Enterprise Application Integration Architecture Recovery

Ossi Galkin

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 21.5.2019

Supervisor

Prof. Mikko Kivelä

Advisor

M.Sc. (Tech.) Minttu Mustonen

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Ossi Galkin

Title Enterprise Application Integration Architecture Recovery

Degree programme Life Science Technologies

Major Complex Systems **Code of major** SCI3060

Supervisor Prof. Mikko Kivelä

Advisor M.Sc. (Tech.) Minttu Mustonen

Date 21.5.2019 **Number of pages** 55 **Language** English

Abstract

Large and medium-sized enterprise has multiple business applications, such as customer relations management (CRM) and enterprise resource planning (ERP) software. Business applications need data from each other, but they rarely communicate with each other out of the box. An enterprise application integration (EAI) platform is used to tie them together. Automating communication between business applications is called orchestration. As EAI is used to implement communication between business applications, in theory, it could be used to map the architecture of the orchestration layer. As companies evolve, so does the software they use. Either documentation is never done, or it is not updated. Overall, the frequent problem is that the company loses track of what software systems it has and how they are used.

This work studies the feasibility of building software that can automatically recover orchestration architecture. The orchestration architecture is then represented as a network to enable further analysis and visualization. This research investigates whether it is possible to use that network to detect business applications that are being used and whether the production environment is separate from the testing environment. In addition, those nodes connecting the testing, and production environments are determined.

The results suggest that it is possible to detect orchestration architecture automatically. Because of a unique characteristic of orchestration layer implementation, such a model can outperform more general architecture recovery models. With a recovered architecture, it is possible to detect whether the production environment is separate from the testing environment and to identify connecting nodes. However, used applications were not discovered using modularity optimization.

Keywords System Integration, Software Architecture Recovery, Orchestration Architecture Recovery

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Ossi Galkin

Työn nimi Järjestelmäintegraation arkkitehtuurin palauttaminen

Koulutusohjelma Life Science Technologies

Pääaine Complex Systems **Pääaineen koodi** SCI3060

Työn valvoja Prof. Mikko Kivelä

Työn ohjaaja DI Minttu Mustonen

Päivämäärä 21.5.2019 **Sivumäärä** 55 **Kieli** Englanti

Tiivistelmä

Suurilla ja keskisuurilla organisaatioilla on useita liiketoimintaohjelmistoja, kuten asiakkuudenhallinta- ja toiminnanohjausohjelmistoja. Yritysohjelmistot tarvitsevat tietoja toisiltaan, mutta ne harvoin kommunikoivat keskenään käyttövalmiina. Järjestelmäintegraatioalustaa käytetään ohjelmistojen yhdistämiseksi. Liiketoimintaohjelmistojen välisen viestinnän automatisointia kutsutaan orkestroinniksi. Koska järjestelmäintegraatiota käytetään viestinnän toteuttamiseen liiketoimintaohjelmistojen välillä, sitä voidaan käyttää orkestrointikerroksen arkkitehtuurin kartoittamiseen. Yritysten kehittyessä myös niiden käyttämät ohjelmat kehittyvät. Usein niitä ei joko dokumentoida ollenkaan tai dokumentaatio jää päivittämättä. Kaiken kaikkiaan yleinen ongelma on se, että yritys menettää tiedon siitä, mitä ohjelmistoja sillä on ja miten niitä käytetään.

Tässä työssä tarkastellaan ohjelmiston toteuttamista, joka voi automaattisesti palauttaa orkestraatioarkkitehtuurin. Arkkitehtuuri esitetään verkostona, joka mahdollistaa lisäanalyysin ja visualisoinnin. Lisäksi selvitetään: Onko mahdollista käyttää tätä verkostoa yrityksen käyttämien liiketoimintasovellusten tunnistamiseksi. Onko tuotantoympäristö erillään testausympäristöstä. Näiden lisäksi määritetään mitkä järjestelmät yhdistävät kehitys-, testaus- ja tuotantoympäristöjä.

Tuloksista voidaan päätellä, että on mahdollista palauttaa orkestrointiarkkitehtuuri ohjelmallisesti. Orkestrointikerroksen toteutuksen ominaisuuksien vuoksi toteutettu malli suoriutui paremmin kuin yleiset arkkitehtuurin palautusmallit. Palautetun arkkitehtuurin avulla on mahdollista havaita, onko tuotantoympäristö erillinen testausympäristöstä ja tunnistaa liitäntäsolmut. Käytettyjä sovelluksia ei kuitenkaan löydetty modulaarisuuden optimoinnin avulla, joka oli työssä käytetty klusterointi menetelmä.

Avainsanat Järjestelmäintegraatio, Ohjelmistoarkkitehtuurin palautus, orkestraatioarkkitehtuurin palautus

Preface

The work for this thesis was carried out in HiQ Finland Ltd. I would like to thank the company for the opportunity to write this thesis.

I would like to thank my supervisor Professor Mikko Kivelä, and my instructor M.Sc. (Tech.) Minttu Mustonen for their guidance during the working on this thesis. In addition, I would also thank all of those who have contributed to work. Especially, I want to thank M.Sc. (Tech.) Erkka Honkavaara, B.Sc. (Econ. & Bus. Adm.) Atso Galkin, B.Eng. Pekka Kinnunen and M.Sc. (Tech.) Sakari Castrén for their support at multiple points during the process writing this thesis and implementing the model.

I want to express my gratitude to my friends and family and everyone I forget to mention by name for their support and encouragement. Finally, I would like to thank the love of my life, Essi Pitkänen, for being compassionate, patient and supporting for all this time.

Otaniemi, May 26, 2019

Ossi Galkin

Contents

Abbreviations and Acronyms

ASCII	American Standard Code for Information Interchange
API	Application Programming Interface
AMQP	Advanced Message Queuing Protocol
BI	Business Intelligence
BPMN	Business Process Model and Notation
COBOL	Common Business-Oriented Language
CSV	Comma-Separated Values
DLL	Dynamically Linking Library
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
ERP	Enterprise Resources Planning
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
FTPS	FTP Over Secure Sockets Layer (SSL)
GUID	Globally Unique IDentifier
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IT	Information Technology
JSON	JavaScript Object Notation
POSIX	Portable Operating System Interface
SaaS	Software as a Service
SOA	Service oriented architecture
SOAP	Simple Object Access Protocol
Regex	A Regular Expression, sometimes shortened to Regexp
REST	Representational State Transfer
SFTP	Secure Shell (SSH) File Transfer Protocol
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
UI	User Interface
UNC	Uniform Naming Convention
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VAN	A value-added network
WWW	World Wide Web
XML	Extensible Markup Language

1 Introduction

Most large and medium-sized enterprises tend to have dozens of different computer systems. Connecting these systems so they can share data and business processes is called enterprise application integration (EAI) or integration for short. The integration enables enterprises to use their systems together to enable business processes. For integration, enterprises typically utilize a special software system called an integration platform. Integration platforms are designed to be efficient to attach other systems through a wide range of communication protocols. They have the means to modify the format of data, for example. Other systems usually provide only a limited means for integration (e.g., they might have only a limited application programming interface (API) or no interfaces at all). Moreover, these systems are not usually directly compatible with other systems. Usually, the format of the data varies in different systems. In addition, EAI platforms provide visibility and monitoring capabilities to integrations. As companies acquire more systems, the number of data flows, and automations between them grow. Automation of business processes, computer systems, and software is called orchestration, which also forms the integration layer that delivers data between computer systems.

As companies grow and evolve, the software they utilize has been changing. Unnecessary legacy systems are deactivated or migrated to new systems. New systems and individual interfaces are introduced. The computer system keeps evolving with the company. With this, the widespread problem is that a company loses track of what software systems are utilized and how they are used. That is due to the failure to thoroughly document new systems or because the company forgets to add to a list of software being acquired. Documentation also needs updating. Otherwise, the documentation becomes outdated or contains systems that are no longer present. Therefore, companies lose information regarding how and what computer systems are being used.

Software architecture recovery is a process where the architecture-level information is formed from a system implementation. Architecture recovery has not been previously applied to integration architecture. However, a specific characteristic of an integration orchestration layer should help to recover its architecture compared to software in general.

1.1 Problem Statement and Scope

The aim of this thesis is to develop software that can recover the integration architecture from an implemented integration orchestration. In addition, this is fulfilled without using any documentation about the implemented orchestration. Software architecture recovery is generally known to pose a problem without a proper solution. However, this thesis studies whether the unique properties of an integration platform can be utilized to recover the integration architecture. Miniservice, as an architecture of the integration layer, makes it possible to recover architecture. Connections to other systems are easily detected from the parameters of the integration flow.

Hierarchical clustering is applied to a network to group different integration endpoints to the same system. The resulting architecture is represented as a network. The properties of the network are compared to the known properties of a scale-free network. Furthermore, different methods of integration, such as spaghetti integration are explored.

The main aim of this work is to recover the orchestration architecture. Once the architecture is known, it is briefly analyzed to detect the business applications adopted by the company. Additionally, the network is analyzed to detect whether the production environment is separate from the testing environment. If it is not separate, then production might depend on systems intended only for testing and therefore might lack monitoring or other elements required from the production environment. In addition, nodes connecting the environments are identified. Furthermore, it is investigated whether it is possible to use that network to detect business applications that are being used.

In this work, only architecture is recovered. Any analysis of the traffic in the integration is not performed. However, it would be possible to combine information regarding the integration architecture to traffic logs. It should also be highlighted that the goal is not to recover the architecture of the used EAI platform. The goal is to recover the architecture of the integration layer built on top of EAI platform (i.e., the architecture of the orchestration layers). Furthermore, the architecture is not recovered for processes connected to the message buses or message queue.

The method is tested using integrations built for a Finnish company that operates in the energy industry. As they have automated large parts of their business, their integration architecture contains highly sensitive business information; therefore, only a brief description of the analysis of the resulting network is provided. In addition, they wished not to be named. The integration is built on top of the FRENDS hybrid integration platform [1].

1.2 Thesis Structure

Chapter 2 presents a literature review on integrations and software architecture recovery. In addition, several principles that have been proposed and tried regarding how to combine several software systems are examined. Furthermore, the impact of the hybrid cloud to integrations is briefly considered. Chapter 3 will give an overview of product FRENDS and how they affect integration architecture recovery. Chapter 4 discuss expected properties of recovered architecture and how standard network analyzing techniques can be exploited to investigate resulting network.

Chapter 5 gives an overview of the methodology used in this work, that is platform independent. The methodology consists of filtering the data, Clustering connections with hierarchical clustering, and searching the connections from implemented integrations with pattern matching. Chapter 6 expands previous Chapter with details of implemented Power Bi model.

Chapter 7 contains the results and analysis for the network that represents the orchestration architecture from the customer's data that was used to test recovery methods. Chapter 8 presents the conclusions of the results of this thesis.

2 Background

2.1 Enterprise Application Integration

Information systems are usually built so that they focus on only the data and business processing in their domains. They can also provide only a limited interface due to a lack of standards [2]. Thus, information sharing between these information systems becomes unavailable. The integration works by connecting these, enabling information sharing between them and enabling them to co-operate.

With well-integrated systems, enterprises can quickly respond to the changing environment. That provides the possibility to utilize all existing systems with greater agility. When the data from all systems are available, they enable faster development of new business processes with reduced costs [3].

A single software can have coherent design principles, applying only selected concepts [4]. When two or more systems are integrated, this is not generally possible, as existing systems can have different design concepts [5] [6]. Therefore, when the data from multiple systems are utilized, the data typically needs to be transformed thoroughly to be utilized appropriately.

As companies grow, evolve, and develop, the software they operate evolve with them. Old unnecessary systems are deactivated or migrated to new systems, and individual interfaces are deprecated. New systems are developed, while the old systems are removed. Overall, computer systems keep evolving with the company [5] [6].

Integration works by moving data from where they are to where they are needed in the right form. Thus, building a common data model or transformation between the existing models is a significant challenge for any software [7] [8] and especially integration software. As Land [9, p. 338] stated: "The evolution, migration, and integration of existing software (legacy) systems are widespread and a formidable challenge to today's businesses."

Like all other software fields, EAI platforms are continually evolving. Despite this, some concepts and technologies have ever-increasing importance and form the basis for which most integration platforms have been built. According to Hohpe [10], they address three fundamental challenges that all integration solutions must deal with: slow, and unreliable networks, the fact that any two applications are different, and that change is inevitable.

2.2 Terminology

The terminology for discussing integration *strategies*, *patterns*, *techniques*, *technologies*, *topology*, *architecture*, and *methods* is not entirely established. The terms are often used interchangeably, as they are closely related. In this thesis, they are used in the same way as Hohpe has defined them [10]. Integration *strategies* and *technologies* are the technological basis on which integration is built. Integration strategies are discussed in Section 2.3. Integration *architecture*, *patterns*, *methods*, *techniques*, and *topology* indicate how systems are integrated and connected. Integration methods are discussed in Chapter 4.

2.3 Integration Strategies

In all the following strategies, it is common that data are stored in a different format in different systems [10]. Therefore, integration must transform or map data from the format of the writer to that of the reader. In practice, mapping is usually the most time-consuming and thus expensive part of a file transfer.

2.3.1 File transfer

Applications can share data via files. The data is written in the file and then delivered where it is needed. The applications must agree on file location, name, and format as well as which will delete the file if necessary and the timing when the file is created. Mainframe computers usually use file systems based on common business-oriented language (COBOL) conventions, whereas Unix and Windows-based systems use flat files, comma-separated values (CSV), JavaScript Object Notation (JSON) or Extensible Markup Language (XML) files. Multiple file transfer techniques have been developed, such as electronic data interchange (EDI) files over a virtual area network (VAN) and File Transfer Protocol (FTP), FTP Over SSL (FTPS) and SSH File Transfer Protocol (SFTP) over the internet. File transfer is often used because it makes integration simple, as integration has only to transfer files. However, it requires that both the writer of the file and its reader agree on the file format. When this is not achieved, transformation must be done in the integration.

2.3.2 Shared databases

When many applications share the same database, no data needs be transferred from one application to another because there are no duplicate data. The significant difficulty of the shared database is that it is incredibly difficult and often impossible for many software vendors to agree on the data schema and format of the data. When applications are distributed across many locations, access to the database across a network is typically too slow to be practical. Distributing databases allows applications to connect via the local network but raises an issue regarding the computer on which the data should be saved. In addition, resolving locking conflicts on the distributed database can quickly become a performance disaster.

2.3.3 Remote procedure invocation

Remote procedure invocation, also known as remote procedure call (RPC), allows one system to expose functionality so that it can be accessed remotely from other systems. If the system must modify data from other systems or perform something, it does so by calling other applications. This allows each application to maintain the integrity and its data format. Furthermore, the application can change its data format without alternating other applications. Multiple technologies, such as Simple Network Management Protocol (SNMP), SSH, .NET Remoting, and Java EMI, implement remote procedure invocation. Nowadays, methods are usually based on

nearly universal internet standards, such as Simple Object Access Protocol (SOAP), on which Web Services-Management is based, for example.

