Bachelor's Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Robotic process automation in Hardware-in-the-Loop (HiL) SW testing

Karolína Sehnalová

Supervisor: Ing. Tomáš Haubert, Ph.D.
Field of study: Cybernetics and Robotics
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Sehnalová Karolína** Personal ID number: **491904**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Robotic process automation in Hardware-in-the-Loop (HiL) SW testing

Bachelor’s thesis title in Czech:

Robotická automatizace procesů v Hardware-in-the-Loop (HiL) testování software

Guidelines:

1) Analyze HiL testing process in selected project and its possibilities for process automation
2) Design platform and whole ecosystem for robots that interconnect already automated tools within new ecosystem
3) Integrate this solution into existing HiL testing process
4) Summarize the activity and usage of the robots compared with previous state

Bibliography / sources:

[1] Tauli Tom: The Robotic Process Automation Handbook, Apress, 2020
[2] Axelrod, Arnon: Complete Guide to Test Automation, Apress, 2018
[3] Bornet, Pascal: Intelligent Automation, 2021

Name and workplace of bachelor’s thesis supervisor:

Ing. Tomáš Haubert, Ph.D. Department of Electric Drives and Traction FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **22.09.2021** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until:
by the end of summer semester 2022/2023

Ing. Tomáš Haubert, Ph.D. Supervisor’s signature	prof. Ing. Michael Šebek, DrSc. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.
Date of assignment receipt Student’s signature

Acknowledgements

I would like to thank to my family and close friends for their endless support during all my studies. I would also like to thank to my supervisor Ing. Tomáš Haubert, Ph.D. and my colleagues, especially Bc. Denis Helienek, Ing. Matěj Beránek and Ing. Martin Novotný.

Declaration

I declare that I have written, developed and implemented this thesis by myself and that I have listed all used sources according to the Guideline no. 1/2009 for adhering to ethical principles when elaborating an academic final thesis.

In Prague,

date   Karolína Sehnalová

Abstract

This thesis elaborates the problematic of Robotic Process Automation in Hardware-in-the-Loop testing. The main objective is design and implementation of a modular platform for creating various automation software robots. The contents of the thesis are the analysis of certain available tools for Robotic Process Automation and the environment used in the implementation process, design and implementation of the platform and finally, evaluation of thesis's results.

The platform for automation tools which allows users to configure software robots to perform various automation tasks had been implemented in the terms of the thesis. The platform's design and particular implementation is elaborated in detail. The platform's implementation also consists of design and implementation of database for the storage of robot configurations. Lastly, the thesis also focuses on developing automation tool for processing reports from Hardware-in-the-Loop testing.

Consequently, the whole platform and the developed automation tool are rated and particular partial results are shown and described. There is a feedback from platfrom's users in the overall evaluation of the thesis.

Keywords: Robotic Process Automation, Hardware-in-the-Loop, Django, database, robot, software engineering

Supervisor: Ing. Tomáš Haubert, Ph.D.
Department of Electric Drives and Traction FEE

Abstrakt

V této bakalářské práci je rozvinuta problematika Robotické automatizace procesů s použitím při Hardware-in-the-Loop testování. Hlavním cílem práce je návrh a implementace modulární platformy pro vytváření softwarových robotů pro automatizaci. Obsahem práce je analýza vybraných dostupných nástrojů pro Robotickou automatizaci procesů a použitých prostředků při samotné implementaci, návrh a implementace platformy a na závěr zhodnocení výsledků práce.

V rámci práce byla implementována platforma pro automatizační nástroje, která umožňuje uživateli konfigurovat softwarové roboty, které vykonávají různé automatizační úkoly. Je rozebrán návrh této platformy i její konkrétní implementace. Součástí platformy je také návrh a implementace databáze pro ukládání konfigurací robotů. Další součástí práce je automatizační nástroj pro zpracovávání zpráv z Hardware-in-the-Loop testování.

Následně jsou tato platforma i automatizační nástroj v práci zhodnoceny a jednotlivé dílčí výsledky jsou ukázány a popsány. V celkovém zhodnocení je zahrnuta zpětná vazba od uživatelů této platformy.

Klíčová slova: Robotická automatizace procesů, Hardware-in-the-Loop, Django, databáze, robot, softwarové inženýrství

Překlad názvu: Robotická automatizace procesů v Hardware-in-the-Loop (HiL) testování software

Contents

Figures

Tables

Chapter 1
Introduction

Every control unit needs to be adequately tested before it is used in a real car. A massive part of the testing is performed in Hardware-in-the-Loop simulation systems that simulate these control units' environment. The testing process is becoming more and more automated to increase speed and efficiency. This is precisely the primary goal of this thesis - designing a whole automation platform to increase the speed and efficiency of the testing process.

This thesis was created in Porsche Engineering Services company. The designed automation platform is currently being used in the HiL testing process.

1.1 The idea and motivation

My motivation for this thesis was to create a software tool that will help employees with the processing of reports created from tests. Similar tests create a report which contains necessary information about the tests. The processing of these reports was a very time-consuming task therefore the automation is beneficial in this case.

This idea was then expanded to create not only a single tool for processing these reports but a whole platform that can incorporate many different tools and combine them arbitrarily while the platform provides all the management of these tools. The idea went even further - the platform will contain some triggers, which will activate these arbitrary combinations of tools (these are called a robot). This means no human interaction is needed once the robot is configured because the robot is executed automatically when one of the configured triggers is triggered. These tools will save much-needed time for the employees because they perform some necessary tasks instead of the employees. Since the tasks meant to be performed by the platform are Robotic Process Automation tasks, this platform is called the RPA platform.

1.2 Main objective and partial tasks

The main objective of this thesis is to create the RPA platform. Partial tasks of this thesis are:

- analyze HiL testing process in selected project and its possibilities for process automation,
- design platform and whole ecosystem for robots that interconnect already automated tools within new ecosystem,
- integrate this solution into existing HiL testing process,
- summarize the activity and usage of the robots compared with previous state.

Chapter 2

State of the art - Hardware-in-the-Loop and Robotic Process Automation

2.1 Hardware-in-the-Loop introduction

The need for testing electronic control units (ECU) is becoming more and more important. ECUs which were not adequately tested cannot be used in any industry branch. The testing procedure is very complex since every possible state of ECU has to be verified. In order to perform a large number of tests and tests in difficult real-time situations, Hardware-in-the-Loop simulation is used. HiL simulates the whole environment of the ECU (i.e., the car) and generates and computes signals which are then sent to the ECU.

There are various reasons for testing with HiL simulation:

- all the tests and test conditions are pre-defined and repeatable,
- error situations can be tested,
- overall testing cost reduction,
- it is possible to run tests in conditions that cannot be accomplished in real systems,
- tests can be run with multiple versions of ECU (e.g., ECU can be used in HiL simulation even though it is not fully prepared to be used in the real system),
- shorter development cycle.

Essential components of every HiL simulation system are: DUT (device under test), real-time processor and user PC. In this case, DUT is the ECU. The ECU environment model is created and loaded into the HiL real-time processor. The model communicates with the ECU by signal generation and signal measurement. Signal generation sends signals (computed in real-time) to ECU inputs, using analog and digital inputs and outputs and bus communication. Signal measurement reads signals from ECU outputs. It can then use these measured signals to compute the successive values of signals connected to ECU inputs.

Modern ECUs are able to self-diagnose. They are checking if their inputs are credible. Those inputs are signals from various sensors and therefore, the simulation has to be so accurate that the ECU is not able to detect that these signals are not coming from real sensors. Suppose any unforeseen time course or value of a signal is detected. In that case, it may enter emergency mode (ECU has many error states and it enters one of them) and further testing may not be possible. Signals from ECU outputs (and potentially other variables) are being compared to expected values defined by testers and based on that the test result is either fail or success. Tests are run and controlled from the user's PC via test automation software. One of the most used automation softwares for testing is EXAM (EXtended Automation Method).

An example HiL simulator is shown in figure 2.1. This simulator is a product of dSPACE company.

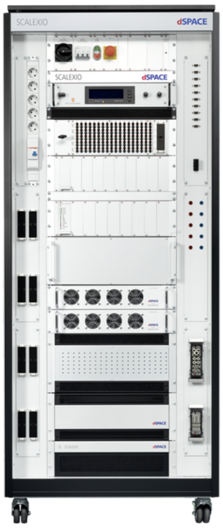


Figure 2.1: HiL simulator - taken from [6]

2.2 Robotic Process Automation introduction

Robotic Process Automation (RPA) is a modern approach to automation of specific tasks. It is often utilized in companies where employees are overloaded with repetitive tasks (usually containing passing data from one application to another one). These tasks can take a significant amount of time and can be very exhausting for the employees. The time spent with these tasks could be dedicated to other more important things if software bots performed these tasks. That is precisely what RPA (Robotic Process Automation) offers.

The term RPA was invented by Pat Geary in 2012. Pat Geary is the chief marketing officer for a market-leading company in RPA softwares called Blue Prism. According to Tom Taulii, RPA can be described as a set of software bots that perform such tasks as listed below:

- the cut-and-paste of information from one app to another,
- the opening of a web site and login,
- the opening of an e-mail and attachments,
- the read/write of a database,
- the extraction of content from forms and documents,
- the use of calculations and workflows. [1, p. 3]

To sum up the information, these bots contribute to many business procedures and disburden employees of many important tasks that have to be done. That allows employees to focus on other tasks and therefore increase their efficiency.

2.2.1 A concern regarding Robotic Process Automation

With all these benefits mentioned in the RPA description it can seem that RPA bots can replace human employees [15]. This is coming mainly from the thing that RPA bots are intended to do tasks previously performed by humans. Employees of the companies which are often using RPA are often overloaded with tasks and therefore, it is beneficial even for them. In addition, RPA can never replace humans entirely. After all, even with the usage of machine learning and neural networks it will encounter problems because its cognitive and learning abilities are not even comparable with humans. Furthermore, RPA opens new working positions for new employees because every RPA software has to be managed and developed constantly by employees.

2.3 Open source softwares for RPA

Open source softwares create a significant part of available software tools. Many companies use it nowadays because it is free which can make the whole development process much cheaper [8]. Since it is free it is possible to try the software tool before the company starts using it. The majority of paid software tools offer trial versions but these are often time-limited and do not provide the full functionality. One of the main disadvantages of open source software is that it is constantly changing because everyone can contribute to it by adding some functionality and it is hard to keep track of these updates and constant development. This characteristic of open source software can also lead to no need to fixing bugs in the software. In contrast, since more developers are involved in the implementation the source code can be higher quality.

2.3.1 Robot Framework

Robot Framework [19] is written in Python (also runs on Jython and IronPython) and behaves like any other Python library. That makes the installation process much simpler for programmers. Robot Framework is also independent of an operating system. Its main advantage lies in allowing the user to use many other Python libraries and modules. Robot Framework is based on keyword-testing, which means its syntax consists of easy-to-use keywords. More keywords for testing can be added by using additional Robot Framework libraries. Robot Framework is easy to integrate with any other Python code. Basic knowledge of Python is necessary for working with this framework.

2.3.2 Robocorp

Robocorp is a cloud platform based on Robot Framework. It has identical syntax as Robot Framework and the only difference between Robocorp and Robot Framework is that Robocorp is extended by a cloud variant so that users can share some codes and bots with each other.

2.3.3 UI Vision

UI Vision is an RPA software tool for the automation of web-based applications or desktop applications. UI Vision is built up as a superior structure of a scripting language. Prior programming experience is not required. On the other hand, UI Vision offers an option to implement scripts in programming languages supported by UI Vision (e.g., Python, Visual Basic). This can obviously be accomplished only with prior knowledge of those languages.

UI Vision differs from other RPA tools in one significant aspect - its main principle - the ability to perform visual tests. Visual tests can be performed thanks to the knowledge gained from computer vision's field of expertise (e.g., image recognition).

2.4 RPA vendors

Companies developing RPA software provide lots of courses and even have academies for developers [1] [16]. These academies and courses offer quick ways to learn how to work with their applications and environments. Another advantage of paid software is that vendors always provide some support for their customers. Customers can communicate about their problems with vendors and some issues can be fixed by vendors themselves. The vast majority of RPA vendors offer a trial version, which is often time limited and does not provide the whole range of functionality. The purpose and goals of these companies are very similar. Hence, customers have to choose which RPA software they will use based on their own requirements for the final RPA system, developer preferences and prior knowledge.

2.4.1 Automation Anywhere

Automation Anywhere is targeted at people with none or minimal programming experience. It has a user-friendly interface that enables its users to create bots in an intuitive environment. Since the integration of AI into RPA is becoming more significant these days, Automation Anywhere provides a system called IQ bot which works with large datasets end provides modern AI features (e.g., computer vision). IQ bot can be easily integrated into a process. Another feature of Automation Anywhere is the bot store. Users can download already developed bots and therefore save a lot of time during the development process of the whole RPA system. One more advantage of Automation Anywhere is its convenient mobile application from which the user can control the bot easily and even provide it with additional data (e.g., for IQ bot).

