Nam Lê

AUTOMATIC TESTING SCHEDULER

Information Technology
2019

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Nam Lê
Title	Automatic Testing Scheduler
Year	2019
Language	English
Pages	42 + 2 Appendices
Name of Supervisor	Timo Kankaanpää

This thesis evaluates Automatic Test Framework's behaviour when scheduling test case to run on real hardware sets. The company has a limited amount of test hardware sets for developers to manually choose to perform different test cases. Each test case requires a different hardware setup and a different execution time. Ideally, a test should be performed on a minimal required hardware set that satisfies the test requirements. As a result, it complicates finding the most optimal way of scheduling the test execution manually. Furthermore, test hardware is expensive, and the company wants to invest more on software solution to utilize test hardware to serve testing purpose instead of buying more hardware.

Hence, a solution, which runs on a web application, is designed to automatically take all the mentioned information into consideration and provide the most optimal schedule for the test execution planning. The result of this work helps to reduce manual work and time spent on the testing phase.

Keywords Automation, testing

CONTENTS

LIST OF ABBREVIATIONS

A&C	Automation and Control
ATF	Automatic Test Framework
API	Application Programming Interface
CI	Continuous Integration
Rack/Racks	Refers to hardware set/hardware sets
HW	Hardware

LIST OF FIGURES AND TABLES

1 INTRODUCTION

1.1 Wärtsilä Finland Oy

1.1.1 Brief information

"Wärtsilä is a global leader in smart technologies and complete lifecycle solutions for the marine and energy markets. By emphasizing sustainable innovation, total efficiency and data analytics, Wärtsilä maximizes the environmental and economic performance of the vessels and power plants of its customers." /1/

Wärtsilä focus on marine and energy business, which can be described as following terms:

- Innovative products
- Integrated solutions
- Environmentally sustainable energy
- Efficient, flexible and economical
- Worldwide network services

In 2019, Wärtsilä's net sales totalled EUR 5.3 billion with approximately 19,000 employees. The company has operations in over 200 locations in more than 80 countries around the world. Wärtsilä is listed on Nasdaq Helsinki. /1/

As of January 2019, Wärtsilä consists of two businesses; Marine Business and Energy Business. Services Business has been incorporated into Marine and Energy Businesses.

Wärtsilä Marine Business enhances the business of its marine and oil & gas industry customers by providing innovative products and integrated solutions that are safe, environmentally sustainable, efficient, flexible, and economically sound. Being a technology leader, and through the experience, know-how and dedication of its personnel, Wärtsilä can customize solutions that provide optimal benefits to its customers around the world.

Wärtsilä Energy Business is leading the transition towards a 100% renewable energy future. As an Energy System Integrator, Wärtsilä understands, designs, builds and serves optimal power systems for future generations. Wärtsilä's solutions provide the needed flexibility to integrate renewables and secure power system reliability. Their offering comprises engine-based flexible power plants – including liquid gas systems – hybrid solar power plants, and energy storage and integration solutions. Wärtsilä supports the

customers over the lifecycle or their installations with services that enable increased efficiency and guaranteed performance. Wärtsilä has 70 GW of installed power plant capacity in 177 countries around the world.

Wärtsilä has an important role in meeting the world's increased demand for energy in a sustainable way. This is the cornerstone of their commitment to sustainability. They provide a sustainability approach which based on economic, environmental and social performance.

1.1.2 Automation and Control department

The department, where the thesis work was carried out, is part of Marine Solutions, Research Development and Engineering department of Wärtsilä. A&C develops engine process performance and functionality with the following guides: /1/

1. Missions
    - Convert customer's requirement and future needs to optimized performance solutions.
    - Provide engine process and controls expertise to ensure the most competitive level of quality, performance and cost throughout the lifetime of the product.
2. Visions
    - Connects engine processes and controls to provide engine performance beyond customer expectations.
3. Values
    - Energy – Pushing the limits
    - Excellence – Knowledge driven
    - Excitement – Passion for engines

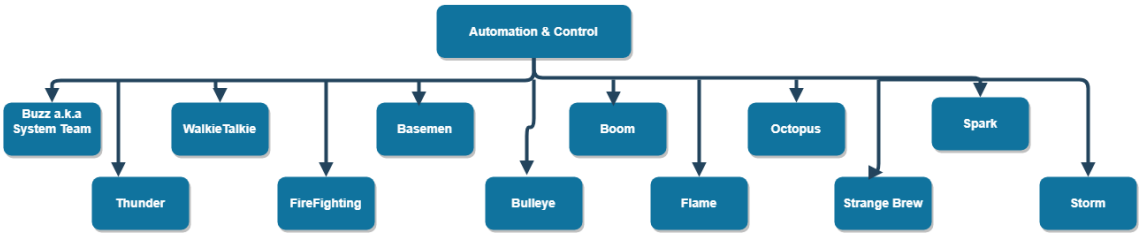

Figure 1. Automation & Control department feature team

A&C department has in total twelve teams which is responsible for variety of engine automation system development. **(Figure 1)**

1.2 Objectives

The primary goal is to apply the solution in scheduling test cases to run on the HW sets. The algorithm needs to be able to evaluate the test case requirements against the HW sets description and find out which sets are suitable for the current test. The algorithm should pick the minimal required HW set that satisfies the test requirements and proceed to run the test on it. Before running the algorithm, developers will be notified about the processes and all the requirements from test cases and HW sets description are collected to serve the evaluation and what to expect is the clear result of which test case would be run on which HW set.