2.3.4 Messaging

Messages from an application are published to a common message channel. Applications must agree on the channel and format so that messages can be read from the channel later. Compared to file transfers, messaging is more immediate. Compared to shared databases, it is better encapsulated. Moreover, compared to the remote process invocation, it is more reliable. However, it requires a more complex programming model and adds overhead to the communication. In addition, it might have synchronization issues or be unavailable due to limited platform support and could result in vendor lock-in after implementation. Messaging systems can be further divided into different parts: send and forget messaging and store and forward messaging [11]. Most new methods count as messaging, such as service buses, queues, and multiple internet-based techniques, such as SOAP, Representational State Transfer (REST), Message Queuing Telemetry Transport, and Advanced Message Queuing Protocol (AMQP).

2.4 Cloud and Hybrid Cloud Computing

The traditional method for enterprise to acquire information technology (IT) assets, hardware, and software is to buy them and maintain them internally. In other words, they use on-premises software. Cloud computing is changing this. Resources, IT assets, hardware, and software are being rented as a service from service providers. Usually, companies move gradually toward the cloud. They have some systems installed on-premises and other in the cloud (i.e., they have a hybrid cloud). Hybrid clouds bring a unique set of advantages and disadvantages, as discussed in this chapter.

A hybrid cloud is a cloud-computing environment that combines on-premises, private cloud, or public cloud services [12] [13]. To enable data sharing between them, orchestration is a necessity. The hybrid cloud gives roughly the same benefits as the single public cloud, such as the ability to grow computing resources as demand grows, but it also prevents giving third parties complete access to the databases and their data. Enterprises gain the flexibility of the public cloud while keeping critical business and sensitive data on premises. The hybrid cloud also provides scalability by eliminating the need to make capital investments when acquiring computing resources. It also makes it possible to free those resources when they are no longer used, which provides cost efficiency. In addition, it is possible to move less sensitive computing to the cloud, freeing up local resources for more sensitive data and applications. According to Won [14] large and medium-sized enterprises face eight types of issues prevent moving to cloud computing: outage (availability), security, performance, compliance, private clouds, integration, cost, and the environment.

In addition to traditional integration challenges, discussed in Section 2.1, cloud integration provides a few unique advantages and challenges. The challenges include

security, data integrity, API limitations, data mapping complexity, and talent shortage [15] [16]. Like other cloud applications, cloud integration provides scalability in terms of end users and the number of applications. It also enables faster integration to software as a service (SaaS) applications and enables faster development and deployment [17].

Cloud integration is built on existing standards and protocols, such as SOAP and HTTP (i.e., REST [18]). Therefore, it is easy to add new connections to applications. Cloud integration provides a more straightforward and more flexible integration that is easier to design, develop, and maintain compared to a traditional EAI.

2.5 Software Architecture Erosion

Software architecture erosion is a term describing the difference between planned or desired architecture and the actual architecture of the implemented software [19]. Software architecture erosion occurs during natural software evolution, when plans are not followed, or when implementation details violate the principles of the selected architecture [20]. This can be a result of a lack of knowledge of the desired architecture or more often a result of too few resources to fully implement a solution that fulfills the architecture preferences rather than just something that works. Modifications are not necessarily documented entirely, and the resulting documentation can become inconsistent or incomplete as the code changes. Software architecture erosion is sometimes also referred to as technical debt. However, that term also refers to old solutions whose architecture is coherent, but that is old and no longer preferred.

2.6 Software Architecture Archaeology in General

Software architecture recovery is a process to extract a high-level architectural model from the available system implementation [21]. Terms such as *reverse architecting* [22] or *architecture reconstruction* [23] are also used. Legacy systems become obsolete with time because these systems have been developed in organizations to address old business requirements. Legacy systems also become obsolete because of financial depreciation and maintenance. Another challenge comes from the constant evolution of systems and incomplete or inconsistent documentation and old documents, which obsolete systems tend to have. Therefore, it is argued that the only way to gain the correct information on how legacy systems work is to re-engineer them [21]. That is achieved first by reverse engineering and then re-implementing them.

Generally, existing software architectural recovery methods are either clustering-based techniques or pattern-based techniques [24]. Regardless of the technique used to gain architecture information, it is mostly accepted that the software systems can be represented as networks or graphs as they are often called [24]. The clustering-based techniques work by grouping the related system entities using a similarity or distance measure and thereby making architectural components. The clustering-based approach is somewhat older, and longer studied. Multiple comparative analyses of different algorithms have been done, and ever better algorithms have been proposed (e.g., by Maqbool [25]) and an improved version has been given by Wang [26].

Pattern-based techniques first create a model architectural pattern, also known as a mental model of the system architecture. They use some modeling techniques, such as a block diagram or a query language [24]. Then, a pattern-matching algorithm is used to gain the architecture of the created model. A block diagram or a query can use information from the application domain and system documentation, and this is seen as a significant advantage if the information is available or can be trusted.

Empirical studies that try to re-identify the architecture of open-source systems and compare known architecture to the found architecture show that even the best algorithms generally perform very poorly despite the metric used to measure them [27]. Two techniques that outperformed the remaining ones were Architecture Recovery using Concern (ARC) [28] and Algorithm for Comprehension-Driven Clustering (ACDC) [29]. The ARC technique uses hierarchical clustering to use the statistical language. In addition, ACDC recovers components using patterns. Despite the inferior performance, architecture recovery methods can be used to correct discovered inconsistencies between the actual implementation and the documentation of the system [30].

2.7 Enterprise Application Integration Archaeology in General

The author does not know of any previous work explicitly trying to recover architecture in integration orchestrations on EAI platforms. The EAI architecture differs from other software systems, and that affects how the architecture can be re-engineered. The critical properties of integration are that they transfer data from where they are to a place where they are needed in the necessary form. Thus, the integration flow consists of three steps: obtaining, transforming, and delivering the data. Transforming the data does not affect the architecture on the integration level. The critical properties of obtaining and delivering the data are the places where the data are obtained and delivered. Therefore, EAI architecture recovery reduces to the question of how different systems are connected. The architecture of the EAI platform is not investigated; only the integration implemented on top of it.

3 Product FRENDS

FRENDS is a complex software, and only parts that are relevant to the recovering architecture of integration orchestration implemented on top of it are covered. This whole section is based on the FRENDS documentation [31], which also describes it in more depth. In addition, a brief history of the product is provided for historical context.

3.1 History of Frends Technology Inc. and FRENDS

The product FRENDS was first developed by ex-members of the Soviet state-owned company All-Union Association Elektronorgtechnica, commonly known as ELORG. The company was also responsible for Tetris [32]. FRENDS was developed to automate updating the price information of uncrewed gas stations that became common in the late 1980s in Finland. Previously, someone had to drive to each station manually to attach a computer to a station. FRENDS replaced this procedure. After the price information was updated by one person, FRENDS was called via a dial-up modem to each station to update the price. The name initially stood for Front End Dialing System.

Frends Technology, Inc was established in 1988. By the beginning of the 2000s, FRENDS evolved into a complete integration platform. Parameterized integration processes could be run on it. It also provided a user interface that allowed a user to define the process execution logic graphically. In 2006, the FRENDS integration platform was migrated on top of the Microsoft BizTalk Server, the EAI platform developed by Microsoft. BizTalk provided basic services, such as basic messaging, data processing, and adapters for connecting to other systems.

HiQ International acquired Frends Technology in 2010 and merged it with HiQ Finland in 2014. In the fall of 2015, a new version of FRENDS was released. It was again a standalone application and was a completely different product. It had a separate code base from the previous FRENDS. However, many of the original concepts were still part of the product. In 2017, HiQ had 1600 employees with 300 in Finland and with 1787.9 million in revenue in Swedish krona [33].

3.2 Architecture

FRENDS is a hybrid integration platform, meaning that it can function fully on premises, in the Azure Cloud environment, or on both (i.e., a hybrid environment), where some parts are in the cloud, and some are on premises. FRENDS consists of three parts: agents, the management UI, and the log service. Any of these can be in the cloud or on premises. Parts communicate through the Service Bus for Windows Server or on the Azure service bus. FRENDS installation consists of one management UI, one log service, and any number of agents. Figure 1 illustrates one possible deployment of FRENDS with five agents.

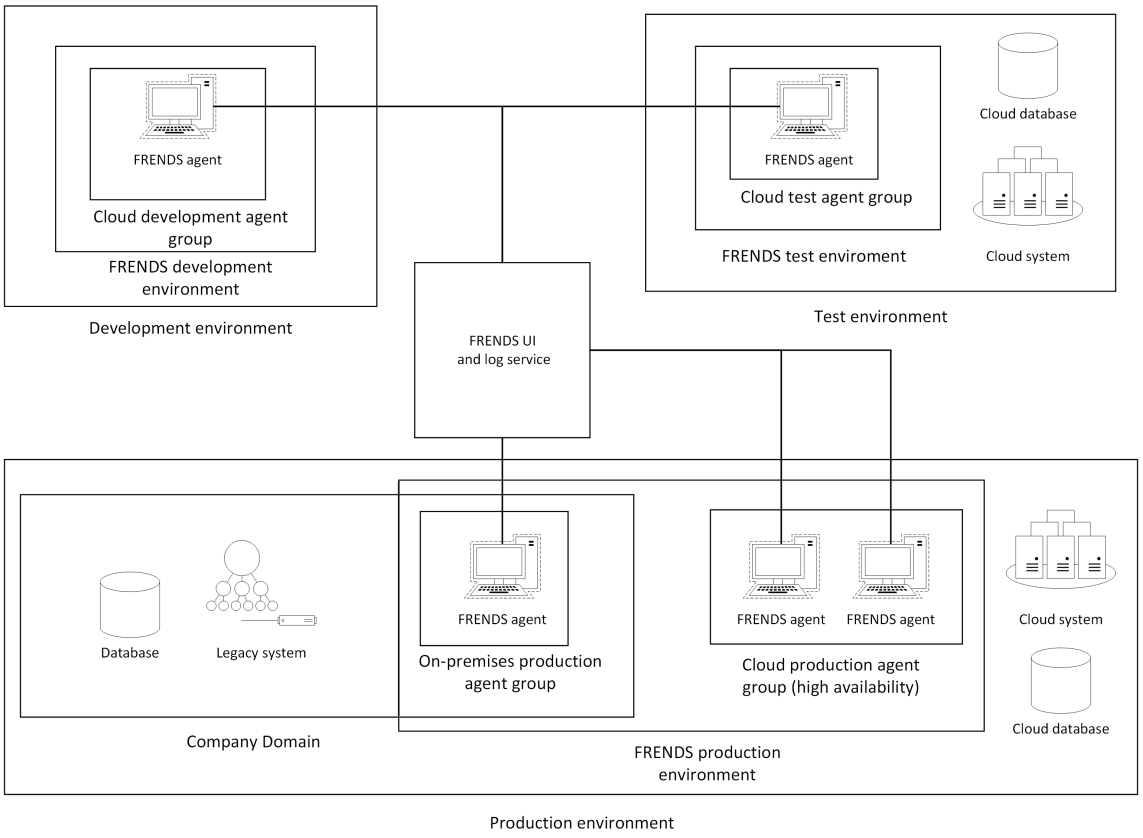

Figure 1: FRENDS architecture with five agents. The figure illustrates the possibility of FRENDS installation where the customer has one development, one testing, and three production agents. This customer has an on-premises production environment, where one FRENDS agent has been deployed. The production environment has also two FRENDS agent in the cloud. These two agents are in the same agent group, and they are configured to work in high availability mode to share the load.

3.3 Process

Processes are how the orchestration layer is built in FRENDS. Each process defines an integration flow. They can be loosely coupled to other systems, so they form a miniservices. The miniservices are discussed in more detail in Chapter 4. The process consists of different elements, such as tasks and triggers. Processes are compiled as executable dynamic-link libraries (DLLs) and distributed to agents as NuGet packages [34]. Each process is versioned and may be deployed to multiple environments and agent groups.

Each process starts with at least one trigger and ends in a return element. The process can also have control elements, such as conditions for each loop, subprocesses, and error handlers. Figure 2 illustrates a process.

A single execution of a process is called a process instance. Both process and process instance can be viewed through the FRENDS UI. Process instance contains all information related to a specific run of the process used to monitor and audit process

runs. There are two kinds of processes available in FRENDS: regular processes (called processes) and subprocesses. Subprocesses are discussed in Section 3.4. Processes are used to create integration orchestration or integration flows. They also visually document the process.

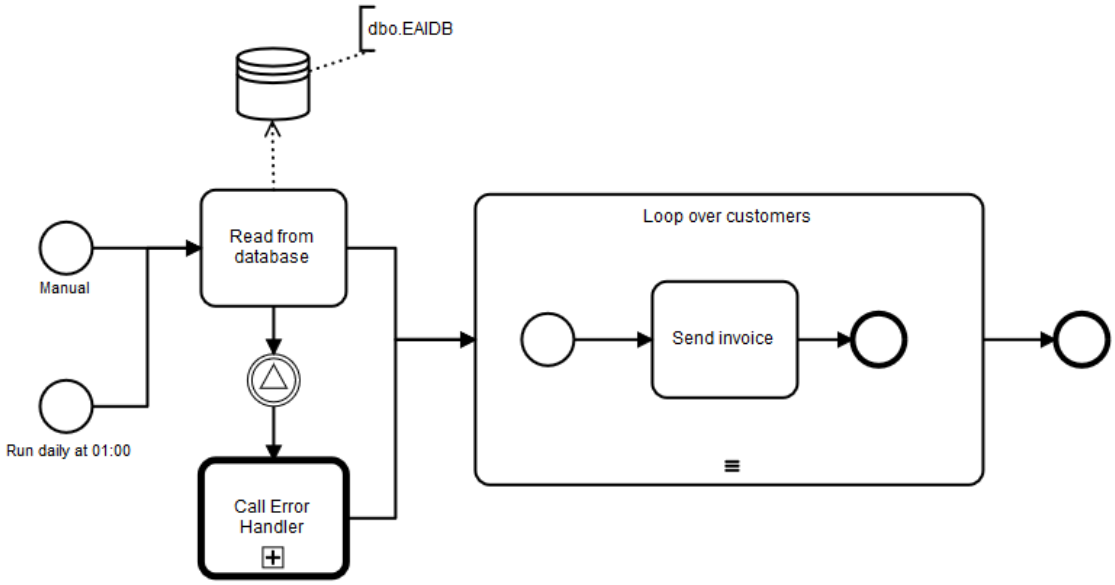

Figure 2: Screenshot of a process from FRENDS UI. This process starts daily at 1 a.m. or whenever manually initiate. It will read customer data from the database and then send invoices to each customer. In addition, the process can detect error happened during reading the database.

3.4 Subprocess

A subprocess is used when different processes need the same functionality, such as error reporting, which is more complex than a single task. In other words, they also create reusable miniservices. Subprocesses can form a hierarchy where the FRENDS process executes a subprocess, which executes another subprocess, and so on. In this way, a process and subprocesses can form an integration orchestration layer where access to different systems is isolated in different subsystems.