2.4.2 Blue Prism

Blue Prism is an RPA development tool based on the programming language C#. Prior programming experience is not required since the software includes a dashboard with drag and drop functionality. Despite that, prior programming knowledge is advantageous because it helps understand Blue Prism's main principles. Blue Prism offers complete data security and it is rated amongst the best RPA tools in this aspect. Data security is one of the main reasons large corporations often use it.

2.4.3 UI Path

UI Path [18] is an RPA tool focused on Windows applications. UI Path requires at least basic programming experience because its designing tool called StudioX is not as user-friendly as for example Automation Anywhere. The main difference between UI Path and any other RPA software is in its architecture. UI Path works with a system called Orchestrator which manages and links all the processes together, while the majority of other softwares leaves the application's architecture more to developers. Orchestrator provides easy and convenient process management. UI Path provides complete data security.

Chapter 3

Analysis of current state and automation design

3.1 TIS (Tools Integration System)

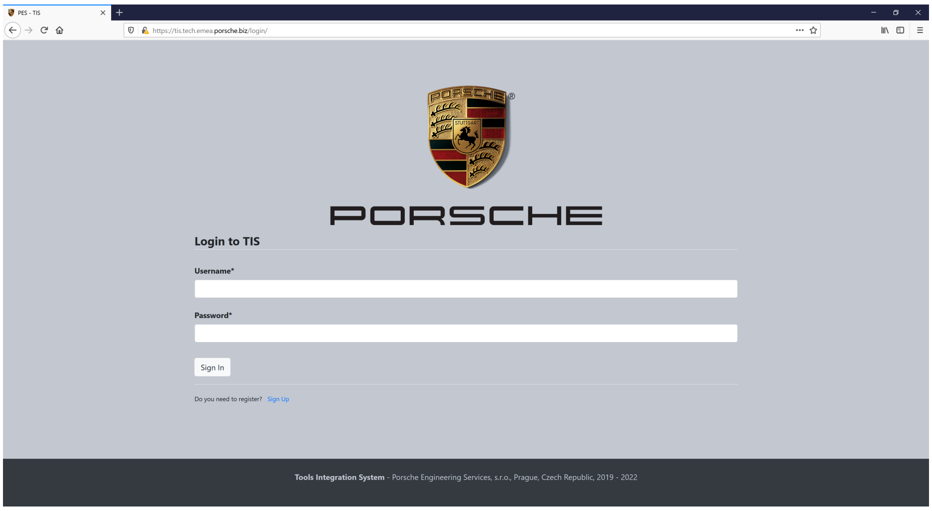
Tools Integration System is a system in Porsche Engineering Services for centralized integration and usage of various automation tools used in the HiL testing process. Developers can easily add new automation tools to TIS. TIS is developed in Django framework in Python. Django framework is a Python module which enables fast and effective development of web applications. TIS backend is mainly developed in python and it is stored in Git (version system) which helps the TIS development and maintenance. TIS frontend is mainly developed in javascript, HTML, CSS and jQuery with the usage of Bootstrap, Vue.js and Charts.js.

TIS runs on *Red Hat Enterprise Linux 7.6* (RHEL 7.6). RHEL 7.6 Linux based opensource operating system which is used mainly for corporate and commercial applications. TIS welcome page can be seen in the figure 3.1.

Welcome at Tools Integration System portal.

Figure 3.1: TIS welcome page

Every employee can use his company account to log in to TIS. Then he can use all the available automation tools. TIS login page can be seen in the figure 3.2.



Do you need to register? Sign Up

Tools Integration System - Porsche Engineering Services, s.r.o., Prague, Czech Republic, 2019 - 2022

Figure 3.2: TIS login page

TIS organization diagram can be seen in the figure 3.3.

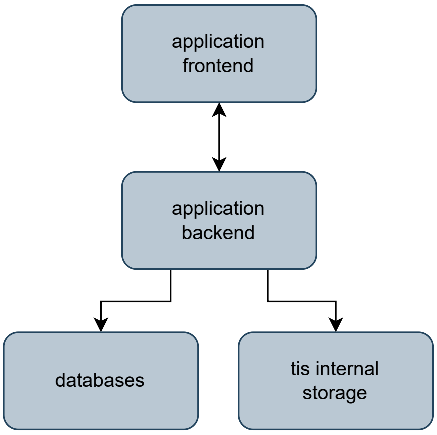


Figure 3.3: TIS organization diagram

3.1.1 Django databases

TIS has a default SQL database written in MariaDB. MariaDB is an open source relational database. Django supports connection to the databases written in database systems listed below [10]:

- PostgreSQL,
- MariaDB,
- MySQL,

- Oracle,
- SQLite,

and it is possible to connect to these by adding the database with all relevant information into the Django file *settings.py*. This file is present in every Django application and is used for various application settings.

EXAM report databases

TIS also has an access to some EXAM report databases written in PostgreSQL. These databases are used to store data from HiL testing. They are accesed via Test Report Database (a graphical user interface in TIS for watching reports and their results) which enables convenient report view. These databases are also used in some automation tools. Currently TIS has an access to two EXAM report databases.

3.1.2 TIS internal storage

TIS also provides internal storage for files. Every file is assigned to a specific user and only the user assigned to the file can access it from TIS. This way it is possible to store either files which are often used in TIS tools (so that users do not have to upload them every time from their computer) or files which were generated by some TIS automation tool. Three types of files can be distinguished based on so called storage class. Storage class can be:

- EPHERMAL (these files are created during a process and deleted immediately after)
- LAST3DAYS (these files are accessible from TIS but are deleted three days after they have been created)
- PERMANENT (user can access these files anytime, they can be deleted only manually)

3.2 Selected project

This subsection describes the project workflow of project HCP1 Powertrain (figure 3.5) in comparison with general project workflow (figure 3.4) in Porsche Engineering Services. HCP1 Powertrain is the selected project for HiL testing process RPA automation in this thesis. The general project workflow in our company is described first, followed by HCP1 Powertrain project workflow description.

3.2.1 General project workflow

Creating a testcase is a complicated procedure consisting of many steps. Tests are created from test specification. Test specification can be created as an output of the Test Specification Generator (TSG) or it can be manually written. TSGs are available in Tools Integration System (TIS) and every employee can use them. They generate the test specification from specific files, however the majority of test specifications is written manually. The test specification is a CSV file which can be imported into DOORS (Dynamic Object Oriented Requirements System). From DOORS it can be imported into EXAM (EXtended Automation Method). This can be done via *DOORS syncer GUI* and *CI plugin* which enables automated DOORS to EXAM synchronization. The usage of DOORS is optional and it depends on the specific testcase and if the testcase needs to be stored in DOORS. EXAM is a test automation software developed by the MicroNova company [7] and is used for graphical development of testcases. EXAM contains libraries which are necessary for running the tests. EXAM creates an executable file which can be run and managed by PTD 2.0. PTD 2.0 prepares everything needed for the experiment (e.g., HiL, signal measurement and signal generation tools, HiL model). EXAM has its own report database which stores information about evaluation (i.e., ERROR, SUCCES, PASS, FAIL) for every testcase along with error or informational messages. This database can be accessed via TIS which is very convenient because the TIS test report database GUI is more user-friendly than the EXAM report database and furthermore, the tester does not have to use EXAM at all to access the report database. The only thing needed to access the EXAM reporting database via TIS is the access credentials.

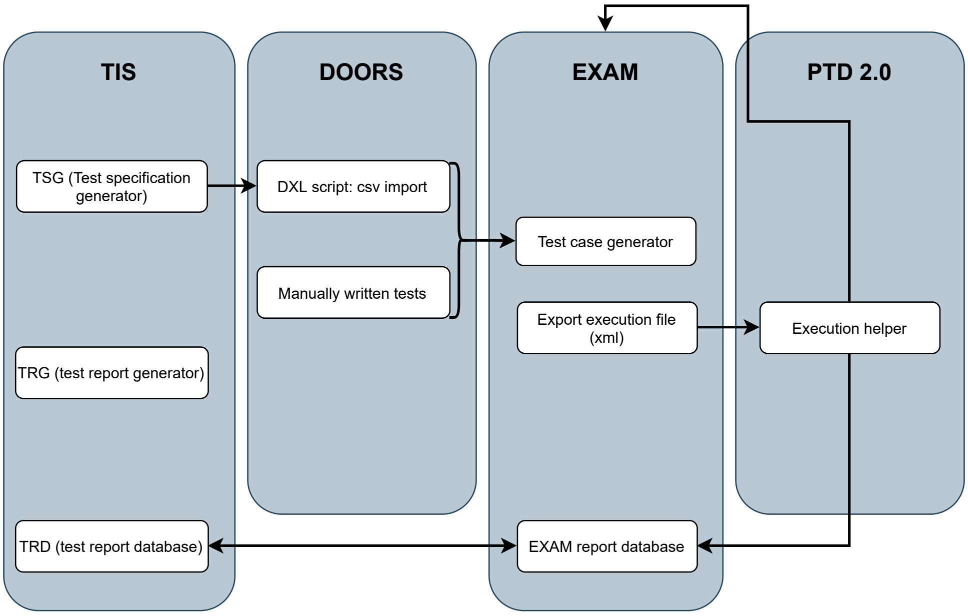


Figure 3.4: General project workflow

3.2.2 HCP1 Powertrain project workflow

Every project differs in at least some aspects since it is impossible to unify every step of the HiL testing project workflow in the whole company. Nevertheless, Porsche Engineering Services tries to keep the project workflow as similar as possible in every project. This improves the understanding of other team's work. In my selected project HCP1 Powertrain the workflow is rather simple. Many steps described in the previous subsection about the general workflow are missing mainly because the project is relatively new and some functionalities have not been integrated into the project workflow just yet. TSGs are not used at all and that means all the tests are manually written. After that they are imported directly to EXAM (no DOORS to EXAM automation is used). The usage of PTD 2.0 is not possible at the moment but is scheduled to be set up in the near future.

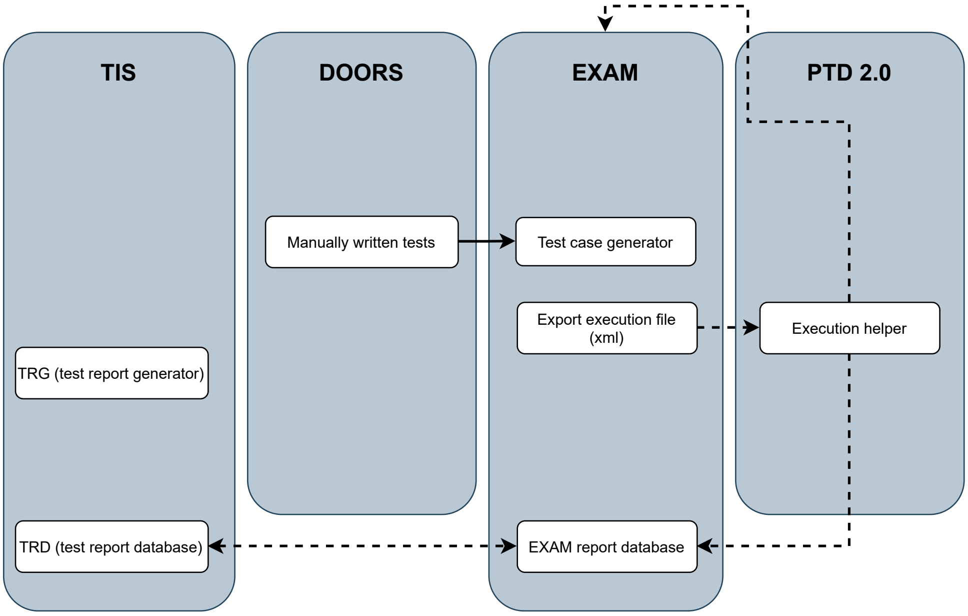


Figure 3.5: HCP1 Powertrain project workflow

3.3 My suggested design

This section describes the RPA design in Porsche Engineering Services created in this thesis. The overall design consists of the process of **choosing the development software** for integrating RPA into the existing ecosystem, **RPA platform** (specific usage and storage of RPA bots in Porsche Engineering Services) and **Robot TIS database** (database for storage of the RPA bots in Porsche Engineering Services). All these parts of the design are further elaborated in the following subsections.

3.3.1 Choosing the most suitable development software

There are many software tools for Robotic Process Automation and it can be really difficult to decide which one is the most suitable option for a specific project. This is a very complex and nontrivial problem since many aspects should be taken into consideration - the final decision depends on many deciding factors. The main deciding factors can be:

- programming experience and knowledge of developers,
- the ease of integration into the existing ecosystem,
- programming language for programming based tools,
- minimization of expenses (e.g., software licenses, paid trainings).

