actual communication with the raw data source. The store translates the relevant parts of a data source into the Source Data Model. As the ICC is essentially and information integration system, unifying data is one of its primary objectives.

The prototype uses JavaScript Object Notation (JSON)[5][77, 78] has seen a rise in popularity in the last decade[79]. JSON is presented as simplified form of Extensible Markup Language (XML). The main reason for choosing JSON as the main encoding-format is that the Nudge Design component is unlikely to ever need huge amounts of complex data. A nudge will typically not contain large amounts of information[59]. JSON also has the advantage of being a slightly more human-readable format than XML, and will require less wrangling to fit into the production of a nudge[80].

The ICC is somewhat similar to a data warehouse, at least in its main purpose. This is to make information from a wide variety of sources available via a global schema. This prototype differs in that its retrieval of data is client-driven, not source-driven as most data warehouses are[5].

5. json.org

Each microservice presents it’s available data via a global schema managed by the API gateway. A request is done via a HyperText Transfer Protocol (HTTP) POST request (Flask inhibits request-body for GET-requests).

Listing 1 shows an example of a request-body is accepted by the API gateway. The body contains a request with any number of requests. The fields in the ’request’-object needs to map to available data-points offered by the microservice, they are null-fields that are to be populated by the corresponding service. The gateway splits and sends the corresponding objects to the correct microservices, awaits the responses and assembles the results. The microservice, will populate these fields with the requested information so long as they fall within the requested time frame and location. Each service is responsible for processing the data and only send the requested data back.

```
{
    "weather": {
        "location": {
            "lat": 69.14,
            "lon": 18.97
        },
        "timeframe": {
            "from": "2020-02-28T12:00:00Z",
            "to": "2020-02-28T13:00:00Z"
        },
        "request": {
            "temperature": null,
            "wind": null
        }
    },
    "bus_eta": {
        "request": {
            "route": 155,
            "bus-stop": 1746,
            "eta": null,
        }
    }
}
```

Listing 1: Example query to the API gateway

Each microservice responds to parts of the whole query and returns a partial response. The collection of partial responses is assembled by the API gateway and returned to the Nudge Design component. See Listing 2 for an example.

The response mirrors the request. The *'location'* and *'timeframe'* objects reflect the geographical location and temporal validity of the data returned.

Handling Partial Responses

The response format needs to account for the possibility of partial responses in various ways. If the bus-service is disconnected, the API gateway will give up and insert a missing data placeholder in the corresponding response fields. Each service will do this for partial responses within its local data. An example might be that the 'temperature' was present, but not 'wind'.

```
{
    "status": {
        "code": 207,
        "message": "Partial response",
        "requested-data": {
            "weather": 200,
            "bus_eta": 404
        }
    },
    "weather": {
        "location": {
            "lat": 69.14,
            "lon": 18.97
        },
        "timeframe": {
            "from": "2020-02-28T12:00:00Z",
            "to": "2020-02-28T12:00:00Z"
        },
        "data": [
            {
                "from": "2020-02-28T12:00:00Z",
                "to": "2020-02-28T12:00:00Z",
                "temperature": 5.5,
                "wind": 8.2
            }
        ]
    },
    "bus_eta": null
}
```

Listing 2: Response example

One of the problems with retrieving data from heterogeneous sources in a distributed system is the potential for partial responses. Parts of query might not be fulfilled, for a number of reasons:

Malformed query A query might not adhere to the schema. Ideally, this would be rejected by the API gateway to save unnecessary network traffic. Note that the ICC should still respond.

Non-Responsive Microservice The 'Weather' microservice might not be responding(which could happen for a variety of reasons). In this case, the response should include a status object, detailing the status of the whole response, what is missing and why.

Missing fields A service might get a request for 'temperature' and 'wind', but only manage to resolve one of these. The response should include a status object, detailing the status of the whole response, what is missing, and why.

5.3 API Gateway

The API gateway, though not implemented, is a mediator between the Nudge Design component and the various microservices that make up the ICC. The API gateway serves several purposes. The API gateway receives a request from the Nudge Design component in the form of a query. The API gateway knows how to parse the query and send the individual parts to the correct services. It then awaits all the responses and assembles the final response. The gateway is also responsible for keeping track of all available services. Keeping track of available services is typically done by having each service sending periodic heartbeats to the gateway[39]. The gateway serves as a single entry point for querying information. If we assume that the Nudge Design component would be running on users private devices, the ICC would likely be a central service, since it does not rely on users private information, but serves as an information repository.

The API gateway would also keep track of any changes in the schema of each individual service. Each service could be required to report a new schema any changes to it were made, either through changes in the composition of the data source it mirrors, or any changes made by developing a microservice such as adding new features that might not have been supported by the data source provider. The Nudge Design component should be able to ask for an updated version of the global schema from the API gateway.

5.4 Data Sources

A lot of the information needed in a nudge is inextricably linked to a geographical position. Access to weather data is all well and good, but it is useless unless it is tied to a geographic location. Knowing when a bus is at a particular stop, is good, but likely to be less useful if it was yesterday.

Two data-sources were chosen for this thesis. The forecast API offered by *The Norwegian Meteorological Institute*[6] and the live bus data API offered by the *Chicago Transit Authority*[7].

The Norwegian Meteorological Institute of Norway

The Norwegian Meteorological Institute of Norway offers a free API providing access to a broad set of data related to weather such as forecasts, wind, waves, tides and air-quality. The forecast-API offer essentially one endpoint[8] and delivers an XML formatted response typically exceeding 14000 bytes. The XML-object used somewhat poorly constructed(See Section 6.2 Evaluation) and is, therefore, an excellent test for the data acquisition and integration aspects of the ICC.

Chicago Transit Authority

The bus-tracking features of the Chicago Transit Authority are supplied by *Clever Device Inc.*[9]. They offer live tracking of busses, estimated time of arrival for busses and several other useful endpoints. The API is available with a developer key which was kindly granted on request to use the API as a case study for this thesis. The bus position information is an excellent case study for the types of challenges the ICC must be able to handle. The data is short-lived (minutes) and includes both the temporal and geographical aspect that is important to make the nudge relevant to the user. It also 'offers' the types of challenges that come with a deployed, live system; like undocumented responses, differing data types for the same field between different responses. These aspects are difficult and time consuming to mock.

6. `met.no`
7. `https://www.transitchicago.com/`
8. The Norwegian Meteorological Institute did a complete API overhaul in May 2020, close to the deadline of this thesis. This likely would have changed the microservice somewhat, though the fundamental architectural model would have been the same. Version 1.9 of the API is still used here for illustrative purposes.
9. `https://www.cleverdevices.com/products/bustime/`

Local busses in Tromsø also offers live GPS tracking and estimated arrivals times for their busses, but does not provide a publicly available API with which to explore different solutions or build new applications. Admittedly, using local bus times would be nice, it is not essential in terms of evaluating the architectural approach to building the IC-component.

/6 Evaluation

This chapter presents the evaluation of the proposed architecture of the ICC. Section 6.1 Architectural Remarks highlights some of the distinguishing features of the system as a whole, the proposed architecture and how it fits into the Smart Nudge setting. Section 6.2 Evaluation makes up the bulk of the chapter. The evaluation is a two-part evaluation, but is mainly based on the established requirements (see Section 4.2 Assumptions and Requirements). Lastly, I offer some important reflections on the project before summarising by reiterating the research questions and the findings.

6.1 Architectural Remarks

The architectural approach taken in designing the ICC is typically applied on larger-scale enterprise distributed systems[39]. The Smart Nudge System is conceivably such a system, but the microservice-based architecture is in this case only applied within the ICC. By only using this architecture in a single part of the system results in some interesting changes in the system that are relevant to the evaluation of the ICC.

Firstly, ICC only has one client. This client is the Nudge Design component (See Figure 4.1 Smart Nudge System architecture revisited). The ICC is explicitly designed to cater to the needs of this particular client, which means that specific rules regarding principles of software design could be bent, or broken. One such rule is that a service should not make assumptions about what a client might need. Or rather, a service should not provide information that has not been specifically requested. In the single-client environment if the ICC, this is not entirely true. We do know what the nudge designer needs, and anticipating those needs does not necessarily break the usability for other clients, because there aren't any other clients.

Secondly, smart nudges are relatively small. A smart nudge is composed of a short (3-4) pieces of information to supply the content as well as the wanted nudge effect[59]. The fact that they are small means that a nudge can be created with relatively few requests. The construction of a smart nudge will never rely on a huge number of requests. The underlying assumption here is that each nudge is produced through ONE single main request, which is not necessarily true either. The Nudge Design component might know what it needs to pre-make the next x number of nudges and proceed to query all the required information in several requests or one huge request. Because a nudge is likely to be most effective just as the user needs the information, it is unlikely that nudges can be pre-produced to any significant degree. The system is very much dependant on the freshness of its data.