1.3 Scope

The thesis's scope intends to serve the purpose of maximizing the efficiency of the HW sets in use for software testing of the System Team, which is one of the Feature Teams. This thesis's work focused on developing a web application, which will take different requirements as input and be able to output, depends on the inputs, HW set ID or both HW set ID and wait time of each set, follow by adding the HW set ID in the schedule to run the test. Different approaches were proposed to deliver the solution to solve the current situation that the testing system is facing, as shown in the following diagrams (**Figure 2** and **Figure 3**):

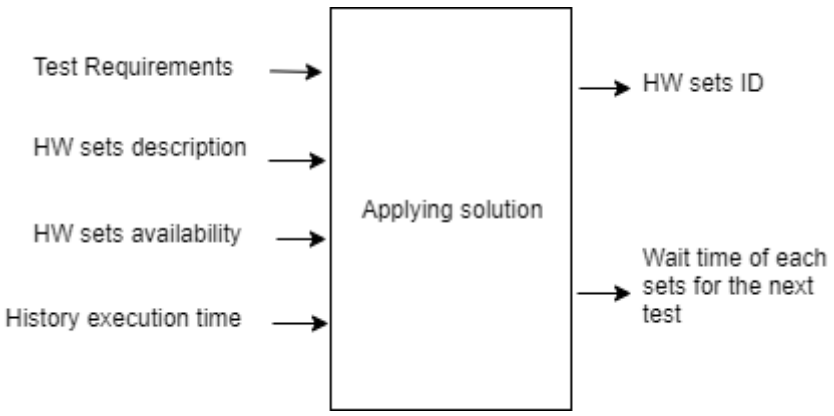

Figure 2. Inputs/Outputs of the program

A simple version of the algorithm will compare the test requirements to the HW set description and find out which set is suitable for the currently waiting test. The algorithm should pick the minimal required HW set that satisfies the test requirements. A more complex version of the algorithm will output HW set ID as a must and wait time of each set as an option and put it in queue for executing the test. With the selected approach, a corresponding solution will be designed and implemented.

1.4 Background technique information

Teams in A&C use a test framework called Automatic Test Framework, a tool developed by Wapice Oy to perform automatic tests. ATF is built upon TestNG, a tool inspired by Junit and NUnit with new functionalities which make the framework easier to use and more powerful. TestNG is designed to create module tests and functional tests in Java, such as unit, functional, end-to-end, integration, etc.… Following is some of the functionalities that TestNG provides: /2/

- Annotations.
- Test multithread safe.
- Flexible test configuration.

- Support for data-driven testing
- Support for parameters.
- Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc.…).

2 THE CURRENT SYSTEM AND PROBLEM ANALYSIS

The thesis is divided into three phases, including pre-study phase, implementation phase and review phase. The pre-study phase takes most of the time since the initial knowledge and analysis have a huge impact on how the problem is defined, and the complication of the solution is based on the problem scope.

The pre-study phase consists of access to the current ATF system, test cases implementation, configuring testing system and address the problem that the current system is facing.

2.1 The current system

2.1.1 Brief information

ATF consists of atf-core, a library written in Java to directly perform tests through the application programming interface. The most important part of atf-core includes System access and Rack control (HW sets control). (**Figure 3**)

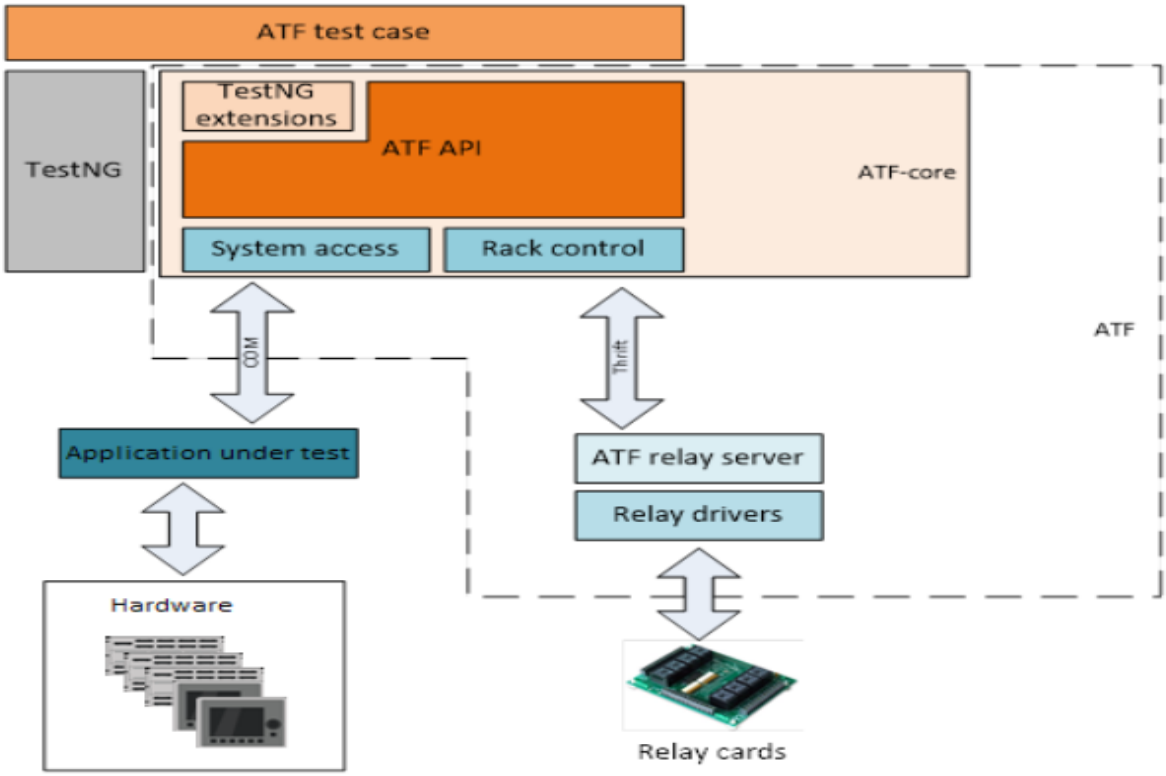

Figure 3. ATF with System access and Rack control /5/

ATF provides a feature to modify the HW set status and control the software of the hardware sets. The Rack control is responsible for the HW control. It can communicate with the ATF relay-server, which runs a separate process to handle the control of the relays that control HW. The detail of how the communication occurs with the application being tested is hidden from the other parts of the system in which all the related code is encapsulated.

The tests are placed in separate classes, which have access to the test framework's functions through ATF application programming interface (API). Subsystems System access and Rack control are used to create communication and control the HW.

Currently, A&C uses Git as version control management system. Thus, both the tests and ATF are managed by Git.

