Abstract—This thesis describes the modernization of a legacy system called Transporter, which is a transport and logistics Enterprise Resource Planning (ERP) implemented by MAEIL. Transporter is a client/server application which has business logic in both parts. MAEIL intends that Transporter becomes more user friendly and easier to develop for it. In order to achieve this goal, it is needed to create the CargoAPI, an Application Programming Interface (API) to expose services. These services will allow the communication between Transporter’s functions and a web browser user interface. Along this thesis, we describe three possible solutions to implement this API as well the pros and cons of each solution. Transporter is an application with a client-server architecture with business logic present on both sides. In this thesis it is also described how the integration is made between Transporter and CargoAPI, how the API integrates with the database, and how requests are made in order to communicate with CargoAPI. We also propose a Representational State Transfer (REST)-like query language as a template to show the different combinations of functionalities provided by the Transporter system.

Keywords—Transporter, Legacy Systems Modernization, API, Integration.

I. INTRODUCTION

Legacy systems are outdated systems that use outdated software. These systems do not have the possibility to upgrade the software to a new version because, among many reasons, there is a fear that the upgrade can disrupt the business. Legacy systems can be a problem because of the following reasons:

- Employees retire, so finding new people to work with old technologies is hard;
- New requirements, e.g., mobile access, are impossible, or very difficult to implement. Occasionally, a legacy system does not fully meet the needs of a client for the interface falling into disuse.
- There are high license costs for the hardware/software and their maintenance
- There is no support available anymore
- Security flaws

Replacing legacy systems is not an easy decision to make, because since these systems usually have years of development, they have enough business logic implemented, which makes it necessary a lot of time and resources to rewrite everything from the beginning. Since companies want to reduce costs, instead of replacing the legacy system they modernize the system. This approach takes less time and consists in creating a well-designed interface known as API [1]. An API is a function library or a REST or Simple Object Access Protocol (SOAP) service specification that are used to access web services, allowing to expose functionalities to other applications. In short, this allows having a newly user interface that communicates with a API which exposes features from a legacy system.

A. Problem Description

Transporter is a legacy system with 17 years of development by MAEIL implemented in a tool called Computer Associates (CA) Plex. This tool is falling into disuse, which does not allow Transporter to follow the evolution of technologies. Since it is an outdated tool, it is also difficult to find programmers who already know how to work with it. Besides, CA Plex has a gentle learning curve. Another limitation is Transporter’s current architecture, which requires it to be installed locally on each computer. In order to follow modern technologies, MAEIL intents to have a web browser interface to replace the current one.

II. TRANSPORTER

Transporter is an ERP focused on transport and logistics, certified by Autoridade Tributaria e Aduaneira (AT) that manages logistics processes. There are more than fifty facilities in Portugal, Angola and United Kingdom. This software has hundreds of entities in its relational model and thousands functionalities in each of its modules. It is developed by MAEIL in CA Plex using a client-server architecture.

CA Plex is a Rapid Application Development (RAD) platform which supports modelation, report creation and code generators in a single environment with its own repository that allows to manage the entire code and database structures. This platform allows the creation of client-server applications without having the need to worry about communication. Transporter is configured to be generated in C++ language and therefore compiled into DLL files.

The Transporter domain model is divided in seven modules: Accounting, Equipment, Logistics, Operation, Sales, Shipping and Electronic Data Interchange (EDI):

- The Accounting module is responsible for billing management.
- The Equipment module is where all types of equipment are managed, such as the demurrage management that refers to the exceeded time of an equipment in a given port.

1 www.transportersystems.com
2 www.maeil.pt
- The Logistics module allows the management of cargo.
- The Operation module is responsible for vessels and voyages.
- The Sales module allows you to create quotes, account management and Customer Relationship Management (CRM) activities.
- The Shipment module is responsible for the management of transports, booking processes and shipments.
- The EDI module is transversal to all previously mentioned modules where all the electronic documentation that can be imported or exported is defined. There is another module that is not intended for the user, but only for administrators to manage users and define paths for reporting.

A. Current Architecture

In this section I will describe the current Transporter’s architecture taking into account the following views: Module Decomposition View, Component-and-Connector View and Allocation View

The current decomposition model view (figure 1) shows how the Transporter is divided into three modules: Client, Server and Database. On the client side, we have the PNL files that are the user interface panels and the code that support these panels in client DLL files. Local files (.INI) in the client and server modules are used for configuring paths. On the server side is where most of the business logic lies in the server DLL files. The Dispatch service manages applications in run-time, therefore, it is responsible for all communication that is performed between the client and server side. Finally, in the Database module are the tables, views, which support the data that are registered and the Table-Valued Functions that are used to perform information searches on the tables/views in Transporter.