Advantages and disadvantages of the architectural approach to building the ICC can be contrasted to an alternative design which might typically follow a SOA approach or a single application monolithic approach. A SOA is similar to microservices but is more coarse-grained in the way services are defined. Wang et al. provide an excellent example of this architectural model in a similar setting[81], Vettor et al. provides another approach[82]. An approach to integrating data sources in this way might see one Extracting service, capable of querying all the different data sources, the next step might then be a Transformation service, next Storage and finally, delivery to the Nudge Design component (see Figure 6.1 SOA architecture example).

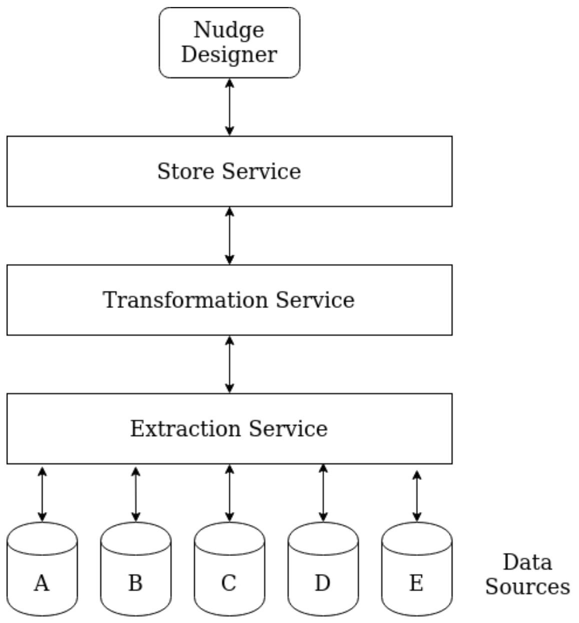

Figure 6.1: An example of a SOA based alternative to the microservice oriented architecture used for this thesis.

Another alternative is a single application, monolithic approach. Many of the disadvantages and advantages with the various approaches are typically tied to aspects of development and deployment of a system[83]. The benefits of a microservice-based architecture are primarily related to developer team organisation, development, deployment and scaling. The following evaluation of the requirements will attempt to establish if the architecture, when employed in the particular setting of the Smart Nudge system, still offers distinct advantages.

6.2 Evaluation

This section presents a critical look at the design of the ICC in relation to the aforementioned requirements and the qualitative aspects and features of the prototype. Before evaluating the architecture and functionality of the ICC component prototyped in this thesis, we will briefly revisit the requirements for this component. These requirements will form the bulk of the evaluation. Each of the requirements will be compared briefly to alternative architectural approaches, namely a SOA approach and a monolithic approach. Table Table 6.1 Overview of requirement-based analysis with regards to different architectural approaches. provides reference to these comparisons. The evalu-

	Monolith	Service Oriented Architecture	Microservices
Insulate from heterogeneiety	Good	Good	Good
Context awareness	Good	Good	Good
Data management	Bad	Medium	Good
Data integration	Bad	Good	Good
Availability	Bad	Good	Good

Table 6.1: Overview of requirement-based analysis with regards to different architectural approaches.

ation will then focus on a more formal analysis of the proposed architecture. This is analysis will be using decomposition techniques derived primarily from Parnas[57] but also applying criteria emphasised in a study of the migration from a monolithic architecture to a microservice-based architecture in various businesses[58].

Insulate from heterogeneity The ICC should function as an abstraction or middleware to hide the complexity and heterogeneity of the underlying data sources. This complexity can manifest itself in various ways, but typical problems are tied to differing data-formats and communication protocols.

Context awareness The information provided by the ICC should be augmented with meta-data that helps the Smart Nudge system match the information to the context of the user. Matching the context of the user is essential because some information is useless without certain types of meta-data. Nudges also rely heavily on being timely and correct.

Data management Data management refers to the need to handle data acquisition, transformation, aggregation and storage (caching). These aspects are related to scalability but should be considered none the less because data management is an aspect which impacts more than just scaling solutions.

Data integration The ICC must ensure a uniform way to access different data sources. The component should present a single endpoint through which all available data can be accessed. When delivered, it should conform to a specific format. The integration also encompasses the acquisition of relevant metadata for each data point queried. This metadata is necessary for the nudge system to produce sensible nudges.

Availability The ICC should account for the lack of infrastructure, processing power and connectivity (a common problem with IoT-devices) and present either a graceful failure or fallback on alternatives. A graceful failure might mean an empty or lacking response, but a response none the less. This requirement is not strictly a non-functional requirement. Presenting an illusion of availability(through fallbacks or graceful failure) could be said to a functional aspects of the component requirements.

6.2.1 Insulate from heterogeneity

As with any middleware, insulating the system from heterogeneity and complexity is one if not *the* core goal of the ICC. One of the main problems with integrating different data sources is the different formats. The Meteorological Institute's forecast API[1] responds to a request with a complicated XML-object where a lot of the data is hidden as attributes of XML-elements (see Listing 3 Example of part of XML from met.no API).

```
<time datatype="forecast"
      from="2020-05-26T12:00:00Z"
      to="2020-05-26T12:00:00Z">
    <location altitude="496" latitude="60.1" longitude="9.58">
        <temperature id="TTT" unit="celsius" value="18.4"/>
        ...
```

Listing 3: Example of part of XML from met.no API

This is generally not recommended practice[2]. To further complicate things, many of the attributes are defined as "optional" in the schema definition[3], which makes constructing a data-model that retains as much of the original information as possible difficult to write. Parsing the XML-object into a usable data-model becomes complicated. In short, the response is a good example of a poorly designed XML-object and illustrates some of the complexities that the ICC need to handle.

The Chicago Transit Authority API responds with reasonably well-formed and defined JSON-objects which more easily translates to the hash-map/key-value data structure standard in most modern languages.

One advantage of having a single service manage each data source is the technological freedom inherent in the approach. If a particular data source

1. The old version of the API, that is.
2. `https://www.w3schools.com/xml/xml_attributes.asp`
3. `https://api.met.no/weatherapi/locationforecast/1.9/schema`

responds with an extremely strange or archaically formatted data, the service responsible for it can be built using languages that fit this better. This fine-grained adaptation is a lot more difficult if a single service needs to handle all types of data sources attached to the system. So long as a single microservice conforms to the agreed-upon interface and schema, it can resolve the communication, extraction and cache-solutions in whichever way suits best (caching is described in subsubsection 6.2.3 Caching). Each service is better able to both handle disadvantages or difficulties inherent in the data-source and exploit advantages inherent in the data source. Each microservice can do this independently.

A monolithic approach suffers the same disadvantages as mentioned above. Moreover, both the SOA and monolithic approaches are less failure tolerant. If the Extraction Service (see Figure 6.1 SOA architecture example) failed, the whole application would fail. Likewise, any changes, made to this service, would require a complete recompile of the whole service. If a microservice failed, it would mean a single data source is unavailable, but the system would still function.

Another complexity problem is tied to communication protocols. In highly specialised distributed systems that integrate and use vastly different applications, databases and sources of information, managing large amount of different protocols can become a problem. Communication protocols are likely to be less of a problem for the ICC for one reason. Since nudges are small, specially constructed pieces of information that need to be consumed quickly, they will typically be short texts or small images. Most of the web-based services offering this type of data use the HTTP as the communication protocol[84]. But one could imagine an image server using the File Transfer Protocol (FTP) as its communication protocol. In a microservice-based architecture, this would not be a problem.

An 'image' microservice, could communicate with a data source via FTP and pass requested images as part of a JSON-object, and still conform to the JSON based schema of the ICC. The image data would need to be base64 encoded. Including images as a data source by base64 encoding them directly in a JSON response would be an exciting challenge. One challenge would likely be that base64 encoding images is slow and takes up a lot of space[85].

However, the possibility to insulate the Nudge Design component from different communication protocols are not unique to the microservice architecture. Both a monolithic approach and an SOA-based approach would handle this equally well.

6.2.2 Contextual awareness

Contextual awareness is a vital aspect of a nudge. Nudges must be relevant for the user, both in terms of geographical location and temporal data. A nudge has to be constructed of information that is relevant at the time. As mentioned, there is no use knowing that a bus will be somewhere in 5 minutes, or that Tromsø will have sun, sometime, if we do not know when. Both the temporal and geographical meta-data fields are incorporated into the responses of the weather and the bus eta microservices. The idea behind having a time-interval is to inform the Nudge Design component of the interval within which the given data is valid. This is to allow the Nudge Designer to store responses as long as they are valid.

There is of course information which is correct and valid, regardless of any time frame or temporal metadata. The bus arriving at a specific time is information which is only valid until the bus has actually arrived. That is not to say that information becomes useless after it's 'validity' expires. Knowing when a bus left the stop, can indeed be useful, but less so in the smart nudge system because the whole premise is to influence users by providing them with helpful information *before* decisions are made. This does not mean that historical data might not feature as part of a nudge. Indeed, historical data offers an interesting insight into a potential problem with providing a 'validity time-frame' for every piece of information integrated.