The comparison of key factors necessary for deciding which software is the most suitable one for a specific RPA project is listed in Table 3.1 below.

	Requires programming experience	**Programming language**	**Opensource**	**Architecture**
Robot Framework	yes	Python	yes	undefined
Robocorp	yes	Python	yes	undefined
UI Vision	no	Python, Visual Basic, other	yes	undefined
Automation Anywhere	no	based on Java	no	client-server
Blue Prism	yes	C#	no	client-server
UI Path	yes (a little)	Visual Basic, C#	no	web based

Table 3.1: RPA tools comparison

There are many more development softwares which can be used for RPA. The softwares which were described in the previous chapter are the most common ones.

Chosen development software

Each one of the previously mentioned development softwares has its positives and negatives which makes the development software decision extremely difficult. It is important to have in mind that it is extremely difficult (might be costly) to change the chosen development software during the development process, which makes the decision even more crucial.

In all implementations mentioned in this thesis I will use Robot Framework. The reasons for choosing this software tool are:

- it has been used in the company before (coherence of used software tools within the company),
- the company has many good Python developers so it is logical to use a framework based on Python language which is good for developers,
- the company supports opensource software.

Robot Framework is a good option for RPA tasks discussed further in this thesis.

3.3.2 RPA platform design

The RPA platform designed in this thesis is a modular platform for the storage of RPA software bots. Every TIS developer in Porsche Engineering Services company can create a new RPA tool and add it to the platform. Every TIS user is then able to use the new RPA tool by creating a robot consisting of this RPA tool. The robot creation and configuration can be done via a user interface in TIS which was created solely for this purpose.

The integration of RPA robots into the existing ecosystem (TIS) is very complex and needs to be planned in detail. It is crucial that the RPA platform does not disrupt TIS and all the tools that are currently uploaded in TIS. The RPA platform is designed with an emphasis on user comfort, focusing on user-friendliness and the possibility to be easily extended to accommodate more robots. The creation of the new software bots is intended to be fast and user-friendly.

The essence of the platform is that the platform will contain *triggers*, *RPA tools* and *finish tasks*. Trigger passes down an impulse to start a robot execution, RPA tool is a specific RPA application and finish task is a simple post-processing task that will process data provided by the RPA tool in a standardized way (defined by the implementation of the finish task). The *robot* consists of triggers, RPA tools and finish tasks. The main aim is the possibility for the user to configure a robot with arbitrary triggers, RPA tools and finish tasks via a user interface on TIS (mentioned in the beginning of this section). This robot will then start the execution of all the selected RPA tools followed by finish tasks when any of the triggers is triggered.

3.3.3 Robot TIS database

The idea is to store information about all available RPA Robots in the default TIS database. New tables and relations have been created for this purpose in the database. This part of the default TIS database is called **Robot TIS Database** and was designed and then implemented solely for the RPA platform.

The entity relationship diagram (ERD) for the Robot TIS database can be seen in the figure 3.6.

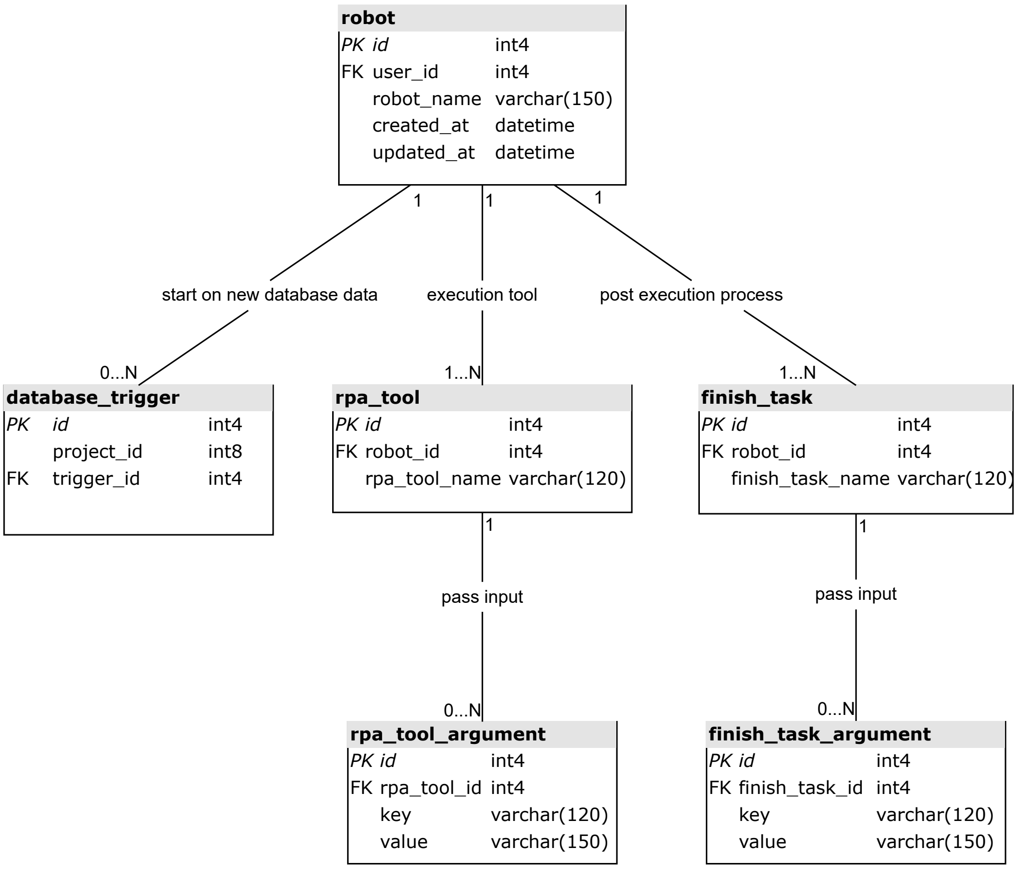


Figure 3.6: Robot TIS database entity relationship diagram

The central thought behind the design of the Robot TIS database is to minimize manual data insertion into this database for individual RPA tools developers. For example, there is no need to access the database when creating a new RPA tool or finish task. The Robot TIS database and the RPA platform are modular and they prepare everything that is needed for successful robot execution. All configured robots by users are being stored in the Robot TIS database. All the data about the robots and their configurations are inserted into the database through the user interface in TIS.

During the database design process (i.e., creating the ERD) it is essential to try to cover all the possible situations that can occur when trying to work with data stored in the database and foresee potential problems. The design is a mentally exhausting process which takes some time and it leads to problems if it is not designed correctly. The database had to be redesigned many times, sometimes even fully reimplemented. Django documentation [10] and MariaDB website [11] were helpful in the design process.

Robot TIS database organization

The head table is the *robot* table. Robot table has multiple records in specific trigger tables (e.g., robot can have many database triggers and many other triggers which have not been implemented yet and for each of them there is a record in the corresponding specific trigger table). That means that when a developer implements a new triggering script he has to create a corresponding table in the Robot TIS database with all necessary information for this new triggering script. Each specific trigger table (e.g., database trigger) corresponds to one trigger script. The robot table also has multiple records *rpa tool* and *finish task* tables. These tables (rpa tool and finish task) can each have multiple records in argument table (*rpa tool argument* and *finish task argument*). These relations are shown in figure 3.6. This database arrangement contains all necessary information and allows convenient data insertion and access.

database trigger

This table differs from other designed tables in one standpoint. It is not general. Every other database table is designed to be general and universal but *database trigger* is a special table. It is used for only one type of trigger called *Database trigger*.

Database trigger is a triggering script that runs periodically in TIS. It has a set of so-called *watched projects* and for each of them their corresponding EXAM report database. These watched projects are folders in EXAM and the database trigger script checks if there is a new report in one of these projects (by searching in the corresponding EXAM report database). If it detects a new report occurrence, the trigger is considered activated and it passes down an impulse to execute a robot (or multiple robots). These watched projects and their EXAM report databases are stored in the Robot TIS database in the *database trigger* table. There is a separate triggering script for each reporting database, although there is an access to only two reporting databases at the moment. Database trigger triggering script is elaborated in detail in subsection 4.3.1.

The triggering script needs to know the information which projects are currently being watched. Therefore the script accesses both the Robot TIS database (for the information which projects are being watched) and the

EXAM report database (for information about new reports) in its every execution and reads necessary information from all database trigger tables. To sum up the information, the database trigger contains all the required information about which projects are currently being watched and which robot they belong to.

Fields of the database trigger table are:

- *project_id* ⟶ project id of watched project (the id refers to the corresponding EXAM report database and not the TIS default database),
- *database* ⟶ a field that specifies the reporting database that should be watched by the database trigger (there are multiple reporting databases),
- *robot_id* ⟶ a foreign key that refers to the *robot* table.

Each new triggering script is intended to have its own table in Robot TIS database with relevant data for the specific triggering script as it was mentioned earlier in this chapter.

robot

Robot is the head table of the Robot TIS database. Fields of this table are:

- *user_id* ⟶ a foreign key that refers to the *user* table which is a fundamental part of the default TIS database and connects the robot to its user in TIS,
- *robot_name* ⟶ a name of the robot inserted by the user,
- *created_at* ⟶ a time of the record in the robot table creation,
- *updated_at* ⟶ a time of the robot's last update.

The robot has to have a name so that the user can name it and recognize it by the name. This name can serve as an identifier of the robot for the user.

Robots cannot be updated at the moment but the robot update feature is meant to be added to the RPA platform in the future.

rpa tool

Rpa tool is a table for relevant data for every tool. Fields of this table are:

- *rpa_tool_name* ⟶ an RPA tool name that is needed in the implementation for deciding which tool is supposed to be executed,
- *robot_id* ⟶ a foreign key that refers to the *robot* table,
- *gui_arguments* ⟶ a field determinig if the tool should use arguments inserted from graphical user interface (GUI) or arguments passed from trigger or other tools.

One robot table can have multiple records in the rpa tool table (i.e., robot can have multiple RPA tools). In the implementation all these tools belonging to the robot are executed in the order they were inserted into the database. (i.e., the order in which they were inserted into the graphical user interface for creating robots).

rpa tool argument

Rpa tool argument is a table for storage of one argument belonging to one rpa tool. Fields of this table are:

- *rpa_tool_id* ⟶ a foreign key that refers to the *rpa_tool* table,
- *key* ⟶ a name of the argument passed to the rpa tool,
- *value* ⟶ a value (stored in a string type) corresponding to the argument name.

The rpa tool argument table can be matched to one item in a python dictionary. It has a key field representing the name of the argument and a value field representing the value of the argument. One rpa tool table can have multiple records in the rpa tool argument table. Arguments stored in this table are "static" which means they are inserted into the database once during the creation of the table. "Dynamic" arguments (mostly obtained from the triggering script) can be passed as keyword arguments as it is mentioned in section 4.2.

finish task

Finish task is a very similar table to the rpa tool table. The only difference at the moment is in the names of the database tables. It is important that these tables are kept separate because in the future RPA platform extensions the finish tasks and rpa tools will probably differ in more aspects. Therefore it will probably be necessary to have different fields in each table which means there have to be separate tables. It is important to think about all the possible future extensions and design the database from the beginning to be as general as possible.

finish task argument

Finish task argument is a similar table to the rpa tool argument table. The only difference is that the parent table of a finish task argument table is a finish task table and the parent table of an rpa tool argument table is an rpa tool table. These tables have the same purpose and they are both used as a storage for arguments belonging to their parent table.

There could be one table called simply *arguments* which would be used to store all arguments belonging to RPA tools and finish tasks together. This would lead to a problem with primary keys and it would be necessary to store the information if the arguments belong to RPA tools or finish tasks. The solution with two separate tables avoids this problem and therefore I consider it slightly better.

Chapter 4

Implementation

4.1 Robot TIS database implementation

The database was implemented with the Django framework. Django has a class *django.db.models.Model* which can be used when creating a database table. Each subclass of the *django.db.models.Model* class corresponds to exactly one table in the database and each attribute of the subclass corresponds to one database column. These tables then have to be added to the Django *settings.py* file as mentioned in section 3.1.1 and then Django creates proper tables in the database itself.

All relationships were implemented with Django as well and according to the Django documentation [10]. The documentation has been extremely helpful during the database implementation process.

As the database is implemented it contains separate tables for each trigger. This can seem as a disadvantage for developers because they need to create separate table in the database for every new trigger they implement. However, the developer inevitably needs to access the database when creating a new trigger and there is no way around that. The reason for accessing the database is that the developer needs to know the triggering condition (when the trigger executes the robot) followed by the execution of the robot. All data about triggering conditions are stored in the database. That means the database knowledge is required from developers and they can create their own tables for their trigger. At this point it becomes an advantage, because the developer can specify his own custom table columns and adjust them to his code which makes the trigger code more straightforward and faster (it is crucial that the trigger script is fast so that it does not overload the server). Last but not least, it is intended to have only a few triggers so the separate table for each of them is adequate.