Example of the Subprocess hierarchy:

```
Main Process
  Subprocess
    Subprocess
      Subprocess
  Subprocess
Main Process
```

In addition, the subprocess can be executed on a different agent group, in the same environment, then calling the process or subprocess. This way, parts of the

process can be on different physical or logical locations. This enables the hybrid capabilities of FRENDS. For example, a process running on a cloud agent can call a subprocess on an on-premises agent. A process can have a subprocess attached that is called when the process is in error. They can be used to report or log unexpected errors.

The subprocess can also act as a conditional trigger. They enable using complex logic to be the condition when the process is executed. The conditional trigger is executed on the configured interval, and depending on the outcome of the subprocess, the parent process is either executed or not.

3.5 Triggers

Triggers define when a process should be executed. Triggers act as a starting point for the process and the first step in the process. They are started based on an event that the agent can receive or on a condition given by a conditional trigger. These events include a web service call received by the agent, a file being created, a message appearing in the queue, and a schedule activated within a time window. Triggers are the first element in the process editor canvas, and the process can have multiple triggers. Only web and file triggers provide some information about the location or path where the process is started. Other triggers, such as schedule or queue triggers, do not provide information about integration locations. Web triggers start processes after they receive Hyper Text Transfer Protocol (HTTP) or Hyper Text Transfer Protocol Secure (HTTPS) requests. Each HTTP trigger has a unique URL. File triggers start the process when a file matching the file filter is saved to the defined directory.

3.6 Tasks

Tasks are the building blocks of FRENDS processes. For example, a task could read a file or write something to a database. They are reusable components and have parameters that can be configured in the FRENDS UI. Configuring can be, for example, giving the file name and directory location to a task that reads files. Technically, a task is a class library targeting either the .NET Standard or .NET Framework. Most FRENDS tasks are open source, and the code is available via GitHub [35] [36].

Figure 3 illustrates a successful process run, where the process execution path and task that have been run successfully have been highlighted in green. As seen in the figure, not all tasks are executed. In addition, the operator could click on any task and see the execution time, depending on the logging setting, and possibly see the task input and return values as well.

3.7 Parameters

A parameter editor is used to tell tasks and other shapes what they should do. Therefore, configuring parameters is the second most important item in a process

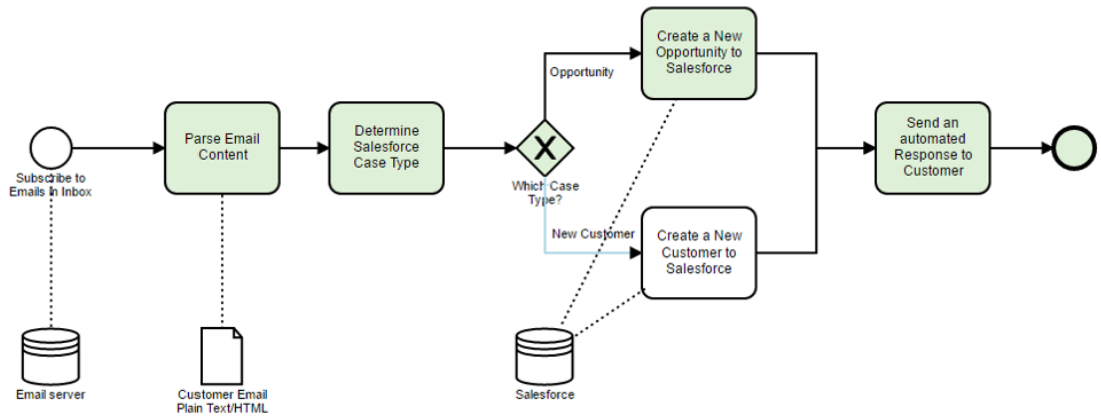

Figure 3: Screenshot of a process execution from FRENDS UI. Executed task are colored with green, marking the path of executed integration flow.

after building the process model itself. The parameter editor is very versatile and features rich code and a text editor, and only the parts relevant to this thesis are presented. Parameters contain all the information of all process endpoints, either straight or via environment variables. Therefore, investigating them and the environment variables is the only step required to acquire information for analysis to perform architecture recovery.

Parameters contain text that depends on the situation, such as structured query language (SQL) queries, XML "files," URLs, or anything else. Figure 4 shows the FRENDS subprocess that contains one task and configuration for that task. The task configuration is on the right side. The task is a SQL query and performs the query shown in the figure. It takes the connection string from the environment variable, and the reference `#env.AdventureWorks.ConnectionString` is shown. The connection string also contains a password needed to connect the database, and it is therefore not revealed anywhere in the UI or logged anywhere. However, it can be obtained from the database. Environment variables are discussed more in Section 3.11. The task performs the SQL query:

```
SELECT TOP 1 [ProductID]
      ,[Name]
      ,[ProductNumber]
      ,[StandardCost]
      ,[ListPrice]
      ,[ProductCategoryID]
      ,[ProductModelID]
      ,[SellStartDate]
      ,CONVERT(varchar(max),[ThumbNailPhoto],2) AS
          ThumbnailPhoto
      ,[ThumbnailPhotoFileName]
  FROM [SalesLT].[Product]
```

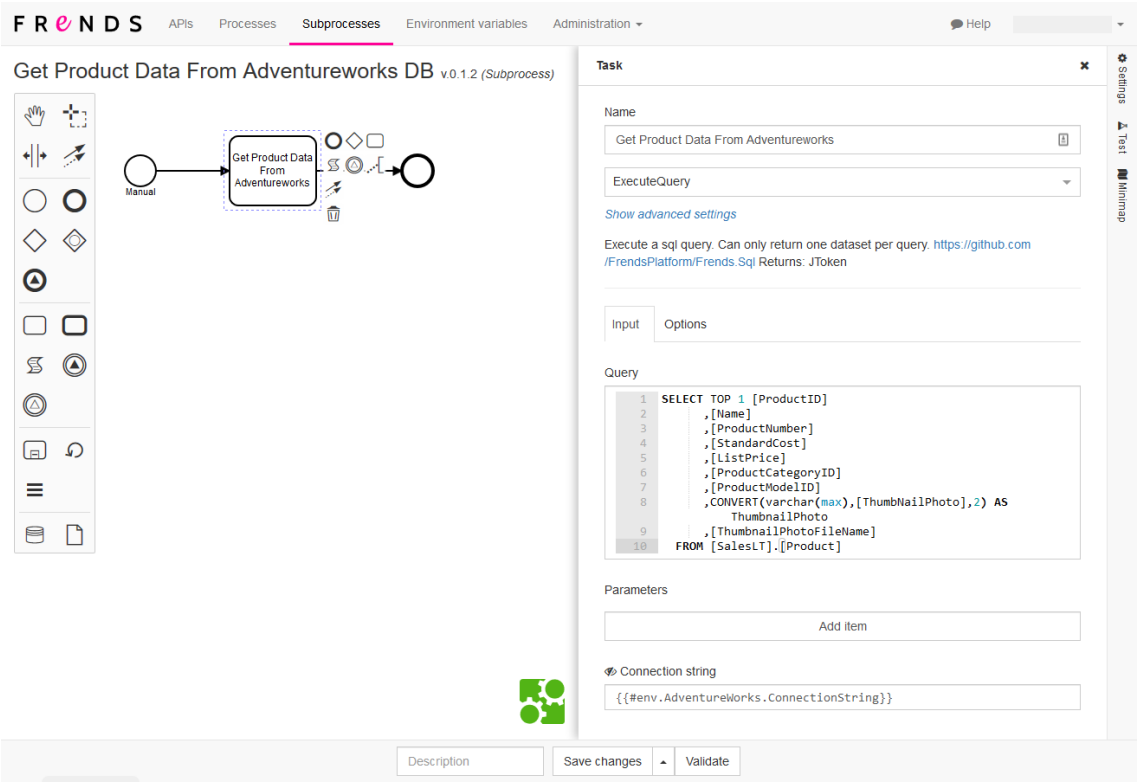

Figure 4: Screenshot of FRENDS UI while editing process that only executes SQL query. View for editing task parameters is open and it shows the SQL query that can be executed.

FRENDS enables writing the C# code right into the parameter editor. It can be inside handlebars `{{}}` or anywhere after the parameter field type is changed to the Expression mode. In addition, FRENDS enables writing multiple different references in Expression mode or inside the handlebars. References always start with a hashtag. These references include, for example, environment variables `#env.name` and variables `#var.name`. When the process is compiled to an executable DLL,

FRENDS resolves all references. It is possible to combine text with expressions. For example, one could write the URL of a destination as follows:

```
https://www.{{#env.CustomerName}}.com/api/v1/petshop .
```

Given that CustomerName contains the correct name of the customer, this would be a working URL. However, it is also possible to combine expressions so that they form meaningful addresses only at runtime. For example,

```
https://www.{{#env.CustomerName}}.com/api/{{#env.Petshop.
ApiVersion}}/{{#var.ApiName}}
```

contains environment variables and variables in such a way that they do not form a valid address unless all environment variables are evaluated. That would be possible, but in this thesis, it is not done because replicating FRENDS's entire parameter expansion would have been tremendous work.

Figure 4 shows an example of the FRENDS UI while editing parameters. Here, subprocess consists only one task: an SQL query. The actual query, task name, and a connection string obtained via the environment variable are present on the right-hand side of the picture. The connection string contains information on how to connect the database to the query.

3.8 Agent

The agent is responsible for executing the integration flows or processes. The agent does not rely on any other FRENDS component to function. Therefore, integration processes keep running, even if some other FRENDS components are offline. The agents are connected to other component databases through a service bus connection. Each agent is assigned to one agent group.

3.9 Agent Group

Agent groups form a farm configuration. They share configurations and states. Agents can share the load with each other and take over the execution from a failed agent. In addition, multiple agents can co-work in an agent group to enable high availability of services by sharing the load.

The process is deployed to agent groups and then delivered to each agent in the agent group. A single environment can contain multiple agent groups (i.e., cloud and on-premises agent groups in production).

3.10 Environments

Environments isolate agents for logical containers that share different roles. Usually, at least three environments are present: development, testing, and production. Each environment has agent groups assigned to it. Each environment also has different environment variable values, monitoring settings, and other settings, such as log and

security settings. Processes can only be created in the development environment. They can be then deployed to other environments. It is possible to deploy a process in some environment without any agents but running them requires at least one agent.

3.11 Environment Variables

Environment variables play a key role in storing configuration information. They store information that is attached to a specific environment. This information is often related to a system where the integration process is connected, such as addresses, usernames, and passwords. Environment variables are organized into three groups: key-value pairs, hierarchical groups, and lists. When information is needed in the integration process, it can be accessed by referring to the environment variable. The correct value for each environment is automatically fetched. These references can be an unlimited number, and if the value must be changed, it must be updated only once in the environment variables.

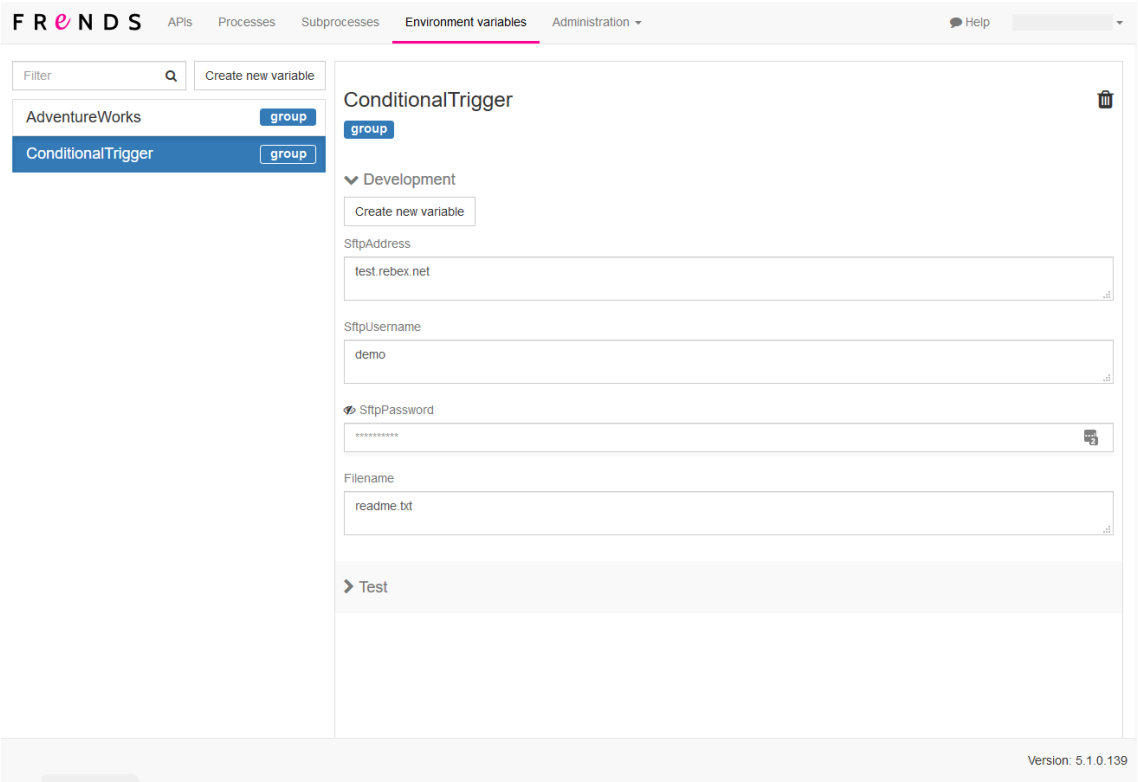

Figure 5: Screensot of FRENDS UI showing the environment variable editing view. The "ConditionalTrigger" group has four fields with their values set in the development environment.

Environment variables can be accessed in a process by references starting with `#env`. For example, in Figure 4, the connection string is taken from the environment variable via the reference `#env.AdventureWorks.ConnectionString`. Figure 5

shows the editing view of the environment variables. For example, `#env.ConditionalTrigger.SftpUsername` is set to have value `demo` in the development environment.

3.12 Log Service

The log service handles all messages coming from agents and evaluate monitoring rules. Messages include heartbeats and log messages. The log Service also handles cleaning up the database by removing old process instances. Process execution status updates are sent to a management UI website by the log service.

FRENDS can log process execution information. Depending on the setting, anything from everything to nothing can be logged (e.g., only errors).

3.13 Database

FRENDS stores everything to its SQL database. Two databases are particularly interesting for this work: FRENDSConfigurationStore and FRENDSLogStore. FRENDSConfigurationStore contains all process and their editing history. FRENDSLogStore contains information about the process instances. By default, any instances older than 60 days are removed to prevent the database from growing too large. Therefore, data in FRENDS does not contain data for an extended period about how processes have been run. This limits the possibilities for what kind of traffic analysis would be possible.

3.14 Traffic Analysis

The limitation of the possibilities of traffic analysis is that, currently, only process-level logs are available. It is possible to extract a task-level analysis from them, but not how much traffic goes to that system. Previously, process- and task-level traffic analyses have been made by Power BI in HiQ, but higher-level analysis first requires information on which processes connect to the same system. The work in this thesis can provide that information. However, the actual implementation of traffic analysis on the system level is out of the scope of this thesis.

4 Expected Characteristics of Integration Architecture

In this chapter, it is first discussed how the chosen method of integration influences the architecture of the orchestration layer. If the chosen method is followed, it defines how different systems are connected. Then more general principles of network tools and their applications to orchestration architecture are discussed.