An example might be a data source that offers snippets of informative texts about historical locations, incorporated into the ICC. These might have a geographical meta-data tag that is relevant, but the temporal validity of historical data does not expire (so long as history isn't rewritten too often, but that is outside the scope of this thesis). It might also be challenging to specify the geographical validity for certain pieces of information. "The Declaration of Independence was signed in July 1776."[4], is presumably valid forever, but how does one define the geographical validity of this sentence? Is it only valid in the US? Or should it signify where it was signed? This is a problem that needs further study but is unlikely to cause problems because the lack of these lack of the time-frame or geographical meta-data tags, does not render the information useless. The schema does not account for historical data not needing a time frame defining its validity, but one could assume that an empty time frame, means it is illogical to include. The status code would tell if this was done intentionally.

There are some problems inherent in supplying a time frame for a piece of information. Firstly, the validity might change and render the time frame false.

4. https://www.archives.gov/founding-docs/declaration-transcript

A bus that we know is 5 minutes away might suddenly stop or break down. One of the main assumptions made about the system is that it is user-driven. That is to say, that nudges are produced by the Nudge Designer knowing when to create one. External events do not drive the production of nudges. In other words, the ICC does not listen to events to let the system know that a previously requested piece of information has become invalid, or a better prediction is available. Because of this, there is no way for the Nudge Designer to know, other than making a new request.

Compared to alternative architectural approaches of SOA and a monolithic approach, the microservice-based architecture offers no particular advantages with regards to supplying meta-data that provides the context that is necessary to produce tailor-made nudges.

One of the key points of interest the project reveals is concerned with the temporal meta-data of a data source. Two data points are of interest, the temporal validity or lifetime of the data and its potential for becoming invalid. The first denotes how long a data point is valid, which is of crucial importance to generating useful nudges. The other says something about how likely it is that the validity of the data changes. Two examples serve to illustrate this further. A bus arriving in 5 minutes, is valid for 5 minutes. It is also highly likely to change and should, therefore, perhaps be re-queried more frequently. A historical fact is valid forever. It is very unlikely to change, and we might not need to re-confirm this with another query at all. Incorporating information about how often a response should be refreshed into the ICC is of vital importance to the Smart Nudge system as it would increase the systems ability to produce timely, correct and accurate nudges. Creating correct and accurate nudges, would in turn likely increase the trustworthiness of the system and increase its capacity to influence peoples behaviour.

6.2.3 Data Management

Handling data acquisition, transformation, aggregation, and storage is an essential part of the ICC. Microservices run independently of each other and can likewise more readily adapt to particular challenges with regards to both data acquisition, aggregation, and storage. Part of the data acquisition stage includes removing data will not be used by the system. The data model for the *met.no* data source models the source as closely as possible, but restructures the list of data from a list of location-points to three lists of predictions with different time frames(one-hour, single moment, and six-hour forecasts). The microservice-based architectural approach does not do this any better, or worse than alternative approaches. But the independent nature of the microservices, allow them to specialise both with regards to formats and technology. Some languages might have better support for parsing a particularly odd data format. Another microservice might run hardware particularly well suited to handling the data it serves.

One drawback with the microservice-based approach in this regard is that sharing software resources is difficult to do. Commonly used utility software resources, such as storing to a simple file, needs to be implemented across all services that need this. One could write custom libraries and host these in a separate server, but this would create a common dependency that brakes with the point of running independent services (if the custom library server crashes, all microservices relying on it, does too).

One advantage over other architectural approaches is that expandability is ingrained in the system architecture. Adding any new resource to the ICC requires only knowledge of the schema-agreement each service makes with the API gateway. So long as the new microservice follows the agreed-upon interface.

Caching

Another major aspect of the data management requirement is the caching of data. One of the most critical aspects of the whole ICC is that it needs to comply with requirements coming from both the Nudge Designer AND requirements/restrictions from each of the different data source providers that it integrates. Each of these source providers will have different requirements and restrictions for anyone wanting to access their data. These requirements might limit how the data is used, how often the data can be queried or collected. This is the case with both the examples of the ICC prototype. The Meteorological Institute imposes a limit of 20 requests per second limit on the endpoint[5]. The Chicago Transit Authority imposes a limit of 10 000 requests per day, per developer key, with options to increase the maximum limit of requests[86]. Both sources advise the developers to cache and store responses to limit traffic to their API's. Both the Meteorological Institute and the Chicago Transit Authority have license agreements tied to the use of their data which defines what it might be used for, where it might be used and how. The legal ramifications of licensing agreements fall far outside the scope of this thesis. Still, they would need to be considered and indeed handled in any deployed version of the Smart Nudge system and the ICC. It might, for instance, impose limits on the physical location of the hardware running a particular microservice. Note that this does indeed speak in favour of a microservice-based architecture. By having each data source be wrapped by one microservice, that microservice

5. https://api.met.no/conditions_service.html

need only ever comply with one set of limitations and restrictions.

In a monolithic approach, the whole application needs to comply with every providers' set of requirements. A monolithic approach typically also uses a centralised database which becomes vulnerable to accidental changes made to it, by different parts of the system. This would, by design never happen in a microservice approach because each service would use its own data management solutions that are decoupled from the other services.

When it comes to resolving the caching requirements, a microservice-based architecture might be well suited to handle this problem. The prototype, though it does not handle scaling, provides an informative example of the issue. A request made to the weather service must always provide a geographical location for the requested data. The service, as is, maps this to the closest of one of 11 000 defined *'important places'* by yr.no[6]. A new request that mapped to a location where a recent response was available, the service would return this, rather than request a neew response from met.no.

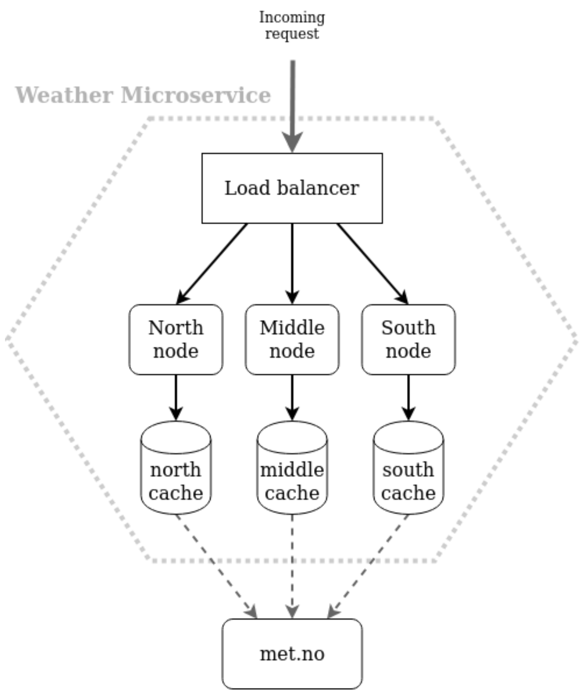

Figure 6.2: Weather service cache solution.

Figure 6.2 Geographical cache solution of microservice illustrates the caching scheme where the load balancer distributes requests based on their location.

6. `http://fil.nrk.no/yr/viktigestader/noreg.txt`

Each running instance or node of the microservice is responsible for caching a dedicated geographical area. The load balancer handles the routing of requests based on location rather than the usual round-robin scheduling algorithm. This might result in a slightly more flexible and scalable system where each cached response can be read individually. In a scenario where a cache is non-responsive, the node handling the response can still go to the original data source(met.no), while the other two (who still have working caches) would still avoid sending requests to met.no.In a monolithic single service approach, we might still use a geographical caching solution like the one illustrated, but here, the part of the system handling the responses becomes a single point of failure. Ultimately, the microservice approach, might in exceedingly rare cases save some requests that other approaches are unable to. But on the whole, a cache-miss is doomed to result in having to ask the original data source.

In the case of the bus tracker microservice, this caching solution would offer few, if any, benefits because the data served is for the most part continually updated live data that loses its value within minutes (such as estimated arrival times). In this case, it is difficult to see how wrapping the service and exposing its interface to the API gateway improves anything.

Two cases illustrate possible caching solutions that might still offer improvements and save some requests that would have otherwise have to have been sent to the BusTime API. One is that some of the data served by the BusTime API are not likely to change very often, as the list of stops belonging to a route, or the list of possible routes. This is data which the microservice can easily store in an embedded cache solution. An embedded cache solution is one where each instance or node of the microservice retains its in-memory cache. With rarely changing data (like the bus-stop list), this data can be deployed with the service and only updated occasionally. This is not to say that caching solutions could not work with short-lived data like live tracking. A single shared cache between each service node could store responses for eta-requests. Each response would only be valid for a few minutes, but during heavy load, it might still remove the need to call the BusTime API. Again, knowledge about typical load-spikes could be a good reason to use one cache solution over another.

In the case with the Chicago BusTime API, getting the estimated time of arrival for the next bus to a particular stop at a specific route requires both the *stopid* and *route* parameters. Both of these parameters need to be acquired by querying two different endpoints for a total of three API calls to get the estimated time of arrival. Since the microservice can handle two of these calls, the BusTime API is spared two calls. The calls would still need to be made, but they are offloaded from the original data source API.

The microservices could also augment the original data source by using rarely changing information stored in an embedded cache. An example of this could be that the bus-service could offer an endpoint which takes a geographical position as an argument, along with a route id, and returns the estimated time of arrival for on this route, to the stop closest to the given geographical position.