2.1.2 Control parameter of the ATF execution

At this moment, there are two configuration files that influence the behaviour of ATF, include an init file, this file is created automatically and defines basic parameters to configure ATF, which is relay-server address/port, rack configuration location…etc. The Rack Configuration is an XML file which describes the HW available on the test HW set. The Rack Configuration XML file helps to validate whether test case hardware requirements can be fulfilled by the set. The two mentioned files, init file and Rack Configuration file are processed during the initialization phase by atf-core and relay-server. /6/

Following is an example of how an XML configuration file is defined (**Figure 4**):

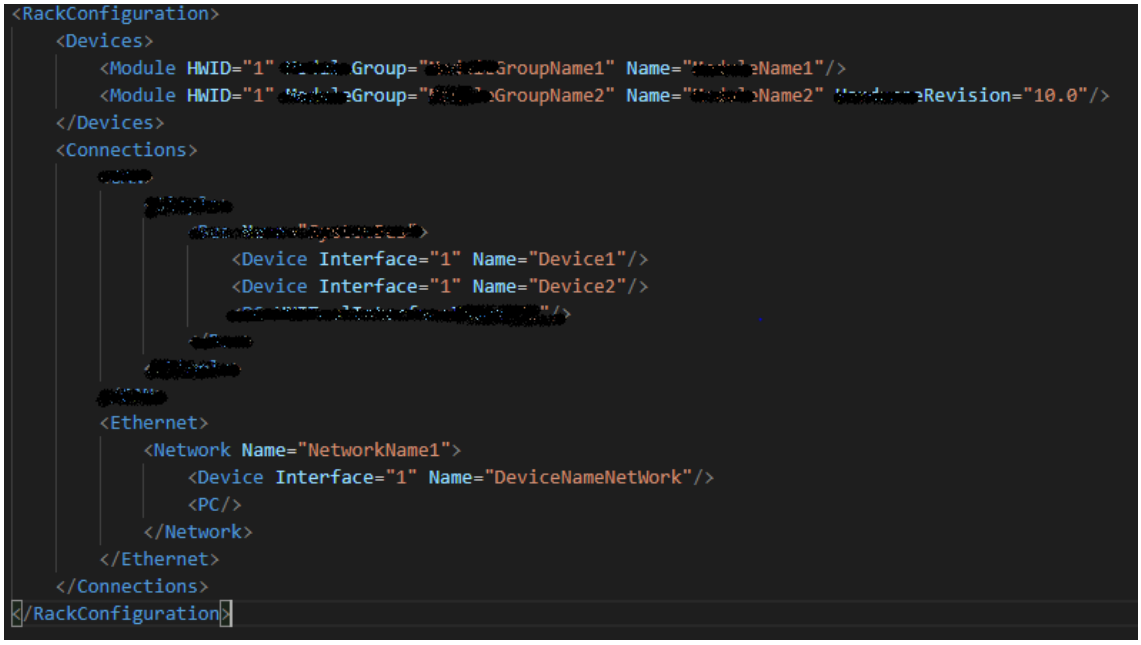

Figure 4. XML configuration file sample /5/

2.1.3 Execution of a test case

All tests are available and organized in TestNG test cases. A valid test Rack Configuration and running relay-server is required when the user (developer or CI server) wants to execute tests. The behaviour of the execution can be modified by TestNG extension class which implements TestNG listener interface.

TestNG executes tests one-by-one with a changeable order by using a parameter. A test report will be generated after all the tests in queue have been executed. TestNG extension are used to modify the default behavior of TestNG and report generation only. The test case instantiates entry ATF class, which contains all the necessary modules and communication with relay-server...etc. Thus, via ATF object, test case can modify system's state and fail the test execution if system did not react to the change in the expected way.

2.1.4 Review of test case execution report

As a result, a test report will be generated after all the tests have been executed. The status of each test is added to the list and produce to one of the available formats.

2.1.5 Add a new test case

In an ideal scenario, a new test should be added to one of the existing TestNG test cases, with all the basic parameters already defined.

2.2 The challenges

Each test HW set consists of multiple expensive HW modules, thus, have limited resources for testing purpose. A different HW setup is designed for a different test case. Each test case has its own HW requirements, this means any HW set can perform the test on condition that the set has enough modules to satisfy the test.

Additionally, the execution time of test cases are not identical, each test requires a certain amount of time to execute. HW sets should be utilized in the most effective use to reduce the cost of an HW purchase, in other words, the company wants to invest more in software solution than purchasing more HW.

2.3 TeamCity as CI system

TeamCity is the CI system are being used in the department. Briefly, TeamCity consists of build case (so called jobs) and build agent, which is a piece of software which listens for the command from TeamCity server /7/. Currently, TeamCity just schedules incoming test cases to run on an agent that is available, without knowledge of HW requirements or HW set capabilities.

A Java class handles the solution for TeamCity. It rejects the HW sets if there are not enough modules and accepts if the set has enough modules. However, it does not have an optimal HW sets selection method. It selects randomly a set that can perform the test, regardless of the number of modules on the set.

The developers manage it manually by assigning specific test suites to a specific TeamCity agent, which is a test HW set in this case (or a test rack in company's term). For users who are unfamiliar with the HW sets, or do not know which modules are available in each set, they need to ask other people to know which set is suitable, or worse is try running the test and see if ATF complains about the HW set being suitable or not.

Ideally, a test should be performed on a minimal required HW set that satisfies the test requirements.

Taking all obstacles into consideration, it is complicated to find the most optimal way of scheduling the test execution manually. Moreover, it added overhead work for users without any added benefits.

2.4 The proposed solution

The solution is to automate the ATF to automatically select the HW set with minimal number of modules which satisfies the requirements of the test case to perform it and spare other HW sets to serve other test cases which require greater HW setup.

A solution based on backtracking algorithm is created to run on a web service application to give the most optimal schedule possible. The essential target is to apply the algorithm in scheduling test cases to run on the HW sets. The algorithm evaluates the test case requirements taking into consideration the HW sets description and find out which set is suitable for the current test.