4.1 Spaghetti Integration

The simplest way to integrate systems is to connect them directly connect them. From the perspective of one system, all connections seem like a star. However, when all systems are presented, the connections look like spaghetti, hence the name. Spaghetti integration is illustrated in Figure 6. Star integration is also called point-to-point integration, as integrations are done from one point directly to another. The overall complexity of the spaghetti integration is not too high for a small enterprise, which has only a few systems. However, large and medium-sized enterprises have tens or hundreds of different systems. Therefore, the different point-to-point connections become unmanageable when old systems are replaced, or new systems are added [37]. Unexpected dependencies will become too common.

Usually, heterogeneous or proprietary interfaces raise the cost of spaghetti integration, but real difficulty comes from the fact that the time and cost to add new subsystems increases exponentially. Therefore, medium-sized enterprises often start seeing the rising cost of integrations and start looking for alternatives to spaghetti integration [37].

4.2 Hub and Spoke Integration

Difficulties with spaghetti integration architecture were first addressed by the hub and spoke systems. They replace spaghetti integration by creating a single system, called a broker, where all other systems were directly connected. The broker allows loosening the coupling between systems by indirect asynchronous communication. The sender does not need to wait for a response but instead could continue its execution after sending the message. In addition, the sender would not need to know the destination of the message.

However, the broker would quickly become a bottleneck and a place of single-point failure. In addition, as it a single destination for messages, it is difficult to scale this model across multiple geographical locations. Moreover, the broker is often supporting only a specific vendor's technology. The support for all kinds of technologies (often legacy) is problematic. Nowadays, the broker model is not generally used due to the advantages of newer approaches [15].

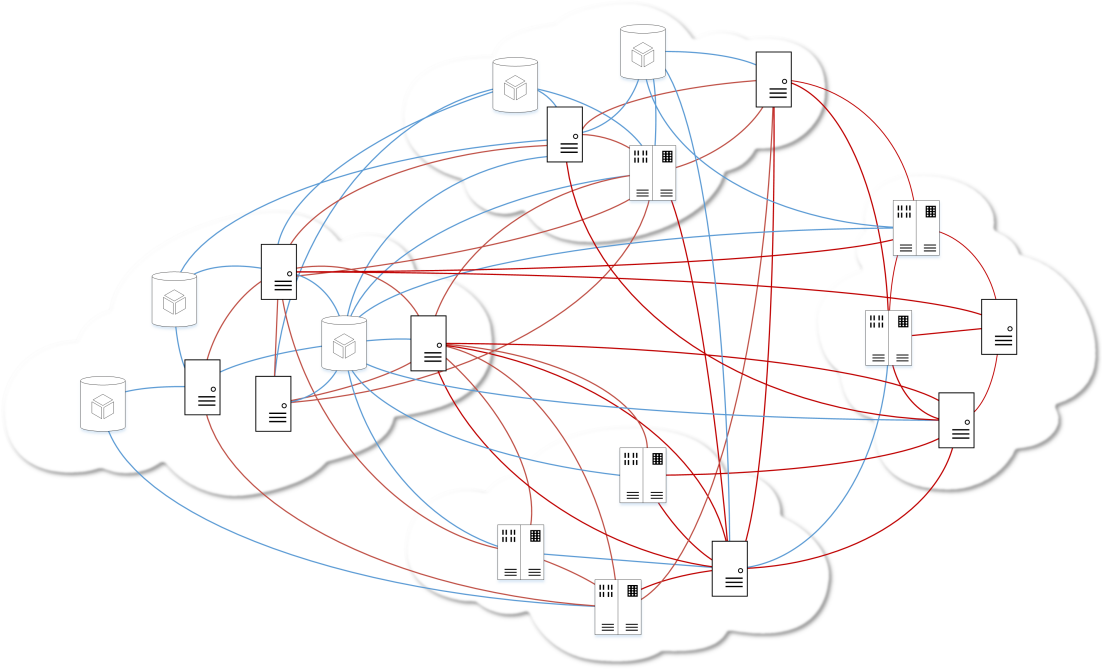

Figure 6: When an enterprise has only a small number of systems, the spaghetti integration remains a viable option. As the number of systems grows, the integrations start to look like a plate of spaghetti. The figure illustrated connections between 20 different systems, divided into four different network locations. In real life, there are usually even more connections, and therefore real-life diagrams are a lot messier.

4.3 Service Oriented Architecture and Macroservices

The Service Oriented Architecture (SOA) was designed to hide different systems with different capabilities and publish them as a general-purpose business service. Hence, services created in one project were available in the following projects, which would reduce the overall costs. For example, a subscription service or customer information service could be one service in SOA. The SOA architecture is usually built on top of the Enterprise Service Bus (ESB) [37].

ESB illustrated in Figure 7 is a specialized subsystem designed to enable communicating between other systems. It connects horizontally to other systems. Horizontal integration means communication between different systems supporting different functional areas of a business. ESB was developed to make it possible to connect various services and thus enabling SOA as a design principle; multiple services could be connected via one bus.

Granularity (i.e., the size of the published service) of services following SOA architecture was high; therefore, these are also called macroservices. The problem with macroservices is the pursuit of generality, and their monolithic nature drives the development costs too high. In addition, when business operations change, the cost of changing macroservices is usually high. A business process using macroservices is illustrated in Figure 8. Granularity is illustrated in Figures 8, 9 and 10. The first one

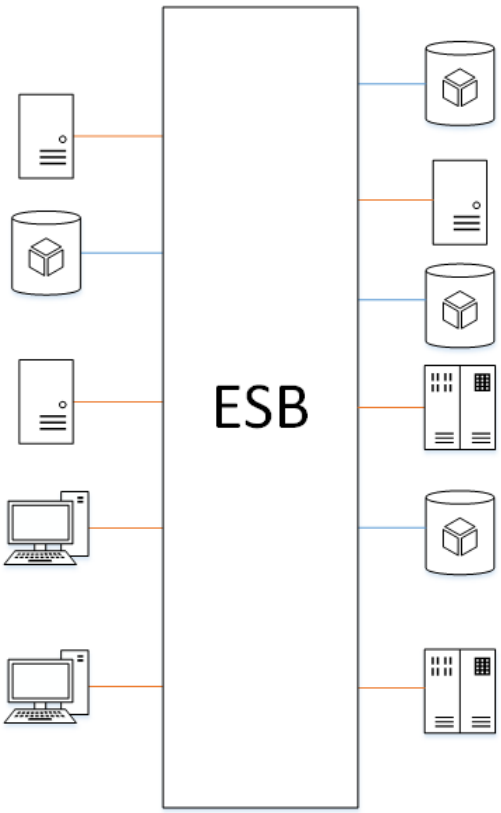

Figure 7: Enterprise Service Bus (ESB). ESB connects multiple different services via one bus, resulting very simple architecture.

describes macroservices, the second is microservices, and the third is miniservices used by a business process.

4.4 Microservices

The microservice architecture was developed to address problems in macroservices. They were developed as a small and independent component. They could be deployed to production without any dependencies on other components or systems. Microservices are technology agnostic, so they do not require any specific technology. That design also provides their ability to scale exceptionally well, and they are used in the largest web services. A business process using microservices is illustrated in Figure 9.

Microservices also contains many aspects that are disruptive to any enterprise [38]. One of the toughest is that each microservice owns its own data. This comes from the fact that the microservice cannot have any dependencies anywhere. Therefore, the microservice cannot rely on data in the enterprise resource planning (ERP) or customer relations management (CRM) systems. Therefore, microservice development and usage require a massive change in the old way of thinking about business processes and developing systems. This independence is a crucial differentiator

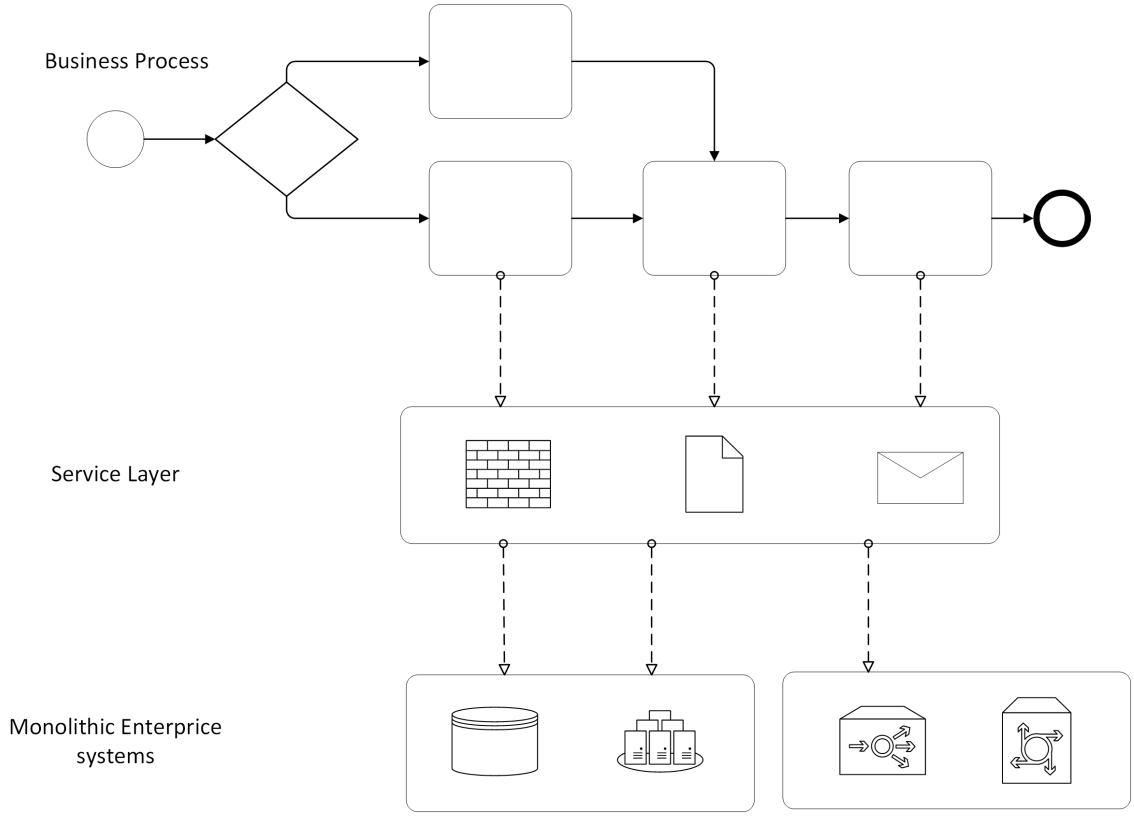

Figure 8: Illustration of a business process connecting to external systems using a service layer (usually an ESB). All connections from integration go to the same destination, that will direct them to the correct systems. Systems are large monolithic services, and therefore they are described as a macroservices.

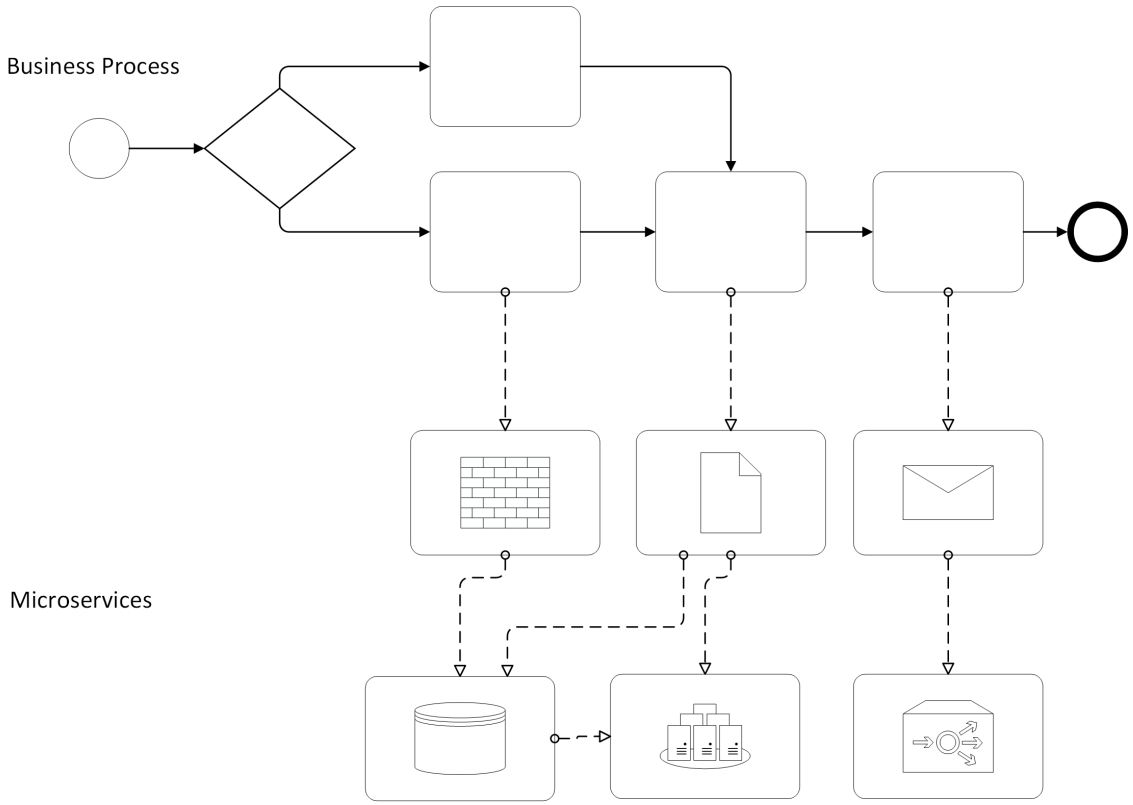

Figure 9: Illustration of a business process connecting to external systems that are implemented as a microservices. Systems are small and individual, and they perform only one thing so that they might call other systems as well.

between microservices and spaghetti integration. In spaghetti integration, everything has dependencies everywhere, whereas no dependencies exist in microservices.

4.5 Miniservices

To address problems in both macroservices and microservices, miniservices were developed. Miniservices try to incorporate the best aspects of both. They differ from microservices in that they can be loosely coupled to other services. Miniservices are easily deployed to production, but even loose dependencies to external systems can form a limitation to scaling. Dependent systems might not scale well. A business process using miniservices is illustrated in Figure 10.