In short, various caching solutions do offer some ways to improve the scalability and efficiency of the data sources. But these caching solutions are not uniquely available to the microservice-based architectural approach. The approach does, however, lend itself better to using different caching solutions than a monolithic and SOA approach as each microservice is decoupled from the logic of any other, it is better able to adapt its storage and caching solutions to suit both the needs of the source AND the Nudge Design component.

6.2.4 Data Integration

The general goal of information integration is to combine various separately available sources of information and present them through a single unified view. The end-user should have no knowledge of the conglomerate nature of the system they interact with[87, 88]. The user should be;

> "...provided with a homogeneous logical view of data that is physically distributed over heterogeneous data sources."[87]

Data integration is a huge, complicated problem which likely does not have a single solution because new applications of data and ways of interacting with it, is continually changing[89]. Integration involves resolving local schemas, solving data structure and semantics problems. These problems stem from the simple fact that most information sources are not specifically designed to be easily integrated, but often considered 'finished' or ready to be used.

One of the main drawbacks with a microservice-based architecture compared to a monolithic approach, is an increase in network communication, both between microservices and the client application. Communication in a monolithic approach is local, and thus much faster and far less likely to fail. The inherent increase in network-based communication requires that a microservice-based system must be able to handle communication failures. The system must also handle completely non-responsive services (either due to a failed microservice or failed line of communication to that microservice)[42].

Mapping, in a data integration system, refers to the relation or road between the data sources and the data as made available through the global schema. The approach to mapping taken in this thesis is that of a *local-as-view*(LAV)[88, 90], meaning that each data source is specified and mapped independently of any other. The reason for this is that it favours expandable systems and would likely play well with the microservice-based architectural approach[88]. To evaluate this, we need to look closer at how query-based API's differ from traditional API's.

Microservices typically use HTTP and REST API's for communication[42, 39, 71]. It is a good fit because each service is built to fit the specific needs of its clients. As an example, the texititem-info microservice of an imagined online store can safely assume how the information will be queried, and the exposed endpoints can be designed to reflect these needs. Endpoints or resources are typically organised in a hierarchical manner around nouns representing a particular resource, such as items. The plural form is used to signify endpoints that return a collection or list of objects[91]. There are some drawbacks to this design, however, which bears on the discussion of query-based API design in the Smart Nudge system. In particular, four problems are often highlighted with traditional REST API's[92, 93], all of which are relevant to the discussion of a query-based API.

Query Complexity Traditional REST API's often require multiple and complex HTTP requests to fetch multiple resources. With a query-based API, any combination of data can be requested with a single HTTP-request. The query might still need to be completed via multiple complex calls to the source data provider, but within the ICC it is only one request.

Overfetching An API will often return more data than needed by the application, which is a waste of resources on the server-side of things, as well as wasting network bandwidth. This can also be solved by making the microservices prune off any unwanted data before returning it.

Underfetching A badly designed API often have endpoints that do not return a sufficient amount of information, which in turn leads to the need to send multiple queries to get all the information needed. Using a query-based API will shield the Nudge Design component from such badly designed API's.

API versioning An API is a contract between two systems (provider and user) and should thus be as stable as possible in terms of interface. But as business goals change, so too must API, often resulting in compatibility problems. A query-based API is more resilient to versioning changes because it can still conform to the agreed-upon schema. This is an important problem that each microservice solves. Even if the API exposed by the data provider changes, the microservice can still provide an unchanged API, provided that the actual data available from the source is still the same, just presented differently.

REST interfaces is a good fit for microservices, but they can lead to increased latency, for example, when dealing with lots of different data from different end-points[74]. On average, the met.no endpoint used in this thesis has a Round Trip Time (RTT) of 180 milliseconds, the BusTime API, slightly more at 316 milliseconds. This illustrates the importance of minimising the number of network calls that have to ask the source data providers. Does a query-based request entry point (the API Gateway) solve this problem?

Some have pointed to the increased computational complexity that comes with the need to parse queries and resolve them as a problem of GraphQL[74, 94]. In short, more computation is moved towards the microservice. If each request requires heavy processing, the gain from only needing one request, might get lost in the increased processing time for each query/request, though given the relatively high RTT's of a request to the data sources used, this is unlikely. Other studies focus on improvements gained by migrating REST interfaces to GraphQL[95]. In particular, they point to massive reductions in the amount of data sent per response.

This was part of the goal of using a similar approach in the ICC. This might, on the face of it, seem like a significant improvement if the 14000+ bytes from met.no can be reduced to a few 100. But wrapping a poorly designed API does not necessarily make it better. From the perspective of reducing network bandwidth on communication *within* the ICC, it certainly helps. But unless the weather-object for a particular place, was cached, no amount of GraphQL or query-based requests will save the service from having to ask for that data from the source. It might even make it worse. The query-language could easily allow for queries that inadvertently put a heavy load on the source REST API. An example could be a query for weather from multiple places, which would necessarily translate to one request for each location. This scenario illustrates that requirements of the ICC do not only stem from the consumer of its services (the Nudge Design component) but also from the core data source providers.

Another advantage of query-based APIs is that they are more robust to change than their REST-based counterparts. There is no need to change the endpoints to accommodate for client-side changes in requirements; the client can quite simply change the query. Likewise, a change in the data source API that doesn't remove access or add anything new likely would require the Nudge Design component to change, only the microservice wrapping that data source. In the Smart Nudge system, this is good because predicting what information might be used to form a nudge is heavily dependant on the users' preferences.

An example might be that a user interested in paragliding, might be induced to take the bus and go flying if provided with meteorological data that shows that the weather is good for flying. This might include rarely used data points like air pressure, atmospheric soundings[7], dew point temperatures or cloud cover. Designing an API with the particular endpoints for all of the myriad different data points that make up just weather, would likely be extremely complicated and difficult to maintain. The alternative would be to group data points into larger objects, which would again increase the amount of unnecessary data being sent between microservices and the API gateway.

The drawbacks of a query-based API is that it needs to manage this schema, which in the Smart Nudge system setting is likely to become quite large and complicated both in terms of scales, but also in terms of typing. The original XML-object served by met.no, contains 17 distinct data points, each with a different ID and unit of measurement. These units would need to be defined in a schema. A good example of a slightly obscure unit is *cloud cover* which is typically measured in an int from 1-8. But the unit is *eighths*, denoting how many eighths of the sky is covered by clouds. Though not implemented, the idea is that each microservice is responsible for reporting any updates to its schema to the API gateway upon startup, thus enabling the API gateway to supply a complete schema of all the currently available information in the ICC, and how it can be queried. Though complicated, this does not necessarily mean that a purely REST-based approach would be any better. This assumes that the API gateway implements endpoints for all available data points served by the underlying microservices. The same information would need to be kept track of in order to query the ICC successfully for information. Brito et al.[95] demonstrates that using this approach can indeed be easier.

Another disadvantage mentioned about a query-based API is the increased complexity that comes with handling partial responses. As mentioned in Section 5.1 Prototype Overview, responses can be partial on multiple levels. If a single microservice is unresponsive, any data information will be missing. But one could conceive of a situation where only some of the requested fields from a microservice where resolved, making the error more fine-grained and difficult to deal with. In the above example, one could imagine a request for atmospheric pressure returning only the number and not the unit of measurement. A pressure of 101,3 would be very different if it was lbs per square inch and not Newton per square meter. The point is that partial failures can render other data useless. This problem is not unique to a query-based setting, however, and there are ways to mitigate this problem. The most common solution is to group pieces of data as an object that will not be served unless complete.

7. A vertical profile of the atmospheric weather conditions at a certain point in time. Measured by air balloons.

One of the core assumptions made about the Smart Nudge system is that the Nudge Design component must be able to function with partial data. If a planned nudge needed a particular piece of information and this was unavailable, the Nudge Designer must necessarily redesign the nudge or otherwise re-start the design process. It is largely up to the Nudge Design component to know whether a nudge can make do without a requested data point. One interesting feature could be to mark certain requested data points as *required*, thus enabling the ICC to fail early if any of the required pieces were missing or unavailable.

The strengths of the microservice approach to data integration proposed come from the loose coupling of services that neither a monolithic nor a SOA approach exhibits. Each service is entirely free to handle extraction, transformation, storage, caching, schema changes and so on in the way that suits best. This puts the microservice approach ahead. To keep this advantage, however, the loose coupling must be maintained. It also often results in complex systems that make fault analysis and bugs difficult to handle[96].

API gateway

Redundancy is one of the main features of a highly available system[97]. Redundancy in a microservice-based architecture comes from running multiple instances within a microservice. A few of these instances are typically not handling requests, but act as redundancies that are continually running and ready, if one of the other instances fail. Service discovery is an essential tool in running a highly available microservice system. As mentioned, each microservice typically runs a handful of instances[43, 39]. The number of instances running at any given time is preferably dynamic based on the current request load of the microservice in question. One of the problems with running multiple instances of a service dynamically - which is key to high availability - is making sure that clients know where to send requests. Bear in mind that a running instance might crash, or more likely, be killed as the load lessens and it is no longer needed. There is a need for service discovery that ensures requests end up where they are supposed to. There are essentially two architectural patterns that provide solutions to this problem: server-side discovery and client-side discovery.

