A case study on low-code integration platforms

Master of Science Thesis
University of Turku
Department of Computing
Software engineering
2023
Elmeri Kuismin

UNIVERSITY OF TURKU
Department of Computing

Elmeri Kuismin: A case study on low-code integration platforms

Master of Science Thesis, 81 p.
Software engineering
November 2023

The low-code approach has been gaining popularity among integration platforms. Low-code aims to make software development so intuitive that regular citizens could build applications. However, there is little research on what effects it has on integration development.
This paper consists of a literature review and a case study. The literature review aims to identify the areas of software that are affected by the low-code approach. Then a case study is conducted with 4 cases and 4 integration platforms to determine how low code affects the comparison areas.
The key advantages of low-code were that the platforms were easier to learn and use, and they improved development times. The disadvantages of low-code were that expanding their functionalities was more difficult and code produced with them could not be reused with other platforms.

Keywords: low-code, integration, integration platform, citizen developer

Contents

List of Figures

1 Introduction

Low-code integration platforms have been gaining popularity amongst integration platforms. Low-code platforms aim to simplify the software development process so that any regular citizen could develop their own programs. This study aims to clarify what advantages and disadvantages low-code brings to integration platforms.

1.1 Research question

This thesis is written in collaboration with the company Netum Oy. They are an IT consultant company, that provides various services from cybersecurity to integration services. The goal of this paper is to assist Netum's integration team by clarifying the effects that a low-code approach has on an integration platform.

The question that this paper aims to answer is as follows: How low-code integration platforms affect the development and maintenance of integrations? This question is so broad that it needs to be further split into comparison areas, which is done with a literature review. The comparison areas are:

- Learnability: how easy the platform is to learn for a new user.
- Extensibility: how easily the functionality of the platform can be extended.
- Functionality: how wide the base functionality of the platform is.
- Scalability: how well the platform scales to handle large amounts of data.

- Maintainability: how easy it is to keep the platform updated.
- Testability: how easy it is to test integrations with the platform.
- Development time: how long the building of the integration takes.
- Reusability: how much of the integrations can be reused in other platforms.

1.2 Methodology

This paper uses two different research methods to answer the research question. A literature review is used to cover the basics of low-code and integration, and to split the research question into smaller comparison areas. Then a case study is used to compare a low-code integration platforms with traditional integration platforms in the different comparison areas.

The main goal of the literature review is to split the research question into smaller pieces by finding the areas which are affected by the low code approach. A literature review was chosen because there is a sizeable amount of prior research on low code platforms, which can be used to identify the areas which need to be compared.

The case study is used to examine how low code integration platforms compare with more traditional integration platforms in the comparison areas found with the literature review. These comparison areas allow for a more detailed comparison of the platforms. A case study was chosen to utilize Netum's existing integrations and because it allows for a more flexibility in comparing the approaches. Netum has years of experience in creating integrations and using existing integration cases in the comparison helps us examine how the low code approach works in practice. As there are many differing areas that need to be compared, a flexible research method is needed.

1.3 Paper structure

This paper is divided into three sections. The first section contains chapters 2 and 3, which include the literature review on low-code and integration platforms. It also defines the comparison areas, which are used in the next sections.

The second section encompasses the case study. Chapter 4 introduces the integration platforms used and explains the cases used in the study. Chapter 5 gives a detailed explanation on how the cases were implemented with each of the platforms. Also the key findings for each platform based on the comparison areas are noted here.

The last section contains chapter 6, which compares the findings from the case study. The comparison areas are used to compare the platforms, and they are used to find the differences between the low-code and traditional platforms.

2 Low-code platforms

Low-code platforms (LCP) are software development platforms that utilize a low-code approach to minimize the programming experience needed for creating applications. [1] The goal of LCPs is to ease the lack of software developers by allowing "citizen developers" to build software with minimal training. They also aim to make software development more efficient by streamlining the process of writing, compiling and deploying software.

2.1 Model-driven development

Low-code platforms take heavy inspiration from model-driven development. [2] As the name suggests, model-driven development focuses on building models of the application, from which the actual program code is generated. [3] Models are closer to the problem domain than the underlying implementation, and thus focusing on them can be a more effective way of solving problems. In software development models are already used to represent most parts of the program, such as the structure of a database or the hierarchy of classes, so generating the application based on them can boost productivity. Model-driven development also ensures that the application and the models used for documentation stay in sync as a form of living documentation.

2.2 Key features

Key features of most LCPs are a GUI designer, a conceptual modeling tool, a way of accessing external data, some standard operations, and a way of deploying the application. [4] LCPs don't necessarily have any innovative new technology, but they combine many common technologies into a easy to use package.

The GUI designer and the conceptual modeling tool are at the core of LCPs. They are the tools that allow the developer to create user interfaces and program flows with minimal coding experience. These tools have a way of accessing external data, usually with APIs, which in combination with standard operations allows for the processing of data. Finally, the deployment of the program is automated, which can take many different forms. For example, the LCP can be installed on a web server where the applications are also deployed, or the LCP can produce self-contained applications to various devices.

The main productivity gains of LCPs stem from the reduction of the efforts of routine tasks. Traditional software development requires different tools for different parts of development, for example IDEs, modeling tools and database management systems. LCP integrates these systems together which reduces the need of switching between tools and helps to keep implementation artifacts consistent with each other.

2.3 Current challenges

While LCPs can bring a lot of benefits to software development, they still have their own unique challenges and limitations. The four main areas of limitations are: interoperability, extensibility, learning curve and scalability. [5] [6]

Interoperability means the ability to communicate with different types of systems. LCPs can do this effectively if the LCP supports the type of communication requested by the communication partner, but problems arise if it is not supported.

Then more traditional code needs to be developed to handle the communication, which hinders the effectiveness of the platform.

The same problem can also hinder the extensibility of LCPs. Extending the capabilities of the platform requires programming experience and knowledge on the inner workings of the platform, which defeats some of the advantages of an LCP.

While the learning curve for LCPs is definitely lower than for learning a programming language, there are issues caused by the lack of teaching materials. Also, LCP still require some knowledge on software development, which goes against the goal of making software development available to citizen developers.

There are also some problems with the scalability of the platforms. They should be run from the cloud, so that they can handle large amounts of data and traffic efficiently. This has been hard to verify and research because of the lack of open standards within LCPs.

Finally, there is a risk of a lock-in effect with LCPs. [1] The models created by a LCP are rarely if ever compatible with other LCPs. Because of this, if the support for an LCP is dropped, most work done with it is lost and the transition to another LCP is costly.

2.4 Comparison areas

This section will explain how the large research question is split into smaller comparison areas. It will also give more details on what those areas refer to in integration platforms.

The list of comparison areas was picked by selecting the attributes that were often noted in the literature. Each of these areas were are mentioned in multiple papers as being improved or hindered by low-code. The list of areas is as follows:

- Learnability

- Functionality
- Extensibility
- Scalability
- Maintainability
- Testability
- Development time
- Reusability

2.4.1 Learnability

Learnability can be defined as: : “The system should be easy to learn by the class of users for whom it is intended”. [7] Learnability is important for low code platforms, as they aim to get non-programmers to develop applications. [4] [1] The term "Citizen developer" is often used in the papers discussing low code, and it means getting non-programmers to develop applications.

The evaluation of learnability is loosely based on a survey produced by Grossman et al. [7] The method presented by Grossman et al. splits learnability into two categories: initial learning and extended learning. For this thesis I will be focusing on the initial learning, as evaluating extended learning would grow the scope of the thesis too much.

Learnability is evaluated in two ways: how difficult learning to use the platform is in general and how much tutorial content is available for it. The integration platforms are selected so that I am familiar with one low code platform and one non low code platform. This split allows for some comparison of how difficult it was for me to learn to use the platform. Tutorial content is key for any development platform and low code platforms are no exception. [8] Comparing the amount and

quality of available tutorial content will give some measure on how easy it will be for citizen developers to use the platform.

2.4.2 Functionality

Functionality refers to all of the available tasks provided by the integration platform. Low code platforms need to provide prebuilt tasks for its functions in order to make it easy to learn and use. [9] Thus it is important for a low code platform to provide a comprehensive set of tasks for its use case.

Functionality will be assessed by counting the number of functions in the platform that don't require writing code. Additionally, it will be assessed case-by-case by evaluating how well the built-in functionalities cover the usecase.

2.4.3 Extensibility

Extensibility means how well the functionality of the platform can be expanded. As low code platforms cannot provide built-in functions for every use case, there has to be some way for the user to expand the functionality of the platform. [5] For example, some platforms allow the user to program their own tasks and add them to the platform.

Extensibility is evaluated in the use cases by assessing how easily any missing functionalities can be added to the platform. As it is not guaranteed that all platforms are missing the functionality of a usecase, it will also be evaluated with a general missing functionality.

2.4.4 Scalability

In this paper scalability refers to how well the integration platform can be scaled to handle an increasing amount of integration traffic. Scalability has many different dimensions, such as the scalability of the number of users or the scalability of the

problem size. [10] With integrations the most important scaling variable is the amount of integration traffic, so the evaluation of scalability will focus on assessing how well the integration platforms scale in that dimension.

As low code platforms are often built as SaaS platforms, they can naturally offer great scalability on the server level. [5] [9] At the integration level they can struggle as they need to provide some additional scaling functionality, such as threading.

Scalability will be evaluated on two levels: the server level and the integration level. The server level is evaluated by assessing how well the system can support multiple servers working together on the integrations. The integration level is evaluated by assessing how individual integrations can be scaled to handle large volumes of traffic.

2.4.5 Maintainability

Maintainability refers to how easy it is to keep the integration platform and the integrations built with it up to date. As low-code platforms are meant to be used by citizen developers, keeping them maintained has to be as accessible as possible [5].

Another important aspect of maintainability is how easy it is to understand the source code and the software. [11] As software has to be maintained for long periods of time, it is often the case that the people maintaining it change. Thus, the understandability of the integrations is an important aspect of maintainability.

Maintainability will be evaluated by assessing how much knowledge is needed to maintain the integration platform. This encompasses updates to the platform itself and the individual tasks in the integrations. Another evaluated aspect is the understandability of the integrations. It is evaluated by assessing how easy it is to figure out the steps of the integration based on the code of the integration.

2.4.6 Testability

Testability refers to how the testing of integrations is done with the platform. There is a lack of general testing tools for low code platforms, and thus the developer has to provide their own methods for testing. [6] [12]

Testability is evaluated by how the integrations are tested with the integration platform. Error messages are important in testing, and as low code platforms have to present them in their own way special attention is paid to them. Additionally, a look will be taken into third party tools if they are available.

2.4.7 Development time

Development time means the time needed for the development of integrations. It is often cited that the key advantage of low code platforms is the reduced development time. [9] [1] [13]

The development time of each case is tracked and compared between the platforms. There will also be an evaluation of the factors that contribute to the final development times.

2.4.8 Reusability

Reusability refers to how easily parts of integrations can be reused. Generally low code platforms allow for easy reusability of components inside of the platform, as they are already split into configurable tasks or functions. [9] [1] On the other hand each low code platform presents the model of the program in its own way, it is rare that a model can be exported from one platform to another. Thus, there is a risk of a lock-in effect, where a company is stuck using one platform because moving to another is difficult and time consuming.

I have split the assessment of reusability into two sections: inside of the platform and between platforms. First it is evaluated how much work can be repurposed be-

tween cases in the same platform. Then, it is evaluated how much of the integration can be reused if it was moved to a different platform.

3 Integration

Integration is a term that is often used when discussing enterprise applications. [14] Generally it refers to making applications work together through some sort of an interface. This section presents the common types of integration and explains what the word integration refers to in this paper.

Integration can be loosely split into three different types: big integration (big I), little integration (little I) and business to business (B2B) integration. Usually, integrations use a combination of these types as a single type is not efficient for complex integration needs.

3.1 Big I

Big I refers to a integrations type where all data is processed by a single software application and data is stored only once. [14] Updates in a single application component or module are spread throughout the whole application without external interfacing.

The main advantage of Big I is that there is always a single source of truth. As all data is stored only once, all modules can trust that the data they receive is correct. A second advantage is that there is no need for complex external interfaces across applications, as all data handling is done by the single application.

The downsides of Big I stem from the fact that all of the user's data handling and storing needs need to be done with a single application. As different users or

businesses have varying data processing needs, it is unlikely that a single application can satisfy all of them. Also, the internet has allowed more options for little I, so using only Big I has become less common.

3.2 Little I

Little i refers to integrating applications through some sort of interfaces. [14] Each application needs its own way of sharing data so that it can be integrated with other applications. For example, two often used interface types are REST APIs and file systems.

There are many ways of integrating applications with little i. For example, you can use a point-to-point approach or a database-to-database approach. The point-to-point approach integrates two interfaces together with some code. This code could be a middleware application, or it can be as simple as a batch script. The database-to-database aims to integrate the databases that the integrated applications use. Again, some code will be needed to transfer the data. For this paper the most important approach is the enterprise application integration approach, which will be covered in a later section.

The biggest advantage of little I is its flexibility. It allows the user to use a set of applications that meet their demands rather than a single large application. Also, it allows for new applications to be added to the set of integrated applications. With big I you would need to add the new applications functionality to the single big I application.

The disadvantages of little I come from the increased complexity of the integration. Each integrated application needs to have its own interface that allows for the transfer of data. Additionally, you need additional software to handle the traffic and logic of the integrations.

In practice neither big I or little I is by itself. Often, they are used together to

gain the advantages of both. In essence, little I is used to connect pockets of big I.

3.3 Integration platforms

Integration platforms are middleware applications that facilitate the transfer of data between other applications. [14] They use preprogrammed connectors to connect the different applications' interfaces together. In practice, this use of customizable connectors allows for any-to-any integrations. Usually, one connector is used as a source and another as a destination, and together they form an integration flow.

Another key feature of integration platforms is the handling of business logic. Most of the time the data cannot be transferred as is and some processing has to be done to it. Integration platforms allow code to be freely written into the integration flows. For example, this can be used to convert SQL query results into a .csv file, or to send an email if a transferred file is too small.

Integration platforms help to reduce some of the downsides of little i. The preprogrammed connectors speed up the integration process, as they are highly reusable. They also gather the integration logic into a single application, which eases the management of large-scale integrations.

4 Case Study

4.1 Integration platforms

Four integrations were selected to be used in the case study. Those platforms are: Mirth Connect, Frends, ArcESB and B2Bi. These platforms were chosen because they are used by Netum and they can be split into low-code platforms and low-level platforms. The low-code platforms are Frends and ArcESB, while the low-level ones are Mirth Connect and Apache Camel. The next subsections will give more detail on each platform.

4.1.1 Mirth connect

Mirth Connect (Mirth) is an open source integration platform developed mainly by NextGen healthcare. [15] As the company's name suggests, NextGen healthcare mainly offers software and services to the healthcare industry. Still their integration platform Mirth has been built to be flexible so that it can be used outside of that market.