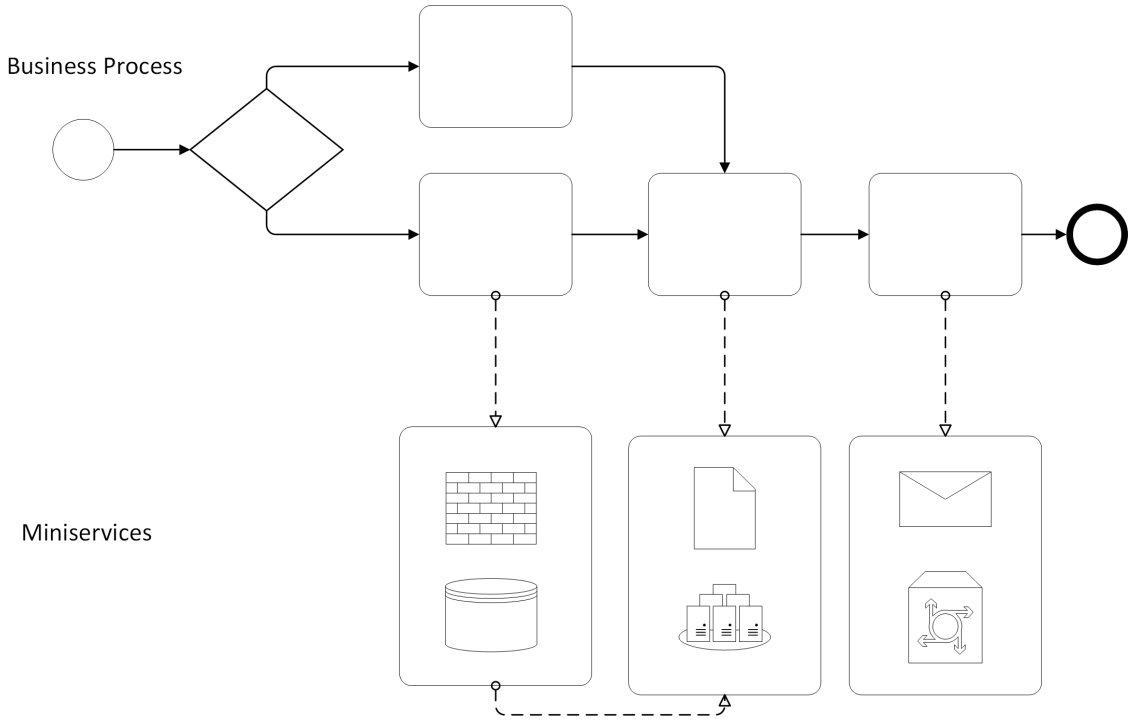

Figure 10: Illustration of a business process connecting to external systems that are implemented as a miniservices. Systems are not as small as in microservices, and they might depend on other systems, but also call them when necessary.

4.6 Macroservices, Miniservices, and Microservices from the Architectural Viewpoint

Whether systems use macroservices, miniservices, or microservices as the designing principle has a significant effect. Macroservice appears as one system regardless of how many operations can be performed, whereas each micro part in a microservice appears as one system. As described in Section 2.1, integration occurs between multiple systems that do not share the same architectural principles. Therefore, the recovered architecture might contain an aspect of multiple methods of integration.

As discussed in Sections 3.3 and 3.4, FRENDS processes, and subprocesses can be viewed as a miniservice. Therefore, most integrations should look like the spaghetti integration illustrated in Figure 6. The crucial difference between the spaghetti integration and microservices and miniservices is that, in spaghetti integration, the integrations are not implemented using any particular system, such as an EAI platform. The EAI provides multiple benefits, like governing the monitoring and reusability of components, regardless of the architecture used. These are discussed in more detail in Section 2.1.

4.7 Centrality Measures

As discussed in Section 2.6, orchestration architecture can be represented as a network. Therefore, the normally used methods to study networks can be applied to orchestration architecture. Usually, centrality measures, such as betweenness, closeness, and page rank, are used to study the importance of different nodes in the network.

However, they are not important in this network because different integration flows are usually separate. The processes can use the same data, for example, one process might involve customer creation, and another can also use that information. Other than handling the same data, the processes do not have a relationship; for example, it is very unusual for a process to depend on another process because it would be a poor design. Subprocesses would be a more viable option. Furthermore, the fact that integration flows are usually separated prevents errors spreading in the network.

4.8 Clustering

Clustering is a technique to detect higher level organization in networks. Networks are said to have a cluster, also known as a community if a network can be grouped by node groups so that each group is more densely connected internally than to other groups. Multiple strategies to form clusters exist and some of them allow overlapping communities. Usually, clusters are detected from this type of network using the stochastic block model or modularity optimization.

The stochastic block model is widely used to detect clusters [39]. It would be a useful tool in solving this kind of problem; however, it was agreed with the customer that their data would not leave the environment. Installing tools such as the graph-tool (a Python module) [40] would have required installing software that was not possible due to the policy limits.

Not being able to use the stochastic block model leaves the modularity measure as a viable option to detect clusters. The modularity method measures the number of edges within the cluster subtracted from the expected number of edges calculated from a random network. The clusters themselves can be found through modularity optimization. For that, the algorithm presented in [41] was used.

4.9 Multipartite Network

At first, the network appears as a multipartite network. However, closer examination shows that this is not the case. Similar methods could nevertheless reveal useful information about the network structure. A k-partite network has k kinds of vertices, and connections are only allowed between different kinds. When $k = 2$, this is called a bipartite network, and when $k = 3$, it is a tripartite network.

This network is partitioned as follows: Data is written or retrieved in the endpoints. Connections can be made between endpoints and processes and between endpoints and subprocesses. Processes can have connections to both endpoints and subprocesses. A major point here is that, in addition to processes and endpoints, subprocesses can have links to other subprocesses. In multipartite networks, nodes are only allowed to have connections to different kinds of nodes. If this were a multipartite network, a subprocess would not be able to connect to another subprocess.

Although the network is not a true multipartite network, it is possible to remove endpoint nodes and then make a new link between the remaining nodes if at least one common neighboring endpoint node exists in the original network. Likewise, processes and subprocesses can be removed, leaving only endpoint nodes. New links are added between endpoint nodes if they could be connected in the original network by any number of processes or subprocesses. Because of formed network resemble unipartite networks markedly, terms *unipartite* is used to describe networks and term *multipartite* to describe the original network.

The result is two different networks: one with only processes and subprocesses that shows how they are connected and another with endpoint nodes that show how they are connected. The first network should clearly show whether different environments are separate and thus even show separate networks for each environment. In theory, the second network could reveal the architecture of the systems connected to EAI, as the EAI platform itself is completely abstracted away.

4.10 Scale-free Networks

Usually, integration orchestrations are separated into three separate environments: development, testing, and production. They should not contain any connection to each other as production should not depend on testing environments that do not have sufficient monitoring, for example. Otherwise, the network might be a small-world network. A topology of the network can be tested by calculating connected components, node degree distribution, and a few other metrics.

Scale-free networks, or scale-free graphs as they are often called, are formed in the dynamic process. Scale-free networks are known to arise from notably different phenomena.They form when new nodes are added to the network, forming new connections. If the probability of a node connecting to a new node depends on the old node connections, a scale-free network will emerge. This leads to the phenomenon of the “rich get richer,” and the nodes with the most links will get even more of them. Most real-world networks develop over time. Thus, scale-free networks are found in various real-life scenarios [42].

5 Methodology

This chapter discusses how integration strategies discussed in Section 2.3 use different technical methods to connect to external systems. In addition, those methods are exploited to recover architecture from the integrations implemented on FRENDS.

The three main ways to represent external systems are paths, URIs, and connection strings. A combination of these is common; for example, SFTP file sharing combines paths and URLs. Various integration strategies use these to connect to external systems, but the details vary.

5.1 Databases

Connection strings specify how to connect to the data source. They are usually used to connect databases or other data sources, such as spreadsheets. The two most-used database languages used in corporations are Microsoft's T-SQL and Oracle's PL/SQL that both extends SQL. Their connection strings share a common characteristic, where parts of the connection string are separated with a semicolon (";") and the parts themselves contain a keyword and value [43] [44]. Databases contain multiple tables containing different data relating to different items. In this work, which table integration process writes the data is not investigated. The whole database is treated as one system and thus, one architectural component.

5.2 File Transfer

File transfer moves files between locations, as the name suggests. Fully qualified paths or absolute paths describe a location of a file or web resources starting from a file system root, drive letter, or full domain [45]. Relative paths contain only part of the absolute path. Thus, relative paths usually locate resources relative to the current location.

5.2.1 Uniform Naming Convention for paths

The Uniform Naming Convention (UNC) is a standard method in Windows to describe the location of a file, directory, or printer. The simplified generic form for remote resources is `\\ComputerName\SharedFolder\Resource`. Local files usually start with the drive letter `C:\\Folder\File` [45]. Relative paths do not start from a computer name or drive letter but from the "middle" of the path, for example, `SharedFolder\Resource`.

5.2.2 Unix-style file system paths

Paths in portable operating system interface (POSIX) systems [46] or Unix-like systems, for example Linux, use notation that is generic for `hostname:/Directory/Resource` for the remote location and `/Directory/Resource` for the local. The leading slash ("/") marks the root directory (i.e., the beginning of the file system

hierarchy). Paths can also be relative, and then they do not start from the root but from the middle of the path, for example, `Subdirectory/Resource`.

5.2.3 File shares

It is possible to share any directory to be available in the network. The technical details on how sharing works varies between Windows and Unix-like systems, but the general principles are the same. It is also possible to give a new address (i.e., a path to access a shared directory remotely). For example, using the UNC convention, a resource in a local location can be found as follows:

```
D:\EAI\FileStorage\ERP\Invoices\Company AB.\VacationFiles\
        Temporary Storage\Incoming files\ ,
```

and could be shared at location

```
\\Servershare01\Company AB.\VacationFiles\in\ .
```

It is also possible to share the same directory or resource multiple times with different paths. For example, it is possible to share a local location:

```
D:\EAI\FileStorage\ERP\
```

as `\\ErpFiles\` , thus enabling access to a previously shared location via the path:

```
\\ErpFiles\Invoices\Company AB.\VacationFiles\Temporary Storage\
        Incoming files\ .
```

From that location, it is impossible to know the underlying hierarchy. Now, it is hard, or even impossible to determine without further knowledge whether these two shared locations lead to the same place or not. Therefore, it is impossible to recover such architecture without additional knowledge.

5.3 Messaging: Internet-based Strategies

Correct URLs have a strict format [47] and represent the location and access method of some resources on the internet. Moreover, URLs support only American standard code for information interchange (ASCII) characters, but that can be extended by canonicalization [48]. Canonicalization enables encoding of a non-ASCII character with ASCII characters. Usually, a URL is not written completely and therefore, is not "correct." One commonly known example is the use of the WWW, World Wide Web prefix in a URL [49], as is in `www.example.com` where `http://` is omitted, for example. Another notable addition is the Uniform Resource Identifier (URI) is a file URI [50] used to locate files on the internet. In general, it is tricky to detect all URI-like strings, but fortunately, it is not necessary because they are usually hierarchical locations and detecting that hierarchy is sufficient, as discussed more in Section 5.5.

5.4 Messaging: Buses and Queues

As described earlier, there are two ways to deliver messages: either by bus, such as the Azure Service Bus or by queue, such as AMQP. Both work the same way in that all messages are sent to one place regardless of where they are going or what they contain. Therefore, it is significantly more challenging to identify the message destination from the integration process implementation. Additionally, identifying the destination of messages sometimes requires information on a system that has a subscribed message from a bus or queue, and therefore, knowing the destination from the implementation of the integration process is sometimes impossible. There might be multiple queues that could be identified but only one bus, and information that the integration process is connected to the bus is not very informative. Therefore, buses are left out of architecture recovery.

At the time of writing this thesis, FRENDS has only limited support to AMQP queues. Messages can be only read from it, and no support exists for other queues. Therefore, integrations that use queues are not made, and these are left out of the architecture recovery.

5.5 Matching with Regex

Paths, URLs, and others are well-structured strings, so it makes sense to use a regular expression, regex, to match them. Regex is a widely used language to find patterns, such as URLs or paths, in the text. In this work, the regex processor called Perl Compatible Regular Expressions [51] (PCRE) was used. It provides a robust and well-documented environment to execute regex.

A significant difficulty in using regex comes from the fact that different systems use different characters with special meanings. For example, if delimiters are present in data, they must be escaped with escape characters to avoid delimiter collision. Usually, backslashes ("\") are used as escape characters. The problem arises when different systems use the same characters as delimiters, and the information goes through multiple systems. Each system adds escape characters that the next system needs to escape. Therefore, the massive number of backslashes needed for escape characters makes the regex patterns nearly unreadable. This phenomenon is known as "the Backslash Plague" or "leaning toothpick syndrome" [52]. For example, to match `//` after the protocol part of the URL, each forward slash is escaped with a backward slash (`\/`). Moreover, when doubled, as in URLs, this becomes `\/\/`.

We consider a similar problem with UNC by example. In UNC, paths might begin with double backward slashes `\\`. Both might be escaped by the external system, resulting in four backslashes `\\\\`. Therefore, the regex matching two backward slashes requires eight backslashes `\\\\\\\\`. Besides that, the backward slashes might have to be escaped twice in UNC, and the UNC might also start with a drive letter followed by a backward slash and colon `\:`. Therefore, the regex matching the start of the UNC path is `[a-zA-Z]:\\{1,2}|\\{2,4}`. The previous regex matches any string that starts with an ASCII letter followed by a colon and one or two backward slashes, or a string that starts with two to four backward slashes. The

regex also matches a string starting with three backward slashes, but generally, they are not presented in the data and can easily be filtered out afterward.

5.6 False Positives

Integration processes can and often do contain a vast number of URLs and file paths that do not represent external systems. These include XML schema locations and the locations of temporary files. They are false positives as they do not represent the location of an actual place where data is written or where the data is read. They have to be filtered out, or otherwise, clustering, discussed in section 4.8, would recognize false positives, such as a temporary file, as an individual system.

False positives are, however, easy to filter out because XML schema locations are prefixed with "xmlns." File paths to local resources can be identified because, at the time of writing this thesis, FRENDS only runs on Windows, where the drive letters C, D, and F are used for the local file system. FRENDS references, or parts of them, might also be detected as URLs under certain circumstances. For example, the environment variable `#env.customerAddress.fi` is roughly like the URL `customerAddress.fi`. FRENDS references will always start with either `#var`, `#env`, `#process`, or `#result`, so they are easy to filter out.

5.7 Hierarchical Clustering

A common characteristic of paths and URLs is that they provide information about the underlying organization by design. They are designed to tell the hierarchy that starts somewhere and leads somewhere. For example, UNC starts from a drive letter, followed by folders. As more folders are added, the location becomes more specific. Therefore, removing folders from the end enables one to move upwards in the hierarchy, thus enabling the clustering of larger units. For example, the path

```
D:\EAI\FileStorage\ERP\Invoices\Company AB.\VacationFiles\
            Temporary Storage\Incoming files\
```

could be reduced to

```
D:\EAI\FileStorage\ERP\Invoices\Company AB.\
```

and, alternatively, even to

```
D:\EAI\FileStorage\ERP\ .
```

The former would represent all file paths to Company AB and, later, all ERP connections. The URL works is a similar fashion. For example, the URL

```
http://www.example.com/api/v1/Petshop/store/order/
```

could be split to

```
http://www.example.com/api/v1/Petshop/ .
```

The removal of levels of the hierarchies is an essential part of how clustering would be performed, as otherwise multiple very similar paths would represent individual unrelated connections. The available literature did not contain any insight into this problem. Moreover, the author did not produce any smart algorithm to determine the number of levels removed, so it was left as manually configurable.

The hierarchical clustering was done before the network was constructed. Therefore clustering technique Modularity Optimization, discussed in Section 4.8 were applied to the network that was already clustered with the hierarchical clustering. This two-phase approach to clustering resulted in much better results than either technique alone.