Client-side discovery requires the client - which in our case would be the API gateway - to keep track of all instances belonging to a particular service. This information is commonly stored in a separate service registry or discovery service. Each instance would register itself (host and port) and thus become part of the currently available services. This quickly becomes complicated because it requires the discovery service to be valid at any given time. It also has the downside of making the service discovery a single point of failure. The solution to this is to run multiple instances of the service discovery, but then which service should take new registers? This brings with it the all too familiar problems of distributed consensus[98] between instances of the discovery service to ensure the validity of the registry[43, 39].

Server-side discovery moves the problem of knowing where to route requests, to the service itself. The analogy here is a help-desk that takes calls and forwards them to whoever is available. This approach is seen in Figure 6.2 Geographical cache solution of microservice, where a request is sent to a load balancer, which in turn round-robins them to all the available instances.

The proposed design for the ICC is somewhat of a hybrid design where the API gateway knows, either through a dedicated discovery service or via a more static lookup table, where each microservice is hosted. Each request is then sent directly to a load balancer, which in turn distributes this between all the available instances of the service. This moves some of the management to the microservices, but also allows a higher degree of autonomy with regards to specialised caching solutions such as a reverse proxy solution. A reverse proxy is where the load balancer checks a cache before even routing the request to a service, thus potentially eliminating the need to call a service instance at all.

As discussed, the underlying data sources often limit the degree to which a microservice within the ICC can scale. This is why server-side discovery managed by each microservice is the best solution in terms of both scaling and availability. Each source is unlikely to be able to scale to a size that cannot be handled by the load balancer (see Section 6.3 Architectural Analysis). Seeing this from a broader perspective strengthens this argument. A nudge is most likely built on extremely local knowledge. They are trying to influence immediate choices. These are choices that rely on knowledge which can only affect a given number of people. Even a bus time arrival system in a big city is unlikely to generate millions of requests per second. There are only so many people whose upcoming choices are touched by bus arrival times or footpath conditions. The underlying data sources also set a soft cap on the level of availability the system can achieve. If a source is unavailable, caching and storing solutions in the microservice will only carry it so far. Eventually, its cache becomes invalid, and the microservice is unable to meet the Quality of Service (QOS) agreement. Another limitation which is somewhat related to this is if a microservice in the ICC used two different data sources to improve on the quality or even offer an augmented set of data to the Nudge Design component. This service would then become less fault-tolerant because it relies on the availability of two different sources. If any ONE of these fails, so too does the microservice that needs both of them.

An unforeseen advantage that comes from decoupling the context of a specific user from the ICC is that the microservices that make up the ICC are stateless. Another advantage is that the whole ICC is read-only, making each microservice stateless. They need not retain any information about the state of the user. Stateless microservices are easier to manage with regards to redundancies.

Fault tolerance is another important factor of availability. The microservice architecture offers several strategies for handling various faults in a system[99]. Timeouts can effectively deal with the connection and networking issues that are more prevalent in a microservice-based architecture. In a monolithic design, this would be less of a problem because the whole application runs as a single executable on the same hardware. Bounded retries is another sensible way to handle transient failures of a microservice. Both of the above methods can be assigned to the API gateway. One of the drawbacks of the microservice-based architecture is that supplying a highly available system requires many different solutions to be implemented and often duplicated across all services. This will, of course, be more challenging to maintain and update.

The microservice-based approach also lends itself to offering partial availability. The architecture offers several places where data faults may be handled to maintain an illusion of availability. An unresponsive data source can be handled by timeouts on the microservice, which then responds according to the protocol for missing/unavailable data. The same goes for partially available data(see Listing 2 Response example). The API gateway offers a similar safeguard for any unresponsive microservices. One downside in this regard is that an unresponsive microservice does not necessarily mean an unresponsive data source. The API gateway cannot feasibly fallback on calling the source directly. There is, therefore, the slight possibility of a false negative, which only emphasises the need for the microservices to be highly available, as per the above discussion. Note that the ability to offer partial yet sensible responses is not a unique feature derived from the microservice-based architectural approach. Nevertheless, it does ensure a high degree of decoupling, which we shall examine next.

6.3 Architectural Analysis

Decomposing a system into modules allows for a more robust and extensible system. Decomposition is one of the main ideas behind the microservices architectural style that has become increasingly popular since 2014[39, 40, 43, 42, 100]. Big enterprise companies like Amazon, Netflix and LinkedIn, have all successfully adopted this architectural approach[100]. This is not to say that this knowledge is new. David Parnas[57] (1971) points to three characteristics of a well-designed architecture. The development time should be short because modules can be worked on without knowing about other modules. It improves product flexibility because major changes can be made to a module without impacting others. Finally, comprehensibility, a well-designed system is simply easier to understand. He writes:

> *This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time.[57].*

This citation sounds almost eerily contemporary when seen in the light of the rise of agile development and microservices. Parnas presents criteria with which to evaluate how a system might be broken into modules that allow for all the desired advantages of well-designed software architecture and warns that unless a system is decomposed into modules correctly, advantages might be lost entirely. This section provides a critical look at how the ICC is divided into modules. One of Parnas' main points is that decomposing a system should not be done by looking at over-arching flowcharts and but introduces the term 'information hiding' as a better criterion. In a flowchart approach to decomposition, each module will typically represent one major step in the execution of the system as a whole. With an information hiding approach to decomposing a system, module boundaries represent design decisions that it attempts to hide from other modules. A good way to illustrate how well a system is decomposed is to examine the consequences of potential changes to the system, both externally and internally.

Looking at the ICC from a broad perspective, we can note a few important points. Firstly, each microservice only wraps one source of information. The primary goal of the service is to hide how this information is acquired, stored and transformed and present it in a unified way to the rest of the system. Each microservice hides the decision about how this is done. The API gateway can be seen as a module that hides decisions about how communication with the services happens. Not just in terms of protocols, but also how faults are handled, non-responsive services and partial responses.

If we look closer at a single instance of the weather service as an example. The WeatherData class represents the raw data as it is sent from met.no. It decides how the raw XML object is parsed and transformed from a string into a programmatic data model of that information. If met.no suddenly removed the XML-endpoint, only this class would need to be updated (provided that the endpoint still supplies the same data).

Caching solutions are another decision which should be modularised and hidden. The business logic within the microservices of the ICC is mainly concerned with extracting data from a data-model and constructing a response. Whether the data model came from a cache or was obtained from the source, should be hidden. This would allow for caching to be turned on and off in a test-scenario, it would enable a separate set of developers to make changes to it, or it could be swapped out completely, without the internal logic of the service needing to change. The prototype does not satisfactorily explore this.

The above analysis can also be seen as an extension of the criteria mentioned in Table 3.1 Criterias for microservice extraction. Coupling refers to the degree to which modules are dependant on each other. The decision to only wrap one source of information decreases coupling in the ICC. The weather service does not need to wait for, nor is it directly dependant upon a response from a different microservice within the ICC. We could conceive of a microservice that incorporated two or more sources of information into an enhanced service offering something the sources did not. This would typically involve some form of processing of data to present an aggregate form of it. Privacy issues and limitations set by sources could be a good reason to do this. But it would create a stronger degree of coupling. That microservice is now dependant on multiple sources which makes microservices less autonomous. Should the supplying microservice be unresponsive, what would happen? The fact that a microservice is dependant on multiple sources could render its logic broken if one of those sources where unavailable. This, in turn, increases the complexity of the system and would require the schema to be able to account for the broken logic. One could, of course, argue that a simple "missing" would solve the problem. This is an exciting aspect worth exploring in the future.

A parallel to the above scenario is a scenario in which the microservices that make up the ICC rely on each other to generate responses or use other microservices as fall-backs, should their data source be unavailable. An example of this might be that the "*bus eta*"-microservice currently cannot get any data on the arrival times to a specific stop. It could then give up on its own source and call the "*bus timetable*"-microservice and for a less accurate response. This approach might, at first glance seem to be highly in tune with the core idea of microservices as independent modules that happily communicate with each other and end-users to build large flexible systems. But this can lead to coupling between services that are difficult to keep track of. It can even lead to so-called cyclic dependencies[100]. What if the "*bus timetable*"-microservice used the "*bus eta*"-microservice as its fall-back? In addition to this, the proposed architecture emphasises one microservice per data source. This allows each service to retain and manage the specific domain knowledge pertaining to the source that it handles. Having services call other services would require them to manage this domain-specific knowledge about their potential list of cal-lees. In other words, it is better to let the Nudge Design component know what it needs. That is not to say that this option should not be explored further as it might if managed nicely, offer a stronger degree of availability and resilience.