Implementation of the Robot table of the Robot TIS database is shown in listing 4.1. Every other table was implemented similarly.

```
from django.db import models
from django.utils import timezone
from django.conf import settings


class Robot(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE,
                             null=True)
    robot_name = models.CharField(max_length=150)
    models.DateTimeField(default=timezone.now)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Code Listing 4.1: Robot table implementation

4.2 RPA platform implemenation

The RPA platform backend is entirely implemented in Python. The first step in the implementation was to create a class diagram for the RPA platform because the relationships of all implemented classes are very complicated and create the core of the RPA platform. The inspiration for the class diagram used in the RPA platform came from a book called Head First design patterns [13] and a website called Refactoring.Guru [12]. Especially the Factory design pattern and Abstract Factory design pattern were the main inspiration. The class diagram designed for the RPA platform can be seen in the figure 4.1.

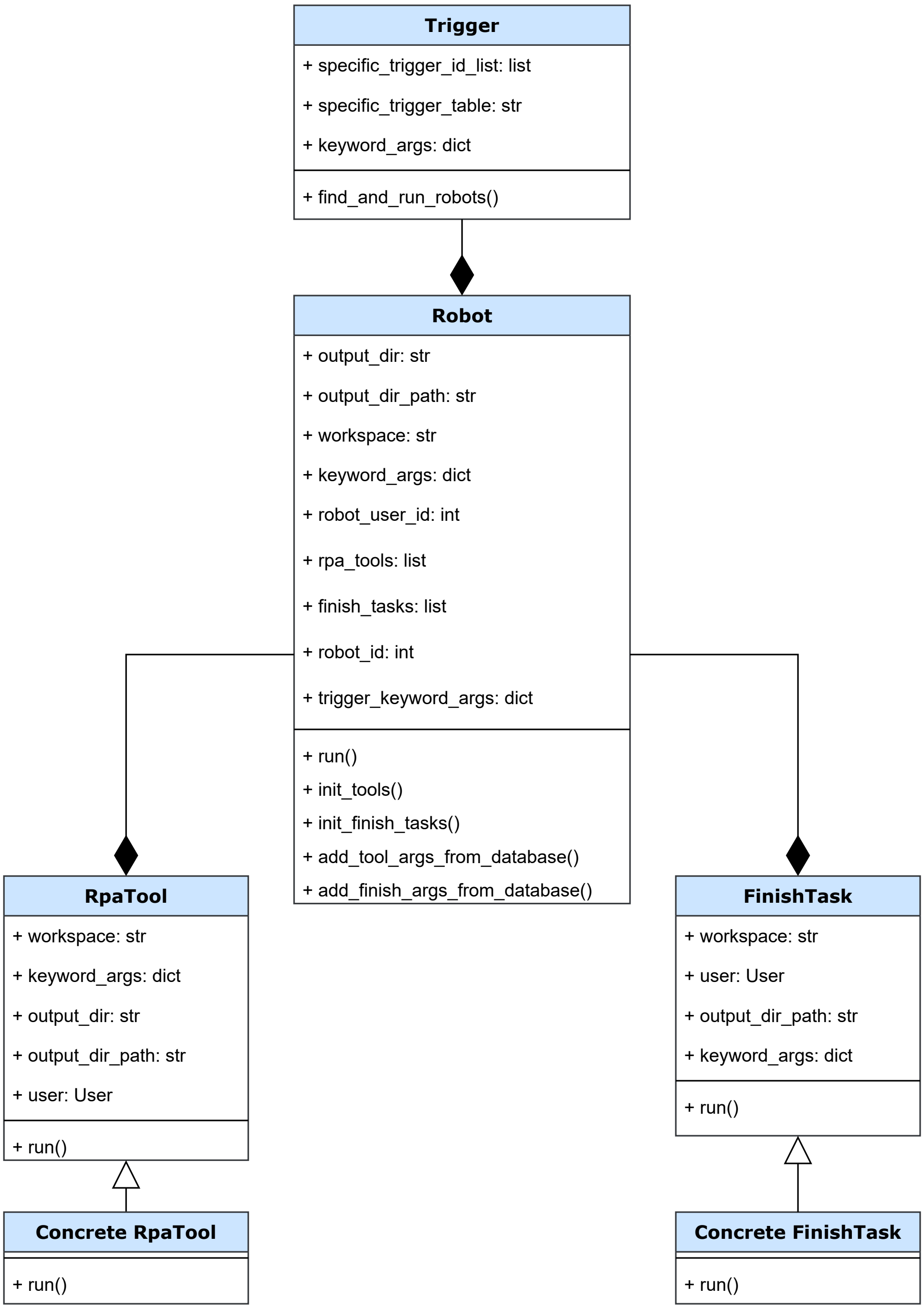


Figure 4.1: RPA class diagram

4.2.1 Abstract classes

For detailed explanation of each class used in RPA platform implementation (figure 4.1) it is essential to understand the concept of abstract classes fully. An abstract class is a template for the declaration of other classes - called the abstract class's subclasses. The subclass inherits the abstract class's methods and attributes. Every abstract class contains at least one abstract method. The abstract method differs from the standard method in a vital aspect - it has a declaration but not a definition. The absence of the abstract method's body means the abstract class cannot be instantiated and can be used exclusively as a superclass. Superclass is a term for a class from which other classes inherit methods and attributes. The method's body is intended to be implemented in the subclass declaration. The correct usage of abstract classes brings modularity and universality to standard programming.

Abstract classes are not implicitly supported by Python but they can be implemented with the usage of Python's standard module called *ABC*. Abstract class can then be created as a subclass of the ABC class and abstract method can be implemented with a decorator imported from the *ABC* module. The abstract method's body is left without implementation as mentioned in the previous paragraph.

The following subsections contain a detailed description of each implemented class.

4.2.2 Trigger class

Upon activation trigger script instantiates the *Trigger* class and runs the *find_and_run_robots()* method. The arguments passed to the Trigger class are:

- *id_list* ⟶ a list of IDs from the *specific_trigger* table of the Robot TIS database (section 3.3.3),
- *trigger_type* ⟶ a type of the trigger script,
- *keyword_args* ⟶ a dictionary of arguments from the trigger script.

The *find_and_run_robots()* method accesses the Robot TIS database and reads the corresponding robot for each *specific_trigger_id*. The *trigger_type* is needed for the selection of the correct *specific_trigger* table (e.g. *database_trigger*) from the database. The Trigger class instantiates a *Robot* class and runs Robot class's *run()* method in separate thread for each robot found in the Robot TIS database. That means *Trigger* class creates multiple *Robot* classes.

4.2.3 Robot class

Robot class is instantiated in *Trigger* class's method and arguments passed to the *Robot* class are:

- *robot_db* ⟶ a subclass of *django.db.models.Model* containing Robot TIS Database information about the robot,
- *keyword_args* ⟶ the dictionary of arguments from the trigger script.

Robot class has a *run()* method which calls more *Robot* class's methods and reads all additional information from the Robot TIS database. That incorporates reading information about relevant RPA tools and finish tasks and their respective arguments from the Robot TIS database. All relevant arguments are added to the *keyword_args* variable before the execution of the corresponding RPA tool or finish task and are passed to their class initialization method as ***kwargs*. The *Robot* class decides which arguments are used from the triggering script (e.i. passed in kayword arguments from the Trigger class) and which are added from the Robot TIS database. This decision depends on the *gui_argumnets* field of the *rpa tool* (resp. *finish_task*) table.

Robot class performs one crucial task. Every robot needs a directory in TIS internal storage to store important files and data structures. This workspace is created in the Robot class. That means every robot has its own workspace. Robot class also creates a separate directory in the workspace passed to RPA tools.

The instantiation of RPA tool (resp. finish task) subclasses is complicated since the class names differ for each RPA tool (resp. finish task). Concrete subclass names are stored in the Robot TIS database. Every RPA tool (resp. finish task) subclass is imported in the *Robot* class python file and thus appears in the global scope of the program. That means the concrete subclass can be called from the global scope of the program as it is shown in listing 4.2.

