Expertise
and insight
for the future

Dmitry Molchin

# Secure Integration Transfer to Cloud-based ERP Using ESB Approach

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master’s Thesis

15 May 2021

## PREFACE

I am lucky to have written this study in 21st century, when a term procrastination is already existing and everyone knows its meaning. Otherwise, I would have to come up with a complex description of a undiscovered yet psychological term of in order to hide my natural laziness. Even writing this part of my thesis before finishing theory part, very clearly shows how bad I am in doing such scientific studies.

Procrastination is not the only challenge I faced with while writing this thesis, as I also had to change the topic of it at a very late stage due to changes at my work, by being assigned with a new role of integration lead for a new ERP implementation project.

That’s not all the challenges, though, as our lives are full of very attractive and much more interesting than writing a thesis things, that easily distract my curious nature from this boring and time consuming process.

During this study, I learned how to make sourdough bread, how to improve my average cycling speed and how to browse adult websites on my work PC without being noticed by the company’s IT security department. Also, I learned, or refreshed my memory on how to dig things deep and put them nicely in a Word document, playing with text size and indents to make it look longer than it really is.

If I had a dog, or a cat or a wild goat with evil red eyes glowing in the dark, I would thank them for helping me in doing this study, but unfortunately, I do not (anymore) have any of them, and the only creature being all this time with me together is my gorgeous and funny wife pushing me all the time to finish this document sooner. So, I thank her and the laptop I have written this on.

Espoo, 15 May 2021
Dmitry Molchin

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Dmitry Molchin<br>Secure Integration Transfer to Cloud-based ERP Using ESB Approach<br><br>37 pages<br>31 May 2021 |
| Degree | Master of Engineering |
| Degree Programme | Information Technology |
| Instructor(s) | Ville Jääskeläinen, Principal Lecturer |

This Master's Thesis was produced for Ramirent Finland Oy which core business is providing rental solutions for construction purposes.

The thesis covers the process of planning, developing and testing of integrations solution for the company's new ERP system.

The thesis describes why the company decided to switch to a modern cloud-based ERP and lists different types of ERP deployment methods, their benefits and challenges. It particularly focuses on securing the integrations of new-coming ERP system and making the process of handling existing data transfers as secure and cost efficient as possible. Different integration approaches and their limitations are also covered in this document, helping reader to understand why certain decisions had been made.

This thesis work describes different stages of building a new integration solution, familiarizes the readers with the chosen ERP's capabilities and technologies used in the field of data integrations. It also lists the finance functions that rely on integrations in the modern business world.

As a result, this document presents the final integration solution that has successfully been operating in production use for the last 8 months at the company.

This paper shows one of possible ways to plan and build the integration processes in medium sized company and does not declare that such solution is the only one suitable for all similar cases. However, the solution presented in this thesis has proven being robust, scalable and secure and does meet the company's requirements.

| | |
|---|---|
| Keywords | ERP, integration, ESB |

## Contents

## List of Abbreviations

| | |
|---|---|
| AD | Active Directory |
| AP | Accounts Payable |
| API | Application Programming Interface |
| AR | Accounts Receivable |
| CRUD | Create Read Update Delete |
| ERP | Enterprise Resource Planning |
| ESB | Enterprise Service Bus |
| FTP | File Transfer Protocol |
| GL | General Ledger |
| JSON | Java Script Object Notation |
| ODATA | Open Data Protocol |
| POJO | Plain Old Java Object |
| REST | Representational State Transfer |
| SEPA | Single Euro Payments Area |
| SFTP | Secure File Transfer Protocol |
| SOAP | Simple Object Access Protocol |
| SQL | Structured Query Language |
| XML | Extensible Markup Language |

# 1 Introduction

In 2018 Ramirent Finland Oy decided that supporting and developing their old custom-built 20 years old ERP is not efficient anymore. It was jeopardizing the business, not allowing it to go digital nor automate various processes.

Most companies have to replace or significantly upgrade their ERP at some point of time to keep up with modern trends and meet new business requirements. ERP replacement usually takes a lot of time before implemented and generates cost accordingly. But it is a long-time investment that can help business to grow and scale faster and/or avoid some unpredictable challenges with supporting older solutions.

ERP replacement projects can also include various improvements to current business processes and help automate routines and make human errors less likely to occur.

One of the most significant part of modern ERP solution is integrations with external parties and vendors. Poorly done it can jeopardize the business in different ways, and thus should be planned carefully and implemented wisely and precisely.

The goal of this project is to plan and develop a reliable and fully automated solution, with advanced monitoring and logging functions, that is also error-proof with a possibility to easily locate and fix issues. The plan is to utilize existing well working processes as much as possible, while automating the new ones in a similar way. In a long-term prospective, this will save the company a lot of money by automating many manual processes, enabling live data exchange and requiring less continuous development and maintenance.

The project can be divided into the following main sections:

1. Getting familiarized with the existing on-prem ERP and its integration solution
2. Getting familiarized with the new cloud-based Microsoft Dynamics 365 ERP
3. Conducting the Dynamics 365 integration capabilities' analysis
4. Collecting business requirements for finance processes' integrations
5. Designing, implementing and testing of the new integration solution

6. Evaluating the new integration solution and suggesting improvements based on more than 6 months production use

This paper will also introduce readers into several related topics, such as ERP systems and their types, integrations and their benefits and choosing and building up the appropriate solution for the company.

# 2 Technical Background and General Information

This section will introduce the readers into a topic of ERP systems and their types and their benefits for enterprises. As ERP systems cannot generally be on their own, the benefits of integrations are also covered.

This part also familiarizes the readers with the main technologies, standards and terms used in data transferring processes.

## 2.1 Introduction to ERP

Enterprise Resource Planning is a consolidated modular software that integrates various business functions and processes into a centralized system. ERP combines different software components, where each digitally represents one of an organization's main functions, such as human resources, finance and accounting, products, and materials, customer relationship management, purchasing, inventory management, and others. The choice of such modules can very much vary from one organization to another.

Usually, ERP solutions are very complex, as they unite many company's business departments into one unified system. When bringing clear benefits of managing the whole business logic from one program, ERPs are difficult to plan and implement, which therefore usually takes months to complete and are very often expensive.

### 2.1.1 ERP Deployment Options

On-premises ERP (shortly on-prem) is an ERP that is hosted locally on the company's own server environment, instead of third-party data center facilities or in the cloud.

Even though cloud-based ERPs are leading the market in the modern world, there are some industries and companies, to which data security and control are more valuable [1].

Benefits of on-prem ERP are:

- Security. The level of IT security can be as sufficient as possible with running the ERP on own servers. A company’s own IT security department is responsible for keeping data safe.
- Control. Companies in highly regulated industries, such as government, nuclear power, bank, and finance prefer and are even obliged to have a full control of data they own and manipulate, which is only possible with on-prem ERP.
- Modification. Having possibilities to apply new functionalities or build up new modules at any time with the shortest delay time are very crucial.

Challenges of on-prem ERP are:

- High upfront investment. Running an ERP on own servers usually requires huge hardware purchases, where a cloud-based system would offer a scalable solution.
- Longer implementation time. Running an ERP on-prem is still dependent on ERP’s vendor, especially when it comes to general updates and modifications.
- High ongoing maintenance. Running on-prem ERP usually requires a company to have a bigger IT department for maintaining and supporting it.

Cloud-based ERP is an ERP that is hosted on a third-party cloud platform, instead of a company’s own data center or server and a firewall. This deployment option is being very popular in the modern world when most of the software is moving to cloud computing.

Benefits of cloud-based ERP are:

- Cost. This is the most winning factor of choosing cloud before on-prem, as it is much more cost-effective to run large software solutions in the cloud, as companies pay only for what they use. Also, other most alluring factors of the cloud are predictable future price, and no need to invest huge amounts of money in hardware.
- Accessibility. This allows the company’s users to access the ERP from anywhere on pretty much any device.

- Stability. Cloud solution providers make continuous updates and take security actions with a very short delay and fix issues in faster and more professional ways.

Challenges of cloud-based ERP are:

- Less control. The company's admins will have restricted control of system updates and customization, as it is usually done by third parties.
- Less security. Since the data is stored on third-party's facilities, the company loses the ability to protect the critical data.
- Compliance. Some highly restricted industries must meet specific data compliances, and with cloud it is usually much harder to achieve this due to the previous two factors, control and security and, also, the fact that data stored in the cloud is usually replicated to other cloud provider's data centers around the world.