Tsilingiris et al.[46] (see Section 2.9 Related Work) also uses a microservice-based approach to integrating heterogeneous data sources. In this approach, the system is layered. Each layer is a set of microservices that handle a specific part of the integration process and handing the results off to a database/storage layer before the next layer of microservices does the next part of the integration process. This architecture differs from the approach presented herein. Tsilingiris et al.[46], takes the decomposition stage a step further. In addition to only dealing with one data source, each microservice also only handles one aspect of the integration process. I would argue that this might create dependencies between layers of microservices that might, to some extent, render the system less resilient and compromise its overall availability. If a service at the first layer (the data acquisition layer) fails, this will render the next layer unable to complete its task of performing data cleanup, mapping and other integration tasks. As stated by Tom Killalea; "*In a mature microservices-based architecture, failure of individual services should be expected, whereas the cascading failure of all services should be impossible[101].*" I argue that by decomposing the system into services by the decisions they need to make about the integration process, we build a system that is better able to withstand a cascading failure due to one service failing.

It is important to note that microservices are not a silver bullet fix for all your software architecture design problems. Though useful decomposition and an effective modularisation makes for a system that is relatively easy to maintain, when changes do occur, they also bring with them several drawbacks that are worth putting into context of the proposed architectural design and the Smart Nudge system.

Handling partial responses and unresponsive sources is an important requirement of the ICC. It is worth pointing out, however, that even if this is dealt with properly, it might still not be good enough. The reason for this is that a nudge is necessarily a composite of several pieces of information all arranged to influence a choice about to be made by the user. This nudge might indeed not work if a key part is missing. Herein lies a weakness not addressed by the proposed architecture. The successful microservices that all responded satisfactorily and within the allotted time, might still have done needless work if the microservice that was supposed to provide the critical piece failed to do so. Since each microservice knows nothing about the complete request, there is no way to avoid potentially needless work being done. I would, however, argue that no amount of clever architectural acrobatics can easily solve this problem. As a side note, one could imagine a sub-par solution where the requesting component can set a level of importance on the various requested sources, thus enabling the API gateway to simply discard a response that does not contain the minimum required information according to the request. This would at least remove the need for the Nudge Design component to also handle this health check after receiving a response.

6.4 Reflections

This section offers a critical reflective look at the project as it comes to a close. As a project ends, it presents a valuable opportunity for self-assessment. The ICC needs to provide a platform that is capable of integrating heterogeneous sources of information. In this regard, I would have liked to use more challenging sources of information, both in terms of data formats, their availability, protocols and restrictions posed by the source provider. Integrating images or video data would be a good example, particularly since the schema assumes anything can fit into a JSON format. In short, the project would likely have offered more meaningful insights if a broader spectre of sources were integrated.

The local bus company in Tromsø[8] does not offer a publicly exposed API to access their live bus trackers, nor any developer version. In the face of this, a more readily available one was chosen since it had no significant impact on the analysis of the system. In retrospect, this might not be entirely accurate. Integrating with such sources is precisely the point of the system and getting permissions, working with an API that might or might not be documented might have provided a more revealing and interesting challenge in terms of evaluating the systems ability to integrate differing sources.

Implementing the API gateway would likely also have contributed valuable insights, specifically with regards to fault tolerance, such as partial responses and slow microservices. This would have allowed a practical test of the systems ability to handle partial failures. What sets this approach apart from similar ones, is that each service handles every aspect of integrating a single source of information. Salvadori et al.[] refers to this as data-driven microservices. The assumption is that this will prevent cascading failures from happening, even if one service crashes because it will not impact any of the others. An API gateway would allow this assumption to be more thoroughly tested.

8. `https://www.tromskortet.no`

6.5 Summary

This section gives an overview of the evaluation and analysis presented in this chapter and ties it into the research goals presented in Section 1.2 Goals. The goal of this thesis was to establish a set of requirements that the ICC of the Smart Nudge system needs to meet and then to evaluate if a microservice-based solution is a good fit. Does a microservice-based architecture in the ICC solve the problems of integrating heterogeneous data sources into the Smart Nudge system? How can this integration component be organised in a way that fits the Smart Nudge system, and does the proposed architecture offer any particular advantages?

The analysis presented in this chapter consists of two parts. The first part is based on the list of requirements produced by analysing the Smart Nudge system and nudging in general. The second part is a more general architectural analysis drawing on classical decomposition techniques and studies. Both the first and second part of the analysis draws on practical knowledge derived from implementing two experimental microservices that each integrates a different source of information.

When comparing the architectural style employed with a monolithic and SOA, respectively, the microservice approach does not necessarily offer any distinct advantages in terms of insulating from heterogeneity or context-awareness. A monolithic approach fares less well in terms of data management, data integration and availability. Both the SOA approach and microservices should fare well overall.

The architectural analysis reveals a potential for fine-grained modularisation that is not present in other approaches. This is one of the most interesting aspects of the proposed architectural approach. The decomposition is centred around a data source. Each microservice is responsible for the complete integration of its data source. The main advantage of this approach is that it gives each service the autonomy needed to incorporate meta-data that the Smart Nudge system relies on to produce nudges. This is an important aspect. The ICC must be able to augment sources with potentially useful meta-data that is important for the system. Examples of this are the time-frame, geographical position and information about the validity period of the data and how often the data should be refreshed. The proposed architecture makes for a system better equipped to handle these challenges in a way that does not compromise any of the established requirements. In short, given the Smart Nudge systems particular need for augmented data, the proposed architectural approach does seem to offer a distinct advantage through the way it is decomposed into services. I argue that this stems from the fact that each microservice is responsible for the complete lifecycle of the data it serves. They are therefore better able to augment the data with whatever strange requirements the Nudge Design component might have in the future. It makes for a more robust system which is better able to incorporate unknown sources and meet hitherto unknown requirements from the Smart Nudge system.

7 Conclusion

This research project aimed to explore the problem of integrating heterogeneous data sources into the Smart Nudge system. The goal was to determine the suitability of a particular architectural approach to designing the system's ICC. By establishing a set of requirements based on the proposed Smart Nudge architecture, a microservice-based architectural approach has been tested via a prototype and analysed theoretically. The analysis seems to indicate that the proposed microservice-based architecture is well suited to resolving some of the requirements that are somewhat unique to the Smart Nudge system. The architecture is analysed using a general theoretical method as well as drawing on the practical experience gained by implementing the prototype. While the example sources of the prototype could have been selected to be more varied, they nevertheless provide insight into how the Smart Nudge system is likely to require augmentation of all data sources it uses. Of particular interest in this regard is the problem of the validity of the information used. Integrated data need to be temporally valid; it needs to be geographically correct. The frequency with which data must be refreshed must be present for the data to be useful to the Smart Nudge system. The proposed architecture places the responsibility of integrating ONE data source in ONE microservice. Having this responsibility adds complexity to that microservice, but also adds the necessary degree of autonomy and independence to better meet the demand for augmentation unique to the Smart Nudge system.

7.1 Future work

This project has revealed several potential areas of research into building a functional information integration platform for the Smart Nudge system. The most interesting ones are listed here.

Request based APIs An important aspect of the ICC is the API gateway. Using an API gateway is a common pattern in microservices. It can manage API-versioning as well as compiling a complete schema for the whole system. This thesis proposes to expose this schema through a request based API similar to GraphQL because it would lessen the bandwidth used between services and the gateway. It would also, to some extent, solve the problem of API-versioning because it is less reliant on static endpoints. These assumptions should be explored at scale. It should also be explored with regards to changing data sources and changing needs from the Nudge Design component.

Scalability Two aspects of the scalability issue are especially interesting in the Smart Nudge system context.

Firstly, the architecture offers each microservice the ability to use tailor-made caching and storage solutions, but does this actually bring any noticeable advantages in terms of performance or availability. As seen, the ICCs ability to scale is inextricably tied to the scalability of the integrated data sources and the lifetime of these sources. Sources that offer short-lived live data might be difficult to scale beyond the capabilities of the source provider, but this should be explored further.

Secondly, the architecture could be said to use an approach to decomposition that is too tightly tied to the data source. Each service handles multiple aspects of the integration logic. While this approach does offer advantages, it might face difficulties in scaling beyond a handful of data sources simply because of the sheer number of microservices needed (one for each data source!), and the added complexities of managing these. This potential problem should also be explored further.

Privacy Privacy and security have on the whole been ignored throughout this project, both in the prototype and theoretical analysis of the proposed prototype. The revised Smart Nudge system architecture (see Figure 4.1 Smart Nudge System architecture revisited) does remove the need for data about users to function. This was done primarily for design reasons. It makes the ICC a read-only information repository, but also ensures that a level of privacy is baked into the system. This is not to say that private information can not be inferred from the requests made to the

ICC. One of the key problems in this regard is the fact that in most cases, ICC will expect a geographical location and expected time-frame as part of a query. This is potentially problematic because this information is highly likely to correspond to where a user is and at what times, at least roughly.

Morally, the Smart Nudge system rests on the assumption that nudges are effective, even when a nudge is completely transparent in every aspect[24, 25]. This is in my opinion and under-explored aspect of the Smart Nudge system which might lend it more credibility when measured against systems based on big data that ignore such concerns[28] (see Section 2.4 The Ethics of Nudging). In order for the Smart Nudge system to rest safely on its ethical foundations, a user must be able to trace the construction of a nudge to the information that leads to its creation. Why was I nudged now? What psychological effects was employed and why? How do you know? Where did this information come from? These are all questions the system should be able to provide to any user, should they want it, which in turn means that provenance must be designed into the system from the start. Exploring how to incorporate data provenance and how this might affect the ICC is an interesting problem.