The current component-and-connector view (figure 2), shows how the Transporter connects its modules. The communication between the client and server is handled by Remote Procedure Call (RPC) which, in turn, is supplied together with the C++ server generator. RPC is a programming interface that allows you to perform invocations of remote procedures. All of this communication is invisible to the programmer because of the way the application is developed in CA Plex. Both client and server functions are drawn automatically as RPCs. In this way, all remote invocations will appear as if they are local invocations. Communication with the SQL Server database that is performed with the server side is through an Open Database Connectivity (ODBC) driver.

All the functions generated by CA Plex, except for panels, can be called and controlled through client controllers using the OLE Automation protocol. This protocol was the first Component Object Model (COM) integration engine provided for the CA Plex. Object Linking and Embedding (OLE) is the old name used by Microsoft that was replaced with COM. This mechanism allows other development environments and other programming languages to invoke CA Plex functions.

A possible integration with the Transporter is through two .EXE files, Ob600RS.exe and Ob600RC.exe, which allow communication with server and client side respectively.

In the figure 3 we have the current Transporter’s allocation view. This whole application is installed on a machine, in this case, in a virtual machine called PAVA where Transporter can be executed on it. There is the possibility of having more computers, in the same local network, running Transporter on the same database. To do this, you must have a local installation on each computer with the local configuration files pointing to the DLL files on the machine where PAVA is located, with the locations of those files shared in the local network.

III. Existing Solutions

In this section I will describe different solutions for the modernization of the Transporter. There are three possible workarounds for implementing the API: Services Windows Communication Foundation (WCF) generated by CA Plex where it is described how to generate WCF services through CA Plex and then compiled With the use of Visual Studio; A solution in which all business logic is rewritten for stored procedures and/or table-valued functions with a service layer WCF or REST implemented in Visual Studio; A solution REST implemented, also in Visual Studio, integrated with the Transporter through an .exe file provided by CA Plex that allows communication with existing Transporter functions. It is important to note that any of the solutions listed above allow other user interfaces to communicate with Transporter as long as they support web services.
A. Solutions Architectures

In this section I will make a description of the architectures of each of the possible solutions.

1) WCF Services generated by CA Plex: The architectural difference of this solution with the current architecture described in the II-A section is that the Transporter functions happen to be in the form of packages generated by CA Plex. The structure of the current database is maintained and two new modules are added: A web browser and services WCF, also generated by CA Plex (see figure 4). The new web browser interface will communicate with the API WCF that is generated by CA Plex containing the services to communicate with Transporter functions (see figure 5). At the allocation level (see figure 6), the entire server part (API in Internet Information Services (IIS), functions in packages and database) may be within the same machine and a user can access the application by having a browser to access the new interface from any other machine.

Fig. 4. WCF Services Module Decomposition View.

Fig. 5. WCF Services Component-and-Connector View.

Fig. 6. WCF Services Allocation View.

2) Data Centric Solution: Portal Transporter: This Portal Transporter solution discards any function generated by CA Plex, just accessing the Transporter database directly and performing operations on it (see figure 7). This project made in ASP.NET has two modules: Client and Server. On the client side, we have a web browser interface and a database of users of this same interface. On the server side, we have queries that are performed on the Transporter database.

At the level of the component-and-Connector view (see figure 8) and allocation view (see figure 9), it is similar to the solution referenced in the subsection III-A1, where users will have a web browser interface that communicates with the serving party. The difference is because there is an additional database that keeps the users information.

Fig. 7. Portal Transporter Module Decomposition View.

Fig. 8. Portal Transporter Component-and-Connector View.

Fig. 9. Portal Transporter Allocation View.

3) REST API integrated with Transporter: In this section I will explain how Transporter architecture is integrated with a API. As it is intended to replace the current user interface, the difference between the current architecture and this one is that the PNL files are replaced with a REST API integrated with the Transporter. This integration allows you to have a modern web browser interface that communicates with Transporter through this API.

The differences between the decomposition module of this solution and the current one (see figure 10), is the addition of two modules: the web browser interface and REST API. In the "Client" module, the PNL files were removed. In module REST API we have controllers, models and a set of classes which we can call as "Plex Engine".

In the component-and-connector view of this solution (see figure 11), we have the "Client Web Browser" communicating with REST API using the REST protocol. This API is hosted on a server IIS and communicates with the Transporter client and server part through a COM application and can use TCP/IP to communicate with the database SQL Server.

In the allocation view (see figure 12), the API can be installed on the same machine as the Transporter. In order to run the Transporter from other machines on the same network, you only need a browser to access the application.
This chapter describes an evaluation of each of the approaches and comparison between them.

A. Implementation

For evaluation purposes, a web browser solution of the CRM module has been implemented adopting the approaches described in the chapter III. As a measure, the time and number of lines required for implementation in each of these approaches were taken into account. The interface that has been implemented is the same for each of the approaches only differing in the way it communicates with the Transporter.