Postmodern ERP is a way to combine the best of on-premises and cloud ERP. It utilizes two ERP strategy types: administrative and operational splitting two main business process pipelines [2]. At a very high level, Postmodern ERP is a philosophy for creating a flexible IT infrastructure, in which different pieces of technology are adopted on their own merits and then networked intelligently with one another to promote connectivity. This can be applied to administrative ERP strategies and operational ERP strategies—with the latter being perhaps more relevant to manufacturers, shippers, freight forwarders, and other businesses that are dealing with a lot of complex moving parts. Smaller organizations, with more limited budgets and smaller IT departments, might find the work necessary to pull off a transition to a postmodern ERP environment too cost-prohibitive. If the new environment requires a great deal of manual intervention and data does not seamlessly flow from one system to the next, then a move to postmodern ERP might be a waste of time and resources. This type of ERPs is not that popular as the previous ones, but usually requires more efforts towards integration and is more expensive and complex to maintain.

### 2.1.2 ERP Integration

The main purpose of an ERP is to automate and consolidate business processes into a unified system and to operate as a centralized data hub. Many business processes are utilizing software from a variety of vendors or exchange data with stand-alone applications or legacy systems. One of the most important tasks during ERP implementation is to integrate isolated applications and provide a united application architecture [3]. By connecting ERP to other systems companies can ensure that consistent information is shared, and workflows are automated.

Benefits of integration:

- Automating processes and workflows. This reduces the number of manual jobs and the time it takes to complete various tasks. As a result, integration saves money and human resources to companies [4].

- Centralized data. Having all business data passing one hub can improve many business processes and reduce the time needed to get access to some business-critical data.

- Human-error reduction. This is achieved by automating business processes, as well-designed and tested software does not make typos and other critical mistakes.

Challenges of integration:

- Complexity. The more applications and services are in the integration scope, the more complex and harder to support the system becomes [5].
- Security. Having an ability to communicate to various vendors and stand-alone applications by using integration concepts increases the risk of potential data loss or system misbehavior.
- Poor implementation or data mapping. When a piece of data passes through an integration service, it is usually being modified or normalized. That is where an error might happen in a poorly designed code. As integration services usually transfer data to other applications, all of them will get corrupted information.

### 2.1.3 ERP Integration Approaches

There are several common methods used for implementing integrations in ERP. They vary from the easier to adopt, that are suitable for small organizations, to complex unified solutions, that are used mostly within mid-size and large companies.

Point-to-point integration. This approach is often considered as the simplest way to integrate ERP with external services. In this case, the ERP is individually connected to each service, which means that each additional tool or application will have a separate integration. That makes the integration process very complex and hard to maintain and support when the companies grow and the number of new vendors and software increase. Point to point integration is not considered the best solution for a long-term strategy, as it is a tightly coupled architecture that quite often becomes a nightmare as the time passes by. Somehow, this method is very popular, as it has a low entry barrier [6].

Custom integration. Some companies prefer to build their own adapters to have their ERP communicate with external services. This integration method shares the same benefits and challenges, as point to point integration, which makes it a bad solution for a long-term strategy.

Enterprise Service Bus or ESB. This approach is the most modern and advanced and is becoming more popular even within small organizations. Basically, an ESB operates as a middle abstraction layer and as such reduces dependences by decoupling the ERP from other applications and tools. As the bus is centralizing all the communication between all components of the large system, the applications and services do not have to be tweaked each time a new component is added. The ESB can handle data transformation, error and exception controls, queuing, and sequencing, and enforce proper quality of messaging between systems. With an ESB, one sets configurations rather than coding integrations.  The ESB also can set up a single point of failure bringing down all systems.  ESBs are also quite complex and require a high degree of maintenance to keep the configurations operating at an optimal level. At this time, most ESB software are expensive and used primarily by large businesses with a high volume of communication to be integrated.  Smaller and mid-sized businesses might not have the resources required to employ an ESB.

### 2.1.4 ERP Integration Strategy and Best Practices

There are several most important points that an integration architect must keep in mind while building up an integration solution for a new ERP:

- Secure all connections and end points. Losing business-critical data is the most harmful IT risk that modern companies face.

- Deploy new integration pieces step by step. Deploying new service integrations in phases gives the company a possibility to fine-tune the processes and optimize the system.

- Design a long-term solution that is easy to maintain, develop and support. Saving money on development on early stages might sound very appealing but might cost a lot in a long run.

- Make integration solutions scalable. Each business aims to grow, which also means that the number of tools will grow too. Making integration solutions scalable will pay back in a long run.

## 2.2 Microsoft Dynamics 365 as the EPR of choice

Ramirent's main business is rental services, which very much limits the ERPs to pick from, as most ERPs are designed to serve sales, not rentals. There are no ready to use ERPs for rental business on the market, and most rental companies use either their own or customized third-party solutions. Usually, it is one of the most popular ERP with a rental module integrated.

As Ramirent Finland Oy's ERP of choice was Microsoft Dynamics 365 Finance and Operations, which is a cloud-based ERP, the further work will only be focusing on this deployment option.

The main reasons, why particularly this ERP was chosen are the following:

- Proved history of using with rental business with an additional module provided by third-party company
- The complete business solution, combining finance and operations modules
- A cloud-based system, allowing to access it from basically any device with a web browser, and more important from mobile devices, such as mobile phones and tablets
- User-friendly interface, familiar from other Microsoft's products
- Powerful reporting possibilities with PowerBI module
- Easy integration with Microsoft other services
- Overall, being a company that uses a wide range of Microsoft products, choosing Dynamics 365 was a very logical and natural decision

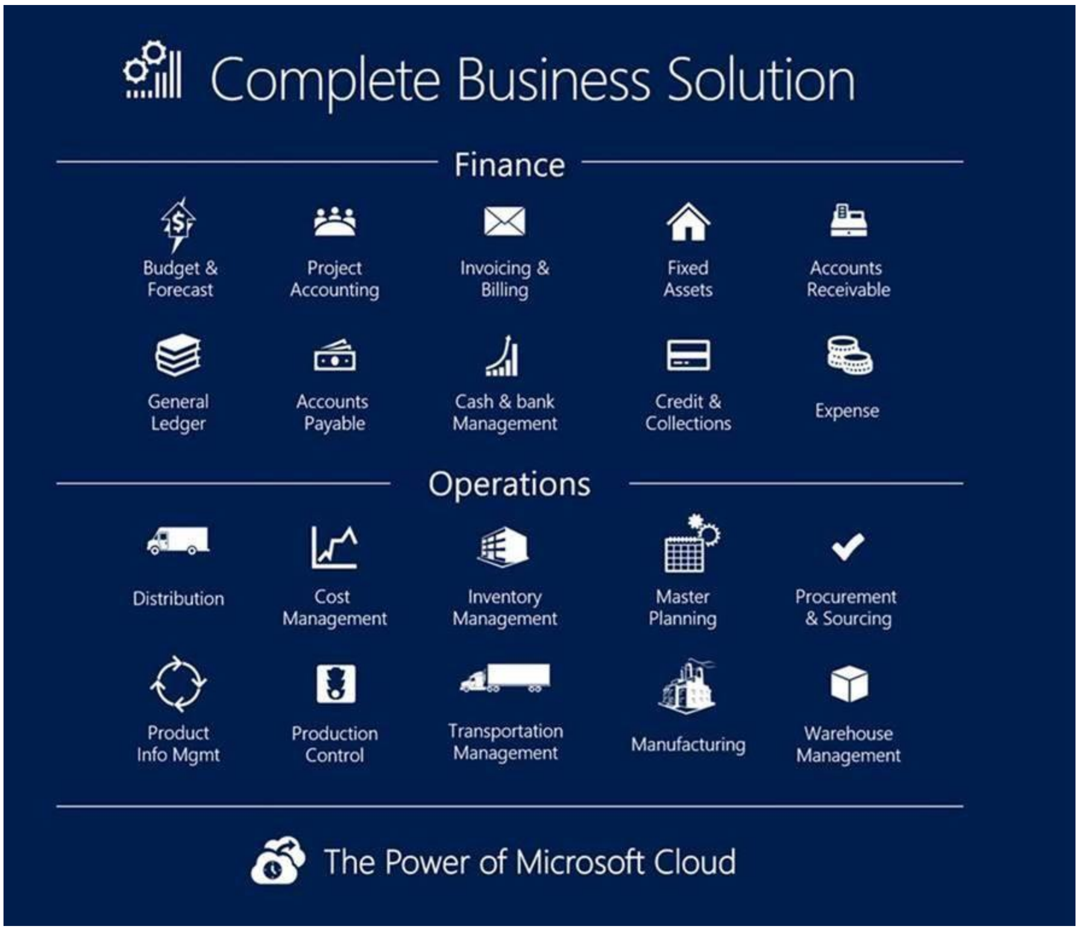


Figure 1. Microsoft Dynamics 365 Business Solution and its business components [7]

Dynamics 365 Finance & Operations is often the right choice for large, complex, or diversified companies that want to manage their finances, inventory, and more in the cloud.