Bibliography

[1] H. T. Richard and S. Cass, *Nudge - Improving Decisions About Health, Wealth, and Happiness*. Yale University Press, 2008.

[2] C. Meske and T. Kroll, “The dinu-model – a process model for the design of nudges,” in *Twenty-Fifth European Conference on Information Systems (ECIS), Guimarães, Portugal, 2017*, 06 2017.

[3] A. Anders, K. Randi, and Y. Weihai, “Green transportation choices with iot and smart nudging,” in *Handbook of Smart Cities: Software Services and Cyber Infrastructure*, pp. 331–354, Cham: Springer International Publishing, 2018.

[4] R. Karlsen and A. Andersen, “Recommendations with a Nudge,” *Technologies*, vol. 7, no. 2, 2019.

[5] S. Abraham, F. K. Henry, and S. S, *Database System Concepts*. McGraw-Hill Education, 6th ed., 2011.

[6] E. J. Johnson, S. B. Shu, B. G. C. Dellaert, C. Fox, D. G. Goldstein, G. Häubl, R. P. Larrick, J. W. Payne, E. Peters, D. Schkade, B. Wansink, and E. U. Weber, “Beyond nudges: Tools of a choice architecture,” *Marketing Letters*, vol. 23, pp. 487–504, Jun 2012.

[7] H. A. Simon, “A behavioral model of rational choice,” *The Quarterly Journal of Economics*, vol. 69, no. 1, pp. 99–118, 1955.

[8] A. Tversky and D. Kahneman, “Judgment under uncertainty: Heuristics and biases,” *Science*, vol. 185, no. 4157, pp. 1124–1131, 1974.

[9] F. Ricci, L. Rokach, and B. Shapira, “Recommender systems: Introduction and challenges,” in *Recommender Systems Handbook*, pp. 1–34, Boston, MA: Springer US, 2015.

[10] N. Xia and K. George, “Recent advances in recommender systems

and future directions," in *Pattern Recognition and Machine Intelligence* (M. Kryszkiewicz, S. Bandyopadhyay, H. Rybinski, and S. K. Pal, eds.), (Cham), pp. 3–9, Springer International Publishing, 2015.

[11] B. Fogg, *Persuasive Technology*. Interactive Technologies, San Francisco: Morgan Kaufmann, 2003.

[12] H. Oinas-Kukkonen, "A foundation for the study of behavior change support systems," *Personal and Ubiquitous Computing*, vol. 17, pp. 1223 – 1235, 2013.

[13] P. Dolan, M. Hallsworth, D. Halpern, D. King, R. Metcalfe, and I. Vlaev, "Influencing behaviour: The mindspace way," *Journal of Economic Psychology*, vol. 33, no. 1, pp. 264 – 277, 2012.

[14] C. Schneider, M. Weinmann, and J. vom Brocke, "Digital nudging: Guiding online user choices through interface design," *Commun. ACM*, vol. 61, pp. 67–73, June 2018.

[15] M. Weinmann, C. Schneider, and J. v. Brocke, "Digital nudging," *Business & Information Systems Engineering*, vol. 58, pp. 433–436, Dec 2016.

[16] S. Gregor and B. Lee-Archer, "The digital nudge in social security administration," *International Social Security Review*, vol. 69, pp. 63–83, 7 2016.

[17] R. Karen and Z. Verena, "Ethical guidelines for nudging in information security & privacy," *International Journal of Human-Computer Studies*, vol. 120, pp. 22 – 35, 2018.

[18] R. H. Thaler, "Nudge, not sludge," *Science (New York, N.Y.)*, vol. 361, no. 6401, p. 431, 2018.

[19] L. Bovens, "The ethics of nudge," in *Preference Change: Approaches from Philosophy, Economics and Psychology* (T. Grune-Yanoff and S. O. Hansson, eds.), pp. 207–219, Dordrecht: Springer Netherlands, 2009.

[20] T. Goodwin, "Why we should reject 'nudge'," *Politics*, vol. 32, no. 2, pp. 85–92, 2012.

[21] W. T. M, "Nudging and manipulation," *Political Studies*, vol. 61, no. 2, pp. 341–355, 2013.

[22] L. Wenar, "John rawls," in *The Stanford Encyclopedia of Philosophy*

(E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, spring 2017 ed., 2017.

[23] C. R. Sunstein, "Do people like nudges?." http://nrs.harvard.edu/urn-3:HUL.InstRepos:16147874, 2015. Preliminary draft.

[24] H. Bruns, E. Kantorowicz-Reznichenko, K. Klement, M. L. Jonsson, and B. Rahali, "Can nudges be transparent and yet effective?," *Journal of Economic Psychology*, vol. 65, pp. 41 – 59, 2018.

[25] L. George, B. Cindy, H. David, and R. Sachin, "Warning: You are about to be nudged," *Behavioral Science & Policy*, vol. 1, no. 1, pp. 35–42, 2015.

[26] C. Schubert, "Green nudges: Do they work? are they ethical?," *Ecological Economics*, vol. 132, pp. 329 – 342, 2017.

[27] C. Sunstein, "The ethics of nudging," *Yale Journal on Regulation*, vol. 32, no. 2, pp. 413–450, 2015.

[28] Y. Karen, "'hypernudge': Big data as a mode of regulation by design," *Information, Communication & Society*, vol. 20, no. 1, pp. 118–136, 2017.

[29] R. Calo, "Code, nudge, or notice? (regulatory methods of influencing citizen behavior)," *Iowa Law Review*, vol. 99, no. 2, p. 773, 2014.

[30] W. Sarah Myers, W. Meredith, and C. Kate, "Discriminating Systems: Gender, Race and Power in AI." `hhttps://ainowinstitute.org/discriminatingsystems.html`, April 2019.

[31] H. S. Sætra, "When nudge comes to shove: Liberty and nudging in the era of big data," *Technology in Society*, p. 101130, 2019.

[32] J. Singh, J. Cobbe, and C. Norval, "Decision Provenance: Harnessing Data Flow for Accountable Systems," *IEEE Access*, vol. 7, pp. 6562–6574, 2019.

[33] L. Etzkorn, *Introduction to Middleware: Web Services, Object Components, and Cloud Computing*. CRC Press, 2017.

[34] B. P. Douglass, ed., *Design Patterns for Embedded Systems in C*. Boston: Newnes, 2011.

[35] B. Chris and B. Peter, *IT Architectures and Middleware - Strategies for Building Large Integrated Systems*. Addison-Wesley, 2nd ed. ed., 2004.

[36] C. George, D. Jean, K. Tim, and B. Gordon, *Distributed Systems: Concepts and Design*. International computer science series, Boston: Addison-Wesley, 5th ed. ed., 2012.

[37] M. A. Razzaque, M. Milojevic, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of Things Journal*, vol. 3, pp. 1–1, 01 2015.

[38] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W. De Meuter, "Middleware for the internet of things, design goals and challenges," 2010.

[39] N. Sam, *Building Microservices - Designing Fine Grained Systems*. O'Reilly, 2015.

[40] F. Martin and L. James, "Microservices." `https://www.martinfowler.com/articles/microservices.html`, 2014.

[41] M. Robert, "Principles of ood." `http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod`, 2005.

[42] W. Eberhard, *Microservices - A Practical Guide, Principles, Concepts, and Recipes*. Manning Publications, 2019.

[43] T. Hunter II, *Advanced Microservices : A Hands-on Approach to Microservice Infrastructure and Tooling*. Berkeley, CA: Apress : Imprint: Apress, 2017.

[44] F. Martin, "Service oriented ambiguity." `https://www.martinfowler.com/bliki/ServiceOrientedAmbiguity.html`, 2005.

[45] T. Cerny, M. J. Donahoo, and J. Pechanec, "Disambiguation and comparison of soa, microservices and self-contained systems," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, RACS '17, (New York, NY, USA), p. 228–235, Association for Computing Machinery, 2017.

[46] A. Tsilingiris, I. Dimitriadis, A. Vakali, and G. Andreadis, "Demo: Diligent — an osn data integration system based on reactive microservices," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 285–287, June 2018.

[47] T.-D. Trinh, P. Wetz, B.-L. Do, A. Anjomshoaa, E. Kiesling, and A. M. Tjoa, "A web-based platform for dynamic integration of heterogeneous data," in *Proceedings of the 16th International Conference on Information Integration and Web-Based Applications & Services*, iiWAS '14, (New York,

NY, USA), p. 253–261, Association for Computing Machinery, 2014.

[48] I. Salvadori, B. C. N. Oliveira, A. Huf, E. C. Inacio, and F. Siqueira, "An ontology alignment framework for data-driven microservices," in *Proceedings of the 19th International Conference on Information Integration and Web-Based Applications & Services*, iiWAS '17, (New York, NY, USA), p. 425–433, Association for Computing Machinery, 2017.