The following diagram shows an example use case (**Figure 5**):

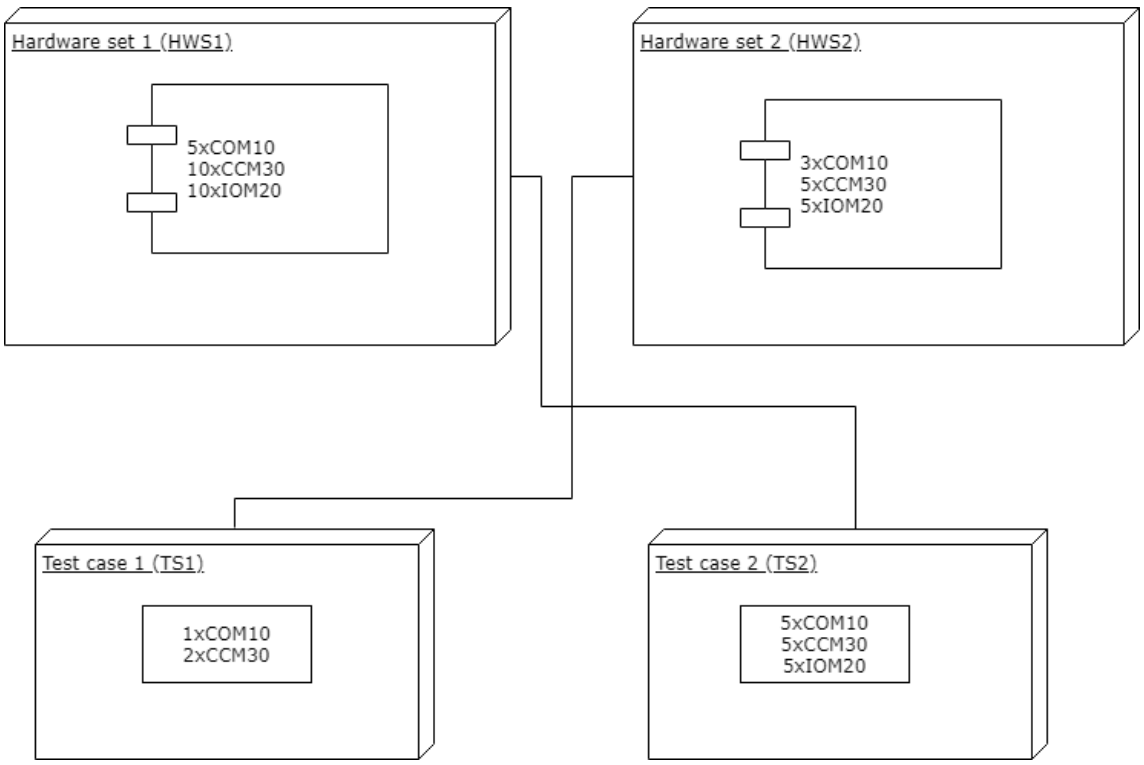

Figure 5. Example use case of matching test case and test HW set

As illustrated in **Figure 5**, HWS1 supports HSR (High-availability Seamless Redundancy) and able to run both test case TS1 and TS2. HWS2 does not support HSR and can run only test case TS1.

TS1 requires 1-unit COM10 and 2-unit of CCM30 module to be able to perform the test case, in the meantime, TS2 requires 5-unit COM10, 5-unit CCM30, 5-unit IOM20 modules and HSR communication to run. The algorithm will select HWS1 to perform TS2 and HWS2 to perform TS1.

2.5 Backtracking algorithm definitions

Backtracking is a general algorithm technique for solving problems, finding solutions recursively by trying to build candidates to the solution incrementally, one piece at a time and abandons those candidates that fail to satisfy the constraints of the problem at any

given time, as soon as the candidate is determined that it cannot conceivably be completed to a valid solution.

Following is a simple example of how backtracking algorithm can be applied to solve a get out of maze problem (**Figure 6**):

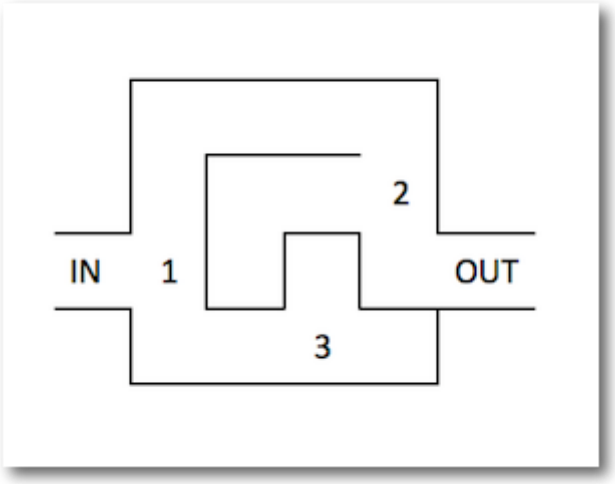

Figure 6. Example of a maze puzzle /8/

The junctions are labelled as 1, 2 and 3. The goal is to find the shortest way out of the maze. All the possibilities available is described in the tree path (**Figure 7**):