```
class_import = globals()[class_name]
class_instance = class_import(*args)
```

Code Listing 4.2: Concrete class instantiation from the global scope of a program

4.2.4 RpaTool class

RpaTool is an abstract class with an abstract method *run()*. Every subclass of *RpaTool* class is a concrete implementation of one RPA tool. Every RPA tool is intended to have its own implementation of *RpaTool* class and most importantly a *run()* method. The *run()* method is an abstract method which invokes the concrete RPA tool source code. The process of the robot initialization and execution can differ from bot to bot and is meant to be written by the RPA tool author to meet his requirements.

The arguments passed to a subclass of *RpaTool* are:

- *workspace* ⟶ a path to a directory where all files and data structures related to the robot are stored,
- *output_dir* ⟶ a name of a directory where all files and data structures related to the tool are stored,
- *output_dir_path* ⟶ a path to a directory where all files and data structures related to the tool are stored,
- ***kwargs* ⟶ keyword arguments - a dictionary of arguments used for modularity and universality of passed arguments; the arguments are added to the dictionary during the run of the whole program.

4.2.5 FinishTask class

FinishTask is a very similar abstract class to the *RpaTool* class (subsection 4.2.4). It also has a *run()* abstract method which is left for the author of the finish task to implement and it is meant to invoke the concrete finish task implementation source code. Arguments passed to every subclass of the *FinishTask* class are:

- *workspace* ⟶ a path to a directory where all files and data structures related to the robot are stored,
- *output_dir_path* ⟶ a path to a directory where all files and data structures related to the tool are stored,
- ***kwargs* ⟶ keyword arguments - a dictionary of arguments used for modularity and universality of passed arguments, the arguments are added to the dictionary during the run of the whole program.

Some of the attributes of the *FinishTask* class are the same as in the *RpaTool* class. It would be possible to implement the whole RPA platform as it is at the moment with only one abstract class for both RPA tools and finish tasks. However, for future development of the RPA platform it is important that these classes are kept separated and there is a possibility to make their implementations diverse in more aspects.

4.3 Proof of concept implementation

The implementation of the Robot TIS database and the RPA platform (mentioned earlier in this chapter) needs concrete triggers, RPA tools and finish tasks for creating robots. For this purpose, I implemented the first trigger, RPA tool and finish task.

The aim was to integrate the created implementations correctly into the designed and implemented RPA platform. This process has two points of view. Firstly, it is the proof of concept of the RPA platform design and implementation. Secondly, it provides a functioning robot that can be used by employees to accelerate workflow in HiL testing company projects.

Concrete implementations of the trigger, RPA tool and finish task are elaborated in the following subsections.

4.3.1 Database trigger

A database trigger is a script that runs periodically and in each run it gets a list of all the watched projects from the *database_trigger* table of the Robot TIS database. Then it searches in the EXAM reporting database for any new report occurrences in these watched projects. If it detects a new report which is complete it executes the robot connected to that specific trigger as it was mentioned in the section 4.2.

As it was mentioned before, the report has to be complete to fulfill the triggering condition. Since there is no information about the report being in the copying process in the reporting database, the triggering script itself needs to determine if the newly occurred report is complete or not. The copying process can take many hours (depending on the size of the report). Database trigger checks the number of tests belonging to the new report and when the number of tests stops increasing the robot is triggered. The database trigger works as a finite automat and the states of the reports and the transitions between them are shown in the figure 4.2.

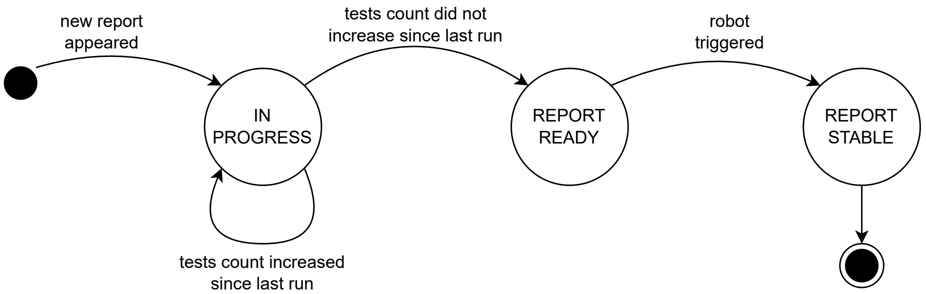


Figure 4.2: Database trigger report state diagram

Database trigger execution

Database trigger in executed periodically as a cron job. It accepts an argument from command-line which determines the EXAM report database which will be searched by the Database trigger in the current run. This solution enables an easy EXAM report database selection and addition. Database trigger is scheduled to run every 30 minutes for both available EXAM report databases. The run of the database trigger with different argument (EXAM report database) is shifted by 15 minutes.

4.3.2 Report Processing tool

Reports are delivered to customers in an excel file. This excel file offers complete and detailed information about the whole test suite and can be generated via TRG (test report generator) tool in TIS. TRG is a tool that a colleague had written prior to this thesis for generating excel files from reports. TRG needs XML files generated from EXAM as its input. The XML download from EXAM can take a very long time (it can be in the order of hours) so it is not fast and convenient. The user then has to manually upload these XML report files to the TRG tool and then generate the excel file. The problem is that the amount of time spent with downloading report XML files makes it often impossible to deliver excel reports to customers the day after the tests have been executed.

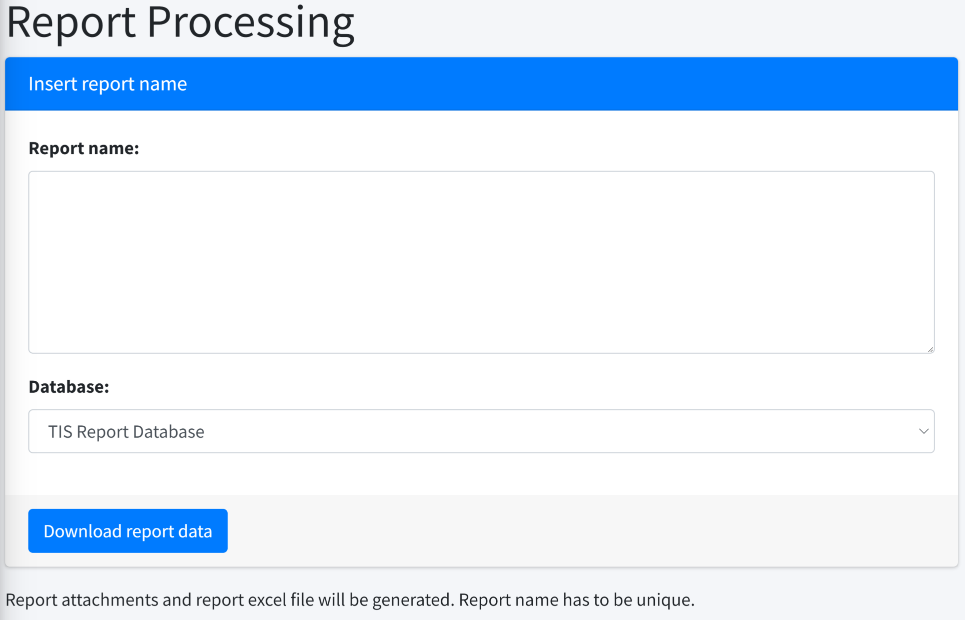
Report Processing is an RPA tool concrete implementation written in Robot Framework which skips the XML download part and gets all the required data regarding the report directly from the EXAM report database. Then the robot executes the TRG tool with data retrieved from the EXAM report database and skips the XML loading part of TRG. Apart from generating the excel report, the Report Processing tool downloads all attachments for the selected report (if there are any). These attachments are measurement files (DAT files) and they contain information about every signal measurement that has occurred during the test suite execution in HiL. Some test suites use INCA measurement software and some use CANape measurement software. Relevant attachments can be downloaded only for test suites using INCA. These attachments are stored in the EXAM report database in *7-zip* compressed data. Report Processing tool saves every one of these compressed attachments to the same folder as the excel report file generated from TRG. The result of the Report Processing tool is this folder containing all the relevant information about the report.

Report Processing tool execution