### 2.2.1 Dynamics 365 Integration Scenarios

Dynamics 365 offers different integration patterns and scenarios. In Microsoft’s documentation, each scenario has data volume associated with it. Data volumes are used by developers and architects to evaluate the integration pattern they are planning to implement, as different integrations processes vary very much in data volumes going through them.

### 2.2.2 Synchronous and Asynchronous Patterns

Processing can be either synchronous or asynchronous. Often, the type of processing that one must use determines the integration pattern to be chosen.

A synchronous pattern is a blocking request and response pattern, where the caller is blocked until the callee has finished its operations and gives a response [8]. An asynchronous pattern is a non-blocking pattern, where the caller submits the request and then continues without waiting for a response.

### 2.2.3 Dual-write vs. Classic Data Integration Patterns

Dual-write provides synchronous, bi-directional, near-real time experience between model-driven applications in Dynamics 365 and Finance and Operations applications. Data synchronization happens with little or no intervention and is triggered by create, update, and delete actions on an entity. Dual-write is suitable for interactive business scenarios that span across Dynamics 365 applications. Dual-write pattern uses OData technology as the way of communicating.

Classic data integration provides asynchronous and uni-directional data synchronization experience between model-driven applications in Dynamics 365 and Dynamics 365 Finance and Operations applications. It is an IT-administrator led experience and one must schedule the data sync jobs to run on a specific cadence. Classic data integration is suitable for business scenarios that involves bulk ingress/egress of data across Dynamics 365 applications.

For real-time data integration using OData technology, Dynamics 365 has a data volume limitation of 1000 records per hour. Exceeding that amount might cause data inconsistency, that could cause serious problems for most of the finance processes. That always should be kept in mind by developers and architects when planning the integration solution for big data amounts.

### 2.2.4 ODATA (Open Data Protocol)

OData is a standard protocol for creating and consuming data. This approach represents a synchronous pattern of data integration. The purpose of OData is to provide a protocol that is based on Representational State Transfer (REST) for create, read, update, and delete (CRUD) operations. OData applies web technologies such as HTTP and JavaScript Object Notation (JSON) to provide access to information from various programs [9]. OData provides the following benefits:

- It lets developers interact with data by using RESTful web services.
- It provides a simple and uniform way to share data in a discoverable manner.
- It enables broad integration across products.
- It enables integration by using the HTTP protocol stack.
- The results of successful and unsuccessful data imports/exports are available immediately.

### 2.2.5 Recurring Integrations

It is based on exchange of files between the Finance and Operations unit and third-party services. Recurring integrations uses data entities to write and read data. It also supports different document formats such as XML and JSON, source mapping and filters. It allows transferring big data amounts but does not provide a feedback or acknowledgement message in a live mode.

### 2.3.1 Old Solution and Legacy System

The company’s old integration solution, which was connecting the old ERP to vendors’ services and other tools, was built up over the last 21 years and became quite a complex system with various legacy tools communicating with it and utilized 3 different integration platforms. These platforms were Microsoft Biztalk, Frends and Ramirent’s own ESB.

Biztalk was only used for payroll data integration between HR vendor and Navision and for sales and customer data update between the old ERP and Navision. Biztalk server was running on one of Ramirent’s service providers’ data center.

Frends, being a third-party product, was used by the company’s modern project management tool SARP and mobile apps Raimo and Ramismart integration purposes. Frends is a cloud-based enterprise integration platform designed for relatively easy to build services that require very little coding.

The ESB was a main and most used platform that was handling the biggest part of data transfer and data mapping between the ERP and vendors’ services and various internal tools and applications. As the ESB was Ramirent’s own platform, it was running in the main service provider’s data center.

All in all, there were more than 50 different processes that were in the integration scope with the old ERP and most of them had to be implemented within the new ERP as well.

Main integrations tasks were the following ones:

Finance:

- Account Receivables (AR)
- Account Payable (AP)
- General Ledger (GL)
- HR data
- Payment order

Project business:

- SARP

Mobile client for rental functions:

- Raimo

Web service for customers:

- RamiSmart

To summarize, there were 3 different integration products that were all running on different computing facilities, managed by different companies, developed by different development teams, and making high operational costs.

In case there was a problem in data integration, first, it had to be identified what integration platform it was utilizing, then finding out who is supporting it. These facts made the response time very slow and the whole solution not reliable.

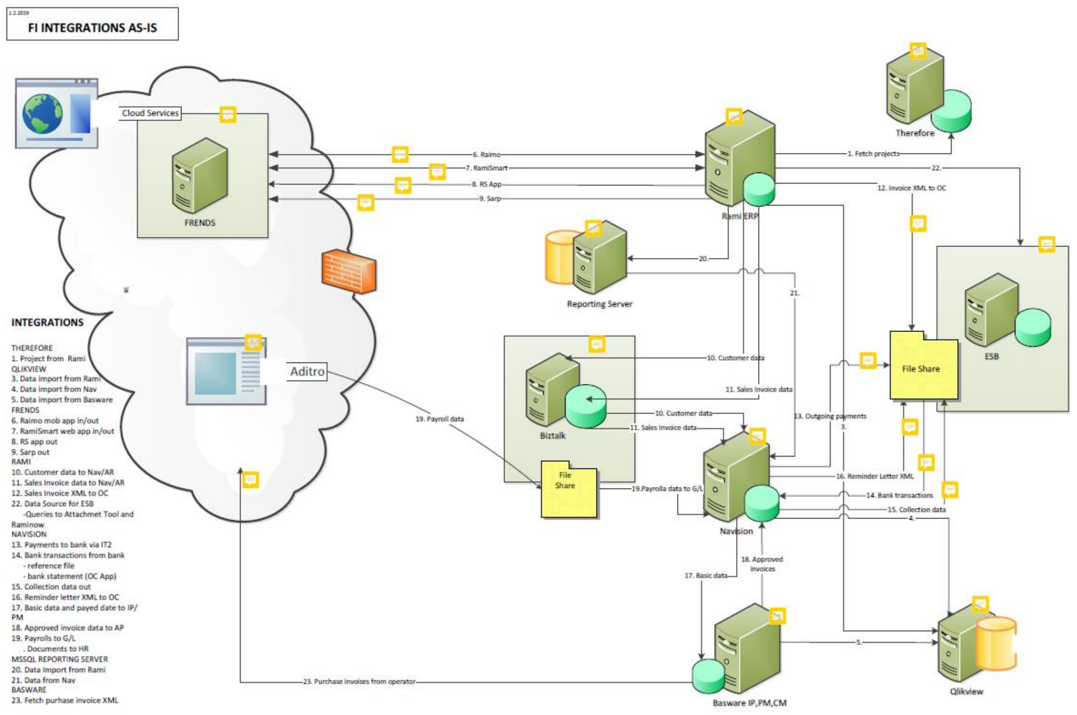


Figure 2. The old integration setup having 3 different integration services

As seen in the Figure 2, Biztalk was handling integrations with HR system Aditro (payroll, vacation pay and travel data - GL), customer and sales invoice data (AR) from RAMI to Navision and outgoing payments (AP). Frends was used to handle modern web services, such as Raimo, RamiSmart and SARP. Customer invoicing and collection functions were managed by the ESB platform.

### 2.3.2 Service Providers and Ways of Communication with Them

Ramirent has been using a wide range of service providers for different business processes. In the following Table 1 each service provider is associated with a business function.

Table 1. Service providers and corresponding business functions

| | |
|---|---|
| Aditro | HR data: payroll, vacation pay reserve, travel data |
| OpusCapita | Customer invoicing |
| OpusCapita Accounts | Bank statements with preliminary bookkeeping for GL |
| Intrum | Collection |
| OP bank | Bank statements |
| Basware | Receiving invoices from customers, purchase management system |
| NETS | Card transactions made on payment terminals |

Data volumes and formats for each business function are very different, which also affects the way data can be transferred from the ERP to the service providers. Some business functions might have only a few records that need to be sent or received daily when some might require hundreds of files and thousands of records. In the following table, approximate data volume, frequency and data transfer volumes are associated with the service providers.