Mirth is run on a single local computer. It is programmed with Java, but it uses the Rhino JavaScript engine to allow for the use of JavaScript in the integrations. JavaScript is the main language used in the programming of the integrations, but the Rhino engine allows for Java classes to be imported and used inside of the JavaScript code.

Mirth uses a common template for building integrations. Each integration has a source and a destination connector, with optional processing steps between them. Mirth offers a varied selection of connectors, and you can also use a block of code as a connector.

4.1.2 Frends

Frends is an integration platform as a service (iPaaS) developed by HiQ. [16] The main features of Frends consist of a low-code integration design GUI and a hybrid cloud infrastructure.

Frends is a hybrid cloud service, meaning that components of the service can be run in the Azure cloud or on premises as needed. The components of Frends can be split into two parts: the control server and the integration agents.

The central component of Frends is the Frends management portal, which is a React based web application running in the Azure cloud. It is used to build, monitor, and control the integrations.

The Frends integration agents are servers that are used to execute the integrations. These agents can be installed either on premises, or they can run in the Azure cloud. They receive the integrations and the orders to execute them from the Frends management portal.

Frends is programmed using C#, which is also the language used in the integrations. The integrations are built by drawing a graph with a GUI. The GUI supports most commonly used programming control structures, such as loops and if-conditions. The platform offers a wide variety of tasks, which can be used to perform common integration tasks, such as file transfers and API calls. C# code blocks can also be inserted into the graphs. A more detailed look into the Frends integration process is given in the case results.

4.1.3 ArcESB

ArcESB is an integration platform developed by CData Software.[17] It features a low-code integration designer and it is focused on business-to-business integrations. ArcESB can be installed locally to a server, or it can be installed to the cloud.

In ArcESB individual integrations are referred to as "flows". Each flow contains connectors, which handle tasks such as downloading files or mapping data from one format to another. A graphical integration designer is used to connect the connectors together, creating a flow where the data flows from one connector to the next.

ArcESB contains a large variety of connectors that can handle a large set of integration needs. As it is aimed at business-to-business integrations, it contains extensive support for data formats and protocols often used in that type of communication. Examples of those are EDI-messages and the AS2-protocol.

ArcESB contains script connectors, which allow custom scripts to be run in the integration flows. [1] ArcESB does not use a pre-existing scripting language, rather it uses its own: ArcScript. ArcScript contains a sizeable number of operations, but it can be further extended with custom Java or C# code.

4.1.4 Apache Camel

Camel is an open-source integration platform developed by Apache. [18] Compared to the other integration platforms Camel is rather low-level, as the integrations are written entirely with Java-code. This allows for extensive customization of the platform, while requiring more effort to set up the platform and build integrations with it.

Individual Camel integrations are called routes. Routes are run within a Camel context, which defines the environment for the integrations. The Camel context can then be deployed as a standalone Java-application, or it can be run inside other

[1] https://cdn.cdata.com/help/AZJ/mft/Scripting.html

frameworks or containers, such as Spring Boot or Kubernetes.

Each Camel route consists of components and processors. Components are used to connect Camel to other systems. For example, the SQL-component can be used to connect to a database and the SFTP-component can be used to download files from a server. Processors are used to handle messages between components. For example, they can be used to transform, validate and route a message. Camel contains dedicated processors for common tasks, but they can also be built using custom Java-code.

4.2 Cases

These cases are picked from integrations built by Netum. Netum has years of experience in building integrations, and I wanted to use that experience in the case study. All of the cases are currently in use in one or more Netum integration projects. All information that could be used to indentify a customer of Netum is removed from the cases.

The cases were picked by balancing the frequency of the integration type with the complexity of it. The first two cases are commonly used, but they are not very complex. The last two integrations are more complex, but they are rarely used. This spread of cases illustrates how well low code integration platforms handle both simple and complex integrations.

4.2.1 Case 1: SFTP transfer and backup

The first integration consists of three steps:

1. Download a file from the source directory to the integration platform with SFTP.

2. Backup the file with the integration platform.

3. Upload the file to the destination directory with SFTP.

While this integration is a simple one, it is one of the most used ones. It provides an example of how a company can transfer files automatically between servers and create backups in the process.

At least one of these steps in this case is used in most more complex integrations. A file system is a common interface for applications, and SFTP is a simple and secure way of transferring those files. Additionally, backups are often needed to increase the reliability of the system. Thus, this case covers the most basic integrations.

For this case all required servers will run on the same Windows machine. The SFTP server will be built using OpenSSH.

4.2.2 Case 2: REST GET and CSV transform

This integration consists of 3 steps:

1. Use a REST GET call to fetch data from a server.

2. Transform the fetched data into a CSV file.

3. Move the file into a local destination folder.

This integration adds a little complexity compared to the first one, while still being fairly simple and common. It provides an example on how a company could fetch data from an application's REST API and transform it into a format that can be used by another application.

REST APIs are another common interface for applications, especially with SaaS-platforms. The data returned from the API is in JSON format, and a transform is performed on it. CSV is a commonly used format for inputting data into applications, so the JSON data is transformed into a CSV file. This file is then moved to a local folder.

For this case the REST request will be made to a external server, while the integration platform and the filesystem will be ran on the same windows machine. The rest request will be made to: https://httpbin.org/json, which returns some JSON format data.

4.2.3 Case 3: Send a EDIFACT message using the AS2 protocol

The third integration shows how an invoice could be transferred between businesses. It consists of 3 steps:

1. Use an SQL query to fetch data for an invoice.
2. Transform the data into the EDIFACT Invoic format.
3. Send the invoice using the AS2 protocol

This integration takes a notable step up in its complexity. This case provides a example on how two companies could exchange invoices securely. SQL databases are another type of interface used by applications, so making SQL queries is a standard feature for integration platforms. The data returned from the query won't match a datatype that is suitable for the transfer of data between companies, so it has to be transformed.

United Nations/Electronic Data Interchange for Administration, Commerce and Transport (UN/EDIFACT) is an often-used standard for electronic data interchange (EDI). [2] It was developed by the United Nations and it was published by the UN Economic Commission for Europe. A EDIFACT message consists of rows which have a segment code and data fields specific to that segment. For example, the row: "DTM+137:20020308:102'" has the segment code DTM, which stands for a

[2]https://unece.org/trade/uncefact/introducing-unedifact

date and/or time. This is then followed by a field containing the code 137, which specifies a date for the document. It is followed by the date/time and a code for the format of the given date/time. Essentially the integration platform needs to map the data from the SQL query into this specific format.

The invoice is sent using the AS2 protocol. It was developed in 2002 by the IETF to replace the older AS1 protocol. [19] It is largely based on HTTP and S/MIME with additional security features such as signing, encryption and return receipts. These transfers also require the trading partners to first exchange X.509 certificates and matching trading partner names for secure transfers. After a successful transfer a Message Disposition Notification (MDN) can be sent to confirm that the message was received successfully. AS2 can send messages synchronously, meaning that the sender waits for the receiver to send the MDN message using the same HTTP connection.

For this case, all of the required servers will run on the same Windows machine. MySQL will be used for the SQL database. The AS2 message will be received using an open source AS2 server "as2-lib" [3]

4.2.4 Case 4: Transform Finvoice to a PDF invoice

The fourth and final case is an example of how a company could serve a PDF invoice for their customers. It consists of 3 steps:

1. Read the Finvoice file from a local directory.

2. Transform the Finvoice into and HTML page.

3. Save the HTML page as a PDF file to a SFTP server.

The last case has a similar level of complexity with the third case. In this case

[3]https://github.com/phax/as2-lib

the transfer of the files is simple, and the complexity arises from the transformation of the Finvoice file.

Finvoice is a e-invoicing standard developed by a group of Finnish banks. [20] It is commonly used by Finnish banks and companies as a format for electronic business messages. Finvoice messages use XML for their structure, with a specific schema dictating the tags and information needed in the message. In this case the Finvoice message is an invoice for a purchased product.

While the Finvoice standard provides a format for transfering data, it is not a format that is easily readable. Companies still need to send readable invoices as PDF files, so the Finvoice message needs to be transformed into a formatted PDF file. For this purpose, the Finvoice standard provides an XSL stylesheet that can be used in a XSLT transform to transform the message into an HTML page. This HTML page can then be transformed into a PDF file.

For this case all of the required servers run on the same windows machine. The source file is located inside of the Mirth server and the destination is the same SFTP server used in the other cases.

5 Implementation

This chapter gives a detailed explanation on how the cases presented in chapter 4.2 were implemented with each of the integration platforms. Each platform has its own chapter, with subchapters for each case. Finally, each platform has a subchapter for discussing the findings from the cases and how they relate to the low-code comparison areas.

5.1 Mirth

5.1.1 Case 1

Case 1 consists of 3 steps:

1. Download a file from the source directory to the integration platform with SFTP.
2. Backup the file with the integration platform.
3. Upload the file to the destination directory with SFTP.

As the case 1 integrations is simple, all of the required steps can be done with a single Mirth channel. Also, as the steps consist of mainly SFTP actions, they can be handled with pre-built connectors.

The download step is handled in the channel's source connector. All source connectors allow for the scheduling of the channel, which can be seen at the top of

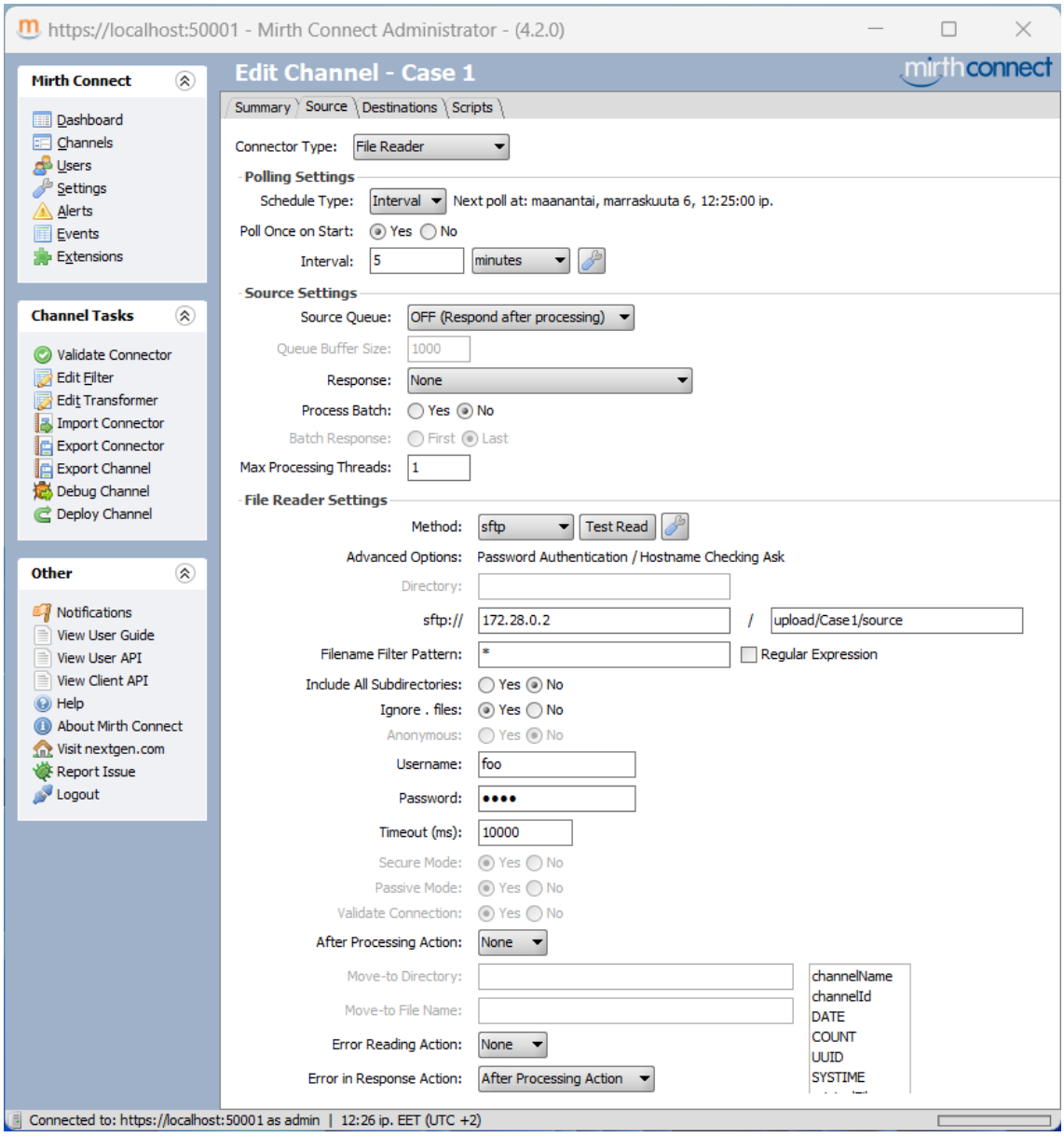

Figure 5.1: Mirth case 1 SFTP transfer source connector

Figure 5.1. For this case, the channel is set to run once it is deployed and then at a 5-minute interval.

The connector uses the "File reader" connector type, which allows for the downloading of files via SFTP. The configuration for this connector is simple, as the main information needed is the address and authentication details of the source SFTP server. For this case the SFTP server is using the IP-address of "172.28.0.2" and the path to the source folder is "upload/Case1/source". This connector reads all files in the source folder and sends them to the destination connector one-by-one.

The backup of the file is also done in the source connector. The "After Processing Action" allows for the moving of the downloaded file to a backup destination. Unfortunately, Mirth does not have a dedicated way of handling backups, so another channel is needed to delete files from the backup folder when they are too old.