Report Processing tool is executed from a *RpaTool* subclass's *run()* method as it was mentioned in the section 4.2. In the *run()* method implementation of Report Processing calls the tool written in Robot Framework in a subprocess. The reason for this solution is that Robot Framework logs and reports work correctly only when the program is called from the main thread. However, this cannot be achieved in TIS (not in an easy way) and the subprocess has its own main thread which is sufficient for correct Robot Framework logging and reporting. This solution is easy and convenient.

Report Processing tool run directly from TIS

Report Processing tool does not have to be used exclusively from a configured robot. It can be run directly from two places in TIS. Firstly it has its own page in TIS which is shown in figure 4.3.



Report Processing

Insert report name

Report name:

Database: TIS Report Database

Report attachments and report excel file will be generated. Report name has to be unique.

Figure 4.3: Report Processing tool's page

Executing Report Processing from this page has one advantage - when multiple reports are inserted (separated with a *newline* character) it will process all these reports into one single excel file. This feature cannot be achieved with the RPA platform from the nature of database trigger.

Second place in TIS where Report Processing tool can be run is in TRD (test report database, mentioned in section 3.2). There is a button called *Export TRG excel and attachments* connected to every report which executes the Report Processing tool with the respective report upon clicking. The report viewed in TRD with highlighted *Export TRG excel and attachments* button in red color is shown in figure 4.4

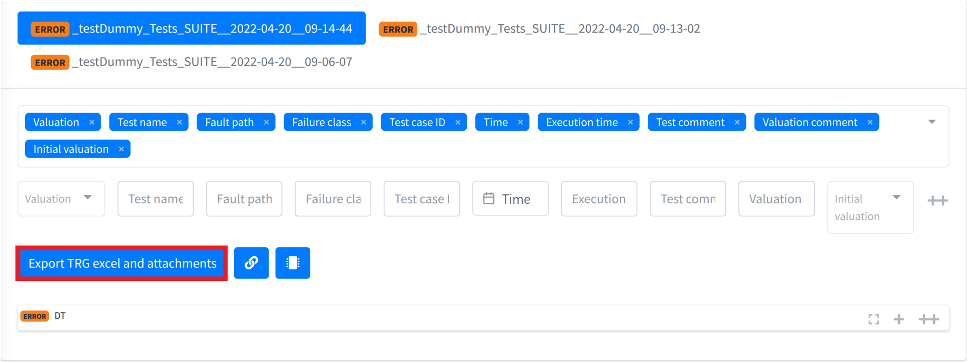


Figure 4.4: Export TRG excel and attachments button in TIS

4.3.3 Zip Files And Send Email

Zip files And Send Email is a concrete implementation of finish task. It takes the files generated by RPA tools (which were executed prior to the finish task) from the current workspace, compresses them in a *zip* format and saves them to the TIS permanent files storage. Then it sends an e-mail with the link to the generated files to the robot owner (the *user* from the *Robot* table of the Robot TIS database). The function to send an e-mail from TIS had also previously been written in TIS backend.

4.4 User interfaces

The user interface offers fast and convenient way to configure robots by users. For this purpose, I created a simple user interface for creating robots which writes the robot data into the Robot TIS database if the configuration is considered valid. Implemented user interface for creating robots is shown in figure 4.5.

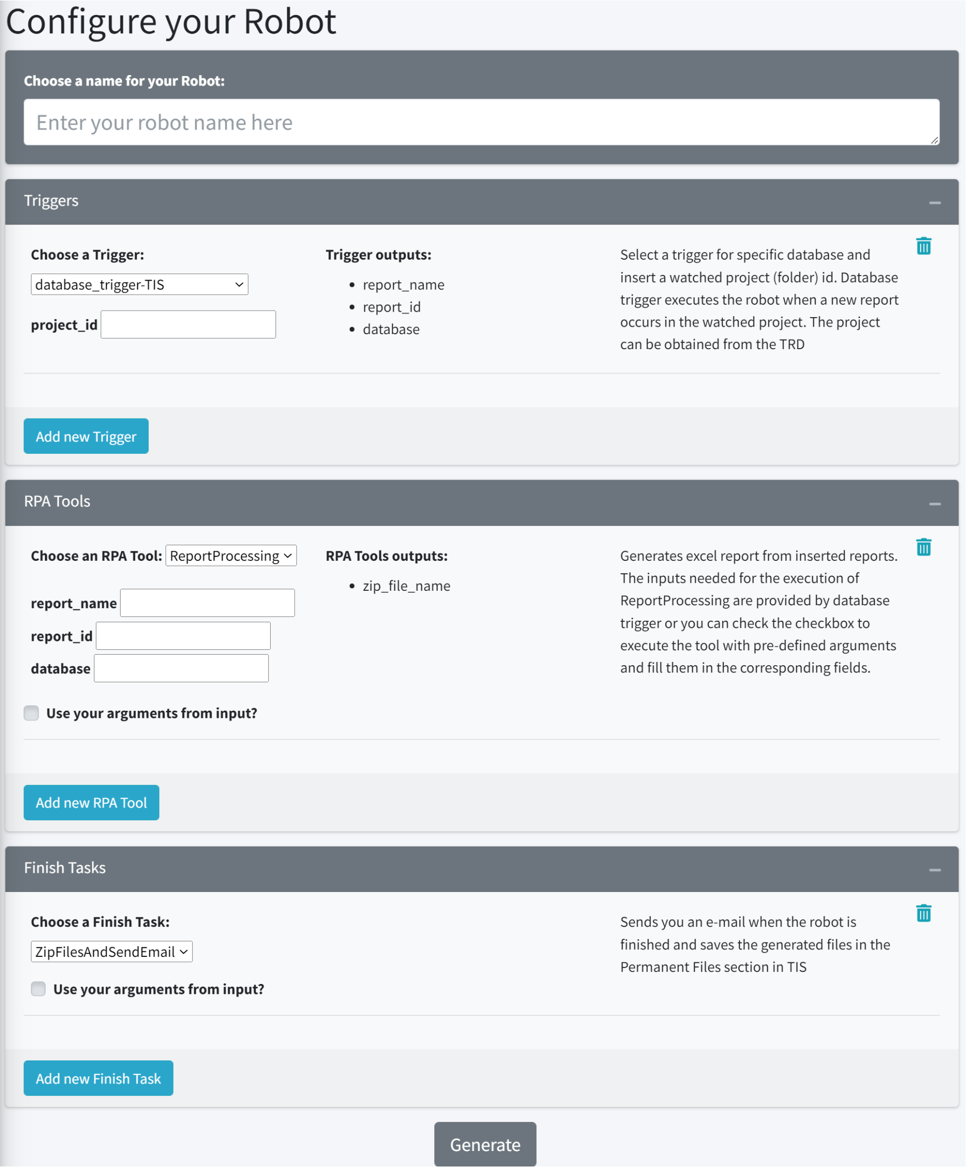
Then there was also created a simple user interface for deleting robots. This user interface shows the user all configured robots belonging to him from the Robot TIS database and it provides the option to delete them. This user interface is shown in figure 4.6.

Both user interfaces were implemented in javascript and HTML with usage of Bootstrap.

4.4.1 User interface for creating robots

Layout

The first field to be filled by the user is the robot name. This is a required field which is later used for robot identification. It does not have to be unique for the user - it is left entirely for the user to choose. Then there is a specific card for RPA tools, finish tasks and triggers. Each of these cards can contain multiple RPA tools (resp. finish tasks, triggers). RPA tool and trigger have three columns in their corresponding card. The first column is for input arguments, the second one is for output arguments and the third one is for a short description of the selected RPA tool (resp. finish task). Finish task has only two columns - one for input arguments and second for the description because finish tasks do not have output arguments. For every configured RPA tool and finish task there is a checkbox that determines if the user prefers to use his inserted input arguments even when they are provided by one of the triggers. The screenshot of the user interface for creating robots is shown in figure 4.5.



Choose a name for your Robot: Enter your robot name here

Choose a Trigger: database_trigger-TIS

project_id

Trigger outputs:

- report_name
- report_id
- database

Select a trigger for specific database and insert a watched project (folder) id. Database trigger executes the robot when a new report occurs in the watched project. The project can be obtained from the TRD

Choose an RPA Tool: ReportProcessing

report_name

report_id

database

□ Use your arguments from input?

RPA Tools outputs:

- zip_file_name

Generates excel report from inserted reports. The inputs needed for the execution of ReportProcessing are provided by database trigger or you can check the checkbox to execute the tool with pre-defined arguments and fill them in the corresponding fields.

Choose a Finish Task: ZipFilesAndSendEmail

□ Use your arguments from input?

Sends you an e-mail when the robot is finished and saves the generated files in the Permanent Files section in TIS

Figure 4.5: RPA platform user interface for creating robots

Implementation

These user interfaces were created in javascript and HTML. Javascript provides dynamic client-side communication and therefore allows the development of complex and dynamic user interfaces and this is exactly the case for the RPA platform.

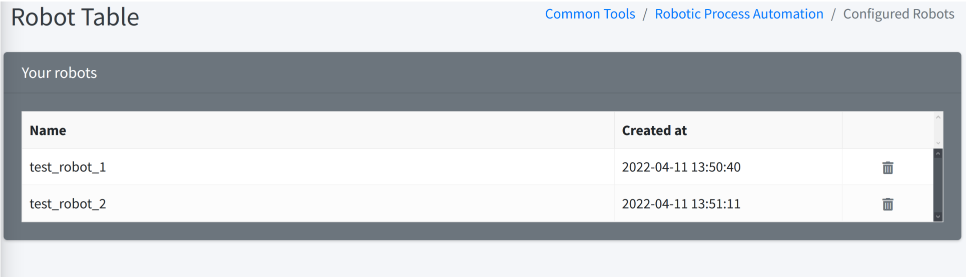
I created a simple HTML template. Specific options for Robot configuration are dynamically loaded from a configuration file based on previously selected options. For example when selecting a different RPA tool (resp. finish task, trigger) the user interface will show different input argument options. Currently, there are no other RPA tools (resp. finish tasks, triggers) but the user interface was created to be modular and is prepared and tested to adapt the form to specific RPA tools (resp. finish tasks, triggers).

The configuration file is the only condition for the user interface to work correctly. The configuration file is stored in TIS. When a developer develops a new trigger, RPA tool or finish task he has to add it to the configuration file so that it is shown correctly in the user interface. When a new RPA tool (resp. finish task, trigger) is selected in the select field of the form, the page will show corresponding input arguments, output arguments and description to the newly selected RPA tool (resp. finish task, trigger). User can add more RPA tools (resp. finish tasks, triggers) by clicking on the *Add new RPA Tool* button (or corresponding buttons for finish tasks, triggers). The user can then delete one of the selected RPA tools (resp. finish tasks, triggers) by clicking on the little bin on the right side from the description.

HTML is very easy to change and thus the validity of the form has to be verified. When the form is being processed (after submission) it has to be reconstructed for these and more security reasons. Data is being written into the database after overall validation of the form. Django framework has a function for verifying these forms but it cannot be used in this case because the form is dynamic and thus the fields of the form are not pre-defined.