Table 2. Data transfer frequency and volume per service provider

| | | |
|---|---|---|
| Aditro | 3 times a week | 3 files with several hundreds of records |
| OpusCapita | Real-time | Up to 200 files |
| OpusCapita Accounts | Once a day | 1 file |
| Intrum | Real-time | Up to 150 files |
| OP bank | Usually once a day | 2 files |
| Basware | Real-time | Thousands of files |
| NETS | Once a day | 1 file |

# 3 Project Planning

This section covers the analysis and planning stages of the project, and how it can be evaluated once delivered.

## 3.1 Requirements’ Collection and Analysis

Requirements’ collection is one of the most important processes of successful projects. It helps understand what the desired result of the project will be and describe how the end deliveries should address client's specific requirements. Quite often this part of project management is done poorly, which causes delays in the project and excess costs.

Within the scope of the project described in this paper, requirement collections started after the project was initiated. Integration specialists, doing business analyst’s work, had to gather the project requirements and convert them to technical tasks, and pass them to the development team.

Before starting to collect requirements, the first task was to identify the right stakeholders for each integration process. That information was received from the company's finance manager. Based on that list, the integration team made a preliminary plan for the next steps.

The main method used to collect requirements was mostly end user interviewing [10]. The interviews were conducted by the digital means, such as Teams and Skype, and by email communication. In many cases following some specific processes step by step was also a part of communication with end users and was done by sharing the screen.

### 3.1.1 System Analysis

Gathering requirements from end users is not enough to plan the whole integration solution, as there are also some process and technical limitations and requirements coming from the ERP system itself, and not taking these into account might make the whole requirement collection worthless.

The ERP service provider was the main source of guidance and knowledge base on the Dynamics integration functionality and requirements. In the very beginning stage of the project, they provided the company with various documents and process descriptions, but during planning and implementation stages they were contacted by the integration team daily. Responsibility zones on the service provider side were divided between several process specialists. The main division was the following:

- General Ledger and Accounts Receivable
- Accounts Payable
- Invoicing

As integration implies that there is some data transfer between two systems, our first task was to recognize the ways of doing it. Dynamics offers two main data transfer channels: REST API (OData) against the ERP database and file transfer for bigger data volumes. A direct connection to the Dynamics database is not recommended by Microsoft, as when done poorly it can corrupt the whole database structure and affect the whole ERP functionality. Basically, some integration processes only include several rows of data, and such are usually utilizing REST API technique.

### 3.1.2 REST API

API, Application Programming Interface, makes it possible for software systems to communicate with each other. As a human is interacting with UI graphics by using a mouse and a keyboard to input orders and eyesight to consume the results, programs use their own way to send and receive data with other programs.

With the use of API's, programmers are significantly more productive than they would be if they had to write the code from scratch. They do not have to "reinvent the wheel" with every new program they write [11]. They can instead focus on the unique proposition of their applications while outsourcing all the commodity functionality to APIs.

The API usage growth is driven by service providers that compete to address this thirst for greater developer productivity by packaging commodity and, often, complex functionality into easily reused API-based components. For each of the various types of functionality that can be invoked via API (such as credit card processing, mapping, navigation and translation), there are often multiple API providers vying for the attention of application developers. In turn, as more componentry is supplied in the form of API-based services, the API economy is accelerating the trend toward a world of applications that are primarily composed of off-the-shelf APIs.

The net result of the API economy's vicious and accelerating circle is that applications that once took months or years to build now take days or weeks or even hours. Developers are not only more productive, but the time it takes a business to make an application available to its customers is dramatically reduced. In turn, those customers benefit from shorter development cycles because the applications they use are being updated with new and innovative features more frequently.

The initially planned finance processes using OData approach included:

- Sending invoice data to a collection agency. This process included the following data fields: date, sum, invoice number, due date, and some other fields. Data volume - not more than 100 rows a day.
- Importing HR general ledger information, such as salary information, travel slips information and vacation pay reserve. Data fields included the following ones: name of persons, cost center, project number, sum, and some other fields. Data volume - not more than 300 rows two times a week.
- Importing bank statements general ledger information, basically all transactions for a previous working day with accounts information. Data fields included the following fields: debit/credit, sender/receiver, sum, account, and some other fields. Data volume - not more than 300 rows a day.

For larger data volume integrations, a file transfer over SFTP was used. The world's standard for data transfers in integration processes is XML format. It allows enterprises to consolidate, organize and store data in both machine and human readable form and helps to automate and integrate data transfers across a variety of systems.

XML stands for eXtensible Markup Language. XML is a self-descriptive markup language that is used to store and transfer data between different IT systems, and at the same time can be read by humans. XML is basically information wrapped in tags but compared to HTML it has no predefined tags.

Since, the XML is only a file type that follows a specific syntax model and has no predefined tags, there was a clear need to have some scandalization for particular business processes and data transfers. That is why the ISO 20022 standard was created. The standard sets the methodology, process, and repository to be used by all financial standards initiatives.

All in all, based on Microsoft documentation and the vendor's guidance, we found out that the following processes in Dynamics use XML format for exchanging data:

- Accounts receivables for bank statements accepts Camt.054 XML files for importing transactions received from the banks and the collection agency
- Accounts receivables for customer invoicing creates Finvoice XML files for storing information of outgoing invoices
- Accounts payables for vendor SEPA payments utilizes pain XML file for exporting outgoing payments

### 3.1.3 CAMT Standard

CAMT is an ISO 20022 Payment Message definition that stands for Cash Management and specifically covers Bank to Customer Cash Management reporting. CAMT reporting allows banks and customers to use a standard way of reporting account information globally and populate the information in a defined and specific manner [12].

CAMT structure:

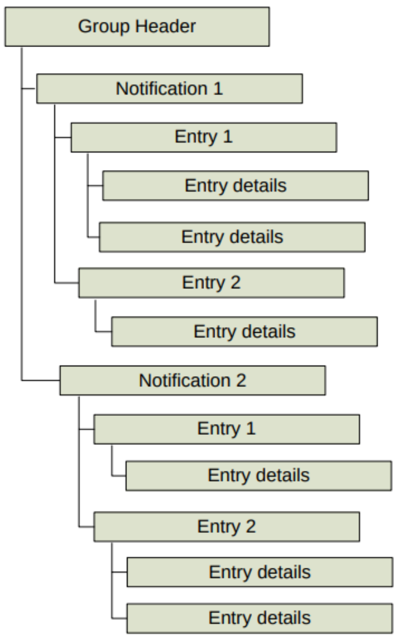


Figure 3. CAMT file structure example [13].

Group header is a mandatory block and can only be provided once in a file. It contains the date and time of the reporting message, and a message identifier.

Notification is also a mandatory block and can be provided more than once, depending on the number of accounts presented in the file. It has the information on the booked credit entries.

Entry is a mandatory part of the notification block and can be repetitive. It contains information related to the entry in the account, such as amount, booking date or value date and a bank transaction code.

Entry details is a mandatory part of the Entry and can be repetitive. This part contains detailed information related to the entry, such as references, amount details, related parties, and remittance information.

### 3.1.4 Finvoice

Finvoice is a publicly available invoicing standard published by Finnish banks. The Finvoice message can be used for invoicing and for other business messages, such as quotations, orders, order confirmations, price lists, etc [14]. Due to easy adoption, it is suitable for invoicing between companies of all sizes and for consumer invoicing. Finvoice is defined using XML syntax. XML enables the invoice to be represented both in a form understood by the application and, using a browser, in a form corresponding to a paper invoice. The browser representation of an invoice may be printed as a hard copy and processed in the traditional way.

### 3.1.5 Human Resources, Requirements and Limitations

Being a non-IT company working in a construction sector, Ramirent Finland Oy did not have an in-house development department. All software development works were sourced from third party companies when needed.

By analysing the new ERP technical aspects, the company’s integration scope and possible changes to the processes, it was concluded that the project cannot succeed without development work. Due to time limitations and a rental specific business model, the requirements to the development team human resources were very strict to succeed. The perfect candidate or candidates for this role would be a person who is familiar with the finance processes and technologies used in them, familiar with Ramirent business model and infrastructure and having experience in finance data integrations and technologies related to them.

## 3.2 Choosing the Suitable Approach

Having various requirements for such important processes as finance services integrations, the steering group of the project had prioritized the most crucial ones for the project to be successful. Those requirements were:

- Smooth, bullet-proof, and uninterrupted data transfers between the ERP and finance services providers. Even a small issue in a process can cause huge consequences to customers, vendors, and employees.
- The integration approach should be capable of handling data transfer technical standards, limitations, and requirements of Microsoft Dynamics 365 and at the same time support the current integration stack with finance service providers. Also, the new integration approach should be ready for future development requirements and not be limited only to the current scope.
- The integration approach should be modern and widely used in order to secure its support in the future and be cost effective in a long time prospective.
- The integration approach should be able not only to transfer data, but also to transform, enrich and modify it, store original and modified data for troubleshooting purposes and to be able to automate routine processes currently handled by hand.
- The development team should be able to pick up the business and system specific requirements and limitations in a fast manner, have experience working with finance processes and integrations and be flexible in implementing business requests.