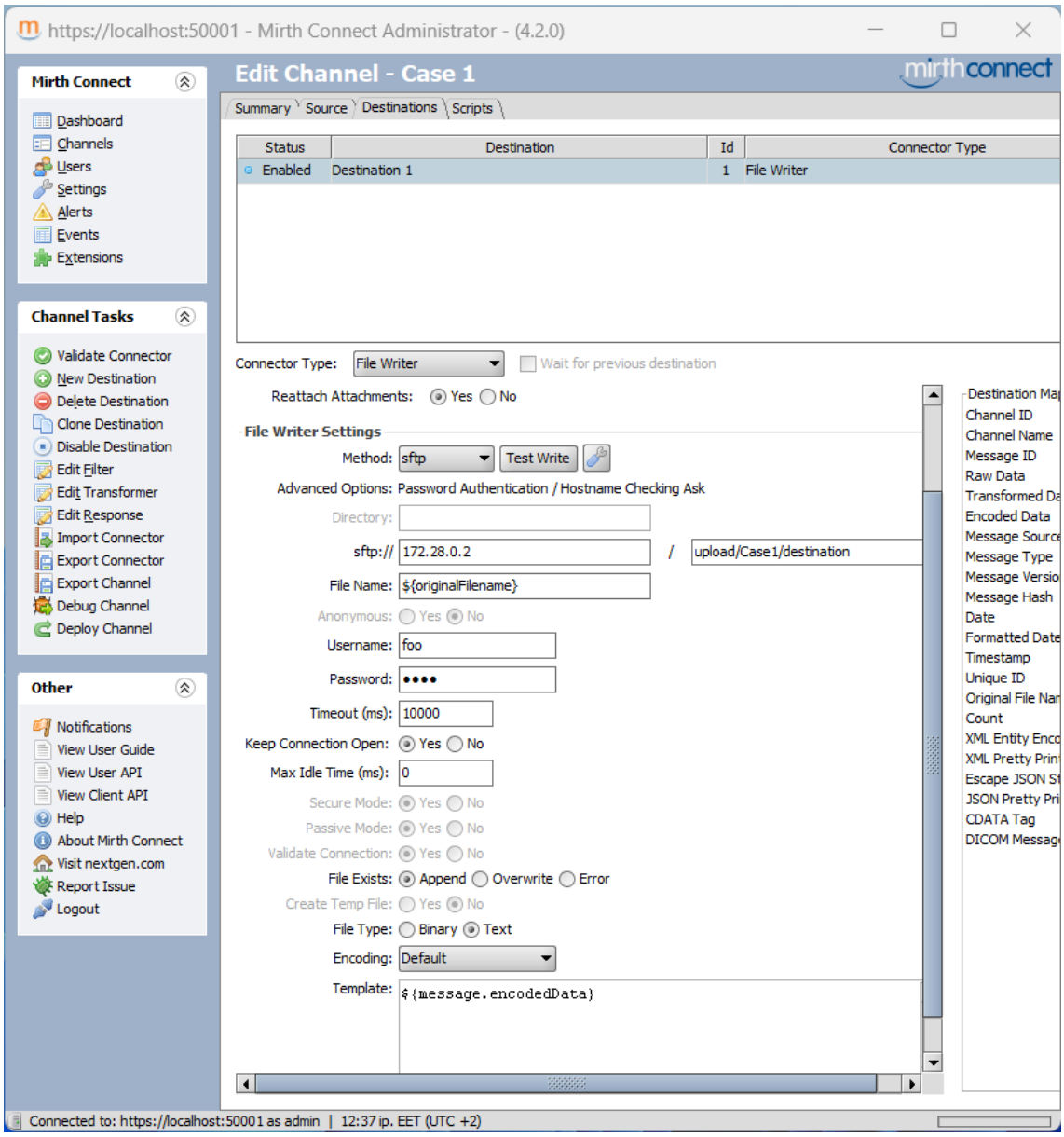

Figure 5.2: Mirth case 1 SFTP transfer destination connector

The destination connector handles the last step of the integration. Mirth supports multiple destination in each channel, which can be used to handle situations where the message needs to be routed to multiple places in multiple ways. In this case only one SFTP destination is needed.

The destination uses the "File writer" connector type, which is almost identical

to the "File reader" source connector. The main differences are that the writer has options for the name and contents of the file to be written. In this case the original file name and content is used, which are accessed from environment variables "originalFilename" and "message.encodedData". Mirth provides shortcuts to these common variables, which can be seen at the right side of Figure 5.2.

5.1.2 Case 2

Case 2 consists of 3 steps:

1. Use a REST GET call to fetch data from a server.
2. Transform the fetched data into a CSV file.
3. Move the file into a local destination folder.

The implementation of this case had to be split into two channels: "Get JSON" and "Map JSON to CSV". This split was made because a REST call can only be made from a destination connector, the result of which can then be used in another channel.

The integration flow starts from the channel "Map JSON to CSV" shown in Figure 5.3. This channel uses a JavaScript connector to launch the "Get JSON" channel, which returns a JSON object from a server.

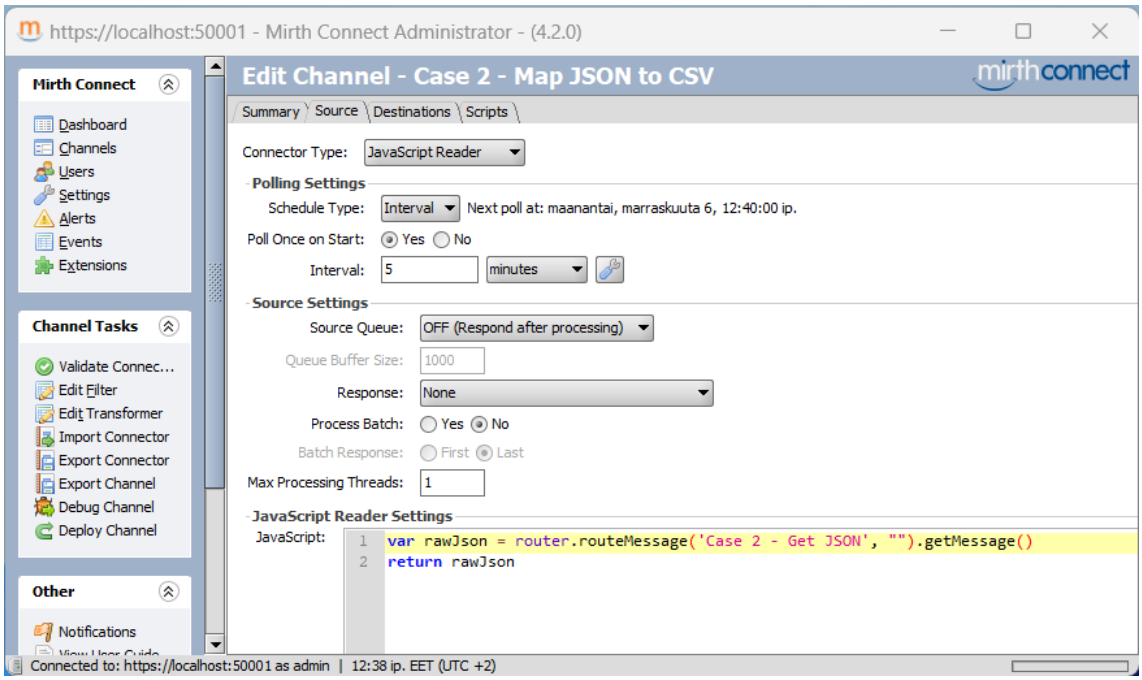

Figure 5.3: Mirth case 2 JSON to CSV source connector

The "Get JSON" channel uses a source connector of the type "channel connector", which allows for other channels to activate it. This connector is shown in Figure 5.4. The most important configuration for it is the "response" option. It is used to configure which destination's response is returned to the callee channel.

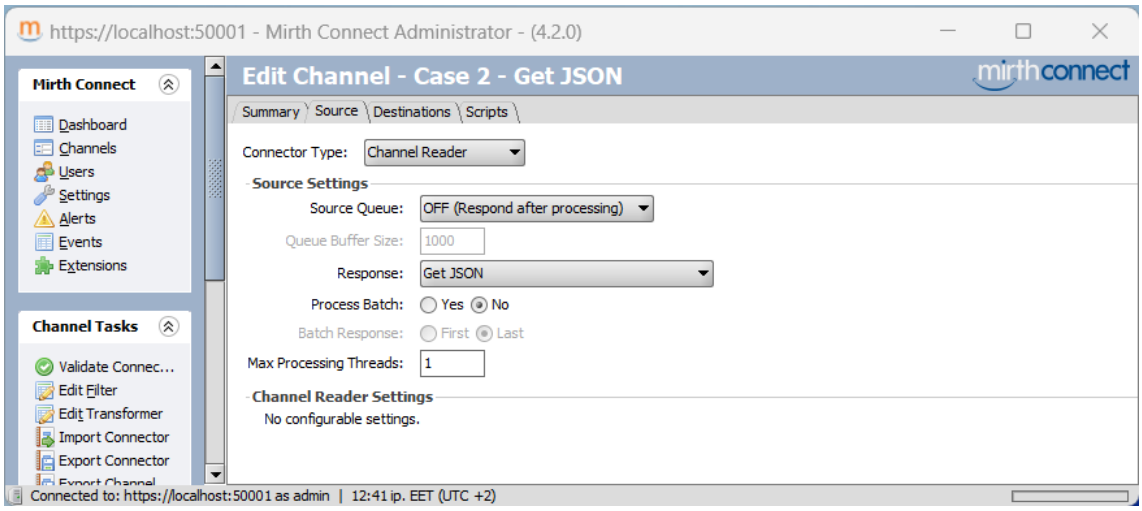

Figure 5.4: Mirth case 2 GET JSON source connector

The "Get JSON" channel uses a "HTTP Sender" destination connector to send a GET request to a server, as shown in Figure 5.5 In this case the request is sent to the address "http://httpbin.org/json", which returns a JSON object. The connector can be configured to send additional information such as headers or query parameters, but they are not needed in this case.

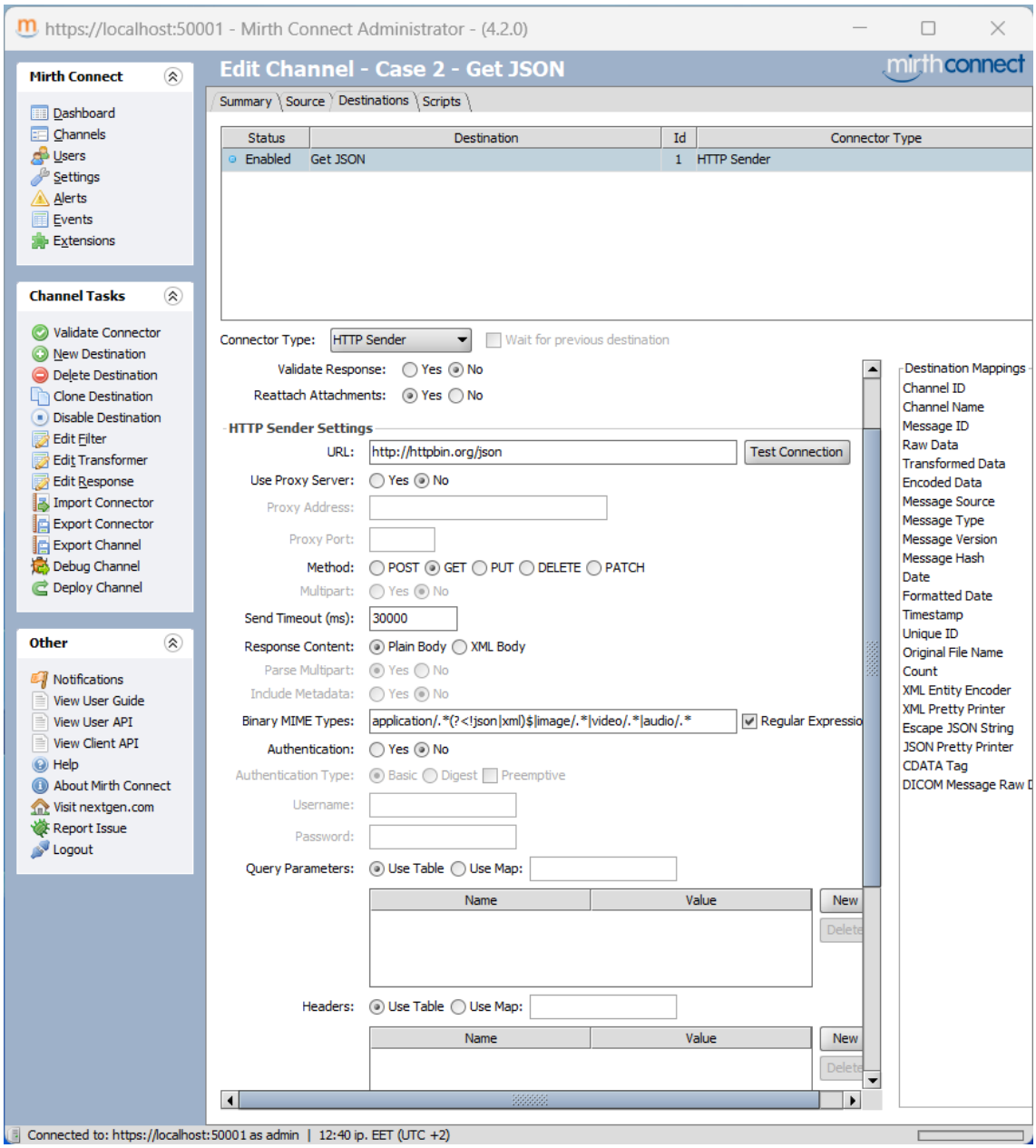

Figure 5.5: Mirth case 2 GET JSON destination connector

After the "Map JSON to CSV" channel has received the JSON response, it maps into a CSV format in the destination connectors transformer, as seen in Figure 5.6. Each destination can use a transformer step to run transform the data received from the source connector. In this case we are using a JavaScript transformer case to loop

through the received JSON and form a CSV string from it. This data is then written to the SFTP server using the "File Writer" destination connector, which is the same as in Case 1.

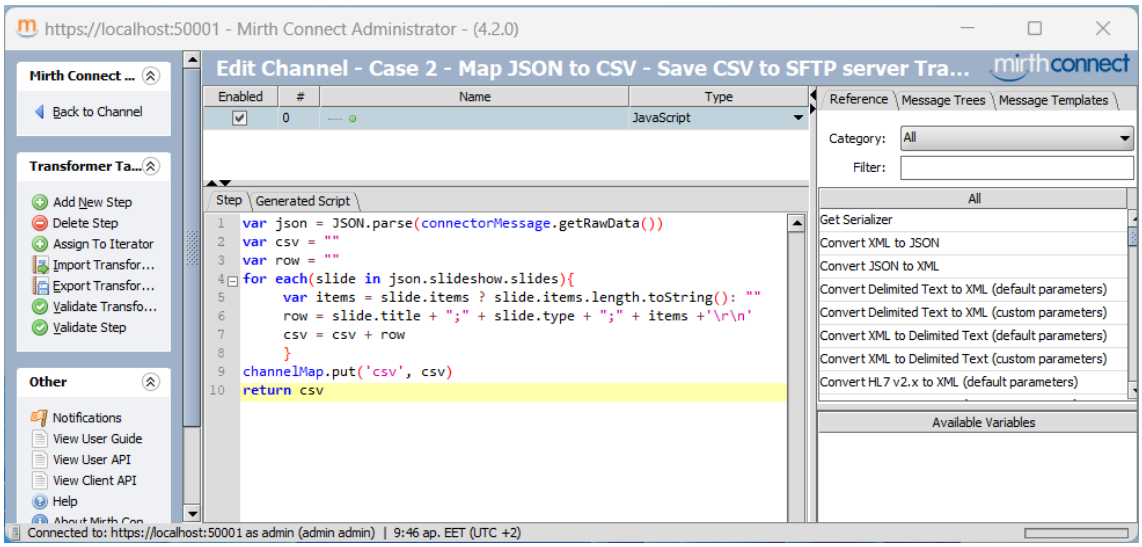

Figure 5.6: Mirth case 2 JSON to CSV transformer

5.1.3 Case 3

Case 3 consists of 3 steps:

1. Use an SQL query to fetch data for an invoice.
2. Transform the data into the EDIFACT Invoic format.
3. Send the invoice using the AS2 protocol

This case is a notable jump in complexity, and thus it requires 3 channels to handle the different steps. Also, Mirth does not support the AS2 protocol, so an external server had to be used for it. The data transfer between the AS2 server and Mirth had to be done with file transfers, which increased the number of channels needed.