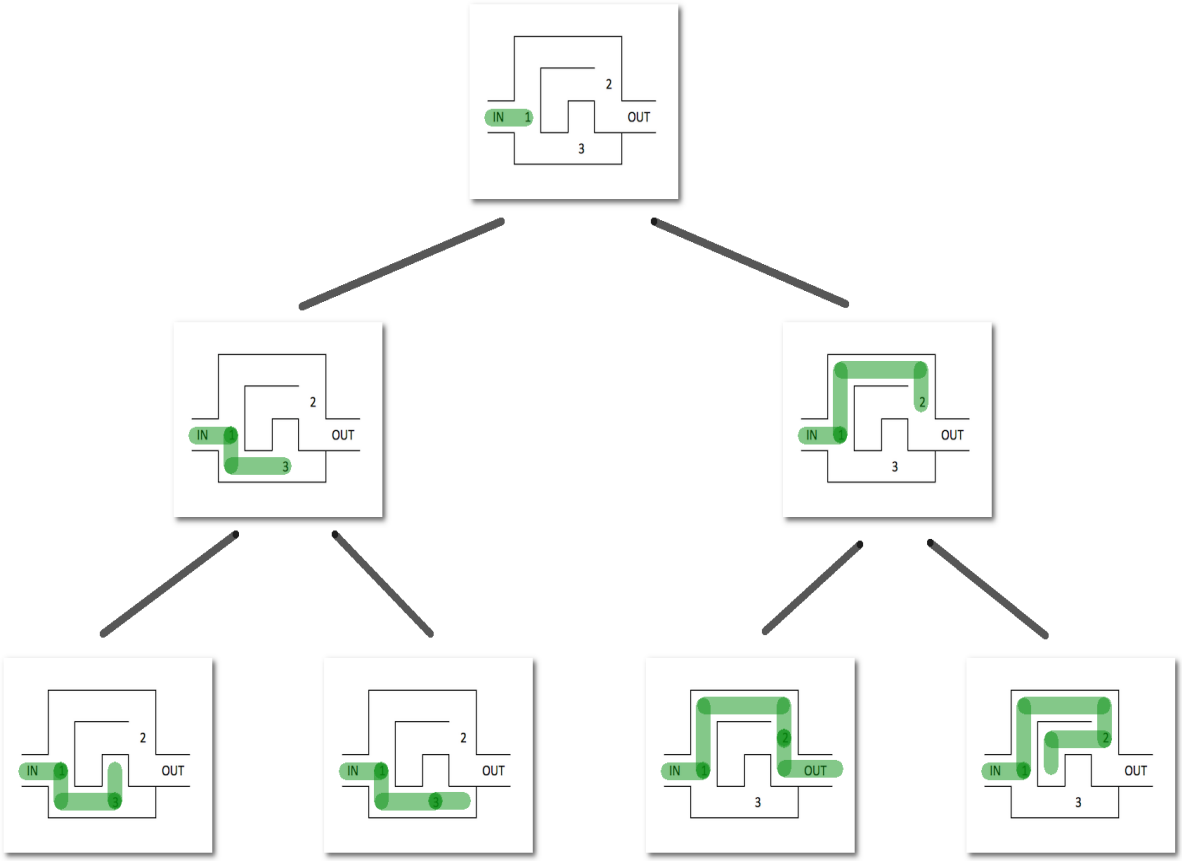

Figure 7. Every possible path in the maze /8/

2.5.1 Algorithm pseudo code

The algorithm can be described in pseudo code format for better understanding, as follows:

```
function backtrack(input):

  if is_exit:
    return true
  for each_possibility:
    if backtrack(input)
      return true

  return false
```

Figure 8. Pseudo code of backtracking algorithm

The solution to solve the problem of finding the shortest way out of the maze is similar to the solution of figuring out the minimal HW set to perform the test case, which is using recursive call function.

The pseudo code in **Figure 8** uses recursive function, a common technique in computer science. A recursive function is a function that revokes itself during its execution, either directly or indirectly. The function repeats itself several times as a routine, outputting the result at the end of each iteration. /9/

The solution is to use recursion to build the algorithm step by step; during the process, if the path is not a valid solution, the algorithm will stop computing and return to the step before. Moreover, the algorithm can realize that it is heading to a non-valid solution and stop the computing before it reached it.

A simple definition of back tracking algorithm, which the program starts and initialize with one or more input, which depends on the requirements. The program continues by checking if the needed criteria are met, it will proceed to end, otherwise it will continue

to run the trial until it finds the potential candidate for the output. The program ends when a candidate is found, and all the requirements are fulfilled.

3 SOLUTION PROTOTYPE DESIGN

3.1 Overview

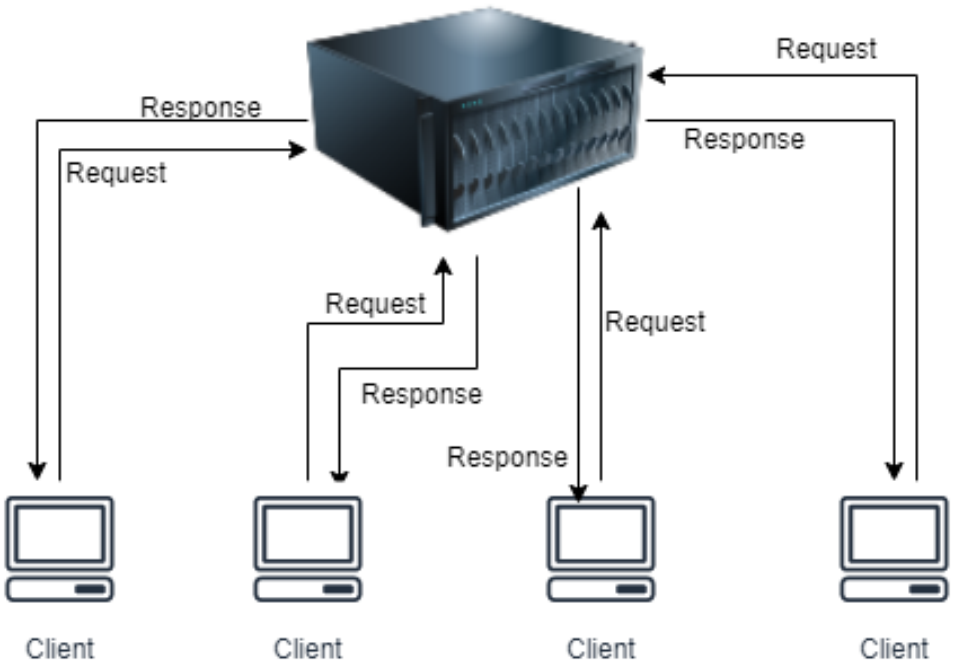

Figure 9. Web Service overview

Applying the algorithm, a web service application will be implemented to process the inputs and outputs. Two separate projects are implemented, one is the server with a specific route to receive the requests, which is the HW requirements of the test and compare it to the list of available HW sets and return the ID of the sets that has the minimal number of modules and satisfy the test as response, and the other is the client that sends the requests, which include all the required HW to the server.

3.2 The server design

Given that the server must accept at least two inputs, which are the available HW set's specification and the HW requirements of each incoming test. To simplify the implemen-

tation, three instances of the available HW sets with different number of HW modules/devices on each of it are included in the server to simulate three available HW sets for easily applying the algorithm and identifying the set.