Additional to the mentioned above, the modern enterprise integration best practices and the current solution were taken in consideration while making the final decision. All steering group members agreed that the ESB approach is the most modern, powerful, and flexible solution for Ramirent needs. Also, since the old ERP had utilized some functionality of the in-house ESB service with very positive feedback, it was decided to continue using that solution as the main and only integration middleware for the upcoming ERP.

The decision to continue using and developing the current ESB service was also beneficial because it already was successfully working with the most of financial service providers and that would save lots of time and money resources within a limited project timeline.

### 3.2.1 Technical Characteristics of the In-house ESB Solution.

For years Ramirent Group was using Oracle ESB solution, which was covering all company needs, but was costing the company several hundreds thousands of euros. The services it was providing were extensive for the company’s business demands, and several years ago it was decided to develop an in-house solution specifically customized to Ramirent’s requirements.

The internal ESB was developed by a small group of integration developers that used to develop and support the previous solution. That was economically efficient, allowing the company to save huge amounts of money in the long run and make a flexible and specifically customized product for every new business need or challenge.

The solution was written in Java utilizing some standard libraries with custom adapters for connections with external services. The architecture and approach were inspired by the world’s most used and valued ESB software, such as MuleSoft ESB and Apache Camel. Compared to those, the in-house ESB did not have a fancy user interface with possibilities to create integration processes by mostly using a mouse but was written in pure Java language and XML configuration files. Yet, the main idea was to create as much abstract integrations as possible to make the system and its logic easy to read and understand and to utilize the loose coupling approach between the ERP and vendors’ services.

### 3.3 Evaluating the Solution

Business software is made to solve business problems or routines, and deliverables should be measurable to evaluate whether the solution is successful or not. The main characteristics of a successful integration solution are smooth and uninterrupted data transfers without any data loss, minimal amounts of routine jobs and human interactions, ease of troubleshooting, flexibility, scalability, transparency, and cost efficiency.

In order to monitor and control some of the above-mentioned characteristics, there was a need to have a dashboard that allows business to react fast to any issues or data loss happening in the integration processes. The main purpose of the dashboard was not only to see currently running or performed integrations, but also to watch divergences between the original and processed data.

Some of the characteristics of a successful solution, such as flexibility, scalability and cost efficiency, can only be seen in the long run and are not measured constantly on a daily basis.

# 4 Solution Building

Based on the project evaluation, business needs and conducted interviews, the key integration processes had to be prioritized to meet the deadline limitations. Task prioritizing must be done for unexpected project progressing scenarios and in order to mitigate the risk for the most crucial business processes.

The money incoming and outgoing flows were granted with the top priorities by the project stakeholders. To make the company's incoming money flow uninterrupted, the customer invoicing must be well designed, implemented and thoroughly tested before the user acceptance test stage. For the outgoing money flow, payment orders to the payment factory were the main focus.

Some important decisions were made to provide the smooth and secure customer invoicing process transfer. The current outgoing invoice document format TEAPPS had to be kept in use to avoid redoing the whole process, as it allowed the development team to only focus on converting the Finvoice invoice files to the TEAPPS. Even though the invoicing service provider supported Finvoice format, the TEAPPS was a better solution as it was already mapped to the invoice image. That allowed the company to meet the deadlines, as such processes usually take lots of time and human resources. In other words, the main task of the integration was to transform Finvoice xml files to TEAPPS xml files. These two formats are compatible with each other and there is official documentation on field mapping available by Finance Finland organization.

Even though the converting seemed to be a straightforward process, there was lots of work done to have a smoothly running solution. Several invoice types had to be prepared and tested, each of them having different file structure and fields. Also, since some customers might have their own requirements for invoice fields being mapped against their invoice processing solutions, there were many changes done also on the ERP side.

With the outgoing payments, the development process was not that demanding, as the original file produced by the ERP and the file required by IT2 payment factory followed the same standard. It still required some changes done on the ERP side, such as netting customers' debit and credit invoices in advance and providing all required address lines, but those were just a matter of system configuration.

The rest of the integrations were developed parallelly and mostly utilized the API:s provided by vendors or the existing file transfer channels, such as SFTP.

## 4.1 ERP Environments for Test and Production

Any software development must go through a series of development stages that are defined in the Software Development Lifecycle (SDLC) methodology. The unique stages will include, requirements analysis, design of the software module, implementation or development of the software module, Testing of the software modules and continuous evolution of the software modules [15].

The last 3 stages - development, testing and continuous improvement (production) require their own software environments separated from each other.

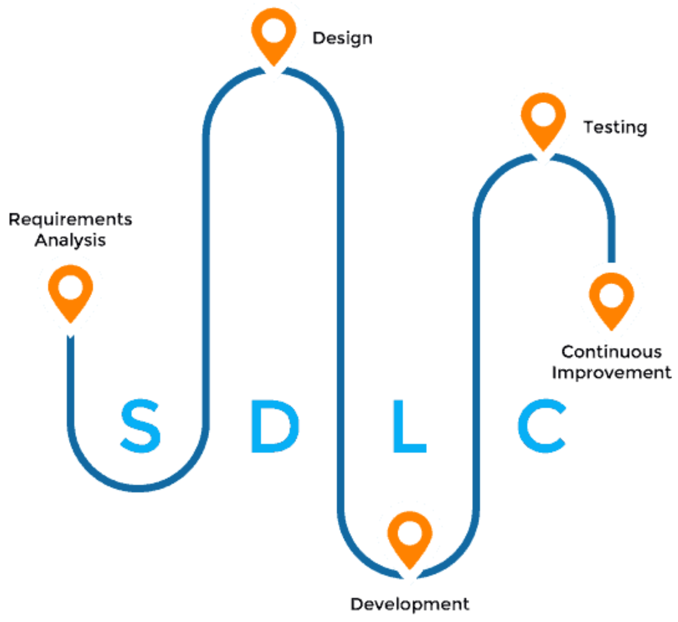


Figure 4. Software development life cycle's stages

1. Development - Only in use by the ERP vendor to deploy and test the ERP's core modules. The environment had no real data in it.

2. Test - Based on the Development ERP but populated with some migrated data from an old Ramirent ERP and also available to a limited group of Ramirent's process specialists, process leads and test users. The whole scope also included

the test ESB connection and a limited amount of vendors' web services. Not all vendors were able to provide test environments.

3. Production - Populated with all essential data migrated from the old ERP. The pipeline had the full scope of integration services set up both between the ERP and the ESB, and between the ESB and the vendors.