1) WCF Services generated by CA Plex: In this approach, an API was implemented that encompasses CRM services generated by CA Plex. A web browser interface was also created in .NET which consumes the services made available through API.

On the API side, there are two services, one to perform a GET operation on one CRM activity and the other one to perform create, update, and delete operations. The creation time of the first service, including the minimum structure that forms the API, took two hours and 75 lines of Plex code. The create, update, and delete service took three hours to implement and 50 lines of Plex code.

On the interface side, a window with a table was created to make available the CRM activities data and its connection to the API to obtain the data. This implementation took two hours and 75 lines of code C#.NET. The update operation of an CRM activity took four hours and 40 lines of code. The delete operation took two hours and 20 lines of code. Finally, the operation to enter a CRM activity took three hours and 40 lines of code. In total, it took 16 hours to implement 300 lines of code.

For better interpretation of the measured values, the following graphs reflect the measurements that were taken into account during the implementation. The graph 13 reflects the time that was consumed in the implementation and the graph 14 the written lines.

During implementation using this approach, the creation of services is not as linear as initially thought. That is, there are cases where you need to encapsulate existing code in panels into functions so they can be used through services.

2) Data centric solution: In this approach a single solution was created in .NET with a connection to a SQL Server database containing the web browser interface which operates directly on a Transporter database.

This solution took a total of two hours with 35 lines of written code. The graph 16 shows the number of lines that were required for the implementation of each of the tasks and the graph 15 shows the time needed to implement them.

The fast development of this solution is due to the tools that .NET framework provides. A GridView was used which is a table that operates on a specific table of a database and...
It can be assumed that this approach can be quite fast compared to the previous one when implementing CRUD operations and operations with some simple business logic. In the case of more complex functions, it can be more time consuming because it is necessary to implement it from scratch and have to know the business.

3) REST API integrated with Transporter: In this solution, it was necessary create client functions in CA Plex and the REST API on C# on the server side and the user interface on .NET on the client side.

This approach took almost 15 hours to implement and 800 lines of code were written. Again, we have the charts that show the time and number of written lines in each of the components of this solution. The graph 18 shows the number of lines that were required for the implementation of each of the tasks and the graph 17 shows the time needed to implement them.

This solution is the less preferable because it requires a lot of code and makes it difficult to maintain.

B. Results Comparison

In this section a comparison of the results obtained during the implementation of the three approaches is carried out.

In the graph 19 we have the total implementation time of each of the approaches and in the graph 20 we have the number of written lines.

Based on the graphs presented, we can conclude that the implementation of solution 1 was the most time consuming and solution 3 was the least time consuming and the one with fewer lines of written code.

Solution 2 was the one that needed more lines of code compared to the other solutions. Due to the greater number of lines, this approach is less preferable as it is more difficult to maintain.

The results of solution 3 are justified by the operations CRUD that have been implemented. Other type of operation that requires a more complex business logic, this approach would not develop as fast as it requires rewriting business logic.

Solution 1, although it has been the most time-consuming, once the base structure is implemented, the development of this approach becomes faster.

In my opinion, in the short term, a hybrid solution that combines solution 1 and 3 would be the best approach. In this hybrid approach, it will allow CRUD operations to be quickly implemented and there is no need to rewrite business logic, using solution 1 for these cases. In the long run, implementing the Transporter from scratch using modern technology so that it is free from the limitations that Plex has.

V. CONCLUSION

This document describes Transporter which is an ERP focused on the area of logistics and transport and it is considered a legacy system. It is a challenge for MAEIL to modernize it so that it opens the possibility of designing and implementing a new interface web browser. The solution to be able to create this interface, is to have an implementation of an API. This approach allows you to take advantage of the developed
functionalities over 17 years and prevent the business logic to be rewritten. There are three possible ways to build an API that have been analyzed and compared to each other. The current architecture is described and how each solution integrates with Transporter.

It is also analyzed how to expose operations of different granularities that exist in the Transporter in which simple operations are considered as nouns and composed operations are like verbs.

It will be MAEIL that will have the final decision of which of the solutions to choose according to the time and resources available. However, my recommendation on which of the solutions to choose, in the long term, will be rebuilding Transporter, so that CA Plex ceases limiting its evolution and allows MAEIL to follow the market technologically. In addition, the use of new technologies will make it easier to find specialized people.

In the short term, for example for any services without the need for an interface, using the service solution WCF generated by CA Plex is a good approach. If there is a need to have a short-term user interface, besides opting the generated WCF services for the more complex operations, also adopt a data centric solution for simpler operations, such as CRUD. Adopting this hybrid solution will be the fastest approach and the application itself will have a better performance.

For future work, there is a possible solution that can be exploited which is a REST generator by CA Plex. This generator is part of the plan from CA to be developed. In addition, to fully modernize the Transporter, it will be necessary to define which modules from it to be exposed and design the user interface.

REFERENCES