In real use case, the number of available HW sets must be dynamic to either allow the clients to manage available HW sets or even further read available HW sets from TeamCity. The HW sets definition is in real XML configuration file format, which is using the same structure with XML configuration file being used by ATF.

The pseudo design can be described with the following classes, each class is created to serve a unique purpose:

- Fetch all HW sets available: A class is defined to fetch the content of the XML configuration files. There is information about the HW sets configuration, which is devices, connections and relay info. Further usage in the real use case requires a class that could fetch the available HW sets from TeamCity.

- HW requirements and comparison: A class is defined to retrieve the HW requirements from the incoming test. It takes data provided by HW sets available, HW requirements and move on to apply the backtracking algorithm to find out and return the most suitable HW sets for the test. In the beginning of the process, the test is set in "*pending*" status when it arrives, after all the requirements are fetched, it will be moved to "*processing*" status and afterward is set to "*processed*" when it has matched with the most promising HW sets.

The flow of the logic being used by the server-side code is applied according to the logic of the general backtracking algorithm, which is the algorithm being applied to this thesis work. First the program gathers the requirements from the incoming test and the information of the HW sets configuration. After all the inputs have been brought together, the program proceeds to compare to select the most suitable sets. If the ideal sets is found, the program ends here. If not, it continues the process from the beginning until the best candidate is found.

3.2.1 Acceptance criteria

- Fetch content of XML configuration files independently and show each XML file as an available HW sets.
- Fetch HW requirements of incoming tests.
- Use recursion to compare the available HW sets and HW requirements of the incoming test. Return the ID of the HW sets which has the least number of devices that can perform the test.

3.3 The client designs

The HW requirements of the tests are listed by annotations. Each test requires a different set of HW modules, connection devices and so on, which is used to compare with the hardware available on HW sets.

The client is a REST consumer, which sends the data gathered from the processing test as request to the server. All the requirements are collected before forming it to a list and attach to the request.

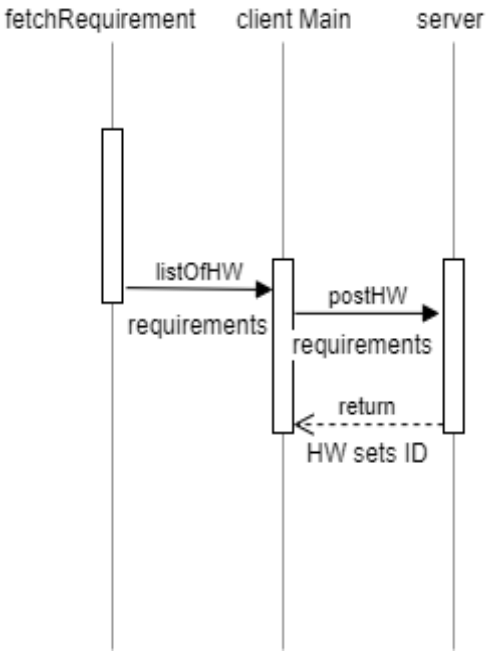

Figure 10. Client flowchart

The client likewise is a trigger that provokes the server to process the comparison. After sending the request to the server, the client expects a response. A response contains the ID of the chosen HW sets if there are ones that qualified. In different circumstances, there is not any HW sets that have enough devices to perform the test, the client will receive a message to say so to the users.

3.3.1 Acceptance criteria

- Fetch the requirements of current test.
- Send requirements to the server.
- Receive the response from server.

4 PROTOTYPE IMPLEMENTATION

This implementation of the prototype is a proof of concept, showing that the algorithm can be successfully used in the development process and help to reduce manual work for users.

RESTful web service is being used to share the responsibilities between the server and the client. The languages being used to implement this prototype are Java and Kotlin, with the support of Spark framework – a micro framework for creating web applications in Kotlin and Java 8 with minimal effort. Following is an example of how a POST route is formed with Spark: /10/

```
post("/", (request, response) -> {
    // Create something
});
```

Figure 11. An example of a POST route

Being shown in **Figure 11** is a POST route, which is one of the main building methods of a Spark application. A route is made up of three pieces: /10/

- A verb (get, post, put, delete, trace, connect). Only POST method is being used in this prototype implementation of the server. It will accept only POST data from the client.

- A path (/hello, /users/:name). This is the exposed path name that will be used by the client to send the requests to the server.

- A callback (request, response) -> {}. This is the callback function where all the requests from the client for that specific route is processed and afterward a corresponding response will be sent back to the client.

Spark helps to create the server, which is an easier way to receive requests and send responses to client. Jersey library is being used to take the requirements and send the request from the client to the server.

4.1 The server implementation

4.1.1 The detail

A Maven project is created for the server. Maven project helps to manage the dependencies needed. The language being used to implement the server is Kotlin programming language.

As mentioned in the prototype solution designs, three XML configuration files with different device setup are included in the server as resources to simulate the initial available HW sets. The contents of these XML files are processed by methods placed in a separate file. The server implementation is different to the client. It needs several methods implemented to process the inputs and output the demanded information.

The XML configurations file contains all the information of each of the HW sets. Each XML configuration file has different number of devices listed, which makes it difficult to read the file dynamically. However, to keep it simple for the demonstration purpose, the following devices are being read and used to make the comparison to the test's requirement:

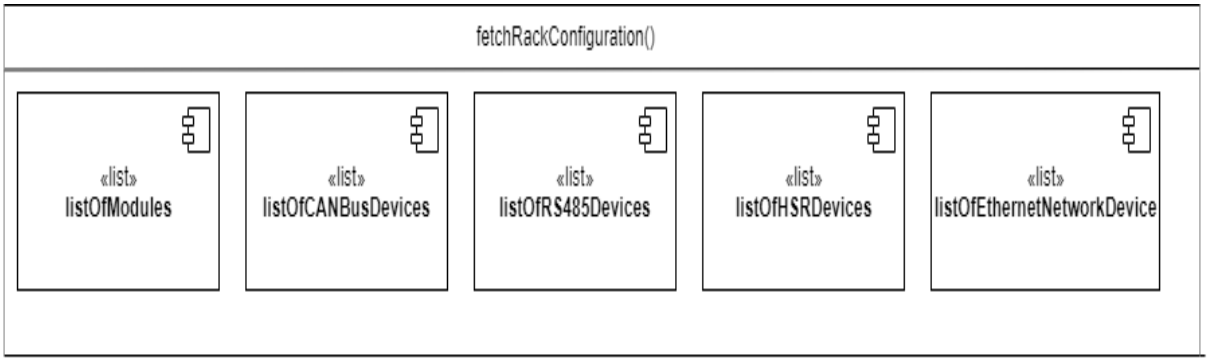

Figure 12. List of devices being fetched from available XML configuration files

Figure 12 shows the initial list of devices of the available HW sets. In this prototype implementation, it will always have three available HW sets from three XML configuration files. Later in the advanced implementation plan, the number of the available HW sets should be dynamic and based on the real HW sets fetched from TeamCity and the test requirement.