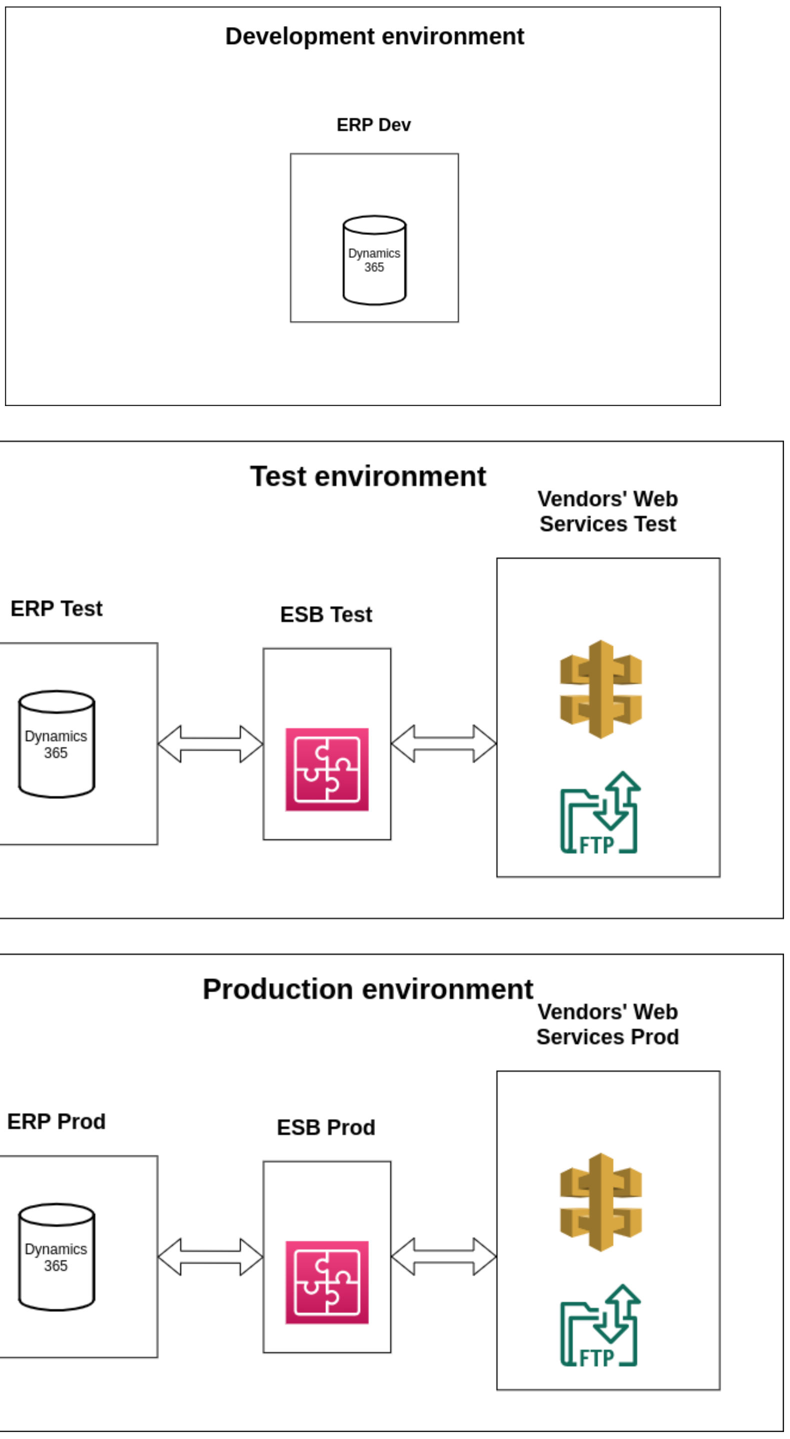


Figure 5. Difference in Development, Test and Production environments' setups

As the development environment was not available to the test user group, the integration workflows were only set up for the test and production environments. There are some main points that need to be kept in mind when setting up those environments:

1. The test and development environments need to be as similar to each other as possible, as the wider the gap between test and production, the greater the probability that the delivered product will have more bugs/defects.

2. Test user group should consist of real software system users, not its developers.

3. You need to configure bug or defect tracking tools well and there should be a process in place to manage their lifecycle.

Fixing bugs or defects is easier at the early stages of the software development process, rather than at later stages. It is also extremely expensive to fix bugs found in production, compared to fixing them in earlier stages such as the testing stage. This is the reason that test environment management is extremely important, and organizations need to take this seriously.

Since most integration processes were intended to continue using the current setups between the ESB and vendors' IT solutions, the main focus was to deliver a working data transfer workflow between the ERP and the ESB.

During system analysis and planning stage, it was concluded to utilize the 3 following integration channels for data transfers:

1. REST API approach that is built on the OData technology and Dynamics 365 entities. That is the only option provided by Dynamics 365 to have a live data transfer possibility. The data is provided in JSON format and can be easily interpreted by most modern web-services. Due to data transfer volume and API calls limitations, this approach was only used for the most live data dependent integrations and monitoring.

2. Recurring data, also utilizing REST API approach, that is built on file transfers followed by a scheduled import/export. That is used for the processes that do not depend too much on live data and immediate feedback for erroneous transfers.

The data can be provided in different file and data formats, such as CSV, JSON, XML and others.

3. SFTP file transfer for bigger data volumes and file-based integration. The file format is XML. This approach was mostly used for billing and collection integrations, that could have several thousands of files transferred in one day.

## 4.2 Authentication to Dynamics 365

Authentication against API's is done utilizing "key+secret" approach, instead of username is password, even though the later method is still used although not recommended. Basically, "key+secret" authentication is very similar to "username+password", but has some differences and advantages over it:

1. When "username+password" method is usually utilized by a user (=person), "key+secret" is used by a service (=application). A key can be used to authenticate an application accessing the API, without referencing an actual user.

2. API keys are usually much longer and more complex, which prevents guessing it and helps against brute force attacks that use dictionaries of usernames against the API.

3. Prevents usernames being compromised in case they are public.

Before creating a key+secret for the ESB, an Azure AD account needs to be set up first. After it is created, the user needs to be created in Dynamics 365 and granted with access rights that allow it to make API calls and read and write data from/to entities. When a user is created in Dynamics365, the system automatically generates an unique ID to it. Then, the administrator must create a secret code for that client ID and choose a period that secret is valid. Each authorization to the API requires an access token that needs to be first received from Azure Active directory. This method is called "Authorization code grant flow" [16].

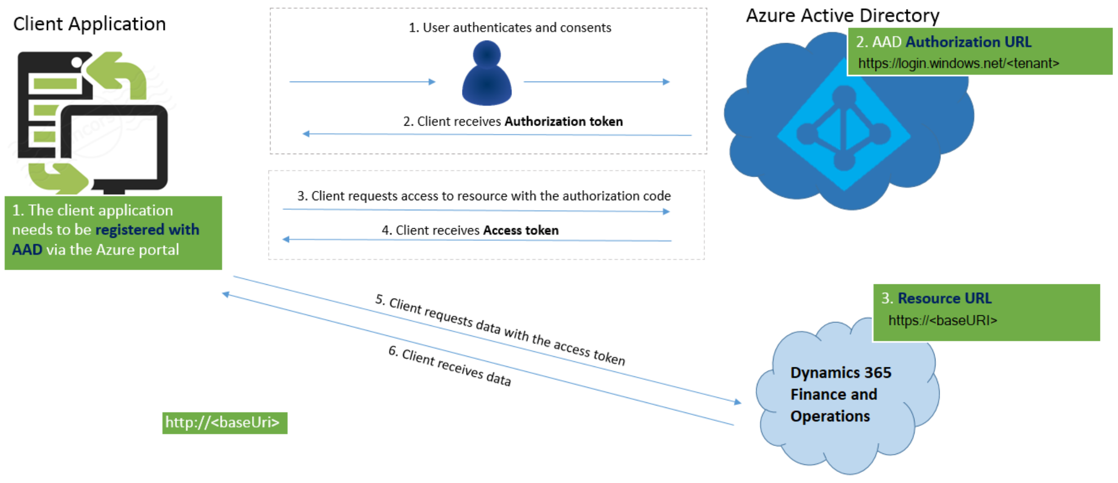

Client Application
1. The client application needs to be registered with AAD via the Azure portal
1. User authenticates and consents
2. Client receives Authorization token
3. Client requests access to resource with the authorization code
4. Client receives Access token
Azure Active Directory
2. AAD Authorization URL
https://login.windows.net/<tenant>
5. Client requests data with the access token
6. Client receives data
3. Resource URL
https://<baseURI>
Dynamics 365 Finance and Operations
http://<baseUri>

Figure 6. Authorization workflow from an application to Dynamics 365

Once keys and secrets are created, there needs to be a connection set up on the ESB side and tested afterwards.

## 4.3 Smoke Tests

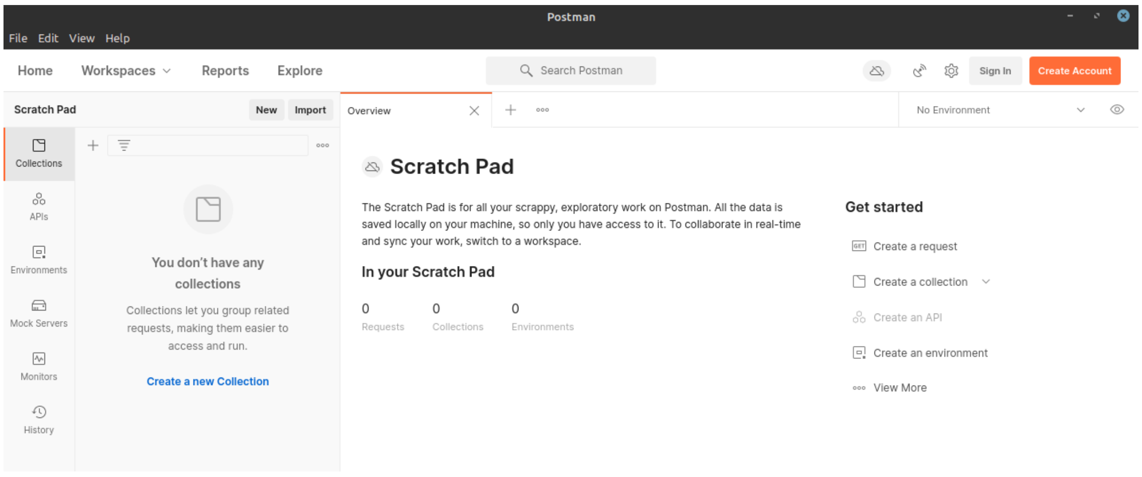
Smoke testing, also known as “Build Verification Testing”, is a type of software testing that comprises a non-exhaustive set of tests that aim at ensuring that the most important functions work. The result of this testing is used to decide if a build is stable enough to proceed with further testing. It can also be used to decide whether to announce a production release or to revert. The term ‘smoke testing’, it is said, came to software testing from a similar type of hardware testing, in which the device passed the test if it did not catch fire (or smoked) the first time it was turned on [17].