6 Power Bi Model and other implemented software

This section provides a simplified description of the implemented Power BI model and other tools. The work is done on the Power BI [53] with the Network Navigator Chart add-on [54] to create an illustration from the networks. Additionally, the R [55] with the rjson package [56] was used inside Power BI and to generate more illustrations. In addition, Gephi [57] was utilized to produce illustrations of the networks.

Unfortunately, the Power BI model with the R code cannot be published due to concerns regarding the accidental release of customer data and business operations. It was necessary to enable native database queries in Power BI, and the database must be marked as public to make it possible to run R scripts on the data obtained from the SQL database. Effectively, it would disable all privacy and data protections inside Power BI. It must be done because Power BI cannot protect data because the data are handled with the R code, which could theoretically do anything. The R language was used because Power BI does not support regex or similar technologies for pattern matching. Rest of this chapter describes what was needed to form network representation of the orchestration layer in addition to things described in Chapter 5.

6.1 Process versions and colors

In the database table `PROCESS_VERSION`, each process that is deployed is listed. It also contains information on which agent group and the environment the processes are deployed. Because that information is used multiple times, it is read in its own table and processed in Power BI only once. After that, each process is attached to a color based on its environment that is used to make networks. Example data after transformations (for clarity the process ids are truncated):

ID	PROCESS_ID	VERSION	ENVIRONMENT	AGENT_GROUP	Color
466	825...86F	22	Default	Default	Red
438	19A...FA1	3	Prod	Prod	Green

6.2 Process names

Usually, a process has the same name in each environment. However, when all processes from all environments are plotted in the same network, it is inconvenient if there are, for example, three processes plotted with the same name. Therefore, the environment name is added in front of process name (e.g., `Prod | Example Process Name` and `Test | Example Process Name`). It is done by expanding information obtained in 6.1 by adding the process name and a similar environment. In addition, the Process ID is attached to the environment because it is necessary to differentiate

the processes when they are deployed to different environments. Example data after transformations:

NameAndAG	IDandAgentGroup	Deployment.Color
Prod | GO: It is network (2)	13823Prod	Green
Default | GO: It is network (2)	13823Default	Red

6.3 Environment variable processing

First, environment variables are read from the database and unnecessary columns, such as old versions of environment variables, are filtered out. The R script is then used to process the JSON, where the environment variables are stored. The script reconstructs the environment variables in the same format as they are written in the Parameter Editor (e.g., `#env.Customer.Name`). In addition, the R script searches the connection endpoints as described in Section 5.5. Environment variable names are then combined with values from the R script. Then, the empty environment variable values and other artifacts, discussed in Section 5.6, are filtered out. Finally, the right environment is placed in the place of #env, so it becomes possible to differentiate the environment variable names between environments. For example, `#env.Customer.Name` is changed to `#Test.Customer.Name` or `#Production.Customer.Name`. It is done because environment variables have different values in each environment. After the change, each environment value name points only to one value. Example data after transformations (for clarity the unique ids are truncated):

ID	UNIQUE_ID	EnvValue	EnvName
1137	ADF...6FE	f:\\kuuukko\pics.jpg	#Test.GO_network.path
1137	ADF...6FE	example2.com	#Test.GO_network.url

6.4 Subprocess Trouble

Subprocesses stored in the FRENDS database are like normal processes but differ on a few points. For example, they use slightly different escaping methods for special charters. Additionally, subprocess can call subprocesses from normal processes or subprocesses. Therefore, subprocesses are handled aside from normal processes. First, subprocess information is read from the database. Then, old subprocesses are filtered out, and the description field of the subprocess is cleaned. The R script is used to determine which subprocesses the process is calling. Then, the result from R is cleaned and modified to the Power BI list. The environment name is added to the process id as well as the process name. An R script is used to search the connection endpoints, as described in Section 5.5.

The Network Navigator and a few other network tools cannot make the connection between two source nodes. Therefore, the subprocess call has to represent an additional node between the process and subprocess. In the database, FRENDS stores only reference in the process, indicating it calls the subprocess, but not another way around. An R script is run to make a connection between the subprocess and

the process that called it. Example data after transformations (for clarity unique ids and names are truncated):

IS_SUBPROCESS	NameAndAG	UNIQUE_ID	VERSION	IDandAgentGroup
False	Default | test...	4CC...A67	15	2546Default
False	Prod | GO: It ...	597...253	2	13823Prod

Moreover, FRENDS saves only a connection from the calling process to a called one, not the other way around. Thus, it must be determined with an R script.

6.5 Processes

In addition to process information, all other information from other steps is merged into the process information to obtain the complete node-edge list. It is done together with the processing process information to avoid the receptive steps in the Power BI model. First, the process information is loaded from the database. Then, the information from Section 6.1 is merged. The old processes are filtered out, and unnecessary columns are cleaned. An R script is run to detect URLs, UNC, and other paths as described in Section 5.5. Environment information is then merged into the Process ID and environment variable name (i.e., `#Production.Customer.Name`). Then, the information about the environment variables and their values are obtained from Section 6.3.

Using environment variables in the process and their corresponding values in each environment, it is now possible to combine the information and obtain an endpoint for each process stored in the environment variables. The following step is to combine the subprocess as endpoints and the reverse subprocess connection that was generated in Section 6.4, so the connection from the subprocess to the process was formed as well. After that, multiple delimiters, characters, and other items must be filtered out, and the endpoints are clustered as described in Section 5.7, preserving only the first four levels in the path. It was manually tested to prove the most satisfactory results. However, some paths would still require stricter cutting, but it would lead other paths to be cut too aggressively.

6.6 Generating networks

After everything is processed in Power BI, in Section 6.5, a complete list of edges is available. The Network Navigator is used to generate an interactive visualization of the network. Its biggest downside is that it does not support showing a large number of nodes, so it cannot show all the environments at the same time. However, all nodes can be plotted with Gephi. These networks are discussed in Chapter 7. Figure 11 shows an example of the visualization of the network. Color of process and subprocess nodes show their environment. Endpoint nodes of different connections are in gray.

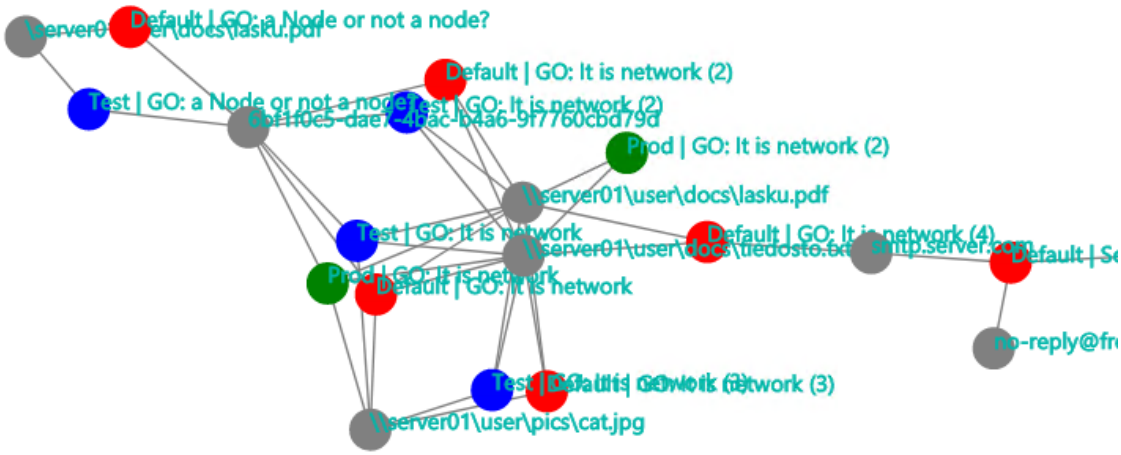

Figure 11: Colored circles are processes on different environments; gray circles are endpoints. Nodes are in force-directed layout and they were allowed to overlap. The figure is a screenshot from Network Navigator.

7 Analysis

The main aim of this thesis was to determine how software can automatically recover orchestration architecture from the implemented integrations. Two different aspects are required to make architecture recovery feasible. The orchestration layer on the EAI platform must be parameterized, and integration flows must follow a microservice or miniservice paradigm as the design choice. Parameters only contain a small portion of code, while still containing all the necessary information, so they make this task practical. This is discussed in more detail in Section 3.7. The microservice or miniservice as an architecture makes it possible to recover architecture. If all integrations were done in the same macrosystems, there would not be any architecture to be recovered at all. In addition, resulting architecture should look more or less like spaghetti, as FRENDS processes are miniservices. These are discussed in more detail in Section 4.6

The orchestration architecture is then represented as a network to enable further analysis. However, for the customer whose data was used to test implementation, FRENDS is used to implement critical business processes. Thus, only the summary is shown. In addition, in many figures, the precise amounts or names are not given.

Figure 12 is an illustration of all the nodes in a network. The colors indicate whether a node is is an endpoint (i.e., a location where data are read or written) or a process and which environment it belongs. This figure illustrates the multiple key properties of the network. Most nodes are connected to each other and form a so-called giant component. The nodes are considered connected if there is some way to connect the nodes by links and other nodes. Only nodes that have a small number of links have been left out of the giant component. The only exception is three nodes that share a huge number of endpoints. They can be seen on the left-hand side of the network. A manual inspection of these processes reveals that they all share the same file backup location that uses a highly hierarchical folder structure, which hierarchical clustering was not able to shrink.

The figure clearly shows that most endpoints in the giant component have links to nodes from multiple environments. Thus, the production environment is not separate from the testing environment. However, due to a high number of nodes connection environments list of nodes connecting the environments did not prove to be valuable for this customer. Manual inspection of the network revealed that it presents the architecture of the integration orchestration is approximately correct. The developed method thus outperforms more general architecture recovery methods that are known for their poor performance. As discussed in Section 4.8, the researcher investigated whether modularity optimization could be used to detect business applications. The resulting groups are used to color nodes in Figure 13. Clearly, modularity can identify groups within the giant component. Manual inspection revealed that they correspond to some degree the actual systems being used, but it was generally not able to detect different systems correctly.

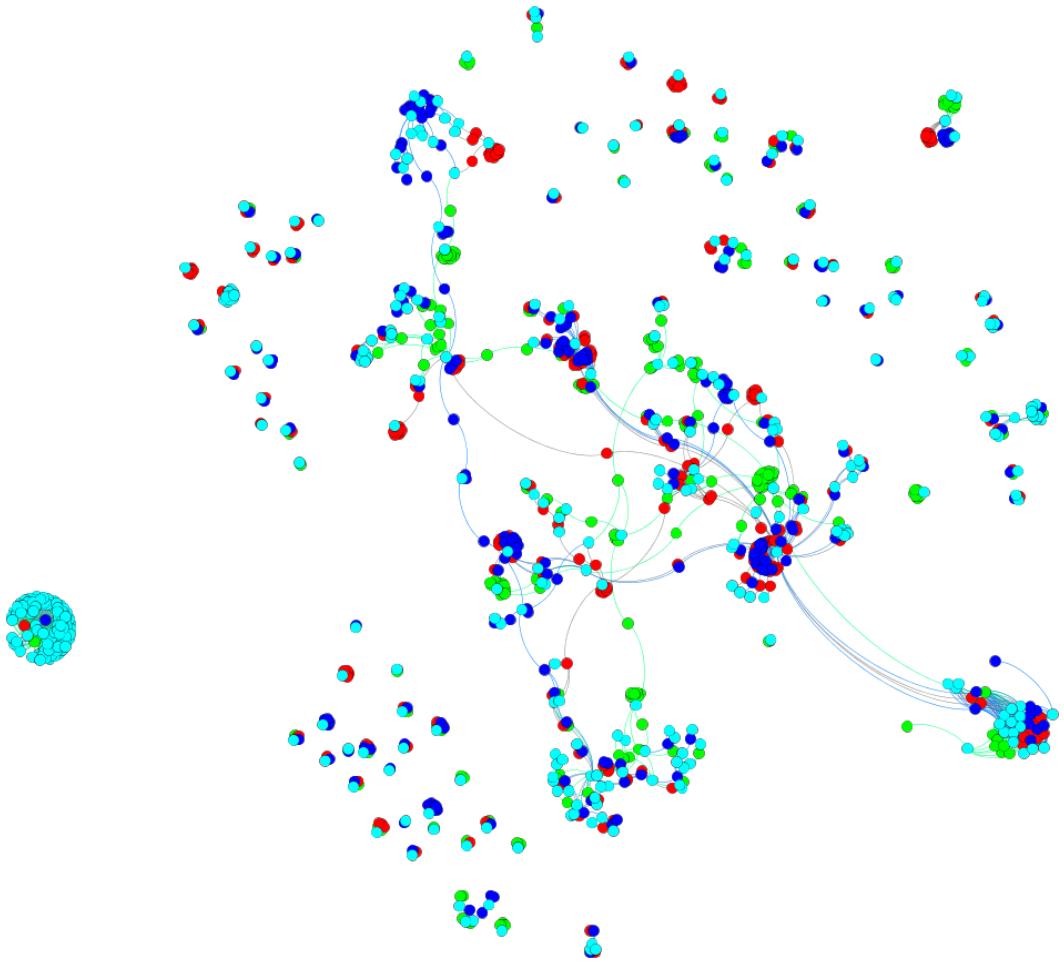

Figure 12: Illustration of a network constructed from the data that was used to test the recovery implementation. All nodes are in a force-directed layout. The colors represent the environment where each node belongs or indicate whether it is an endpoint. The division of these is intentionally left out. The nodes are allowed to overlap, and their labels are intentionally left out. One giant component and only a few small components exist. On the left side, one isolated component with only three process nodes and numerous endpoint nodes are shown. Endpoint nodes are repeatedly connected to process nodes from multiple environments.