The integration starts from the channel "Send csv with AS2". It can be considered the main channel, as it is used to launch all other channels. The source

connector used is of the type "Database reader", which is used to read data from the MySQL server. The configurations needed for it are the connection details and the SQL query.

For the destination connector a JavaScript connector is used to handle the transfer logic, which can be seen in Figure 5.7. First a unique message ID is assigned to the message that will be used to track it. Then a list is initialized to the Mirth global map. The global map is used to transfer data between channels. Here it is used to track which messages have been sent to the AS2 server. After the global map has been initialized, it is checked if the current message ID is found in the global map. If it is in the map, the message has already been sent, and the integration stops there. If it is not in the map, the message is transferred to the channel "Transfer message to AS2 server" and the message ID is added to the global map.

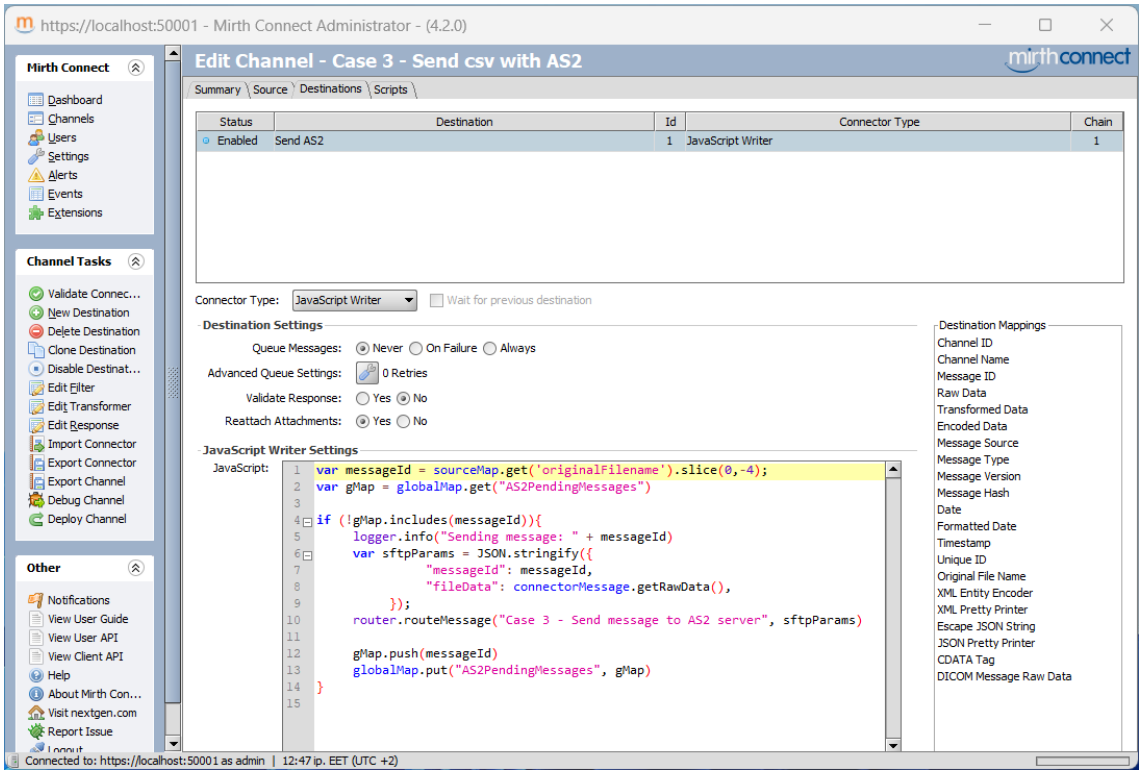

Figure 5.7: Mirth case 3 AS2 transfer destination

The channel "Transfer message to AS2 server" is a simple one, as it is only used to send data to the AS2 server. It uses a "File writer" destination connector to send the CSV file to the AS2 server. An example of this connector can be seen in Figure 5.2

The AS2 server sends the message to the receiver from a specific folder. To confirm that the message has been received correctly, an MDN response is sent back to the AS2 server. These responses can be found in the folder "AS2Server/MDN" on the server. These responses have to be processed in order to confirm if the transfer was successful or if there was an error.

The channel "Process MDN response" is used to handle the MDN responses from the AS2 server. The source connector is a "File reader" which reads the MDN responses. The destination is a JavaScript writer, which handles the MDN processing. The destination connector is shown in Figure 5.8 If the ID of the MDN matches an ID in the global map and the MDN was successful, we remove the ID from the map and update the database to reflect that the message was sent. If the message was unsuccessful, we simply remove the ID from the global map without updating the database. Thus, the integration will automatically retry the sending of that message.

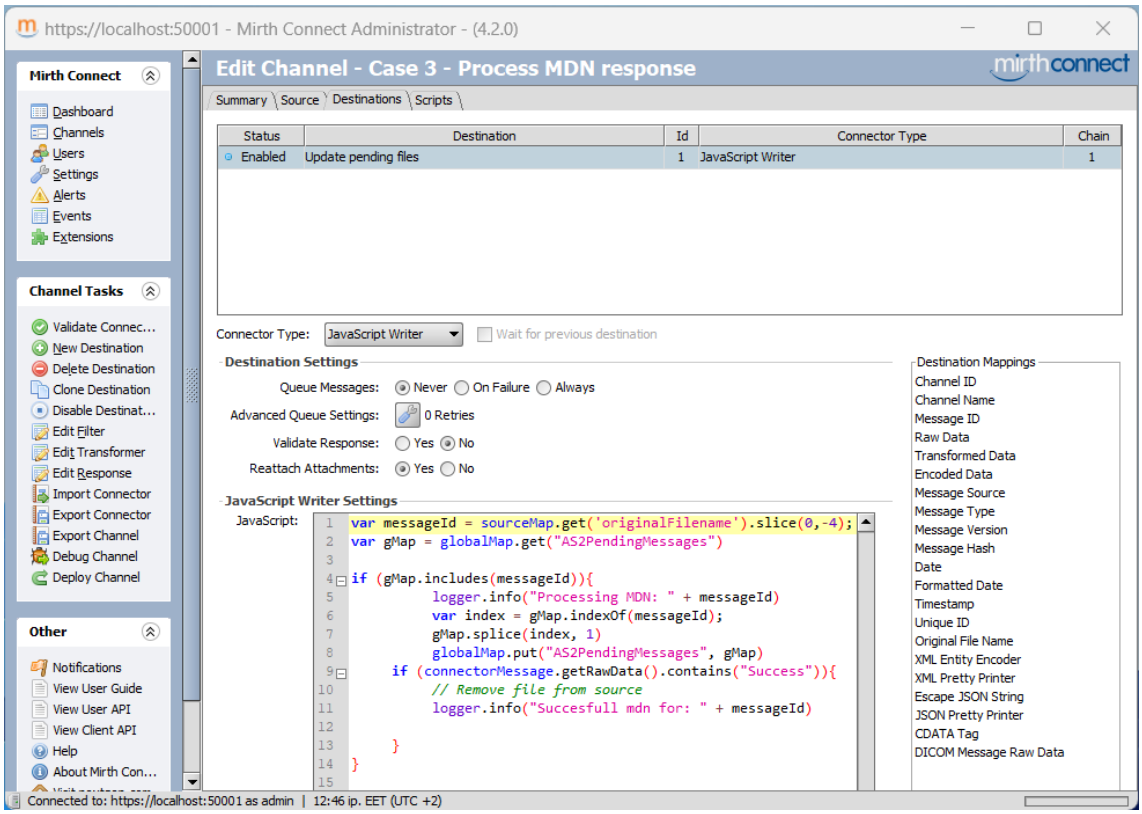

Figure 5.8: Mirth case 3 MDN processor

5.1.4 Case 4

Case 4 consists of 3 steps:

1. Read the Finvoice file from a local directory.
2. Transform the Finvoice into and HTML page.
3. Save the HTML page as a PDF file to a SFTP server.

In this case the 3 steps fit neatly into a single channel. The source connector is used to read the file, a transformer transforms the file to the correct format and a destination connector is used to send the file. Some complexity is caused by the need for external Java libraries to handle the message transformations.

The source connector is a simple "File Reader", which has been used in the earlier cases. It reads the contents of a file that contains a Finvoice message.

The destination transformer is used to transform the Finvoice XML data into a PDF file. The transformation is done in two steps: XML to HTML and HTML to PDF. The XML to HTML transformation is done with a XSL transformation, which uses a standard Java library. The HTML to XML transformation cannot be done with a standard Java or JavaScript library, so the jsoup Java library is added to Mirth and used for this transformation. The transformer code can be seen in Figure 5.9

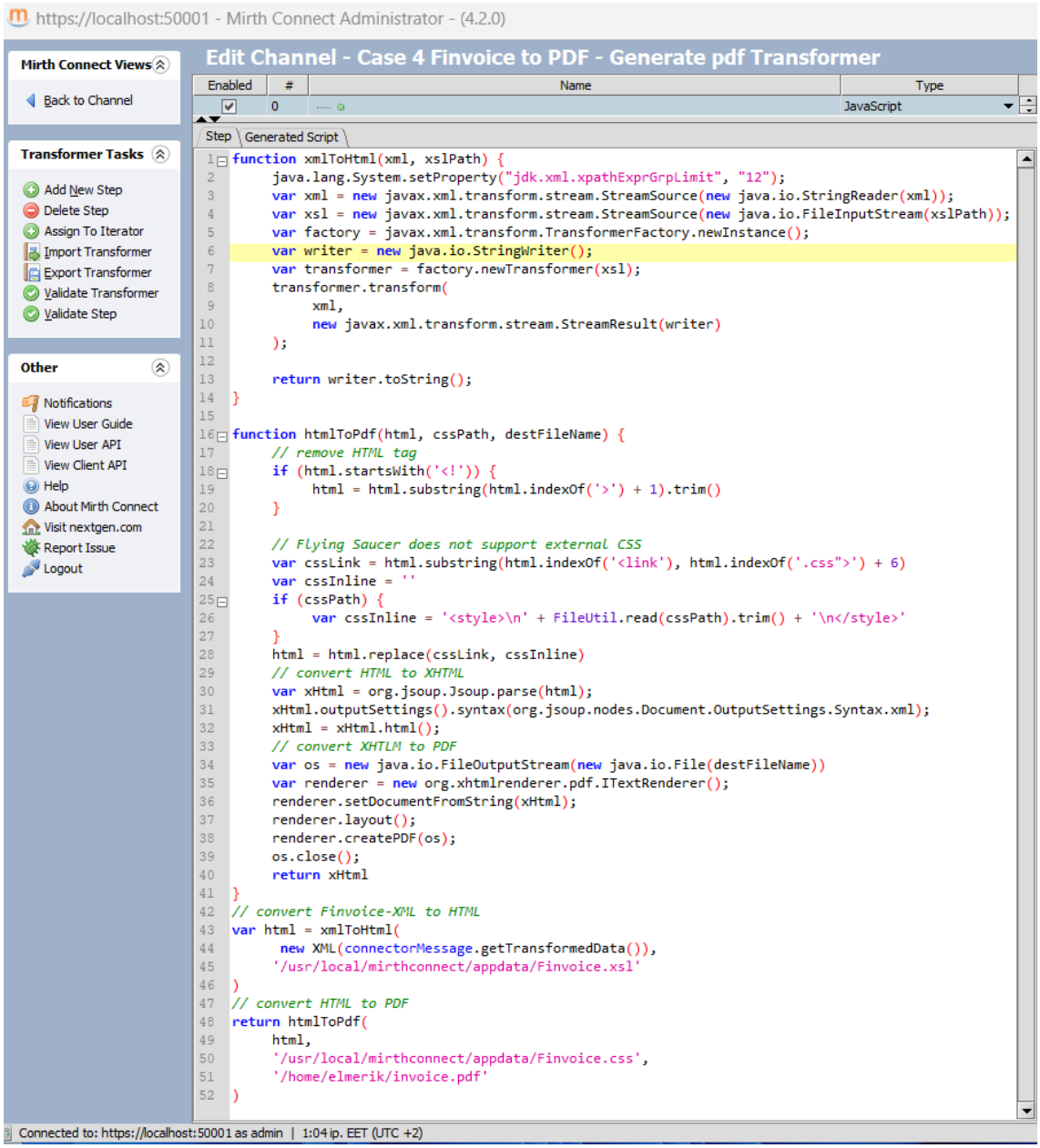

Figure 5.9: Mirth case 4 XML to PDF transformer

After the transformer the transformed data is used by a "File writer" destination connector. It is used to transfer the file via SFTP similarly to the earlier cases.

5.1.5 Findings

Learnability

During the implementation of the cases, I found that the overall learnability of Mirth is poor. Building integrations with the prebuilt connectors is simple and easy to learn, but when you have to do more complicated integrations, problems arise. Additionally, I found that the tutorial content for the platform is lacking.

I think that the biggest problem relating to the learnability of the platform is that it isn't always clear how data should be moved between different steps in the integrations. An example of this would be accessing data returned from the transformer in the destination controller. In some cases, the data is accessed from the variable "msg", while in other cases it is accessed from a separate object. Because the documentation on this is lacking, I had to resort to trial and error to find the right variable to access.

While the user guide for the platform is comprehensive, I found it to be difficult to find the answers to my questions. The guide explains all aspects of the platform in detail, but it rarely gives examples or recommendations on how to use it.

Additionally, the tutorial content for the platform is severely limited. NextGen does not provide any tutorial content and the amount of content from other sources is limited. I found StackOverflow and the Mirth forums to be decent places to find advice on using the platform. StackOverflow has 491 questions tagged with the Mirth platform, and the official Mirth forums have over 15 thousand topics on the support of the platform.[21] [22] While these sources can answer most problems with the platform, they do not offer great tutorial content.

Functionality

Overall, Mirth covered the basic functionalities needed in the cases, but there were a couple rarer functionalities that were missing. Namely these were the support

for the AS2 protocol and a function for HTML to PDF conversions. On the other hand, as Mirth is aimed at the healthcare industry, it supports a datatype used in the specific to it; Health level 7 (HL7)

Extensibility

I evaluate that the extensibility of Mirth is decent. The main ways that Mirth's functionality can be extended is by using an external service or importing Java libraries to the platform. Examples of these can be found in case 3 with the AS2 server and in case 4 with the imported Java HTML to PDF transformer. These methods provide a way for users to extend the platform's functionality to handle most tasks.