The smoke testing performed against the test environment, was basically consisting of several GET and POST requests from a couple of entities exposed to the OData interface.

While GET requests worked as they were intended to, the POST requests were more challenging, as not all entities were available for writing to them, and requests body were supposed to follow a very strict specific structure.

One of the benefits of using the OData approach, is the ability to immediately get the acknowledge messages of successful deliveries and error messages for the rows that were not written to the database. Such errors can happen when a non-existing parameter is being fed to a field accepting only predefined values. For example, when a vendor's data has a cost center that is not registered in the ERP, that will raise an error and the whole row will not be imported. That is usually a matter of data mapping or mismatch between data provided by vendors and data registered in the ERP. Due to immediate erroneous rows report, such issues could be fixed manually by a person responsible for the business unit data is being fed in, which is not the best solution, as it leaves a room for human error. The best way to overcome these issues is communicating the correct parameters to vendors in advance and making the predefined parameters “synchro-nized” between organizations involved.

There are several tools to perform API tests on the market, and the most popular ones are Postman and SoapUI. As Postman is frequently updated with new features and bug fixes and has a modern interface and a possibility to have variable collections, it was chosen for Dynamics’s and vendors’ API testing. Postman use cases are not only limited for testing already existing API’s, but also for API design and development, monitoring, automated testing and creating documentation. It also supports all kinds of API’s: SOAP, REST, and plain HTTP, has a broad community and extensive documentation, has desk-top and online clients and is free.



Scratch Pad

The Scratch Pad is for all your scrappy, exploratory work on Postman. All the data is saved locally on your machine, so only you have access to it. To collaborate in real-time and sync your work, switch to a workspace.

In your Scratch Pad

0 Requests 0 Collections 0 Environments

You don’t have any collections

Collections let you group related requests, making them easier to access and run.

Figure 7. Postman application’s user interface

Postman application allows to create collections, which makes it much easier and safer to test requests against different software development environments (Development, Test, Production, etc.). It also allows to create environments that store variables that makes working with the API's much more user friendly, as long and complex URL's and parameters can be hidden behind more easy-to-read variable names [18]. Example: when working with different API endpoints, that have the same beginning part the same, it can be stored in a collection by just a short self-describing word: *https://testenvironment.xyzwebservice.com/data/* can be used by just calling a "testurl" variable. When using existing variables in Postman, they are put in double curling brackets: {{testurl}}.

Postman also allows to create some scripts written in JavaScript and using Postman's own library to automate some routines. For example, it can save an authorization token and automatically put it in a token variable in a collection, which saves some time and makes the authorization process automated.

Different API's require different requests, parameters, and request bodies. Usually, this kind of information is found on the products' web documentation. Microsoft has well covered documentation for configuring API requests both for OData and recurring integration approaches. As all the Dynamics integration data flows are based on reading or writing from and to entities, it is just a matter of parameters, a body and a type of request provided for each particular integration workflow and data mapping for XML-body based recurring integrations.

Running smoke tests and documenting the results make the way of communicating with developers very much improved and speeded up, as they do not need to spend time doing initial troubleshooting since that can be done beforehand, providing already tested and defined sets of data that must be provided for each particular integration job. The developers then just need to develop the process in a chosen integration platform and a programming language and orchestrate the whole set of processes together.

Communication with developers is usually done in some of modern communication and collaboration platforms, such as Teams by Microsoft or Slack by Slack Technologies. Both have quite similar functionalities, but Teams is getting very popular since most enterprises are using Microsoft tools and Teams is well integrated with them and Windows OS and is a part of some versions of enterprise Microsoft Office packages. During the project Slack was mostly used for messaging services and sharing documents, when Teams was a main way to have voice calls and meetings.

Additional to collaboration tools, there is also a need for a solution to register, control and document tasks that are then assigned to the development team. It helps to have a structured and transparent way to keep track of what is delivered and what is to be delivered as a software product. One of the most popular tools for that is Jira by Atlassian. It allows stakeholders to plan, track, and manage agile and software development projects. Jira uses terms epics, stories, and tasks, where epic is a large body of work that can be broken down into a number of smaller stories, or sometimes called “Issues”, a story is something that is generally worked on by more than one person, and a task is generally worked on by just one person. For the project there were created one epic for the whole project and tasks for each particular integration process.

# 5 Results

This section describes the delivered integration solution with technical details included. It also covers the solution’s evaluation based on several months of production use.

## 5.1 The In-house ESB Solution

For several years the company had been using a popular commercial ESB product, that was costing several tens of thousands of euros a year, which was lately replaced with an in-house solution developed by the integration development team. The solution was written in Java programming language and its architecture and functionalities were inspired by such integration services as Apache Camel and Mulesoft.

The company’s ESB solution is characterized by a publisher/subscriber pattern architecture, leveraging the JMS middleware API and POJOs as integration points.

JMS - The Jakarta Messaging API (formerly Java Message Service or JMS API) is a Java application programming interface (API) for message-oriented middleware. It provides generic messaging models, able to handle the producer–consumer problem, that can be used to facilitate the sending and receiving of messages between software systems.

POJO - is an ordinary Java object, not bound by any special restriction.

The input/output solution is primarily relying on file storage (File servers with FTP clients) and on Web Services. The POJOs are processing the files inbound and outbound through the usage of XSLT pipelines. The logging is happening at the level of the integration server, POJO observer, JMS schedule.

XSLT (Extensible Stylesheet Language Transformations) is a language for transforming XML documents into other XML documents.

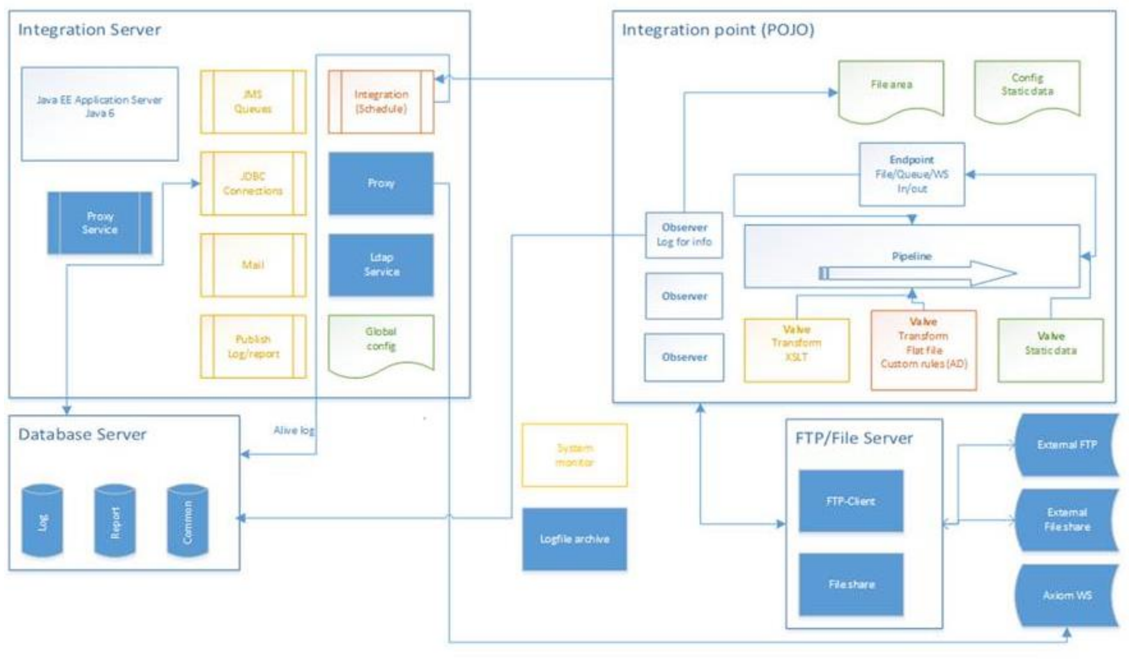


Figure 8. The ESB solution’s infrastructure

A load balancer (Kemp) is used to direct the request to the right servers based on the origin of the request. Observers have a configured frequency based on the desired frequency as per business requirements.