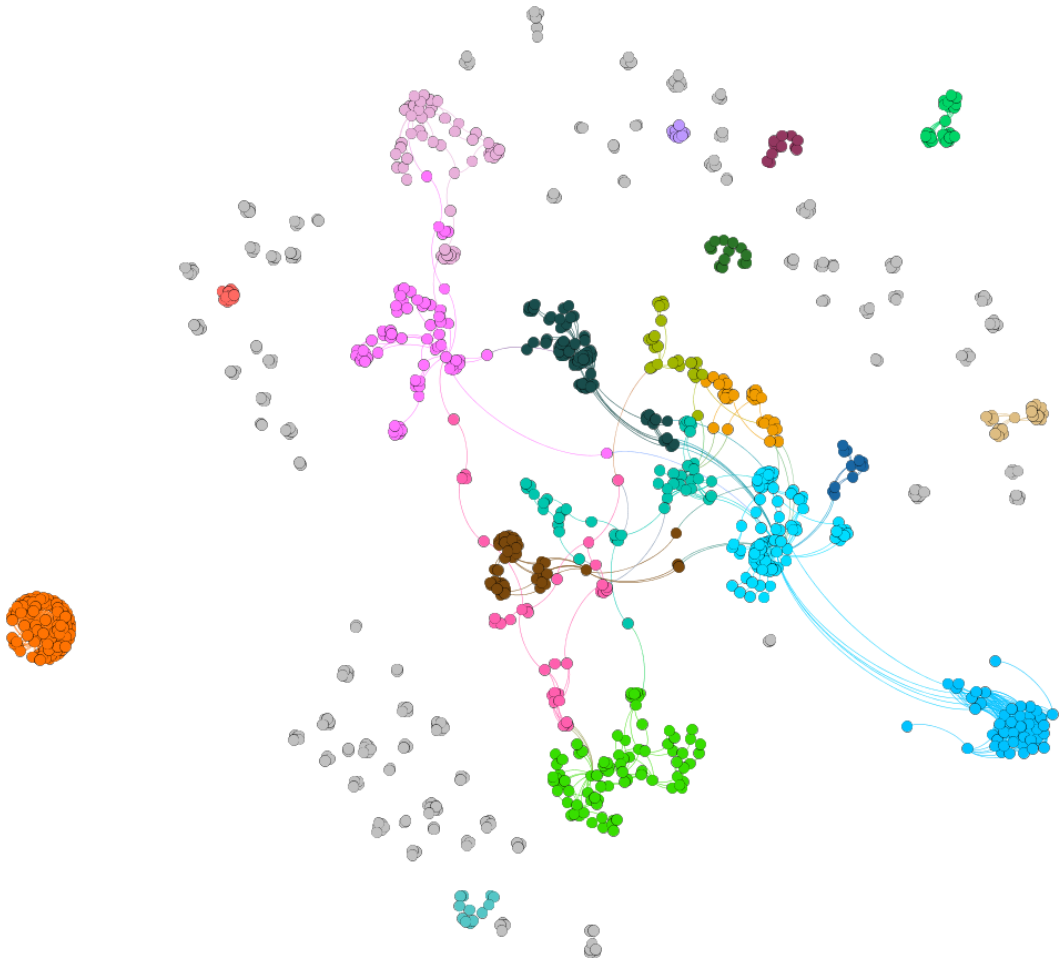

Figure 13: This is Figure 12, but the colors are assigned to the 30 largest groups detected using modularity optimization. All nodes are in a force-directed layout. Nodes can overlap, and their labels are intentionally left out. Clearly modularity optimization can detect some structure from network, but detected groups did not corresponded to actual systems based on a manual inspection.

7.1 Multipartite Analysis

As discussed in Section 4.9, the network was divided into two parts. In the first part, all endpoint nodes were removed, and the neighbors of each removed node were connected to each other. In Figure 14, the endpoint nodes are preserved, and the process and subprocess nodes are removed in the unipartite projection. The structure is nearly absent in this network, as most nodes have only very simple connections to each other. Therefore it can not be used to detect systems as speculated in the Section 4.9.

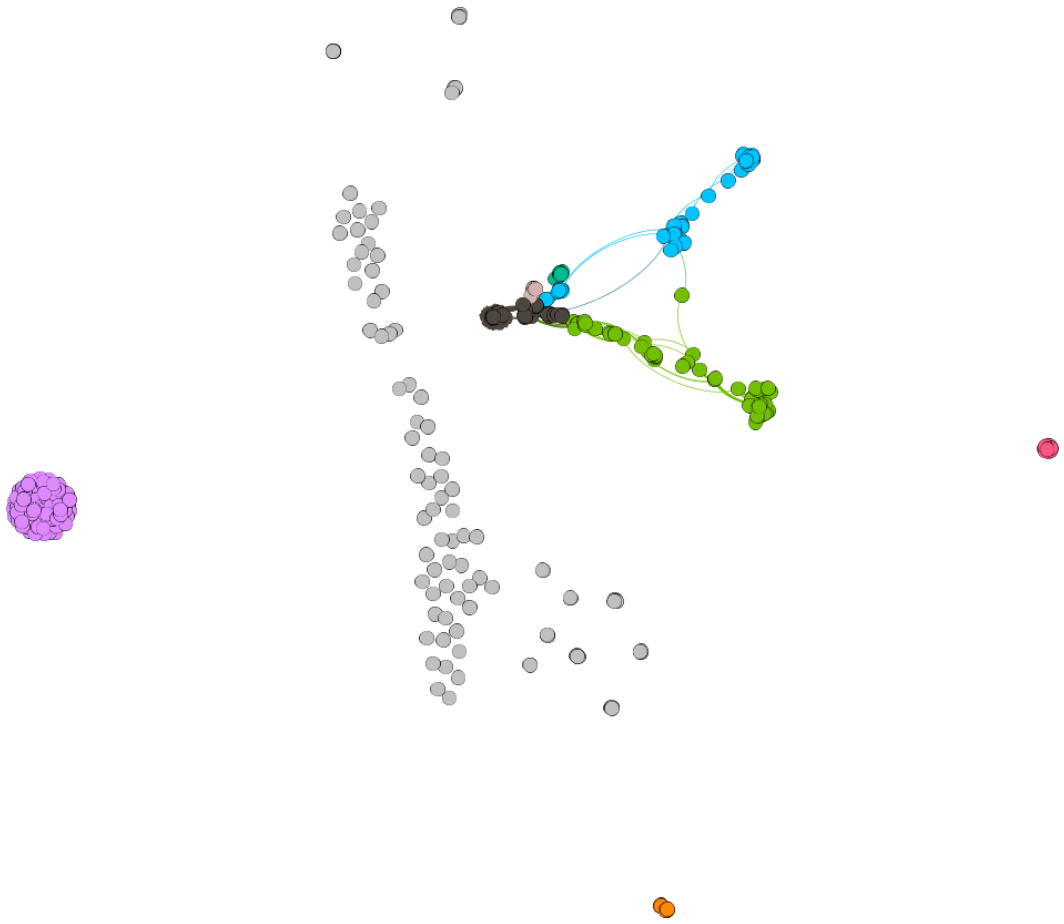

Figure 14: A unipartite projection of a network shows connections between endpoint nodes in a force-directed layout. A color is assigned to all six groups found using modularity optimization. Nodes are allowed to overlap, and most nodes are collapsed to a few very tight groups. The node labels are intentionally left out. Clearly, this projection lacks most of the structure that was visible in the Figure 12.

Figure 15 shows a unipartite projection, where process and subprocess nodes are preserved, and endpoint nodes are removed, and the neighbors of each removed node were connected to each other. The colors represent the environment where each node belongs. This network has a much clearer structure than the structure in Figure 14. However, it shows only how processes and subprocess are related to each other. Therefore, it can be used to group processes and subprocesses together. The original network would also provide this information easily. Clearly, nodes from different environments have multiple connections.

The degree and component distribution were also done with unipartite networks, but they did not provide any additional information. Therefore, in its entirety, the multipartite analysis did not provide new information.

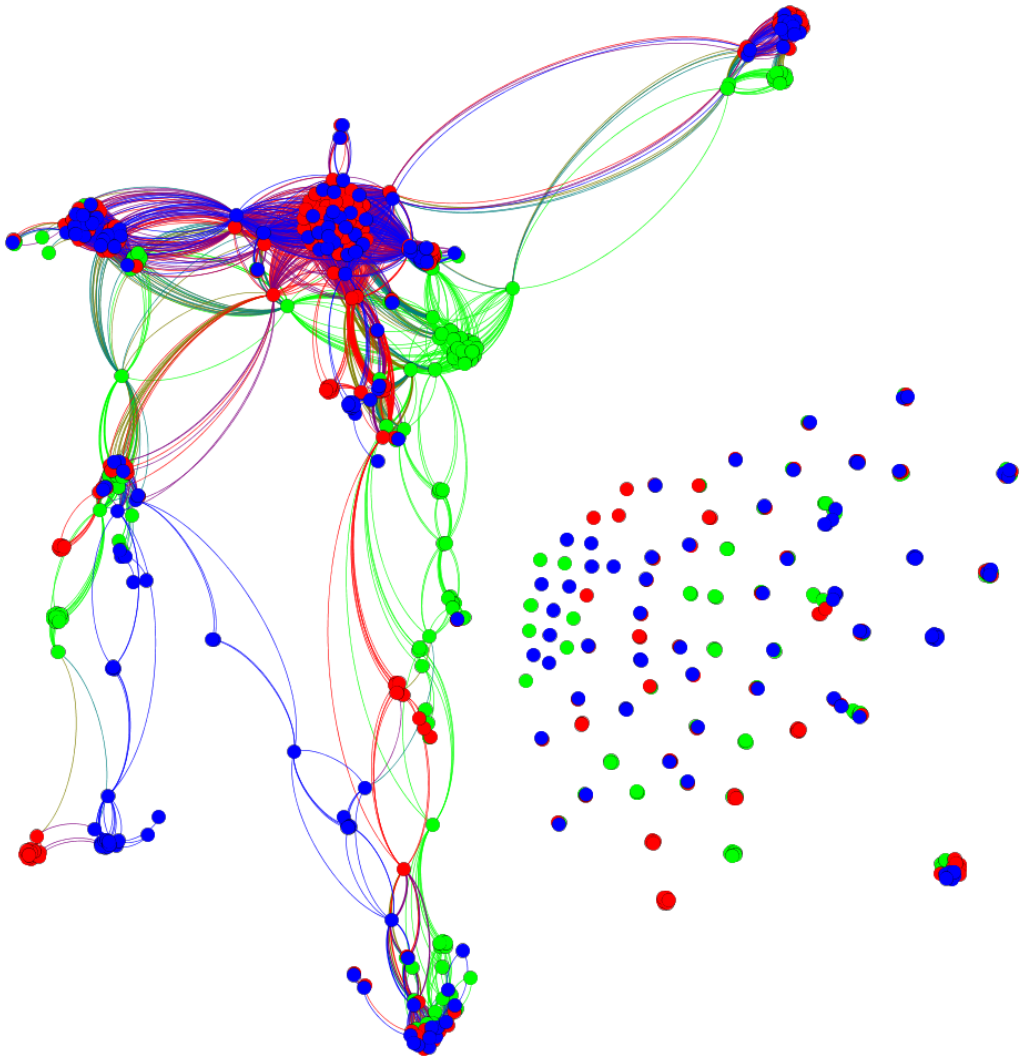

Figure 15: A unipartite projection of a network shows connections between process and subprocess nodes in a force-directed layout. The colors are the same as in Figure 12. The nodes are allowed to overlap, and their labels are intentionally left out. Clearly this projection have much richer structure than that visible in Figure 14.

7.2 Degree and Component Distributions

The degree of the node is the number of links the node has. The degree distribution is the probability distribution of different degrees that nodes have in a network. It is well known that, in scale-free networks, the degree distribution follows a power law. Thus, when plotted on a logarithmic scale, the degree distribution forms a line if the network is scale-free. Scale-free networks are discussed in more detail in section 4.10.

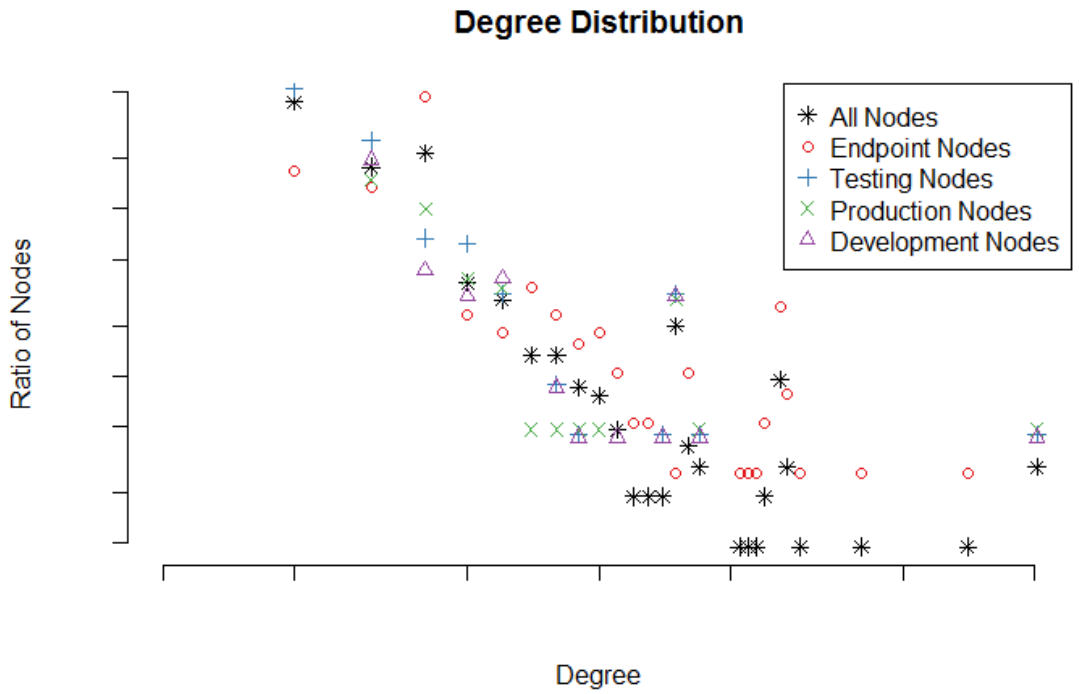

Figure 16: The degree distribution of every node in the network is shown. The axes are in logarithmic scale, and their labels are intentionally left out. The degree distribution for all nodes is shown with stars. Other points show the degree distribution for a subset of nodes (e.g., for the production environment). There seems to be a clear trend, but the points do not form a line; thus, the degree distribution does not follow a power law. In general, it seems that endpoint nodes have a higher ratio of high degree nodes than other nodes. There is no clear difference between development, testing, and production nodes.

In Figure 16, the degree distribution is presented for the whole network and each environment. The points do not form a line; that is a sign that degree distribution does not follow a line. The Kolmogorov–Smirnov test is utilized to test if degree distribution follows a power-law. Table 1 summarizes the results. It shows that the degree distribution does not follow a power law; thus, the network is not scale-free. Smaller values of D and p in the table indicates better fit [58] [59] [60]. However, there are multiple nodes with a high degree that act as a hub in the network. The hubs are over-represented compared to a power-law: the network is more long-tailed

(i.e., it has more high degree nodes) than a scale-free network. Manual inspection revealed that, for example, the backup location was among the nodes with highest degrees, giving a natural explanation to the high amount of links acquired by some nodes.

Exponent	D	p
1.570712	0.15194	0.924982

Table 1: The Kolmogorov–Smirnov test results on the degree distribution show that the estimated exponent for a power law is roughly 1.57. The relatively high value (> 0.05) on D and the very high p-value ($\gg 0.05$) indicate that the degree distribution does not follow a power law.