A shortcoming with Mirth's extensibility can be found in the source and destination connectors, as new connectors cannot be added easily. The prebuilt connectors were often the most useful ones, so providing a way of adding new ones could promote the extensibility of the platform.

Scalability

Overall, the scalability of Mirth is limited. Mirth does offer threading for each channel, allowing the parallel processing of the incoming messages of a channel. Mirth's processing capacity is limited to the processing speed of the host server, as it does not support multiple servers working together.

Maintainability

Overall, the maintainability of Mirth is decent. Keeping the platform updated requires some technical knowledge, as the current Mirth service has to be manually stopped before running the installer for a new version. The sometimes-limited functionality of the platform causes some downsides on the maintainability of it. As

functionalities have to be added to the platform, those have to be kept up to date separately.

As for the readability of the integrations, I would evaluate it to be poor. The simple integrations that require one or two channels are quite easy to understand, but the more complex cases require more detailed documentation. Also, the complex cases require the use of JavaScript, and thus the maintainer would need a basic understanding of the language to understand them.

Testability

The support for testing of the Mirth platform is poor. Mirth does not provide a way to set up multiple environments for testing, development, and production, which makes testing more time consuming. The user has to set up separate servers for testing and development and move the integrations between them manually. Also, there are no tools for testing parts of the integrations with manual inputs.

Development time

Building the integrations took longer than I anticipated. The development times for the cases were:

- Case 1: 30 minutes
- Case 2: 1 hour and 30 minutes
- Case 3: 3 hours
- Case 4: 2 hours
- Total: 7 hours

In all cases the most time-consuming parts were the programming of the JavaScript connectors. As stated in the learnability section, the way that data

is moved between channels and connectors was unclear and required some trial and error, which increased the development time. Note that cases 3 and 4 required the use of external software and libraries, the setup time of which is not considered.

Reusability

Inside of the platform the reusability of integrations is decent. Channels can be exported and imported between servers easily, improving the reusability of the integrations. Smaller components, such as pieces of JavaScript code can also be reused between integrations if reusability is considered when designing the channels.

The reusability of the integrations between different platforms is poor. The channels can be exported, but no other platform uses the same channel structure as Mirth. The only reusable part is the JavaScript code, but even that has some limitations. Mirth uses a mixture of JavaScript and Java, and the JavaScript connectors often use a combination of both. This mixture is rare among integration platforms, which limits the reusability of the code.

5.2 Frends

5.2.1 Case 1

Case 1 consists of 3 steps:

1. Download a file from the source directory to the integration platform with SFTP.
2. Backup the file with the integration platform.
3. Upload the file to the destination directory with SFTP.

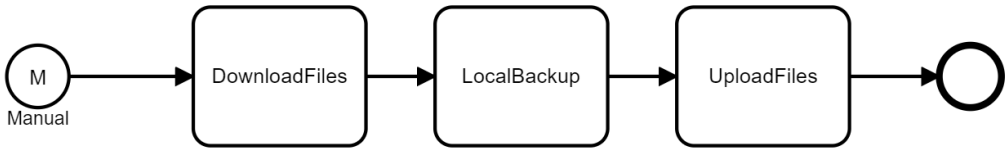

Figure 5.10: Frends case 1

The first case is simple and each of the steps can be done with a single Frends task. The first step downloads all files from the source folder on the SFTP server to a specific folder on the Frends agent. The next step backups all transferred files to a backup folder. The last step moves all files on the agent to the destination folder on the SFTP server. Figure 5.10 is a screenshot of the Frends integration. Each block or task represents one function in the integration flow. In this case, each task completes one step of the case. Figure 5.11 shows how the download task is configured.

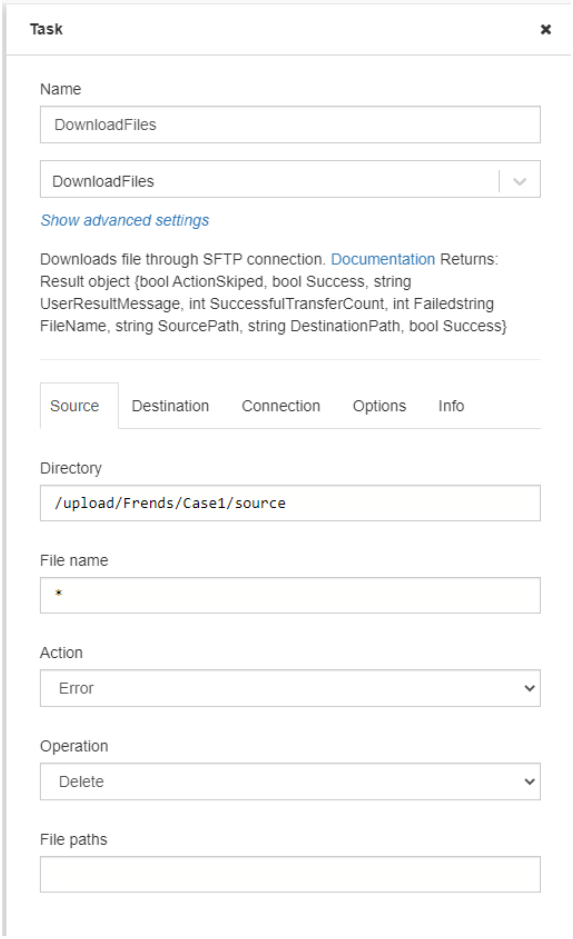

Figure 5.11: Frends SFTP connector configuration

5.2.2 Case 2

Case 2 consists of 3 steps:

1. Use a REST GET call to fetch data from a server.
2. Transform the fetched data into a CSV file.
3. Move the file into a local destination folder.

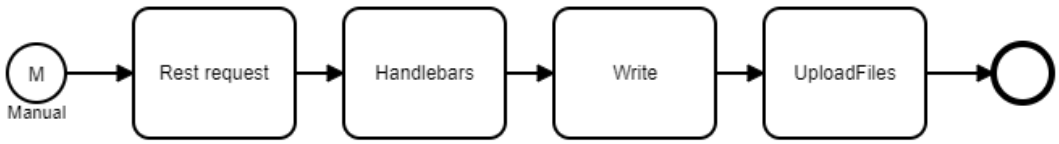

Figure 5.12: Frends case 2

The second case can also be implemented with a small number of tasks. The first task "Rest request" is used to fetch the data with a GET request. The integration flow can be seen in Figure 5.12

The result is then passed to the "handlebars" task, which is used to transform the data into the correct format. The name "handlebars" comes from a notation used in Frends, where code can be inserted into text by encasing it between two curly brackets. The handlebars task uses this notation to format data objects into freeform text. In this case the JSON object is used to create a simple CSV-file. The handlebars task configuration can be seen in Figure 5.13

The formed CSV string is then written to a file on the Frends agent using the "Write" task. That file is then transferred to the destination using the "UploadFiles" task.

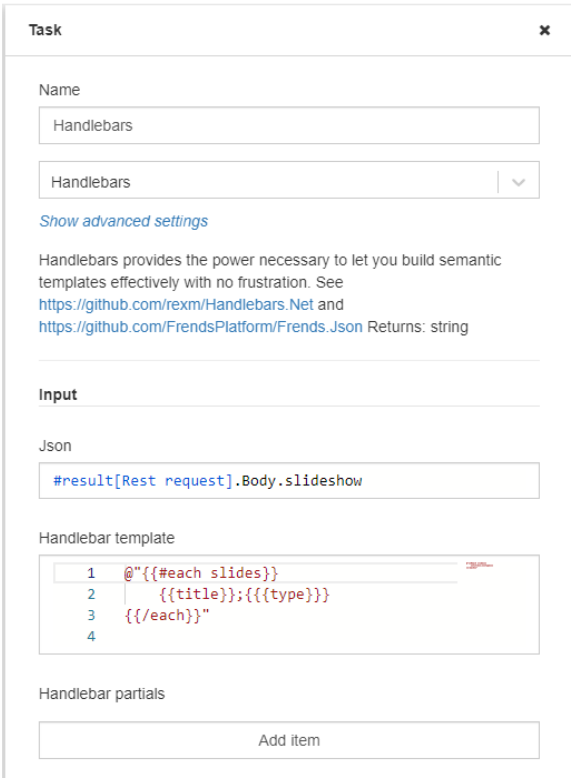

Figure 5.13: Frends handlebars transformation

5.2.3 Case 3

Case 3 consists of 3 steps:

1. Use an SQL query to fetch data for an invoice.
2. Transform the data into the EDIFACT Invoic format.
3. Send the invoice using the AS2 protocol

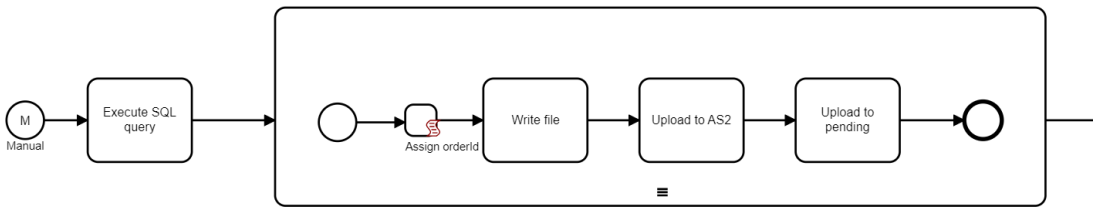

Figure 5.14: Frends case 3 part 1

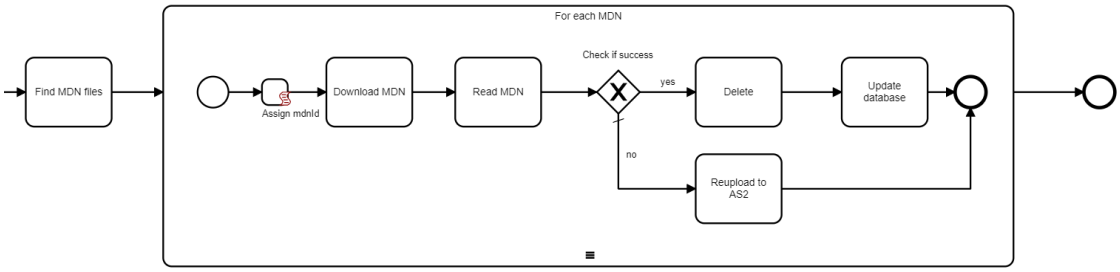

Figure 5.15: Frends case 3 part 2

Frends does not support the AS2 protocol, so the integration has to be done by utilizing an external AS2 server. The server works by reading files from a directory to send and writing the responses to another folder.

The implementation can be split into two parts: the sending of the message and the handling of the response. These could be split into two different processes, but for the sake of simplicity they are combined into a single process, which can be seen in Figures 5.14 and 5.15.

The first task sends an SQL query to a MySQL server. The response to the query is then looped through with a for each loop. For loops are represented in Frends with a rectangle containing the tasks that are looped, as seen in Figures 5.14

and 5.15. In the beginning of the loop the order ID is saved from the query to a variable. Then a file named with the order ID is written with the contents from the query. The file is then copied to the AS2 server for sending. It is additionally moved to a "pending" folder, where it is stored until a successful response is received by the AS2 server.

The second part of the integration handles the MDN-response received by the AS2 server. First the integration reads all files in the MDN folder of the AS2 server, which are then looped through with a for each loop. In the loop the MDN ID is saved to a variable and the MDN file is downloaded and read. If the file contains a successful response the file that contains that MDN ID is removed from the "pending" folder, and the database is updated to reflect that the order has been sent correctly. If the MDN is not successful, the file in the "pending" folder with the same ID is copied back to the AS2 server for a resend.

5.2.4 Case 4

Case 4 consists of 3 steps:

1. Read the Finvoice file from a local directory.

2. Transform the Finvoice into and HTML page.

3. Save the HTML page as a PDF file to a SFTP server.

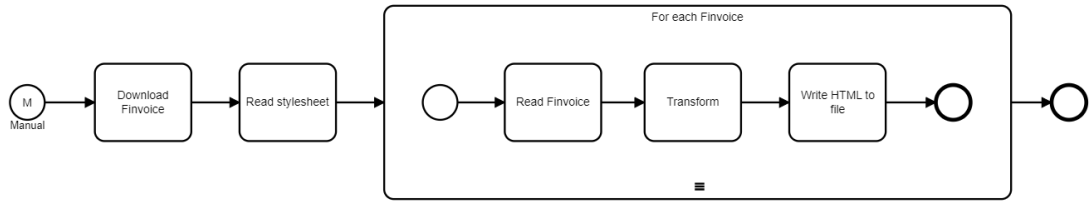

Figure 5.16: Frends case 4

The implementation of the fourth case is relatively simple, but it does not cover the case fully. Frends does not have a task for transforming a HTML page into a PDF-file. This transformation can be done using a C# library, but Frends does not support using external libraries in its code blocks. To utilize an external library, a custom task would have to be developed. While it would be possible to develop such a task, it is outside of the scope of this paper. Another way of circumventing the issue would be by developing a XSL transformation file to transform the XML file straight to a PDF format.

The Frends integration flow can be seen in Figure 5.16. The integration starts by downloading all of the Finvoice XML files from the source directory. Then the stylesheet for the XSLT transformation is read from a local file. A “for each” loop is used to process each of the downloaded Finvoice files. Each file is first read, and then inputted into an XML transform task, which will transform the XML data into a HTML format. The resulting HTML data is then written to a local file.

5.2.5 Findings

Learnability

During the implementation of the cases, I found the learnability of Frends to be great. Frends offers a large group of tasks, most of which are simple to use. Additionally, the documentation of the platforms is great and there is a lot of tutorial content available.

Most tasks in Frends were easy to use and I had little need for external documentation. With most tasks all you have to do is fill well explained text boxes with simple data, such as server addresses. For example, the SFTP task requires you to fill in the fields for the source and destination folders, and the details for the SFTP server.

I would say that the most difficult parts to learn with Frends are the special notations that it uses. Environment and process variables are accessed using "#env" and "#var" notations. Additionally, C# code can be inserted into text by encasing it in the handlebars notation. For example, an environment variable "fileName" could be inserted into a file path with the handlebars notation: "c:/example/#env.fileName". Still this notation is not too complicated and there is a great amount of documentation explaining it.

The documentation for Frends is excellent. Each task has its own page containing details in the usage of it. There is also a docs page with explanations and tutorials on most aspects of Frends. Additionally, Frends has a large number of webinars and tutorial videos presenting examples on how to use the platform. These materials answered all questions I had when implementing the cases.

Functionality

Overall Frends covered the functionalities of the cases decently. The missing parts were the AS2 support and the transformation from HTML to PDF. On the other

hand, the handlebars notation provided a great way for transforming data between different formats.