Business logic can be implemented in the pipeline when the ERP is not flexible enough to do so. Sizing of the messages is configured based on the payload size and total batch size. Few large files with small messages can be produced or many small files with small messages. Archiving and log files at the level of the FTP/files server is performed.

The ESB is deployed to two main environments:

- Test environment
  - ESB Server
  - SQL Server
- Production Environment
  - API server
  - ESB Server
  - Failover ESB server

    - SQL Server cluster
- File shares and sFTP servers
- FTP server manager
    - Silvershield SFTP Server
    - FTP ESB
    - ESB FTP manager
    - Fileserver. Qlik.
    - Infrastructure
- Load balancer: Kemp

The ESB is running on a Windows server, but can be run in any other server OS supporting Java.

There is an integration between the ESB and the company's Active Directory and shared disc resources, that made access control and management very easy to support.

As the ESB is written and being operated in Java and XML files for describing pipelines, configurations, and data mapping, it does not have any user interface nor a possibility to create and edit current integration processes visually using drag and drop technique that such tools as Mulesoft can provide. This makes the ESB usage very limited to the development team only.

## 5.2 Solution Evaluation

This part of the thesis work is being written 6 months after the new ERP solution was delivered. Half a year is a sufficient period to evaluate the delivered product, its flaws, and a need in future development. The system ever since has been used daily and has successfully passed a year closing and quarters' closing procedures. The core ERP has had a few bigger updates by Microsoft and various changes and updates by the vendor. The integration solution has also had various updates to almost all the processes involved and is still under continuous development and improvement.

All in all, the integration solution delivered has proved its efficiency, flexibility, and reliability, also helping to focus all finance data transfers to a single product that improved the system monitoring and control.

Due to very comprehensive and precise planning and testing processes, there were no serious issues in the ESB itself. However, the first weeks after the ERP was taken in production use had brought some challenges and ideas for improvements, that caused minor and medium changes to several integration pipelines.

The solution that has been delivered has proved to be very scalable, as bigger data volumes after going live with the new ERP did not affect its performance. There are some days in a month, usually a middle month and end/beginning of a month that have bigger incoming and outgoing data volumes due to the invoicing and payment cycles. Sometimes there could be more than 15000 invoices sent in one batch. Such volumes run as smoothly and stable as the smaller ones.

Also, the integration solution can easily be extended to support new software products and data transfers. There were at least 2 new integrations implemented after the system's launch, that were not in the original scope of the project. Those were implemented in the same manner as the previous ones, and that did not affect the ESB performance either.

The system is very accurate in data manipulations and transfers, and there were not discovered any data loss ever since the system went live.

# 6 Conclusions

This part focuses on defining whether the targets of the project were met and whether its deliverables fulfilled the business requirements. It l also interprets the results from the research point of view and suggest the future development improvements to the solution.

## 6.1 Meeting Targets

The main goal and business requirement of the project was to build up a reliable and fully automated solution with wide monitoring and logging solution and no data loss. At this point it can be concluded that all these goals were met, as the solution has been operating reliably for more than 8 months without any down times nor other issues. The ESB does not require much maintenance nor continuous development which makes it cost efficient. It is flexible and capable of implementing new process without much development, utilizing modern API techniques. It was also concluded, that the ESB's scalability is sufficient enough even for unexpectedly big data volumes.

## 6.2 Research Prospective

Being a non-IT company with traditional "waterfall" project approaches, this particular project brough a lot of new knowledge and best practices to Ramirent. All project participants were able to adopt agile techniques in their daily tasks, which has helped to look on business processes from another perspective.

By doing very detailed analysis of the modern integration practices and the Microsoft Dynamics technical specifications, it can be concluded that the ESB approach is possibly the best way to handle data transfers and modifications for medium-sized companies.

Modern API economy allows to integrate any systems together without spending much time on development. There are many tools on the market that allow people without much programming background testing integrations and doing MVPs.

## 6.3 Future Development

Living in the “API economy” era, it makes it very easy to expand the system to support new software services and data sources. Adding a new system to be integrated with the company’s ERP is usually just a matter of data mapping and configuring API credentials at the ESB side, since business logic is better to happen outside the ESB, better on the source system’s side.

Thus, the system’s scalability and expansion are limited to the server performance and data manipulation scheduling, more than on the ESB flexibility itself.

However, there are some improvements and changes to be done in the future to the integration solution:

1. Monitoring for active integrations and their dependencies. Sometimes some API’s do not respond, or database connections do not work. Indications of these can be added to the dashboard.

2. Department-specific dashboard views with AD security groups authentication. Currently there is only one dashboard for all users, which allows them to view data they are not supposed to see.

3. Visual programming and configuration for integrations. As all the logic is currently written and supported in Java language, it makes development of the integrations very limited to Java programmers. Making a visual interface for adapters and integration logic would speed up the integration development and allow users without programming knowledge support the system and make changes.

## References

1 Adam Hughes, *The Importance of ERP System Integration*, https://www.cleo.com/blog/erp-system-integration Accessed May 31, 2021

2 Panorama Consulting Group, *Is Postmodern ERP Right for Your Organization*?, (Feb 18, 2019) https://www.panorama-consulting.com/is-postmodern-erp-right-for-your-organization/ Accessed May 31, 2021

3 MuleSoft, *ERP Integration - A united application architecture*, https://www.mulesoft.com/resources/esb/erp-integration-application-architecture Accessed May 31, 2021

4 Lindsey Jenkins, *ERP Integration: Strategy, Challenges, Benefits, and Key Types,* https://www.selecthub.com/enterprise-resource-planning/best-erp-software-integrations/ Accessed May 31, 2021

5 Zach Hale, *The Benefits, Challenges, and Best Practices Associated With Postmodern ERP Integration*, (May 15, 2019) https://www.softwareadvice.com/resources/erp-integration/ Accessed May 31, 2021

6 Tom Miller, *A beginner's guide to ERP integration*, (June 17, 2019) https://www.erpfocus.com/beginners-guide-to-erp-integration-1736.html Accessed May 31, 2021

7 i-Neti, *Benefits of Upgrading to Microsoft Dynamics 365 for Finance and Operations*, (October 8, 2019) https://i-neti.com/stories/benefits-dynamics-365-for-finance-and-operations/ Accessed May 31, 2021

8 Microsoft*, Integration between Finance and Operations apps and third-party services,* (Nov 23, 2020), https://docs.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/data-entities/integration-overview Accessed May 31, 2021

9 Microsoft, *Open Data Protocol (OData)*, (June 19, 2020) https://docs.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/data-entities/odata Accessed May 31, 2021

10 TutorialsPoint, *Requiremt Collection*, https://www.tutorialspoint.com/management_concepts/requirement_collection.htm Accessed May 31, 2021

11 David Berlind, *How Web and Browser APIs Fuel The API Economy*, (Dec 3, 2015) https://www.programmableweb.com/news/how-web-and-browser-apis-fuel-api-economy/analysis/2015/12/03 Accessed May 31, 2021

12 SEPA for Corporates, *The Difference Between a camt.052, camt.053 and camt.054 3*, (Feb 27, 2018) https://www.sepaforcorporates.com/swift-for-corporates/the-difference-between-a-camt052-camt053-and-camt054/ Accessed May 31, 2021

13 Handelsbank*, ISO 20022 Credit Notification camt.054 version 2*, (Mar 20, 2019) https://www.handelsbanken.se/shb/inet/icentsv.nsf/vlookuppics/a_filmformatbeskrivningar_fil_iso_20022_xml_camt_054_ver_2_credit_notification/$file/iso_camt_054_001_02_credit_notification.pdf Accessed May 31, 2021

14 Finanssiala, *FINVOICE E-INVOICING STANDARD,* (May 24, 2021) https://www.finanssiala.fi/en/topics/digital-services-and-payments/finvoice-standard/ Accessed May 31, 2021

15 Plutora, *Test Environment Management Best Practices*, (Nov 23, 2020) https://www.plutora.com/blog/test-environment-management-best-practices Accessed May 31, 2021

16 Microsoft, *Service endpoints overview*, (Jun 22, 2020) https://docs.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/data-entities/services-home-page Accessed May 31, 2021

17 Software Testing Fundamentals, *Smoke Testing*, (Sep 2, 2020) https://softwaretestingfundamentals.com/smoke-testing/ Accessed May 31, 2021

18 Postman software, https://www.postman.com/ Accessed May 31, 2021