4.4.2 User interface for deleting robots

The user interface for deleting robots shows the user all his configured robots with their names and dates of creation. These robots can be sorted by their name or the date of creation. Any robot can be deleted by clicking on the bin located in the right column of the table. When clicking on this bin an alert pops up to verify utter erasure. The screenshot of the user interface for deleting robots is shown in figure 4.6.



Robot Table

Common Tools / Robotic Process Automation / Configured Robots

Your robots

Name	Created at	
test_robot_1	2022-04-11 13:50:40	🗑
test_robot_2	2022-04-11 13:51:11	🗑

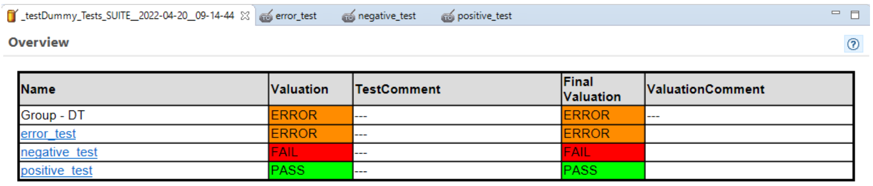
Figure 4.6: RPA platform user interface for deleting robots

Chapter 5

Results

5.1 Report Processing results

For the purpose of the thesis I run three tests that together create a report and this report is used in this thesis to show the functionality of the Report Processing tool and the whole RPA platform. Production tests and reports cannot be shown due to data confidentiality. The reports can be viewed in EXAM and this created report viewed in EXAM is shown in the following figures. Figure 5.1 shows the overall report result and the results of the tests.

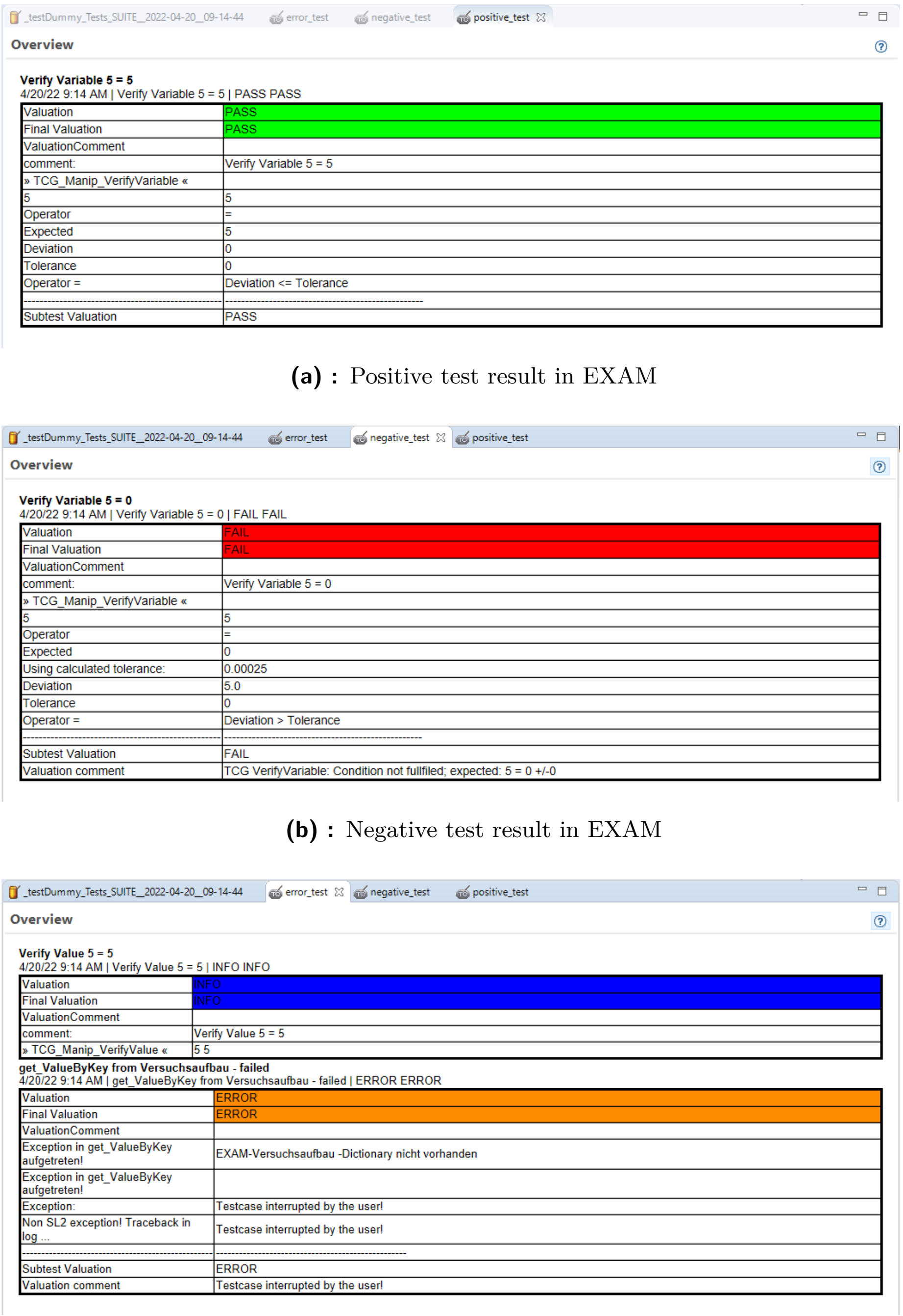


Overview

Name	Valuation	TestComment	Final Valuation	ValuationComment
Group - DT	ERROR	---	ERROR	---
error_test	ERROR	---	ERROR	
negative_test	FAIL	---	FAIL	
positive_test	PASS	---	PASS	

Figure 5.1: Overall report result in EXAM

The next figures show the EXAM view of positive (5.2a), negative (5.2b) and error (5.2c) test results. A positive test result means that everything in the testing process went according to plan. A negative test result means that the test failed and the result was not expected. An error test result means something happened during the testrun that caused the test to crash.


_testDummy_Tests_SUITE__2022-04-20__09-14-44
error_test
negative_test
positive_test
Overview

Verify Variable 5 = 5
4/20/22 9:14 AM | Verify Variable 5 = 5 | PASS PASS

Valuation	PASS
Final Valuation	PASS
ValuationComment	
comment:	Verify Variable 5 = 5
» TCG_Manip_VerifyVariable «	
5	5
Operator	=
Expected	5
Deviation	0
Tolerance	0
Operator =	Deviation <= Tolerance
--	--
Subtest Valuation	PASS

(a) : Positive test result in EXAM

_testDummy_Tests_SUITE__2022-04-20__09-14-44
error_test
negative_test
positive_test
Overview

Verify Variable 5 = 0
4/20/22 9:14 AM | Verify Variable 5 = 0 | FAIL FAIL

Valuation	FAIL
Final Valuation	FAIL
ValuationComment	
comment:	Verify Variable 5 = 0
» TCG_Manip_VerifyVariable «	
5	5
Operator	=
Expected	0
Using calculated tolerance:	0.00025
Deviation	5.0
Tolerance	0
Operator =	Deviation > Tolerance
--	--
Subtest Valuation	FAIL
Valuation comment	TCG VerifyVariable: Condition not fullfiled; expected: 5 = 0 +/-0

(b) : Negative test result in EXAM

Verify Value 5 = 5
4/20/22 9:14 AM | Verify Value 5 = 5 | INFO INFO

Valuation	INFO
Final Valuation	INFO
ValuationComment	
comment:	Verify Value 5 = 5
» TCG_Manip_VerifyValue «	5 5

get_ValueByKey from Versuchsaufbau - failed
4/20/22 9:14 AM | get_ValueByKey from Versuchsaufbau - failed | ERROR ERROR

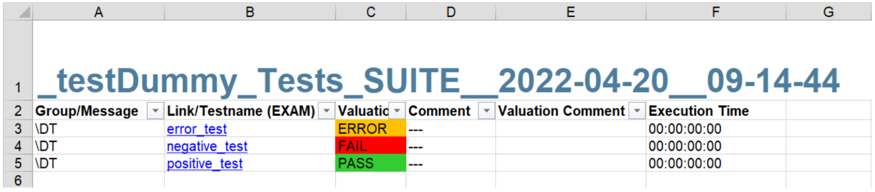
Valuation	ERROR
Final Valuation	ERROR
ValuationComment	
Exception in get_ValueByKey aufgetreten!	EXAM-Versuchsaufbau -Dictionary nicht vorhanden
Exception in get_ValueByKey aufgetreten!	
Exception:	Testcase interrupted by the user!
Non SL2 exception! Traceback in log ...	Testcase interrupted by the user!
--	--
Subtest Valuation	ERROR
Valuation comment	Testcase interrupted by the user!

(c) : Error test result in EXAM

Figure 5.2: Each test of report shown in EXAM

EXAM contains more detailed information about each test.

The purpose of the robot using Report Processing tool is the automated generation of an excel file containing all report relevant data from EXAM. The result of the Report Processing robot (the excel file) is shown in the following figures. Overall report result is shown in figure 5.3.

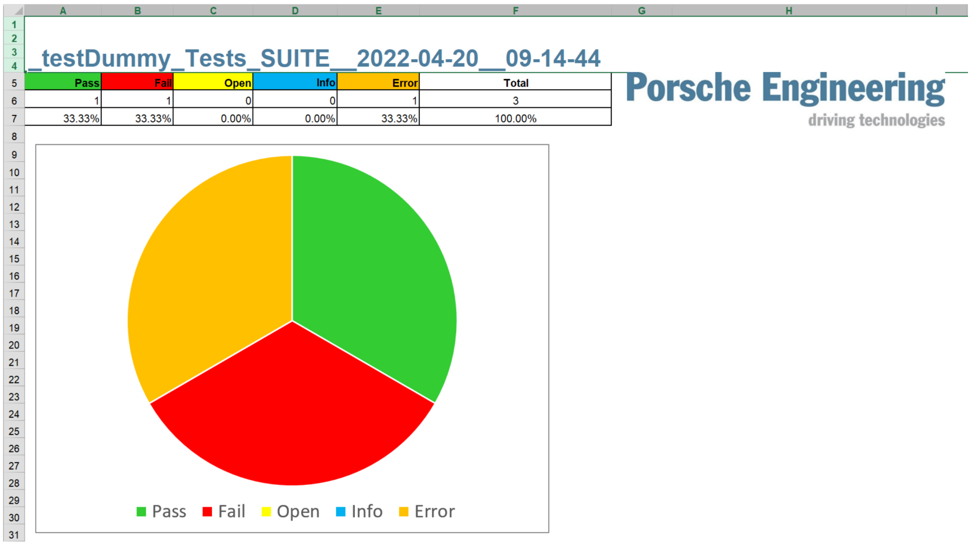


_testDummy_Tests_SUITE__2022-04-20__09-14-44

Group/Message	Link/Testname (EXAM)	Valuatic	Comment	Valuation Comment	Execution Time
\DT	error_test	ERROR	---		00:00:00:00
\DT	negative_test	FAIL	---		00:00:00:00
\DT	positive_test	PASS	---		00:00:00:00

Figure 5.3: Overall report result in excel

The title page is shown in figure 5.4 and it shows the overall test results in a graph.

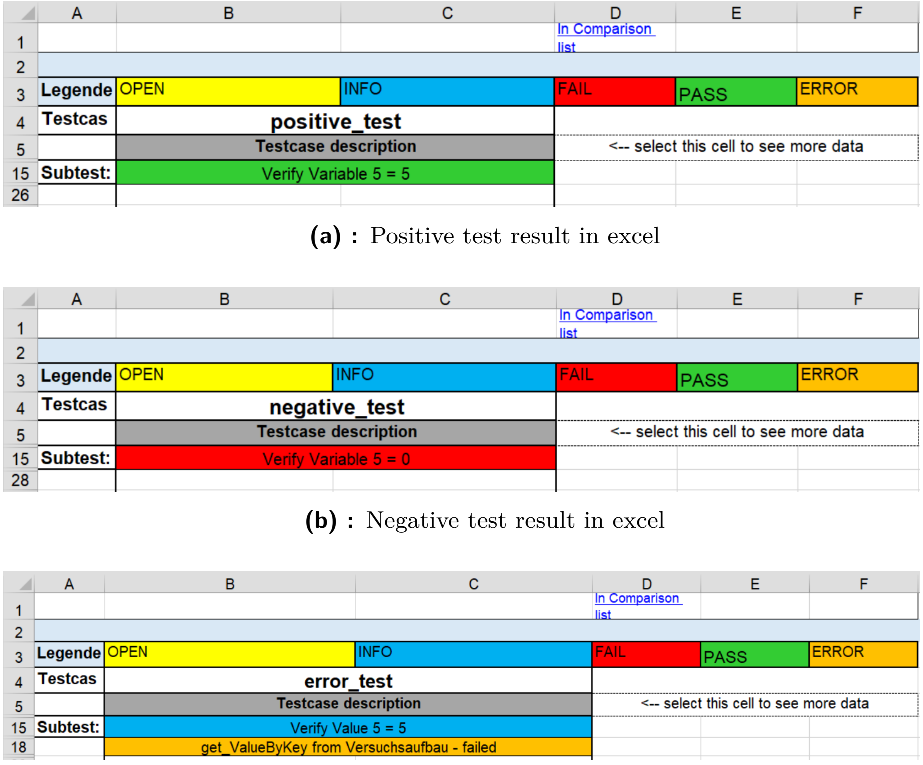


_testDummy_Tests_SUITE__2022-04-20__09-14-44

Pass	Fail	Open	Info	Error	Total
1	1	0	0	1	3
33.33%	33.33%	0.00%	0.00%	33.33%	100.00%

Figure 5.4: Excel file title page

The following figures show the same positive (5.5a), negative (5.5b) and error (5.5c) test results as were shown in the beginning of this section in the generated excel file.



	A	B	C	D	E	F
1				In Comparison list		
2						
3	Legende	OPEN	INFO	FAIL	PASS	ERROR
4	Testcas	positive_test				
5		Testcase description		<-- select this cell to see more data		
15	Subtest:	Verify Variable 5 = 5				
26						

(a) : Positive test result in excel

	A	B	C	D	E	F
1				In Comparison list		
2						
3	Legende	OPEN	INFO	FAIL	PASS	ERROR
4	Testcas	negative_test				
5		Testcase description		<-- select this cell to see more data		
15	Subtest:	Verify Variable 5 = 0				
28						

(b) : Negative test result in excel

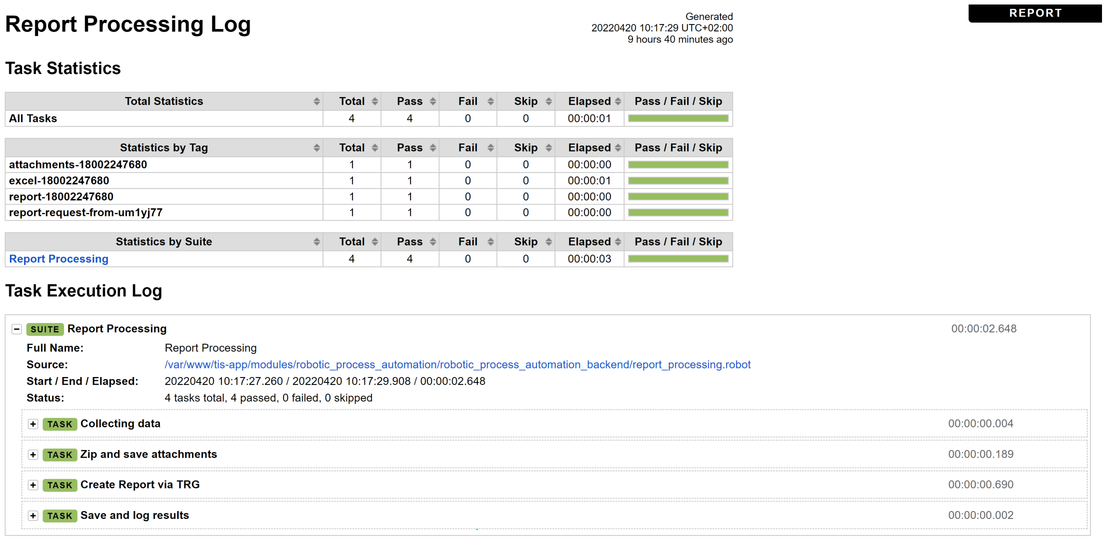
	A	B	C	D	E	F
1				In Comparison list		
2						
3	Legende	OPEN	INFO	FAIL	PASS	ERROR
4	Testcas	error_test				
5		Testcase description		<-- select this cell to see more data		
15	Subtest:	Verify Value 5 = 5				
18		get_ValueByKey from Versuchsaufbau - failed				

(c) : Error test result in excel

Figure 5.5: Each test of report shown in excel

Excel file also contains more detailed information about each test.

These tests did not contain attachments, so the Report Processing tool did not generate them. Along with the generated excel files the Report Processing tool also returns an HTML log file which shows the information about the Report Processing program execution. This log file generation is a feature of Robot Framework and is shown in figure 5.6. The log contains information about every task and its result. The log file in the figure below shows that every task had been executed successfully. Robot Framework also creates a report from performed tasks. This report is similar to the Robot Framework log file.



Report Processing Log

Generated
20220420 10:17:29 UTC+02:00
9 hours 40 minutes ago

REPORT

Task Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tasks	4	4	0	0	00:00:01	🟩

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
attachments-18002247680	1	1	0	0	00:00:00	🟩
excel-18002247680	1	1	0	0	00:00:01	🟩
report-18002247680	1	1	0	0	00:00:00	🟩
report-request-from-um1yj77	1	1	0	0	00:00:00	🟩

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Report Processing	4	4	0	0	00:00:03	🟩

Task Execution Log