Extensibility

The extensibility of Frends is great. It provides a way of building additional tasks in an open-source manner, which allows for seamless extension of the platform. At times I hoped that I could extend the functionality by importing C# libraries, but I think forcing the use of custom tasks is a more maintainable solution in the long term.

The main way of extending the functionality of Frends is by building custom tasks. Frends has a community GitHub repository for custom tasks, which can be added to by the community. The building of the tasks is not too complicated, but it requires experience with C# programming. The advantage of this approach is that the custom tasks work just like the official Frends tasks, which makes them easy to use for people familiar with the platform.

Scalability

Overall, the scalability of Frends is excellent. The hybrid platform model allows for multiple servers to handle the processing of integrations. For example, three agent servers can be added to a production environment, which will then be handled by the Frends cloud agent.

Maintainability

The maintainability of the Frends platform is great. Frends informs you when new versions are available for the tasks, and they can be installed using a simple interface. As the custom tasks are built just like the official tasks, they use the same interface, making maintainability uniform between them. The only downside with Frends is

that the updating of the platform has to be done with the assistance of HiQ.

As for the maintainability of the individual integrations, I would assess it to be excellent. The graphical representation of the integration flows makes them easy to understand. Text annotations can also be added to the integrations, further increasing the readability of them. With the addition of the great documentation of the tasks, the integrations are simple to maintain.

Testability

The testability of the integrations with Frends is decent. Frends supports multiple environments, allowing for separate development and production servers. Frends contains version control, assisting the management of integrations between different environments. Frends also has a function for quickly testing tasks with different inputs.

The only downside of the testability of Frends is the sometimes-lacking error messages when building integrations. If there is a part of the graph that will not compile, the error message given can be vague. Often it does not point to a single task, but instead gives a general error message, such as: "Cannot convert from Object to String". With smaller integrations the source of the errors can be found quickly, but searching through larger integrations can be cumbersome.

Development time

Overall, the implementation of the cases didn't take much time. The development times were:

- Case 1: 20 minutes
- Case 2: 1 hour
- Case 3: 1 hour and 30 minutes

- Case 4: 1 hour
- Total: 3 hours and 50 minutes

There weren't many problems with the implementations, so most of the time was spent building and testing the integration flows. Note that the case 4 implementation is missing the conversion from HTML to PDF, reducing the implementation time. Also, the setup time of the AS2 server in case 3 is not considered.

Reusability

The reusability of integrations inside of Frends is excellent, but the reusability outside of it is poor. Parts of integrations can be copied inside Frends with the browser's regular copy/paste functionality, allowing for fast reusing of integrations. However, there isn't a way of utilizing the integrations in another platform, as the integration format is specific to Frends.

5.3 ArcESB

5.3.1 Case 1

Case 1 consists of 3 steps:

1. Download a file from the source directory to the integration platform with SFTP.
2. Backup the file with the integration platform.
3. Upload the file to the destination directory with SFTP.

The first case was easy and quick to implement with ArcESB. The only hindrance to the implementation was that ArcESB does not have a dedicated backup connector.

Thus, the backup is done with a simple file transfer. The integration flow can be seen in Figure 5.17

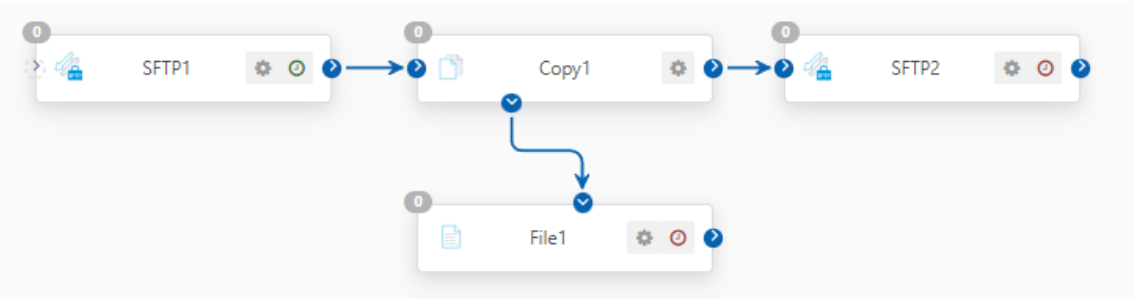

Figure 5.17: ArcESB case 1

The first step of the integration is done with an SFTP connector. The source file is downloaded to ArcESB and passed on to a "copy" connector. This connector copies the file to two different connectors. The first connector uses SFTP to send the file to the destination folder. The second connector saves the file to a local backup folder.

5.3.2 Case 2

Case 2 consists of 3 steps:

1. Use a REST GET call to fetch data from a server.
2. Transform the fetched data into a CSV file.
3. Move the file into a local destination folder.

ArcESB provides all of the required functionalities for the second case. The data transformation required some research, but overall, the implementation didn't take much time. The finished integration flow can be see in Figure 5.18

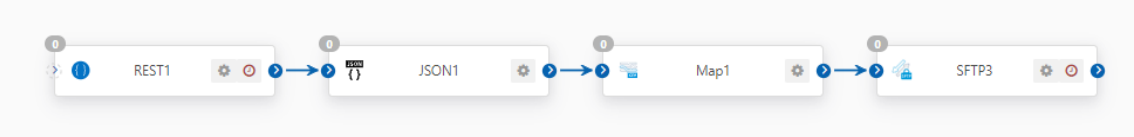

Figure 5.18: ArcESB case 2

The first connector in the flow is a REST connector. This connector sends a GET request to the target URL: "https://httpbin.org/json". The connector saves the response to a JSON-file, which is then forwarded to the next connector in the flow.

The second connector transforms the JSON-object to an XML format. This is done because ArcESB uses XML as its preferred data type in the connectors. For example, the map and EDIFACT connectors only accept XML data.

The transformed XML-file is then passed on to a "CSV map" connector. This connector is used to map the incoming XML-file to a CSV-file. The connector settings for this mapping can be seen in Figure 5.19. This connector takes two files as a source and a destination template. The mapping can then be done based on the fields in the two files. ArcESB does provide a graphical mapper for simple files, but unfortunately the XML-file in this case contains lists, which are not supported by the graphical mapper. Thus, the mapping has to be done with a script written in ArcESBs own scripting language: ArcScript. In this case the script loops through all "slides" objects and adds the "title" and "type" fields into the CSV.

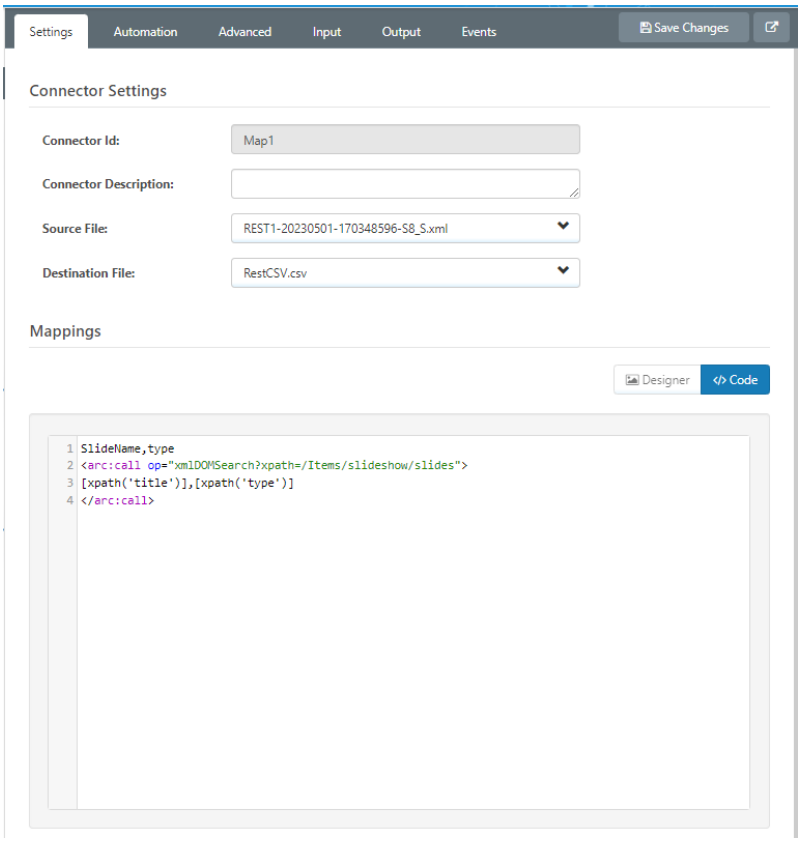

Figure 5.19: ArcESB case 2 mapping

Finally, the resulting CSV-file is transferred to an SFTP connector. This connector is the same as in the first case and transfers the file to the destination folder.

5.3.3 Case 3

Case 3 consists of 3 steps:

1. Use an SQL query to fetch data for an invoice.
2. Transform the data into the EDIFACT Invoic format.
3. Send the invoice using the AS2 protocol

The third case is a considerable increase in complexity, but ArcESB still provided the required functionalities for it. ArcESB has strong support for both EDI

messages and the AS2 protocol, which made the implementation relatively simple. The finished integration flow is presented in Figure 5.20

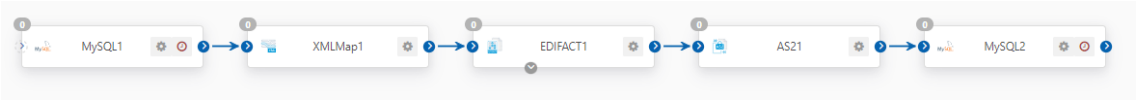

Figure 5.20: ArcESB case 3

The first connector in the flow is a "MySQL" connector, the configuration of which is shown in Figure 5.21. This connector sends an SQL query to a MySQL server, which responds with some data to be inserted to the EDIFACT message. The "MySQL" connector provides a graphical editor for the SQL query, which made the building of the query much simpler. The connector returns an XML-file that contains the response to the query.

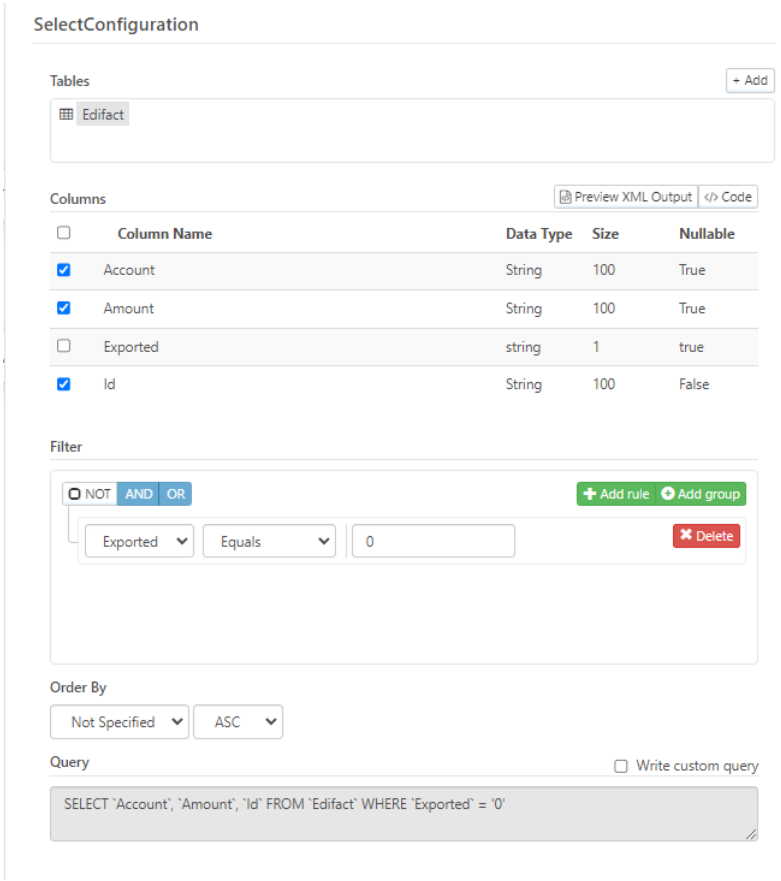

Figure 5.21: ArcESB case 3 SQL query

The XML-file containing the SQL-query response is forwarded to an “XMLMap" connector, which is used to map values from an XML-file to another XML-file. The configuration of this connector can be seen in Figure 5.22 The destination file in this case is a template for an EDIFACT INVOIC message. The data is mapped using a graphical mapping interface, which made the mapping simple. The mapped XML-file is then forwarded to the next connector.

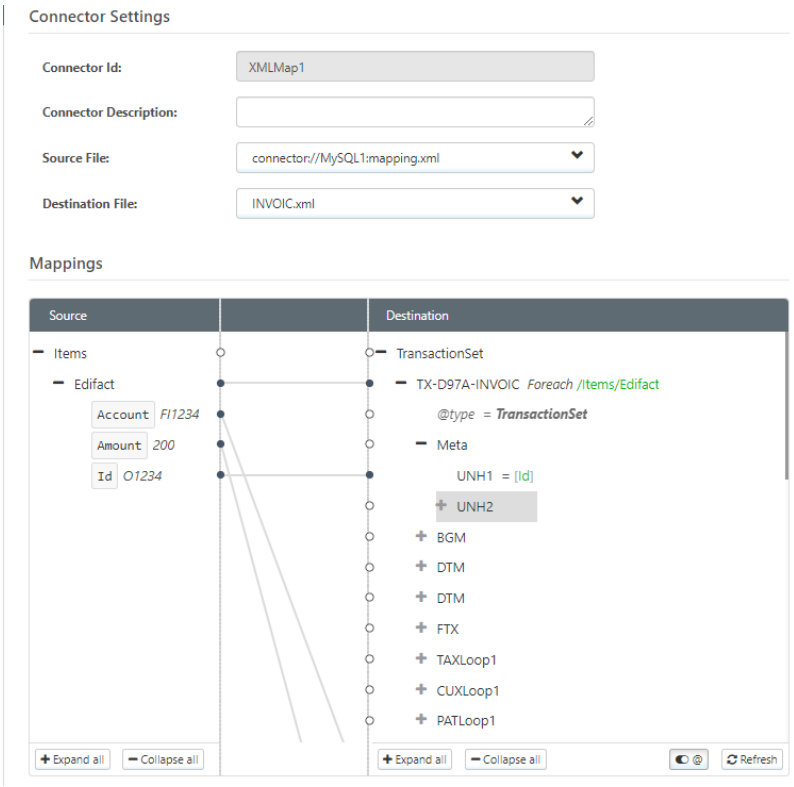

Figure 5.22: ArcESB case 3 mapping

The flow continues with an "EDIFACT" connector. This connector is used to transform XML formatted EDIFACT message into the correct EDI format. Additionally, this connector can be used to validate the structure of the resulting message. In this case we are doing a simple transformation with minimal validation.