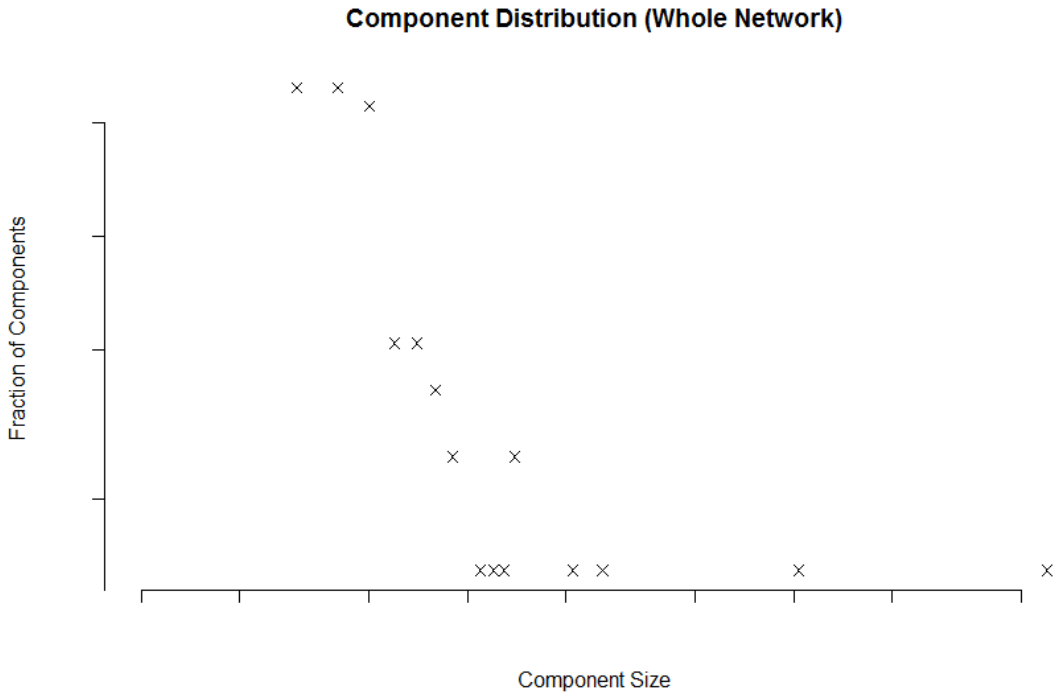

Figure 17: The component size distribution is shown. The axes are in logarithmic scale, and their labels are intentionally left out. The figure reveals that there is one slightly larger component and another giant component. The second largest component is visible in Figure 12 as an artefact.

8 Summary

The primary purpose of this thesis "Enterprise Application Integration Architecture Recovery" is to develop software that can recover the integration architecture from an implemented integration orchestration. The method is tested using real data from one company. In addition, architecture is presented as a network and briefly analyzed to detect general properties regarding network structure and isolation of production environment in implemented integrations.

The thesis is composed of eight chapters, each of them dealing with a different aspect of developing the necessary methods and software. Chapter 1 provides an introduction to the topic and describes the background of the problem: why companies do not always know what computer systems they are using. In addition, the chapter explains why this information can be regained by recovering the architecture of integration orchestration. Furthermore, it defines the scope of the thesis and reviews the thesis structure. Chapter 2 presents a literature review on integrations and software architecture recovery as well as how hybrid cloud computing will affect integrations in the future. Chapter 3 discusses Enterprise Application Integration platform FRENDS that was used to implement the orchestration layer of integrations and how characteristics of FRENDS affected the recovery process. Chapter 4 provides an outline of relevant expected properties of the recovered architecture and how standard network analyzing techniques are exploited to investigate the resulting network.

Chapter 5 presents a general methodology needed to implement architecture recovery from the orchestration layer. The methodology consists of searching the connections from implemented integrations with pattern matching, clustering connections with hierarchical clustering, and filtering the data. The methodology can be used to implement orchestration architecture recovery on any platform. Chapter 6 expands used methodology with details of implemented Power Bi model.

Chapter 7 concentrates on problems resulting from the network produced from real life data that was used to test the method. Conclusions are drawn in Chapter 8.

The main aim of the thesis has been reached since the software that can recover the integration architecture from an implemented integration orchestration was developed. This work used the Power BI to build the software, but it turned out that it was quite incapable of handling polymorphic data; therefore, the author suggests that in future similar works should use programming languages, such as an R or a Python, to implement the software. The author suggests that the developed methods should be applied to built-in architecture recovery capabilities directly in FRENDS.

References

[1] HiQ Finland Oy, *Frends | hybrid integration platform*. [Online]. Available: https://frends.com (visited on 02/08/2018).

[2] X. Song, X. Wang, and X. Liu, "Research on enterprise application integration architecture and development approach", in *2008 International Symposium on Intelligent Information Technology Application Workshops*, Dec. 2008, pp. 215–218. DOI: 10.1109/IITA.Workshops.2008.243.

[3] C. Finkelstein, *Enterprise architecture for integration : rapid delivery methods and technologies*. Boston: Artech House, 2006, ISBN: 978-1580537131.

[4] F. Brooks, *The mythical man-month : essays on software engineering*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1995, ISBN: 978-0201835953.

[5] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is so hard", *IEEE Software*, vol. 12, no. 6, pp. 17–26, Nov. 1995, ISSN: 0740-7459. DOI: 10.1109/52.469757.

[6] ——, "Architectural mismatch: Why reuse is still so hard", *IEEE Software*, vol. 26, no. 4, pp. 66–69, Jul. 2009, ISSN: 0740-7459. DOI: 10.1109/MS.2009.86.

[7] P. Aiken, *Data reverse engineering : slaying the legacy dragon*. New York: McGraw-Hill, 1996, ISBN: 0-07-000748-9.

[8] J. Estublier, "Software configuration management: A roadmap", in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00, Limerick, Ireland: ACM, 2000, pp. 279–289, ISBN: 1-58113-253-0. DOI: 10.1145/336512.336576.

[9] R. Land and I. Crnkovic, "Software systems integration and architectural analysis - a case study", in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, Sep. 2003, pp. 338–347. DOI: 10.1109/ICSM.2003.1235441.

[10] G. Hohpe, *Enterprise integration patterns : designing, building, and deploying messaging solutions*. Boston: Addison-Wesley, 2004, ISBN: 978-0321200686.

[11] O. Zimmermann, C. Pautasso, G. Hohpe, and B. Woolf, "A decade of enterprise integration patterns: A conversation with the authors", *IEEE Software*, vol. 33, no. 1, pp. 13–19, Jan. 2016, ISSN: 0740-7459. DOI: 10.1109/MS.2016.11.

[12] Microsoft Corporation, *Cloud computing terms | microsoft azure*. [Online]. Available: https://azure.microsoft.com/en-us/overview/cloud-computing-dictionary/ (visited on 09/20/2018).

[13] ——, *What is hybrid cloud computing - definition | microsoft azure*. [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-hybrid-cloud-computing/ (visited on 09/20/2018).

[14] W. Kim, S. D. Kim, E. Lee, and S. Lee, "Adoption issues for cloud computing", in *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM '09, Kuala Lumpur, Malaysia: ACM, 2009, pp. 2–5, ISBN: 978-1-60558-659-5. DOI: 10.1145/1821748.1821751.

[15] MuleSoft, Inc., *Understanding enterprise application integration - the benefits of esb for eai mulesoft*. [Online]. Available: https://www.mulesoft.com/resources/esb/enterprise-application-integration-eai-and-esb#esb-advantages (visited on 09/20/2018).

[16] Carol Hildebrand, *The top five roadblocks to cloud integration*. [Online]. Available: https://www.oracle.com/corporate/features/cloud-integration-challenges/index.html (visited on 09/20/2018).

[17] R. Mungrah and Z. Cadersaib, "Cloud application integration methodology using enterprise application integration", in *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, Dec. 2017, pp. 327–333. DOI: 10.1109/ICTUS.2017.8286027.

[18] X. Jin, "Research on the model of enterprise application integration with web services", in *Proceedings of the 3rd WSEAS International Conference on Computer Engineering and Applications*, ser. CEA'09, ISBM number is invalid, but still actually used by conference., Ningbo, China: World Scientific, Engineering Academy, and Society (WSEAS), 2009, pp. 105–109, ISBN: 978-960-474-41-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1519432.1519450.

[19] R. Terra, M. T. Valente, K. Czarnecki, and R. S. Bigonha, "Recommending refactorings to reverse software architecture erosion", in *2012 16th European Conference on Software Maintenance and Reengineering*, Mar. 2012, pp. 335–340. DOI: 10.1109/CSMR.2012.40.

[20] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture", *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992, ISSN: 0163-5948. DOI: 10.1145/141874.141884.

[21] G. Rasool and N. Asif, "Software architecture recovery", *International Journal of Computer, Information, and Systems Science, and Engineering*, vol. 1, no. 3, 2007. [Online]. Available: http://scholar.waset.org/1307-6892/10758 (visited on 08/31/2018).

[22] R. L. Krikhaar, "Reverse architecting approach for complex systems", in *1997 Proceedings International Conference on Software Maintenance*, Oct. 1997, pp. 4–11. DOI: 10.1109/ICSM.1997.624225.

[23] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy", *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573–591, Jul. 2009, ISSN: 0098-5589. DOI: 10.1109/TSE.2009.19.

[24] K. Sartipi, "Software architecture recovery based on pattern matching", in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, Sep. 2003, pp. 293–296. DOI: 10.1109/ICSM.2003.1235434.

[25] O. Maqbool and H. Babri, "Hierarchical clustering for software architecture recovery", *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, Nov. 2007, ISSN: 0098-5589. DOI: 10.1109/TSE.2007.70732.

[26] Y. Wang, P. Liu, H. Guo, H. Li, and X. Chen, "Improved hierarchical clustering algorithm for software architecture recovery", in *2010 International Conference on Intelligent Computing and Cognitive Informatics*, Jun. 2010, pp. 247–250. DOI: 10.1109/ICICCI.2010.45.

[27] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques", in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13, Silicon Valley, CA, USA: IEEE Press, 2013, pp. 486–496, ISBN: 978-1-4799-0215-6. DOI: 10.1109/ASE.2013.6693106.

[28] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic, and Y. Cai, "Enhancing architectural recovery using concerns", in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, Nov. 2011, pp. 552–555. DOI: 10.1109/ASE.2011.6100123.

[29] V. Tzerpos and R. C. Holt, "Acdc: An algorithm for comprehension-driven clustering", in *Proceedings Seventh Working Conference on Reverse Engineering*, Nov. 2000, pp. 258–267. DOI: 10.1109/WCRE.2000.891477.

[30] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidović, and R. Kroeger, "Comparing software architecture recovery techniques using accurate dependencies", in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15, Florence, Italy: IEEE Press, 2015, pp. 69–78. [Online]. Available: http://dl.acm.org/citation.cfm?id=2819009.2819022.

[31] HiQ Finland, *Integration platform documentation and user guides | frends docs*. [Online]. Available: https://frends.com/docs (visited on 02/08/2018).

[32] M. Weisberger. (Oct. 13, 2016). The bizarre history of 'tetris', Live Science, [Online]. Available: https://www.livescience.com/56481-strange-history-of-tetris.html (visited on 03/08/2018).

[33] HiQ International, *Hiq financial information*, 2018. [Online]. Available: https://www.hiq.se/en/investor-relations/financial-information/ (visited on 10/26/2018).

[34] Microsoft Corporation, *Nuget documentation | microsoft docs*. [Online]. Available: https://docs.microsoft.com/en-us/nuget/ (visited on 02/08/2018).

[35] *Community hiq*, Open source project, GitHub, 2018. [Online]. Available: https://github.com/CommunityHiQ/ (visited on 10/26/2018).

[36] HiQ Finland, *Frends tasks*, GitHub, 2018. [Online]. Available: https://github.com/FrendsPlatform (visited on 10/26/2018).

[37] N. Josuttis, *SOA in practice: The Art of Distributed System Design*. Sebastopol, Calif: O'Reilly Media, Inc., 2007, ISBN: 978-0596529550. [Online]. Available: http://www.soa-in-practice.com/.

[38] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture", *IEEE Software*, vol. 33, no. 3, pp. 42–52, May 2016, ISSN: 0740-7459. DOI: 10.1109/MS.2016.64.

[39] E. Abbe and C. Sandon, "Community detection in general stochastic block models: Fundamental limits and efficient algorithms for recovery", in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, Oct. 2015, pp. 670–688. DOI: 10.1109/FOCS.2015.47.

[40] T. P. Peixoto, "The graph-tool python library", *figshare*, 2014. DOI: 10.6084/m9.figshare.1164194. [Online]. Available: http://figshare.com/articles/graph_tool/1164194 (visited on 09/10/2014).

[41] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks", *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, P10008, Oct. 2008. DOI: 10.1088/1742-5468/2008/10/p10008. [Online]. Available: https://doi.org/10.1088%2F1742-5468%2F2008%2F10%2Fp10008.

[42] M. Buchanan, *Nexus: Small Worlds and the Groundbreaking Science of Networks*. New York, W. W. Norton & Company, 2002, ISBN: 0-393-04153-0.

[43] Microsoft Corporation, *Connection string syntax | microsoft docs*. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-string-syntax (visited on 09/14/2018).

[44] Oracle Corporation, *Connecting to oracle database*. [Online]. Available: https://docs.oracle.com/cd/B28359_01/win.111/b28375/featConnecting.htm (visited on 09/14/2018).

[45] Microsoft Corporation, *Naming files, paths, and namespaces | microsoft docs*. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/FileIO/naming-a-file (visited on 09/13/2018).

[46] IEEE and The Open Group, *Definitions*, POSIX.1-2017 Standard. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap03.html#tag_03_271 (visited on 09/14/2018).

[47] P. Hoffman, "The telnet uri scheme", RFC Editor, RFC 4248, Oct. 2005. DOI: 10.17487/RFC4248.

[48] Microsoft Corporation, *Handling uniform resource locators | microsoft docs*. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/WinInet/handling-uniform-resource-locators (visited on 09/14/2018).

[49] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform resource locators (url)", RFC Editor, RFC 1738, Dec. 1994. DOI: 10.17487/RFC1738.

[50] M. Kerwin, "The "file"uri scheme", RFC Editor, RFC 8089, Feb. 2017. DOI: 10.17487/RFC8089.

[51] P. Hazel, *Pcre - perl compatible regular expressions*. [Online]. Available: https://www.pcre.org/ (visited on 09/13/2018).

[52] *Perldoc.perl.org - official documentation for the perl programming language*, Sep. 22, 2017. [Online]. Available: http://perldoc.perl.org/perlop.html#Regexp-Quote-Like-Operators (visited on 09/13/2018).

[53] Microsoft Corporation, *Power bi | interactive data visualization bi tools*, comp. software, version 1.0.0.0, 2018. [Online]. Available: https://powerbi.microsoft.com/.

[54] ——, *Network navigator chart*, comp. software, version 2.0.2.0, Apr. 17, 2018. [Online]. Available: https://appsource.microsoft.com/en-us/product/power-bi-visuals/WA104380795.

[55] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2018. [Online]. Available: https://www.R-project.org/.

[56] A. Couture-Beil, *Rjson: Json for r*, R package version 0.2.15, 2014. [Online]. Available: https://CRAN.R-project.org/package=rjson.

[57] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks", 2009. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

[58] T. Nepusz and G. Csardi, *Fitting a power-law distribution function to discrete data*, Apr. 2019. [Online]. Available: https://igraph.org/r/doc/fit_power_law.html.

[59] M. Newman, "Power laws, pareto distributions and zipf's law", *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005. DOI: 10.1080/00107510500052444.

[60] A. Clauset, C. Shalizi, and M. Newman, "Power-law distributions in empirical data", *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009. DOI: 10.1137/070710111.