```
−  SUITE  Report Processing                                                    00:00:02.648
   Full Name:            Report Processing
   Source:               /var/www/tis-app/modules/robotic_process_automation/robotic_process_automation_backend/report_processing.robot
   Start / End / Elapsed: 20220420 10:17:27.260 / 20220420 10:17:29.908 / 00:00:02.648
   Status:               4 tasks total, 4 passed, 0 failed, 0 skipped
   +  TASK  Collecting data                                                     00:00:00.004
   +  TASK  Zip and save attachments                                            00:00:00.189
   +  TASK  Create Report via TRG                                               00:00:00.690
   +  TASK  Save and log results                                                00:00:00.002
```

Figure 5.6: Report Processing log file

5.2 RPA platform results

RPA platform was being tested for full two days before it was released to production TIS and it works as expected. Various employees configured many robots and when a new report appeared in the watched project configured by the employee it started the robot execution. All the robots configured in the testing process consisted of an arbitrary number of *database* triggers, *Report Processing* RPA tools and *Zip Files And Send Email* finish tasks. Performed tests are shown in table 5.1. The table shows that for various numbers of triggers, RPA tools and finish tasks the robot execution was successful.

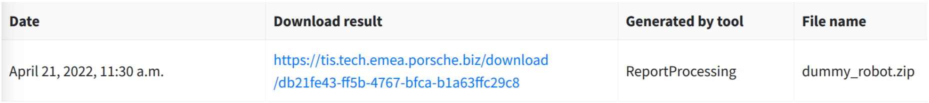
Number of triggers	Number of RPA tools	Number of finish tasks	Success
1	1	1	yes
1	1	2	yes
1	2	1	yes
2	1	1	yes
2	2	1	yes
2	2	2	yes
2	3	1	yes
4	1	1	yes
4	2	1	yes

Table 5.1: Performed tests before upload to production TIS

These tests were performed (as was said before) with only one type of trigger, one type of RPA tool and one type of finish task. Robot configured with more than one type of the same trigger may not make sense but it proved the modularity and universality of the whole RPA platform. It proves the concept that the RPA platform can be used with multiple different triggers, RPA tools and finish tasks which was one of the main goals to achieve.

5.2.1 Configured robot to demonstrate the process

To demonstrate the process I created a simple robot consisting of one *database* trigger, one *Report Processing* RPA tool and *Zip Files And Send Email* finish task. I configured the database trigger to watch one specific project. Then a colleague copied the report into the watched project. After two runs of the triggering script I received an e-mail with a link to the zipped folder with the created report and the report also appeared in TIS permanent files storage which is shown in figure 5.7. From TIS permanent files it can be downloaded and the downloaded folder contains the generated excel file from the report (described in section 5.1 in detail) and Robot Framework log and report. There are no attachments since no attachments were belonging to the report.



Date	Download result	Generated by tool	File name
April 21, 2022, 11:30 a.m.	https://tis.tech.emea.porsche.biz/download /db21fe43-ff5b-4767-bfca-b1a63ffc29c8	ReportProcessing	dummy_robot.zip

Figure 5.7: Generated zipped folder in TIS permanent files storage

5.3 Comparison with the previous state in Porsche Engineering Services

Report Processing saves an enourmous amount of time of the employees. I compared the time needed to process reports before RPA automation tool Report Processing and after to demonstrate that fact. The time needed for processing necessary reports consisted of downloading XML files and attachments from EXAM and then manual TRG execution. After RPA automation it consists only of Report Processing execution. The comparison of the times before and after RPA automation with the Report Processing tool is shown in table 5.2. Each row of the table corresponds to one report. I measured the time needed for downloading all necessary files from EXAM and the time needed for TRG execution separately and the sum of these times (red color column in the table) represents the overall time needed for processing reports before RPA automation. The overall time for processing reports after RPA automation consists of the execution time of Report Processing (green color in the table).

EXAM download time [min]	TRG time [min]	Overall time before RPA [min]	Overall time after RPA [min]	Saved time [%]
7:22.14	2:23.15	**9:45.29**	**4:12.71**	**56.82 %**
0:16.35	0:50.83	**1:07.18**	**0:13.27**	**80.25 %**
1:03.30	0:51.92	**1:55.22**	**0:45.75**	**60.30 %**
2:22.53	1:40.94	**4:03.47**	**1:18.89**	**67.60 %**
1:59.62	1:30.21	**3:29.83**	**1:50.86**	**47.17 %**
3:08.78	1:03.97	**4:12.75**	**0:34.46**	**86.37 %**
22:35.47	4:21.97	**26:57.44**	**2:33.64**	**90.50 %**
23:11.08	4:51.36	**28:02.44**	**2:42.85**	**90.32 %**

Table 5.2: Measured time of report generation before and after RPA

Data in the table above are only indicative. All times presented in the table depend on the TIS workload, EXAM workload, the speed of access to databases and other factors. The time needed for processing reports before usage of RPA automation additionally depends on the speed of the employee clicking on the buttons needed to download XML files from EXAM and then upload them to TRG. Furthermore, there may be a time lag between EXAM XML files download and their upload to TRG.

The following subsections describe the advantages of using Report Processing RPA automation.

5.3.1 Indicative lower estimate of time savings per week

The first advantage is that it saves time for employees. I found every report generation via TIS in the TIS default database and there were 616 report generations in 10-week interval (14.2.2022 - 22.4.2022). That means there is an average of **61.6 report generations per week**. The number is relatively consistent every week. The average time of report generation is extremely difficult to determine since these data are nowhere to be found, they are not stable and they are changing from report to report.

To calculate overall time savings, only the time of downloading XML files from EXAM is taken into consideration since this task eliminates the employee's ability to work and the employee cannot focus on other important tasks until the downloading is finished. The time of TRG execution and Report Processing execution does not block the employee because it is executed in TIS and he can do other things while waiting for the report to generate.

The average time of downloading XML files from EXAM from measured data in minutes is 7:44.91. This average time is only indicative since the report sizes and the download times differ in wide range as was already mentioned. There are some reports which were not measured because their download time is in the order of hours and it would be a waste of time for employees to measure these times. After consultation with employees **7 minutes** seem like a reasonable lower estimate of average time spent with downloading XML files from EXAM.

The lower estimate calculations of saved time per week is calculated as the average count of report generations per week multiplied by the lower estimate of average time spent with downloading XML files from EXAM. The lower estimate calculations of saved time per week is:

7 hrs 11 min 12 sec.

5.3.2 Faster report delivery to customers

Another significant advantage is the ability to deliver reports to customers faster than it was before RPA automation. The time difference displayed in table 5.2 may not seem significant at first, however, for example when many reports are finished during the weekend it may take hours to generate final excel reports from all of them. Since employees have to focus on other tasks as well, they are not able to generate all necessary reports for one customer at once and the whole process may take several days. This leads to customers dissatisfaction.

With the Report Processing RPA automation tool it is possible to process all reports in parallel and it is faster in general. For example, before Report Processing, some reports which were finished on weekend were being delivered on Thursday and with the Report Processing tool it can be done on Monday right after the weekend.

5.3.3 Overall employee's comfort

Another advantage is that employees are satisfied with Report Processing since it is a more user-friendly solution of generating reports. There is no need to use EXAM when generating the reports which is very convenient because EXAM sometimes stops responding and when it happens in the middle of downloading XML files it is necessary to start the whole process again which may be frustrating. Downloading XML files can take even longer when performed from home office via VPN (virtual private net) because the connection is a lot slower. For this reason, there is a computer in the office used exclusively for generating reports by employees working from home via remote connection. With Report Processing, there is no need for this computer which is also a benefit. Furthermore, employees can focus on other tasks while generating the report compared to the previous state without RPA automation as was mentioned before. Employee's satisfaction underlines the importance of Report Processing.

5.4 User experience questionnaire

For the evaluation of the whole RPA project I created a user experience questionnaire. This questionnaire has been filled out by 10 employees from Porsche Engineering Services and it provided a valuable both positive and negative feedback for the whole RPA platform and Report Processing. In addition, some employees also presented their opinions and ideas for possible improvements.

The survey lasted from 11.4.2022 to 27.4.2022. Employees had enough time to try out the RPA platform and the Report Processing tool. They could have tried it out even before the user experience survey started since it was put into production before the survey.

Important aspect to mention is that the user experience questionnaire focuses mainly on what users can see - the graphical user interfaces. Implemented graphical user interface was not a subject of thesis but still it had to be made to prove the concept of the implemented platform's functionality.

5.4.1 Positive feedback

Positive findings gathered from the questionnaire:

- 9 out of 10 people were able to configure a robot,
- every created robot worked according to user expectations,
- 9 out of 10 people found the short documentation in user interface helpful,
- 7 out of 10 people understood the functionality of checkbox present in every RPA tool and finish task,
- 8 out of 10 people found the Report Processing tool helpful; 2 remaining people do not work with reports,
- people rate the overall project highly - they gave it a grade 2.00 on scale 1-5 (perfect-awful) in average,
- some users also shared their opinion in textfield and they wrote:
  - *"Time saver, process simplifier tool. Thank you"*,
  - *"UI(Colors used) and alignment of texts in Create robot page"*,
  - *"nice option of configuring own robots"*, [1]
  - *"process acceleration"*,
  - *"overall the tool works good"*.

[1] translation from Czech language

5.4.2 Negative feedback

Negative findings gathered from the questionnaire:

- only 4 out of 10 people understood the functionality of the triggers, RPA Tools and Finish Tasks (mainly people who have not been informed about the project before filling the questionnaire)

- people were not sufficiently satisfied with the user interface for creating robots - they gave it a grade 2.30 on scale 1-5 (perfect-awful) in average

- people were not sufficiently satisfied with the user interface for deleting robots - they gave it a grade 2.30 on scale 1-5 (perfect-awful) in average

- some users also shared their opinion in textfield and they wrote:
  - *"I would suggest to change some words used to describe how to use RPA Tools(RPA tools section) to make it very well undestandable. + once deleting configured robot, in the alert displayed that user needs to confirm, I think was supposed to display created at 'date when robot created' but its showing undefined (maybe can be fixed.)"*,
  - *"not sure what does the 'Use your argumnets from input' mean"*, [2]
  - *"I think for the configuration, the interface could be more self-explanatory, as well for the application, I did not really understand how the robot works until I asked for further info."*.

5.4.3 Suggested improvements

Some people also suggested some possible improvements:

- *"Copying the report to archive. Generate pdf Rename the report with SW version and calendar week for example SW0551_b02___KW31 Frontend should be redesign again"*,

- *"Would be mind blowing to have the option to create the RPA over the context menu in EXAM, so user can create robot right after report was checked by test engineer."*,

- *"I would appreciate more specific illustrations what to insert to every field with a more detailed description or an illustrative example."*, [2]

- *"Time will show"*,

- *"I think just more information should be added so it would be more understandable to use."*.

[2] translation from Czech language

5.4.4 Questionnaire contribution to the project

Named improvement suggestions, positive and negative feedback are very helpful and they can help with the improvement of the project a lot. Since this was the platform's first release and even the first release of such project in Porsche Engineering Services, I consider the questionnaire outcome great and valuable.

The immense problem seems to be a misunderstanding of some features, especially the specific functionality of implemented triggers, RPA tools and finish tasks. Both user interfaces should be more intuitive since they both achieved exactly the same rating 2.30 which is not satisfactory. The most significant positive aspect of the whole project is that it is considered helpful amongst employees and it is working as expected, which was precisely the main goal of the thesis in contrast with the user interfaces, which were not.

The importance of the user quentionnaire lies in one more aspect. The programmer cannot know what is easily understandable for the users and what is not. For example, I thought that everyone would understand the functionality of each implemented trigger, RPA tool and finish task. It turned out to be the opposite since only 4 out of 10 people understood that. Then I anticipated that the functionality of checkbox for using custom arguments would be misunderstood. Once again the users surprised me because the majority of them interpreted the checkbox's function accurately. The collected feedback helps the programmers to focus on the aspects which represent real problems for users.

Chapter 6
Conclusion

6.1 Summary of results

The main result of this thesis is the implemented RPA platform. This platform provides a whole ecosystem which allows employees to create a software robot that performs Robotic Process Automation tasks. The robot consists of an arbitrary number of triggers (which start the robot execution), RPA tools (which perform the specific task) and finish tasks (which do some post-processing of the RPA tools results). The user makes his custom combination of triggers, RPA tools and finish tasks during the robot configuration process. After the configuration, if any of the robot's configured triggers is triggered, the robot is executed. This platform is currently in usage in Porsche Engineering Services.

One trigger, one RPA tool and one finish task were implemented to prove the concept of the platform's functionality and to help employees with the first RPA task - generating reports automatically after the report is ready to be generated. This first automation tool turned out to save time and energy of employees who were generating those reports manually before the RPA integration.

6.2 Fulfillment of partial tasks of the thesis

The partial tasks of the thesis are:

- analyze HiL testing process in selected project and its possibilities for process automation - elaborated in section 3.2,
- design platform and whole ecosystem for robots that interconnect already automated tools within new ecosystem - elaborated in section 3.3,
- integrate this solution into existing HiL testing process - elaborated in chapter 4,
- summarize the activity and usage of the robots compared with previous state - elaborated in chapter 5.

I consider all these partial tasks fulfilled and, therefore, the thesis in its entirety.

6.3 Future work

There are many suggestions for future improvements to the designed RPA platform. The platform is meant to be extended to contain more triggers, RPA tools and finish tasks to be more general and to provide more functionality. Therefore, future work can be implementing additional triggers, RPA tools, and finish tasks and adding them correctly to the platform (the platform has documentation with a guide to add new RPA tools and finish tasks). The implemented user interface for creating robots (which was not a subject of the thesis) could be improved to be more user-friendly and to have a more appealing design. Another possible future improvement can be implementing a feature to edit already configured robots and a feature to share robots with more users (create groups for sharing robots).

Bibliography

[1] TAULLI, Tom. The Robotic Process Automation Handbook: A Guide to Implementing RPA Systems. 1. Monrovia, CA, USA: Apress, 2020. ISBN 978-1-4842-5729-6.

[2] AXELROD, Arnon. Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. 1. Matan, Israel: Apress, 2018. ISBN 1484238311.

[3] BOOT, R., J. RICHERT, H. SCHUTTE and A. RUKGAUER. Automated test of ECUs in a hardware-in-the-loop simulation environment. Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No.99TH8404). IEEE, 1999, 587-594. ISBN 0-7803-5500-8. Available at: doi:10.1109/CACSD.1999.808713

[4] Design and implementation of HIL simulators for powertrain control system software development. Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251). IEEE, 1999, 1999, 709-713 vol.1. ISBN 0-7803-4990-3. Available at: doi:10.1109/ACC.1999.782919

[5] JOSHI, A., Hardware-in-the-Loop (HIL) Implementation and Validation of SAE Level 2 Autonomous Vehicle with Subsystem Fault Tolerant Fallback Performance for Takeover Scenarios, SAE Technical Paper 2017-01-1994, 2017, Available at: doi:10.4271/2017-01-1994

[6] SCALEXIO [online]. dSPACE [cit. 2022-05-02]. Available at: https://www.dspace.com/en/inc/home/products/hw/simulator_hardware/scalexio.cfm

[7] MicroNova - Sofware and Systems. MicroNova [online]. [cit. 2022-02-18]. Available at: https://www.micronova.de/en/home.html

[8] VEN, Kris, Jan VERELST and Herwig MANNAERT. Should You Adopt Open Source Software?. IEEE Software. 2008, 25(3), 54-59. ISSN 0740-7459. Available at: doi:10.1109/MS.2008.73

[9] LESHOB, Abderrahmane, Audrey BOURGOUIN and Laurent RENARD. Towards a Process Analysis Approach to Adopt Robotic Process Automation. 2018 IEEE 15th International Conference on e-Business Engineering

(ICEBE). IEEE, 2018, 2018, 46-53. ISBN 978-1-5386-7992-0. Available at: doi:10.1109/ICEBE.2018.00018

[10] Django: The web framework for perfectionists with deadlines [online]. 2022 [cit. 2022-03-04]. Available at: https://docs.djangoproject.com/en/4.0/

[11] MariaDB. MariaDB [online]. Tekniikantie 12, 02150 Espoo, Finland, 2022 [cit. 2022-04-25]. Available at: https://mariadb.com/

[12] SHVETS, Alexander. Refactoring.Guru. Refactoring.Guru [online]. Khmelnitske shosse 19 / 27, Kamianets-Podilskyi, Ukraine, 32305, 2022 [cit. 2022-03-10]. Available at: https://refactoring.guru/

[13] FREEMAN, Eric, Elisabeth ROBSON, Kathy SIERRA and Bert BATES. Head First design patterns. Beijing: O'Reilly, 2014. ISBN 978-0596007126.

[14] BISHT, Sumit. Robot Framework Test Automation: Create test suites and automated acceptance tests from scratch. 1. Birmingham, UK: Packt Publishing, 2013. ISBN 978-1-78328-303-3.

[15] GAMI, Manishkumar and JETLY, Parth and MEHTA, Nidhi and PATIL, Sunita, Robotic Process Automation – Future of Business Organizations: A Review (April 8, 2019). 2nd International Conference on Advances in Science & Technology (ICAST) 2019 on 8th, 9th April 2019 by K J Somaiya Institute of Engineering & Information Technology, Mumbai, India, Available at: SSRN: https://ssrn.com/abstract=3370211 or http://dx.doi.org/10.2139/ssrn.3370211

[16] ISSAC, Ruchi, Riya MUNI and Kenali DESAI. Delineated Analysis of Robotic Process Automation Tools. 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC). IEEE, 2018, 2018, 1-5. ISBN 978-1-5386-3785-2. Available at: doi:10.1109/ICAECC.2018.8479511

[17] SIVAJI, Ashok, Rosnisa Abdul RAZAK, Nur Faezah MOHAMAD, et al. Software Testing Automation: A Comparative Study on Productivity Rate of Open Source Automated Software Testing Tools For Smart Manufacturing. 2020 IEEE Conference on Open Systems (ICOS). IEEE, 2020, 2020-11-17, 7-12. ISBN 978-1-7281-9020-4. Available at: doi:10.1109/ICOS50156.2020.9293650

[18] Automation Platform - Leading RPA company: UiPath [online]. 2022 [cit. 2022-01-24]. Available at: https://www.uipath.com/

[19] Robot Framework [online]. Helsinki, Finland, 2022 [cit. 2022-02-12]. Available at: https://robotframework.org/

Appendix A

Content of enclosed codes

Enclosed codes are divided into 3 folders:

- *database_trigger_script* - this folder contains only database trigger script source code,
- *django_application_files* - this folder contains user interfaces implementation shown only for completion (since it was not the main objective of the thesis),
- *robotic_process_automation_backend* - this folder contains source codes for the platform's backend implementation.

All enclosed codes cannot be executed without TIS. TIS source codes are confidential and cannot be enclosed. Some codes implemented as a part of this thesis were not enclosed due to data confidentiality as well.