The resulting EDIFACT file is then moved to a "AS2" connector. This connector handles the whole AS2 messaging process, from sending the message to receiving the MDN confirmation file. If the received MDN response is successful, the message is passed onto the last connector.

The final connector in the flow is another "MySQL" connector. It is used to update the database to reflect that the message has been transferred successfully. If there are any errors in the flow, the database is not updated, and the message will be reprocessed during the next run of the flow.

5.3.4 Case 4

Case 4 consists of 3 steps:

1. Read the Finvoice file from a local directory.
2. Transform the Finvoice into and HTML page.
3. Save the HTML page as a PDF file to a SFTP server.

For the last case ArcESB is missing the functionality to convert HTML-pages into PDF-files. Thus, the last step is omitted, and the HTML-page is saved to the SFTP server as the final step. The finished flow can be seen in Figure 5.23

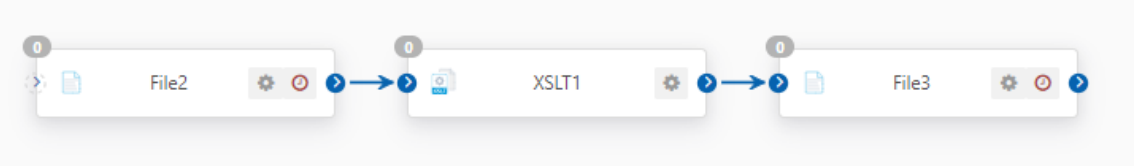

Figure 5.23: ArcESB case 4

The flow starts with a file connector, which reads the local Finvoice file. The file is then passed to an “XSLT" connector, which will transform the XML Finvoice file into a HTML format. XSLT could be used to transform the XML-file straight into a PDF-file, but the Finvoice standard only has the HTML transformation XSL file available. To complete this integration, a XSL file that transforms to PDF could be developed, but it is outside of the scope of this study.

5.3.5 Findings

Learnability

The learnability of ArcESB is great. Building the flows with the connectors is easy to learn, as one needs to only place the needed connectors and connect them with

arrows. There is a small learning curve with the configuration of the connectors, as each one of them have their unique settings. For this, ArcESB provides a comprehensive help-page, which covers the usage of each of the connectors.

ArcESB also provides a good number of how-to guides straight from the platform. These guides show how to build some common sample integrations. There is also an online knowledge base, which contains articles and videos about the usage of the platform. [23] Lastly ArcESB hosts an active community forum, where you can ask questions from the community. [24]

Functionality

Overall ArcESB did an excellent job at covering the functionalities required by the cases. The only part missing was a way of transforming a HTML-page to a PDF-file, which could be circumvented with an XSLT transform. ArcESB is focused on EDI messages and the common protocols that are used to transfer them, which was important for the complex third case.

Extensibility

The extensibility of ArcESB is decent. The main way of extending the platform's functionality is by using the ArcScript connectors. These scripts can include custom C# or Java code that implement the RSBOperation interface. While building these custom operations is simple for programmers, for citizen developers it will require a lot of additional studying. Also, there isn't a way to build custom connectors, which function like the pre-built connectors.

Scalability

The scalability of ArcESB is excellent. It provides support for running multiple servers in a cluster, allowing for multiple servers to work together. Also, each con-

nector can be run in parallel using multiple workers.

Maintainability

The maintainability of ArcESB is decent. Connectors are updated when the whole platform receives a new build, meaning that connectors cannot be updated individually. CData releases multiple builds yearly, and updates are done using an installer or with the assistance of CData, depending on if the server is installed on premises or in the cloud.

The integration flows are easy to maintain, as the visual representation provides an easy way of learning the integrations functions. Each connector can include a description, allowing for more detailed documentation straight in the platform. Unfortunately, ArcESB does not provide a way of adding additional annotations straight into the visual form of the integration.

Testability

The testability of the ArcESB integrations is poor. ArcESB does provide a way of separating a development and a production environment from each other. However, it does not provide an automatic way of pushing flows from the development environment to production, which has to be done manually. The error messages given during the implementations were clear and made the testing of the cases simple.

Development time

The development times for ArcESB were:

- Case 1: 20 minutes
- Case 2: 30 minutes
- Case 3: 1 hour

- Case 4: 30 minutes
- Total: 2 hours and 50 minutes

After a small initial learning curve, I found the building of integration flows to be fast. I encountered a few errors, which were quickly solved with the great documentation provided by the platform. Note that the last case is missing the conversion from HTML to PDF, which shortens the development time here.

Reusability

The reusability of ArcESB integrations is poor. Inside of the platform connectors can be easily reused by copy-pasting them from one flow to another. However, there isn't a way to export the integrations into a reusable format. Additionally, ArcESB uses its own scripting language, meaning that even they cannot be reused.

5.4 Camel

5.4.1 Case 1

Case 1 consists of 3 steps:

1. Download a file from the source directory to the integration platform with SFTP.
2. Backup the file with the integration platform.
3. Upload the file to the destination directory with SFTP.

The implementation of the first case was quick and simple. The first step is done with an SFTP component, that downloads all files from the source directory. Then the next step creates a backup to a local folder. Then the SFTP component is used

again to move the file to the final destination. The code for case 1 can be seen in Listing 5.1.

```
public class Case1Route extends RouteBuilder {
    public void configure() {
        from("sftp:localhost:22/upload/Camel/Case1/in?password=ShinySofa517
            &username=foo&delete=true")
                .log("Case␣1␣source␣downloaded")
                .to("file:target/Case1/backup")
                .to("sftp:localhost:22/upload/Camel/Case1/out?password=
                    ShinySofa517&username=foo");
    }
}
```

Listing 5.1: Case 1 route

5.4.2 Case 2

Case 2 consists of 3 steps:

1. Use a REST GET call to fetch data from a server.
2. Transform the fetched data into a CSV file.
3. Move the file into a local destination folder.

The second case took noticeably more time than the first one. There isn't a fast built-in way of transforming JSON-data into the CSV-format, which meant that the transformation took more time to develop. The code for this case can be seen in Listing 5.2.

```
public class Case2Route extends RouteBuilder {
    public void configure() {
```

```
        from("timer:foo?fixedRate=true&period=1000")
                .to("http:httpbin.org/json")
                .log("Case␣2␣source␣downloaded")
                .unmarshal(new JacksonDataFormat(Root.class))
                .process(exchange -> {
                    List<Slide> slides = exchange.getIn().getBody(Root.
                        class).getSlideshow().getSlides();
                    StringBuilder csvBuilder = new StringBuilder();
                    for (Slide slide:slides){
                        csvBuilder.append(slide.getTitle() + ";" + slide.
                            getType() + "\n");
                    }
                    exchange.getMessage().setBody(csvBuilder.toString());
                })
                .to("file:target/Case2?fileName=case2.csv");
    }
}
```

Listing 5.2: Case 2 route

The integration starts with a timer that launches the route once per second. A timer had to be used, because an HTTP-component cannot be used as the starting point of a route. The timer launches an HTTP-component, which uses a GET-request to fetch the JSON-data from a server.

The JSON-object is then unmarshaled into a custom Java-object "Root". This object was built manually, and it represents the data structure of the JSON-object received from the server. This allows the data to be handled as a standard Java-object, from which it can be transformed to other formats. In this case we loop

through each of the slides in the object and add the data in them to a CSV-string.

After each slide has been processed, the resulting CSV-string is assigned as the body of the current exchange. This body is then written to the local destination folder as a CSV-file using the file-component.

5.4.3 Case 3

Case 3 consists of 3 steps:

1. Use an SQL query to fetch data for an invoice.
2. Transform the data into the EDIFACT Invoic format.
3. Send the invoice using the AS2 protocol

For the third case the configuration of the SQL and AS2-components took a lot of time. For the SQL-component the recommended way of configuring it is by creating a DataSource object and storing it in a registry in the Camel context. The code for the registry addition can be seen in Listing 5.4. The correct way of saving the configuration in the registry took some time to research. For the AS2-component, the documentation of it is rather limited and there weren't any great examples of configuring it. Thus, the AS2-component took a lot of trial and error to configure correctly. The code for this case can be seen in Listing 5.3.

```
public class Case3Route extends RouteBuilder {
    public void configure() {
        AS2Configuration as2Config = new AS2Configuration();
        as2Config.setTargetHostname("localhost");
        from("timer:foo?fixedRate=true&period=1000")
            .setBody(constant("SELECT␣*␣FROM␣Edifact␣WHERE␣Exported␣
                =␣'0'"))
            .to("jdbc:mysql")
```

```
.split(body())
.process(exchange -> {
    HashMap<String, Object> sqlResult = exchange.getIn().getBody
        (HashMap.class);
    System.out.println(sqlResult.toString());
    Path filePath = Path.of("src/main/resources/INVOIC.edi");
    String content = Files.readString(filePath);
    content = content.replaceAll("INVOICEID", sqlResult.get("Id").
        toString());
    content = content.replaceAll("AMOUNT", sqlResult.get("
        Amount").toString());
    exchange.getMessage().setBody(content);
}).setHeader("CamelAS2.ediMessage", constant("test"))
.setHeader("CamelAS2.as2MessageStructure", constant(
    AS2MessageStructure.PLAIN))
.setHeader("CamelAS2.as2To", constant("TestReceive"))
.setHeader("CamelAS2.as2From", constant("test"))
.setHeader("CamelAS2.ediMessageContentType", constant(
    ContentType.create("application/edifact")))
.setHeader("CamelAS2.from", constant("test"))
.setHeader("CamelAS2.requestUri", constant("/pub/Receive.rsb"))
.setHeader("CamelAS2.subject", constant("test"))
.setHeader("CamelAS2.as2Version", constant("1.1"))
.setHeader("CamelAS2.targetHostName", constant("localhost"))
.setHeader("CamelAS2.targetPortNumber", constant("8001"))
.to("as2:client/send?targetHostName=localhost&requestUri=/pub/
    Receive.rsb&targetPortNumber=8001")
```

```
            .choice()
            .when(body().contains("Success"))
                .setBody(constant("UPDATE␣Edifact␣SET␣Exported␣=␣'1'␣
                    WHERE␣Id␣=␣test"))
                .to("jdbc:mysql")
            .otherwise()
            .log("Error");
    }
}
```

Listing 5.3: Case 3 route

```
public class MainApp {
    public static void main(String... args) throws Exception {
        Main main = new Main();
        SimpleRegistry registry = new SimpleRegistry();
        BasicDataSource ds = new BasicDataSource();
        ds.setUsername("root");
        ds.setPassword("ShinySofa57");
        ds.setUrl("jdbc:mysql://localhost/dicase3");
        registry.bind("mysql",javax.sql.DataSource.class, ds);
        CamelContext camel = new DefaultCamelContext(registry);
        camel.addRoutes(new Case3Route());
        camel.start();

    }

}
```

Listing 5.4: Case 3 Camel configuration

The case 3 route starts with a timer, which launches the route once per second. At the start of each exchange the exchange body is set as the SQL-query that fetches all orders that are not exported from the MySQL-database. This body is then forwarded to a jdbc-component, that has been configured with the information of the MySQL-database. The result from the query is then split, so that each individual order can be processed separately.

Each order is processed by first saving the query into a HashMap. The data in this map is then used to fill out fields in an EDIFACT string read from a template file.

The next steps are used to configure the AS2-component. The component uses header fields to set all of the required information for the AS2-protocol. After the configuration is done, the EDIFACT message is sent using the AS2-component.

The next steps are to handle the MDN-response from the receiving AS2-server. If the response contains a success message, the MySQL database is updated to reflect that the order has been exported successfully. This is done with the already configured jdbc-component. If the message does not contain a success message an error message is logged.

5.4.4 Case 4

Case 4 consists of 3 steps:

1. Read the Finvoice file from a local directory.

2. Transform the Finvoice into and HTML page.

3. Save the HTML page as a PDF file to a SFTP server.

The first step of the integration is done with a file-component. It reads the source finvoice message, the contents of which are converted into an XML-String. The resulting string is passed to a processor, which handles the conversion from XML to HTML, and from HTML to PDF.

The processor uses two methods to transform the Finvoice message between the different formats: xmlToHtml and htmlToPdf. The xmlToHtml-method uses a javax transformer to perform a XSL transformation from XML to HTML. The resulting HTML string is then passed to the htmlToPdf-method, which uses the Jsoup package for the transformation [1]. This method also creates the final file to the target destination. The code for the fourth case can be seen in listing 5.5.

```
public class Case4Route extends RouteBuilder {
    public void configure() {
        from("file:target/Case4/in")
            .convertBodyTo(String.class)
            .log("Case 4 source read")
            .process(exhange -> {
                String htmlString = xmlToHtml(exhange.getIn().getBody().
                    toString(), "src/main/resources/Finvoice.xsl");
                String pdfString = htmlToPdf(htmlString, "target/Case4/
                    TestPDF.pdf");
            })
            .to("file:target/Case4?fileName=case4.pdf");
    }
    private String xmlToHtml(String xml, String xslPath) {
        try {
            java.lang.System.setProperty("jdk.xml.xpathExprGrpLimit", "12");
```

[1] https://jsoup.org/

```
            StreamSource xmlStream = new StreamSource(new StringReader(
                xml));
            StreamSource xsl;
            xsl = new StreamSource(new FileInputStream(xslPath));
            TransformerFactory factory = TransformerFactory.newInstance();
            var writer = new StringWriter();
            var transformer = factory.newTransformer(xsl);
            transformer.transform(
                xmlStream,
                new StreamResult(writer)
            );
            return writer.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
    }
    private String htmlToPdf(String html, String destFileName) {
        try {
            // remove HTML tag
        if (html.startsWith("<!")) {
            html = html.substring(html.indexOf('>') + 1).trim();
        }
        // convert HTML to XHTML
        Document xHtml = Jsoup.parse(html);
        xHtml.outputSettings().syntax(Syntax.xml);
        String htmlString = xHtml.html();
```

```
        // convert XHTLM to PDF
        FileOutputStream os = new FileOutputStream(new File(destFileName))
            ;
        var renderer = new ITextRenderer();
        renderer.setDocumentFromString(htmlString);
        renderer.layout();
        renderer.createPDF(os);
        os.close();
        return os.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
    }
}
```

Listing 5.5: Case 4 route

5.4.5 Findings

Learnability

Overall, the learnability of Camel is poor. The biggest hurdle of learning Camel is learning to program with Java, as it is the core language of the platform. Also, as Camel is a relatively low-level system, it is often necessary to know how to run it inside of a framework, such as Spring Boot or Kubernetes. On the other hand, the documentation for Camel is extensive, which helps development. [2]

For a developer who has some Java experience the building of integration routes

[2]https://camel.apache.org/camel-core/

is fairly simple. The routes are built from smaller component methods which are well documented and simple to use. Some problems arose with the more complicated components, the configuration of which differed from the simpler ones.