[49] I. L. Salvadori, A. Huf, and F. Siqueira, "Semantic data-driven microservices," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 402–410, Jul 2019.

[50] B. D. Horne, M. Gruppi, and S. Adali, "Trustworthy misinformation mitigation with soft information nudging," in *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 245–254, 2019.

[51] Y. Wang, P. G. Leon, K. Scott, X. Chen, A. Acquisti, and L. F. Cranor, "Privacy nudges for social media: An exploratory facebook study," in *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, (New York, NY, USA), p. 763–770, Association for Computing Machinery, 2013.

[52] H. Mun and Y. Lee, "Analysis and nudging of personally identifiable information in online used markets," in *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 120–129, 2019.

[53] T. Saad and F. Khan, "Nudging pakistani users towards privacy on social networks," in *2016 SAI Computing Conference (SAI)*, pp. 1147–1154, 2016.

[54] P. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. B. Tucker, A. J. Turner, and P. R. Young, "Computing as a discipline: Preliminary report of the acm task force on the core of computer science," *SIGCSE Bull.*, vol. 20, p. 41, Feb. 1988.

[55] P. Nick, "What is computer science?," *SIGCSE Bull.*, vol. 37, p. 24–25, June 2005.

[56] A. Håkansson, "Portal of research methods and methodologies for research projects and degree projects," in *Proceedings of the International Conference on Frontiers in Education : Computer Science and Computer Engineering FECS'13*, pp. 67–73, CSREA Press U.S.A, 2013. QC 20131210.

[57] D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” August 1971.

[58] L. Carvalho, A. Garcia, W. K. G. Assunção, R. de Mello, and M. J. de Lima, “Analysis of the criteria adopted in industry to extract microservices,” in *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*, CESSER-IP ’19, p. 22–29, IEEE Press, 2019.

[59] D. Sandor, “Designing dynamic and personalized nudges - a model,” Master’s thesis, University of Kaiserslautern, jul 2019.

[60] A. Caraban, E. Karapanos, D. Gonçalves, and P. Campos, “23 ways to nudge: A review of technology-mediated nudging in human-computer interaction,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI ’19, (New York, NY, USA), Association for Computing Machinery, 2019.

[61] K. Daniel, *Thinking, Fast and Slow*. 18 West, 18th Street, 10011 New York: Farrar, Strauss and Giroux, New York, 2012.

[62] E. Johan and E. Mathias, “Can indifference make the world greener?,” *Journal of Environmental Economics and Management*, vol. 76, pp. 1 – 13, 2016.

[63] R. Gouveia, F. Pereira, E. Karapanos, S. Munson, and M. Hassenzahl, “Exploring the design space of glanceable feedback for physical activity trackers,” 09 2016.

[64] L. Colusso, G. Hsieh, and S. A. Munson, “Designing closeness to increase gamers’ performance,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI ’16, (New York, NY, USA), p. 3020–3024, Association for Computing Machinery, 2016.

[65] C. Pinder, J. Vermeulen, R. Beale, and R. Hendley, “Exploring nonconscious behaviour change interventions on mobile devices,” in *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, MobileHCI ’15, (New York, NY, USA), p. 1010–1017, Association for Computing Machinery, 2015.

[66] M. A. A. da Cruz, J. J. P. C. Rodrigues, J. Al-Muhtadi, V. V. Korotaev, and V. H. C. de Albuquerque, “A reference model for internet of things middleware,” *IEEE Internet of Things Journal*, vol. 5, pp. 871–883, April

2018.

[67] N. Madaan, M. A. Ahad, and S. M. Sastry, "Data integration in iot ecosystem: Information linkage as a privacy threat," *Computer Law & Security Review*, vol. 34, no. 1, pp. 125 – 133, 2018.

[68] M. Magnani and D. Montesi, "A survey on uncertainty management in data integration," *J. Data and Information Quality*, vol. 2, July 2010.

[69] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[70] Y. Martin and A. Kung, "Methods and tools for gdpr compliance through privacy and data protection engineering," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pp. 108–111, 2018.

[71] H. Akos, *Designing Microservices with Django - An Overview of Tools and Practices*. Apress, 2019.

[72] F. Roy Thomas, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine CA, 2000.

[73] C. Pautasso, E. Wilde, and A. Marinos, "First international workshop on restful design (ws-rest 2010)," in *Proceedings of the First International Workshop on RESTful Design*, WS-REST '10, (New York, NY, USA), p. 1–3, Association for Computing Machinery, 2010.

[74] O. Hartig and J. Pérez, "Semantics and complexity of graphql," in *Proceedings of the 2018 World Wide Web Conference*, WWW '18, (Republic and Canton of Geneva, CHE), p. 1155–1164, International World Wide Web Conferences Steering Committee, 2018.

[75] B. Lee, "Graphql: A data query language." `https://engineering.fb.com/core-data/graphql-a-data-query-language/`, 2015. Published on FaceBooks development blog.

[76] A. Vázquez-Ingelmo, J. Cruz-Benito, and F. J. García-Peñalvo, "Improving the oeeu's data-driven technological ecosystem's interoperability with graphql," in *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM 2017, (New York, NY, USA), Association for Computing Machinery, 2017.

[77] B. T., "Internet engineering task force (ietf) the javascript object notation (json) data interchange format." Internet Engineering Task Force (IETF), March 2014.

[78] E. International, "Standard ecma-404 the json data interchange syntax." Ecma International, 2017.

[79] M.-A. Baazizi, D. Colazzo, G. Ghelli, and C. Sartiani, "Schemas and types for json data: From theory to practice," in *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, (New York, NY, USA), p. 2060–2063, Association for Computing Machinery, 2019.

[80] M. Kleppmann, *Designing Data-Intensive Applications*. O'Reilly, 2017.

[81] J. Wang, A. Yu, X. Zhang, and L. Qu, "A dynamic data integration model based on soa," in *2009 ISECS International Colloquium on Computing, Communication, Control, and Management*, vol. 2, pp. 196–199, Aug 2009.

[82] P. D. Vettor, M. Mrissa, D. Benslimane, and S. Berbar, "A service oriented architecture for linked data integration," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pp. 198–203, April 2014.

[83] T. Cerny, M. J. Donahoo, and M. Trnka, "Contextual understanding of microservice architecture: Current and future directions," *SIGAPP Appl. Comput. Rev.*, vol. 17, p. 29–45, Jan. 2018.

[84] J. F. Kurose, *Computer networking : a top-down approach*. Always learning, Harlow: Pearson Education, 6th ed., international ed. ed., 2013.

[85] W. Muła and D. Lemire, "Faster base64 encoding and decoding using avx2 instructions," *ACM Trans. Web*, vol. 12, July 2018.

[86] CleverDevices, "Chicago transit authority - bus tracker api documentation." `https://www.transitchicago.com/assets/1/6/cta_Bus_Tracker_API_Developer_Guide_and_Documentation_20160929.pdf`, 2016.

[87] P. Ziegler and K. R. Dittrich, *Data Integration — Problems, Approaches, and Perspectives*, pp. 39–58. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[88] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceed-*

ings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '02, (New York, NY, USA), p. 233–246, Association for Computing Machinery, 2002.

[89] B. Golshan, A. Halevy, G. Mihaila, and W.-C. Tan, "Data integration: After the teenage years," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, (New York, NY, USA), p. 101–106, Association for Computing Machinery, 2017.

[90] A. Y. Halevy, "Answering queries using views: A survey," *The VLDB Journal*, vol. 10, p. 270–294, Dec. 2001.

[91] S. Patni, *Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS.* Apress, 2017.

[92] S. K. Mukhiya, F. Rabbi, V. K. I. Pun], A. Rutle, and Y. Lamo, "A graphql approach to healthcare information exchange with hl7 fhir," *Procedia Computer Science*, vol. 160, pp. 338 – 345, 2019. The 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops.

[93] J. Li, Y. Xiong, X. Liu, and L. Zhang, "How does web service api evolution affect clients?," in *2013 IEEE 20th International Conference on Web Services*, pp. 300–307, 2013.

[94] M. Seabra, M. F. Nazário, and G. Pinto, "Rest or graphql? a performance comparative study," in *Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse*, SBCARS '19, (New York, NY, USA), p. 123–132, Association for Computing Machinery, 2019.

[95] G. Brito, T. Mombach, and M. T. Valente, "Migrating to graphql: A practical assessment," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 140–150, 2019.

[96] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[97] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Microservice based architecture: Towards high-availability for stateful ap-

plications with kubernetes," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 176–185, July 2019.

[98] H. Howard, *Distributed consensus revised*. PhD thesis, University of Cambridge, 09 2018.

[99] N. Wu, D. Zuo, and Z. Zhang, "An extensible fault tolerance testing framework for microservice-based cloud applications," in *Proceedings of the 4th International Conference on Communication and Information Processing*, ICCIP '18, (New York, NY, USA), p. 38–42, Association for Computing Machinery, 2018.

[100] D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," *IEEE Software*, vol. 35, pp. 56–62, May 2018.

[101] T. Killalea, "The hidden dividends of microservices," *Queue*, vol. 14, p. 25–34, May 2016.