The learnability of the platform is helped by the excellent documentation and good amount of answered questions on Stack Overflow.[2] [25] The official documentation from Apache contains detailed information on the configuration and usage of each of the Camel components, which was sufficient for most of the cases. The official documentation also contains plenty of examples and sample projects that help you to get started with the platform. For the cases where the official documentation wasn't enough, I resorted to Stack Overflow questions. Stack Overflow has around 7000 answered questions about Camel, which should offer help with most common issues with the platform.

Functionality

The functionality of Camel is great. It contained all of the needed components to build all four cases. It also contains methods for handling a lot of the needed datatypes, which helped development. Note that the official Camel components are split into different packages, thus needing some extra configuration to achieve the needed functionality.

Extensibility

The extensibility of Camel is excellent. As Camel is running with plain Java-code, it can easily be extended using Java-classes. For example, the Camel process-component can be used to process messages with other Java-classes. As Java is an old and widely used programming language, the library of usable Java-classes is large. Camel also provides great documentation on how to build your own custom Camel-components.

Scalability

Overall, the Scalability of Camel is decent. Camel allows for threading inside of routes, allowing multiple messages to be processed in parallel. Camel does not provide a simple way of utilizing multiple Camel servers to process messages using the same routes, but as a low-level platform it can be extended to allow for it by using additional Java-packages.

Maintainability

The maintainability of Camel is poor. A fully configured Camel platform will include multiple packages, which all need to be kept up to date. This can be done with external package managing tools, such as Maven. This will require some knowledge of the used package manager and how the platform is deployed.

The maintainability of individual routes can be difficult. The readability of routes that only use the official components is decent, but details on the configuration of them will often be needed from the official documentation. Also, if additional Java-packages are used in the routes, the reader will need to be familiar with the functionality of them, or the code has to be well commented.

Testability

The testability of Camel is great. As the routes contain Java-code, Java testing packages can be used to run automatic tests on them. For separating a development and a production environment, standard Java-devops tools can be used. However, this requires the developers to be familiar with how these features can be implemented in Java.

Development time

The development times for the cases were:

- Case 1: 1 hour
- Case 2: 2 hours
- Case 3: 4 hours
- Case 4: 1 hour
- Total: 7 hours

By far the most time was spent on the third case. It required the configuration of the SQL-component and the AS2-component, both of which were difficult to configure. The other cases were more straightforward, with most time spent on handling and transforming the transferred data.

Reusability

The reusability of integrations inside of Camel is excellent. The routes are split into smaller components, which can be easily reused in other routes. Parts of Camel integrations can also be used in other platforms if they support the Java-language. For example, many of the processing steps contain plain Java-code, which could be repurposed in other applications.

6 Comparison

In this chapter the different platforms are compared with each other based on the case study. The comparison will be made individually for each comparison area. It is also evaluated how the low code approach affected the results for each area.

6.1 Learnability

In the case study there was a clear difference between the two low-code platforms (Frends and ArcESB) and the two lower-level platforms (Mirth Connect and Apache Camel). The low-code graphical integration designers were more intuitive to use, and both platforms had great documentation. The two other platforms require knowledge on programming, severely hindering their learnability.

The low-code approach aims to create tools that could be used by citizen-developers, and the two low-code platforms succeeded in that goal. Building integrations with the low-code platforms mimics drawing graphs of a process, which felt intuitive. Additionally, the documentation did a great job at answering any questions that arose during the development process. It is important to note that the two low-code platforms are paid products, while the two low-level platforms are open-source developed, which likely explains some of the documentation quality difference.

For the two low-level platforms the biggest hurdles came from the fact that some or all of the integrations had to be implemented with code. Without programming

knowledge, using the two low-level platforms would be difficult and time consuming. Additionally, the documentation of Mirth was lacking and partly outdated, causing more problems with development.

6.2 Functionality

Overall, the integration platforms fulfilled the needed functionalities at a similar level. Most platforms were missing one or two needed functionalities, while only Camel had all of the needed functionalities. Camel is a low-level integration platform, so it can easily utilize existing Java-code to increase its functionality. The low-code platforms require more work from their developers to increase their functionality, which might explain some of the differences.

6.3 Extensibility

During the case study it became clear that extending the functionality of the two low-level platforms was much easier compared to the low-code platforms. The two low-level platforms allowed for external packages to be imported to the platform and used straight in the integration code.

The functionalities of the low-code platforms could also be extended, but it requires much more work. Additional code has to be wrapped with an interface used by the platforms, requiring additional development work. This highlights how important it is for low-code platforms to offer a wide set of functionalities, as extending them is not ideal.

6.4 Scalability

Overall, the scalability of the two low-code platforms was better compared with the low-level platforms. All platforms supported threading of individual platforms, but only the low-code platforms supported scaling of the server resources as they could be run in the cloud natively. The two low-level platforms require additional development if the server performance has to be scalable.

6.5 Maintainability

There wasn't much of a difference between the platforms with keeping them updated. Most platforms had an installer for updating to new versions. The only exception to this is Camel, which is kept updated by updating individual code packages.

The readability of the integrations was much better with the low-code platforms. As the integrations are built with graphs, they automatically produce a visual representation of the integration process, which makes understanding them much easier. To understand the low-level integrations, you would need some experience with the used programming language and the platform itself, as the integrations consist of mostly code. Thus overall, it is easier to maintain integrations built with low-code platforms.

6.6 Testability

In the case study there was a lot of variance between the testability of the platforms, with most platforms performing poorly. The low-code platforms don't provide a way to test the integrations automatically, and with ArcESB the support for a testing environment was limited. This was also the case for one of the low-level platforms, Mirth. The only platform with good support for testing was Camel. Camel is a

very low-level platform, so it can easily take advantage of existing tools for testing Java-code.

While testability is not a problem only for low-code platforms, they seem to have unique problems with it. Already existing testing tools are made for programming languages, and low-code platforms would need to develop their own ways of utilizing them with their programs. Building universal testing tools for low-code platforms seems difficult, so each low-code platform has to spend time on developing their own tools if they want to improve their testability.

6.7 Development time

There was a big difference in the development times between the low-code and low-level platforms. The low-level platforms took around 7 hours of development time each, and the low-code platforms took between 3 and 4 hours.

I had experience with one low-code and one low-level platform, which didn't make much of a difference in the development times, as the times were similar with the platforms that were new to me. Another thing to note is that the low-code platforms were missing the functionality for the last transformation in Case 4, which made that case faster to develop for them. Still, it can be concluded that low-code platforms can bring significant improvements to the development times of integrations.

6.8 Reusability

For the low-code platforms, there were no parts that could be reused with another platform. Each low-code platform has their own way of building the integration graphs, which are not transferrable to other platforms. The only reusable pieces were the small scripts used in the integrations, but for ArcESB even they couldn't be reused as they were written in ArcESB's own language.

For the low-level platforms reusability is better as larger code pieces can be reused. In the case study both low-level platforms ran on Java, so code could be transferred between them with some modifications. For example, a lot of the code that transformed the data was built using plain Java code, which could be easily reused.

It is clear from the case study that reusing low-code integrations in another platform is extremely difficult. It was noted in the literature that lock-in effects are a concern with low-code platforms [1], and it was confirmed with this study. If a low-code platform stops getting support, moving to another will be costly as most work done will be lost.

6.9 Summary

The main advantages of the low-code approach were found in learnability, development time and maintainability. The low-code approach aims to create platforms that are so easy to use that they can be used by citizen developers. The three mentioned advantages are all important for reaching that goal. The main feature of low-code platforms that advance all three areas is the graphical user interface used to build the integrations. Drawing graphs is more intuitive than writing lines of code, leading to better learnability and reduced development time. They are also easier to understand, leading to better maintainability.

The disadvantages of the low-code approach were found in extensibility and reusability. Compromises had to be made in order to create streamlined development experiences, and those caused some inflexibility in the platforms. Extending the functionality takes some programming knowledge and it cannot be done by a citizen-developer. Additionally, it was cited in the literature that software created by a low-code platform is locked into that specific platform and cannot be moved to another low-code platform [1]. This was found to be the case in the case study, and it is a

considerable downside for low-code platforms.

For the rest of the areas the low-code approach didn't seem to make a huge difference. Functionality was similar across all of the platforms in the study. Integrations limit the number of needed functionalities, and the low-code platforms covered them well. For scalability low-code had no effect, as scalability is handled at a higher architectural level. Lastly testability was poor on three of the four platforms, so it can be seen as a wider problem for integration platforms.

7 Conclusion

The goal for this thesis was to assess how low-code affects integration platforms. Many integration platforms employ the low-code approach, but there is little research into the effects of it.

To find the effects of low-code in integration platforms, a literature review was conducted to find the areas which are often affected by low-code. The found areas were: learnability, functionality, extensibility, scalability, maintainability, testability, development time and reusability.

To assess how low-code affects the found areas, a case study was done using two low-code integration platforms and two low-level platforms. The low-code platforms were Frends and ArcESB, and the low-level platforms were Mirth and Camel. The platforms were chosen from the platforms used by Netum based on their availability and how often they are used by Netum.

The four platforms were used to implement four integration cases of varying complexities. The cases were chosen from integrations developed by Netum based on their commonality and complexity. The cases contain some common integration tasks, such as API-calls and SFTP-transfers, and some less common ones, such as AS2-transfers.

The key findings from the case study were that low-code has a positive effect on the platform's Learnability, Maintainability and Development time. On the other hand, it has a negative effect on the platform's Extensibility and Reusability. In

general, the low-code platforms were easier to learn and use, which lead to quicker development times.

The downsides of low-code were that the functionalities of the platforms were harder to extend and the integrations developed with them were difficult to reuse in other platforms. To extend the capabilities of a low-code platform, custom low-code components need to be developed, which is more complex when compared with extending the capabilities of the low-level platforms. Also, the integrations built with low-code platforms use platform specific formats for the integrations, meaning that very little of them can be repurposed, leading to being locked-in with the platform.

Overall low-code seems like a good fit for integration platforms. The main downsides occur when the platform does not provide the needed functionalities for the integration task. Integrations often need a limited set of functionalities, which limits the need for extending the platforms capabilities. On the other hand, the advantages of low-code can be sizeable as it can lead to quicker development times. There is also the possibility of utilizing so-called "citizen-developers" to develop integrations, whom have better domain knowledge of the needed integration.

References

[1] A. C. Bock and U. Frank, "Low-Code Platform", en, *Business & Information Systems Engineering*, vol. 63, no. 6, pp. 733–740, Dec. 2021. [Online]. Available: https://doi.org/10.1007/s12599-021-00726-8.

[2] J. Cabot, "Positioning of the low-code movement within the field of model-driven engineering", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '20, New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–3. [Online]. Available: http://doi.org/10.1145/3417990.3420210.

[3] B. Selic, "The pragmatics of model-driven development", *IEEE Software*, vol. 20, no. 5, pp. 19–25, Sep. 2003, Conference Name: IEEE Software. DOI: 10.1109/MS.2003.1231146.

[4] A. C. Bock and U. Frank, "In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms", in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Oct. 2021, pp. 57–66. DOI: 10.1109/MODELS-C53483.2021.00016.

[5] V. Säfsten, *Low-code vs traditional codeApplication of low-code solution in ERP systems*, eng. 2022. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-478456.

[6] M. A. Al Alamin, S. Malakar, G. Uddin, S. Afroz, T. B. Haider, and A. Iqbal, "An Empirical Study of Developer Discussions on Low-Code Software Development Challenges", in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, May 2021, pp. 46–57. DOI: 10.1109/MSR52588.2021.00018.

[7] T. Grossman, G. Fitzmaurice, and R. Attar, "A survey of software learnability: Metrics, methodologies and guidelines", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09, New York, NY, USA: Association for Computing Machinery, 2009, pp. 649–658. [Online]. Available: https://doi.org/10.1145/1518701.1518803.

[8] L. Mew and D. Field, *A Case Study on Using the Mendix Low Code Platform to support a Project Management Course*. Nov. 2018.

[9] S. Käss, S. Strahringer, and M. Westner, "Drivers and inhibitors of low code development platform adoption", in *2022 IEEE 24th Conference on Business Informatics (CBI)*, vol. 01, Jun. 2022, pp. 196–205. DOI: 10.1109/CBI54897.2022.00028.

[10] L. Duboc, D. Rosenblum, and T. Wicks, "A framework for characterization and analysis of software system scalability", in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC-FSE '07, New York, NY, USA: Association for Computing Machinery, 2007, pp. 375–384. [Online]. Available: https://doi.org/10.1145/1287624.1287679.

[11] K. Aggarwal, Y. Singh, and J. Chhabra, "An integrated measure of software maintainability", in *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318)*, Jan. 2002, pp. 235–241. DOI: 10.1109/RAMS.2002.981648.

[12] F. Khorram, J.-M. Mottu, and G. Sunyé, "Challenges & opportunities in low-code testing", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '20, New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–10. [Online]. Available: http://doi.org/10.1145/3417990.3420204.

[13] D. Dahlberg, *Developer Experience of a Low-Code Platform: An exploratory study*, eng. 2020. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-176361.

[14] T. Gulledge, "What is integration?", *Industrial Management & Data Systems*, vol. 106, no. 1, pp. 5–20, Jan. 2006. DOI: 10.1108/02635570610640979.

[15] N. Healthcare, *Healthcare integration engine | mirth® connect by nextgen healthcare*, en. [Online]. Available: https://www.nextgen.com/solutions/interoperability/mirth-integration-engine (visited on 02/15/2023).

[16] *Frends Integration Platform iPaaS Home*, en. [Online]. Available: https://frends.com/frends-integration-platform-ipaas-home (visited on 02/15/2023).

[17] *Arcesb*. [Online]. Available: https://arc.cdata.com/ (visited on 06/06/2023).

[18] *Apache camel*, en, May 2023. [Online]. Available: https://camel.apache.org/ (visited on 05/31/2023).

[19] *What is as2?*, en. [Online]. Available: https://www.seeburger.com/info/what-is-as2/ (visited on 02/15/2023).

[20] *Finvoice standard*, en-US, Feb. 2021. [Online]. Available: https://www.finanssiala.fi/en/topics/finvoice-standard/ (visited on 02/15/2023).

[21] N. Healthcare, *Healthcare integration engine | mirth® connect by nextgen healthcare*, en. [Online]. Available: https://www.nextgen.com/solutions/interoperability/mirth-integration-engine (visited on 02/15/2023).

[22] *Mirth Forums*, en. [Online]. Available: https://forums.mirthproject.io (visited on 03/28/2023).

[23] *Cdata knowledge base*. [Online]. Available: https://arc.cdata.com/kb/ (visited on 05/03/2023).

[24] *Cdata community*, en. [Online]. Available: https://community.cdata.com/?utm_source=arc.cdata.com&utm_medium=referral (visited on 05/03/2023).

[25] *Stackoverflow apache-camel questions*, en. [Online]. Available: https://stackoverflow.com/questions/tagged/apache-camel?tab=Newest (visited on 05/31/2023).