The initial list of devices will later be used to make the comparison to the test's requirements sending from the client of the web service.

A POST route with path **/incoming-test** is created to get the request from the client. The request from the client list the detail of the test's requirements as a string. However, the string contains several lists, which makes it difficult to read the data as Java's list. It needs to be converted to Java's list for later usage.

GSON is the Java library being used to convert JSON string array to Java lists. GSON works with arbitrary Java objects, which make it the ideal library to handle this situation. /12/

A diagram is created to demonstrate and for a better overview of the flow of the whole process, as being shown in **Figure 13**:

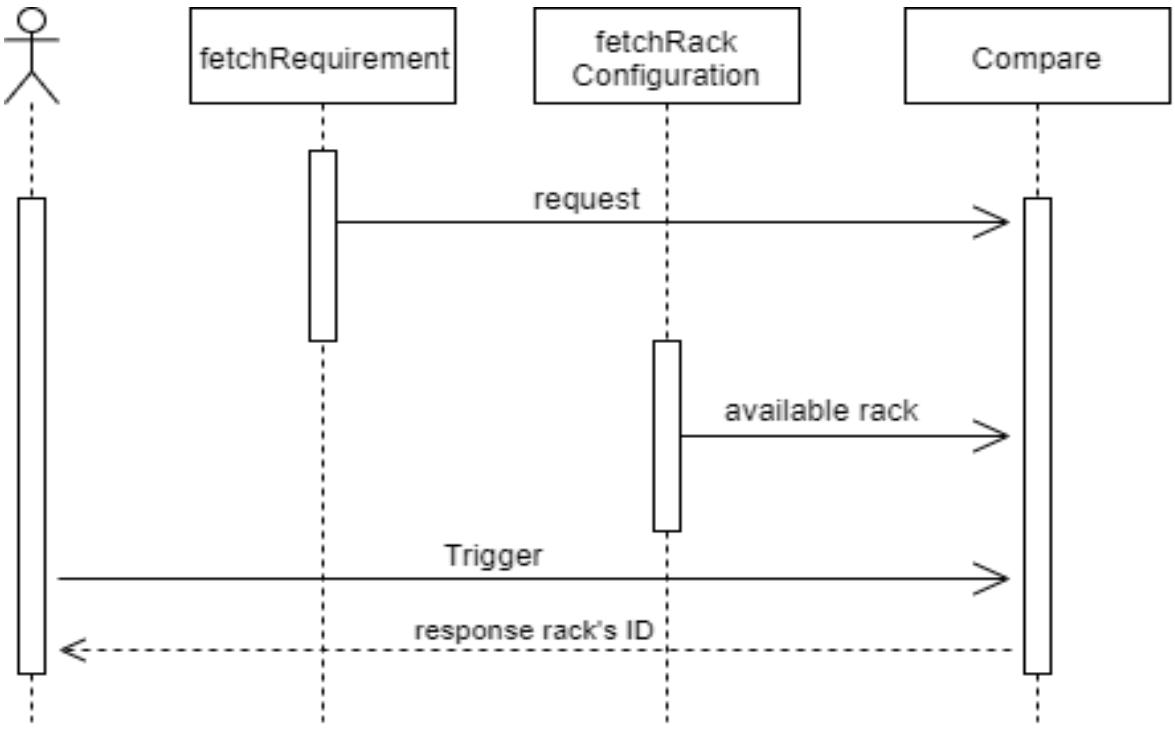

Figure 13. Flow of the comparison process

As stated in **Figure 13**, fetchRequirement and fetchRackConfiguration will gather information about the test's requirements and the HW sets configurations (available devices on a HW sets), respectively. The detailed information is passed as input to make the comparisons.

There are two comparisons needed to be made. The first one is to compare the test's requirement and the available HW sets configuration to pick out all the sets that has enough number of devices/modules to perform the incoming test. The second one is to compare the qualified HW sets to each other and mark out the one with the minimal number of devices/modules.

Recursive function is being used in the first comparison. The logic is expressed by mean of the following:

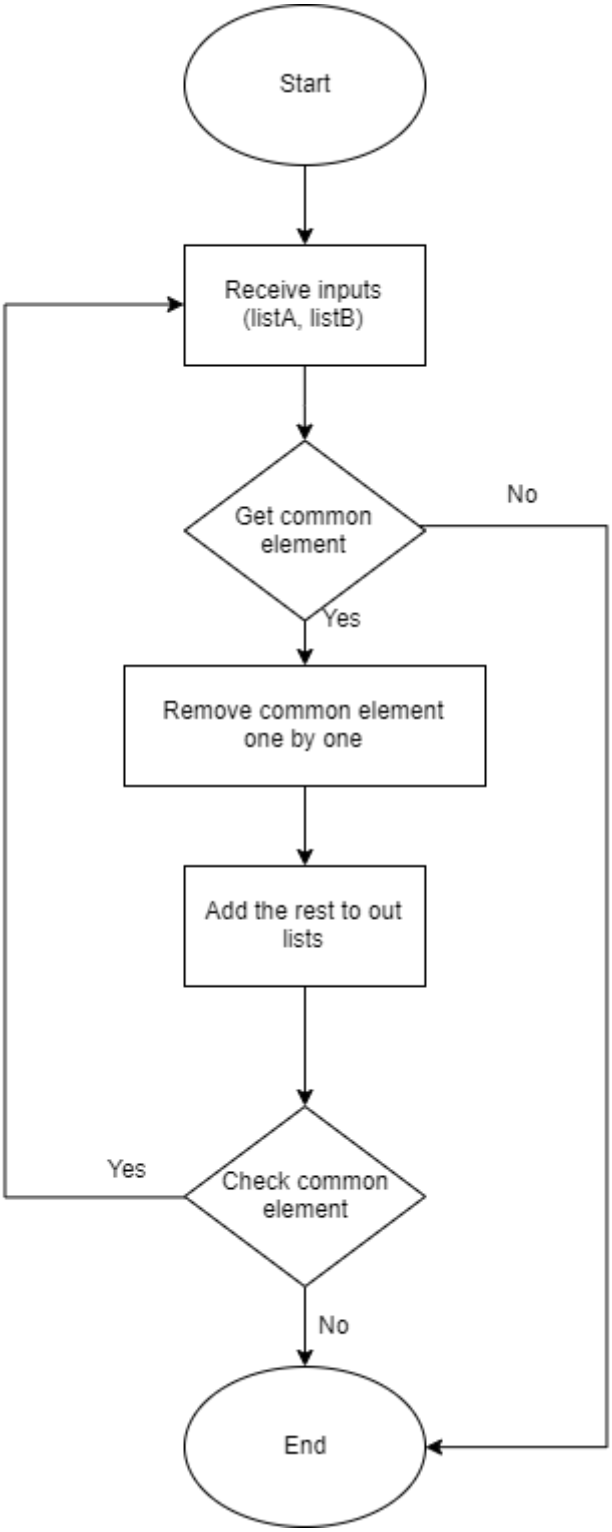

Figure 14. Logic of the recursive function to remove common element between lists

Figure 14 shows the logic behind the recursive function being used to remove to common element between the lists of the inputs.

The function takes two lists, which is the test's requirement list and the list of devices from available HW sets (both are categorized by type) as argument and process to compare the elements of each one against each other. If it does not find any common element, it means that there is no match for the devices/modules in between the input lists, the function will proceed to the end immediately. If there is a similar element in between of the input lists, the common element will be taken out of the input list, one by one. The rest of the elements of each list will be added to two separated lists, which is the next input to the recursive function to continue.

The function continues to check for the common element, if there is no common element left, the function ends at this point. Otherwise, it will go back to the beginning of the function and continue with the mentioned process.

After the recursion progress is finished, it will return the two separated lists, which is the leftover of the test's requirement and the list of devices, exclude the common element, one by one. The two separated lists will later help to choose the ideal HW sets. Another comparison is needed to make the final output, which is the information of the most qualified HW sets.

If the first list between the mentioned lists has the size more than one (the smallest size is one because there is one added element from the client's request to distinguish the type of devices/modules and easier to match with the same type from available HW sets), it means that there are one or more devices on the test's requirements that list of devices from available HW sets cannot cover, in other words, the HW sets is not qualified to perform the test, at this level. A string (for example, "Null") is added to the after-compare list in this case. In other case, when the first list has the size of one, it means that all the devices from the test's requirements of this type are covered by the devices from the list of available HW sets, of the same type. Put differently, the set of devices of this type from the HW sets having a proven capacity to accomplish the required devices of the same type from the test's requirements. The list of the remaining of the devices

from the HW sets is added to the after-compare list. Later, in the next step, all the resulted lists will be examined again to choose the most efficient HW sets.

As stated clearly in the prototype implementation design, to serve the purpose of demonstration only, there is three available XML configuration (HW sets) at any given times to make the comparison to the incoming test.

The recursive function above is applied multiple times on the five chosen type of devices/modules of the available HW sets and the test's requirements type, which led to the result of three distinct after-compare lists, each represents one of the processed available HW sets.

The second comparison function is created with the cited three different lists as parameter. In accordance with **Figure 15**, the logic of the second comparison is described step by step with pseudo code:

```
function compareResult(input0, input1, input2):
  listA:
  listB:

  if(input0 contains "Null"):
    //do something
  else:
    listA.add(sum_of_all_size_of_element_input0)
    listB.add("0")

  if(input1 contains "Null"):
    //do something
  else:
    listA.add(sum_of_all_size_of_element_input1)
    listB.add("1")

  if(input2 contains "Null"):
    //do something
  else:
    listA.add(sum_of_all_size_of_element_input2)
    listB.add("2")

return if(listA.isEmpty):
  "No qualified HW sets found"
  else:
    The required HW sets.
```

Figure 15. Pseudo code of the results comparison

After all the comparisons has been made, a response is sent back to the client. The response should contain a message to show the information of the chosen HW sets. This HW sets should be able to perform the test with the minimal setup of devices on it. If

there are HW sets with the same number of devices and able to perform the test (which is hardly the case), the server will pick one randomly and send to the client.

4.1.2 The server source code examples

```
listOfModules.add(listOf(i.toString()))
listOfModules.add(getInfo(readXml(paths[i]),  elementName: "Devices/Module",
listOfModules.add(getInfo(readXml(paths[i]),  elementName: "Devices/Module",
listOfModules.add(getInfo(readXml(paths[i]),  elementName: "Devices/Module",
```

Figure 16. Adding list of HW sets configuration

Figure 16 shows the process of adding selected information of HW sets to a list, which later will be used to compare with the test's requirements.

```
fun <T> removeCommonValues(listA: List<T>, listB: List<T>): Pair<List<T>, List<T>> {
    tailrec fun removeCommonValues(a: MutableList<T>, b: MutableList<T>) {
        val common : Set<T>  = a.intersect(b)
        if (common.isEmpty())
            return
        for (item : T  in common) {
            a.remove(item)
            b.remove(item)
        }
        removeCommonValues(a, b)
    }

    val out : Pair<MutableList<T>, MutableList<T>>  = listA.toMutableList() to listB.toMutableList()
    removeCommonValues(out.first, out.second)

    return out
}
```

Figure 17. Recursive function used in the solution

Being shown in **Figure 17** is the implementation of a recursive function, used to take out the common element between two lists and returns list of the remaining items of each initial list.

4.2 The client implementation

To be able to examine the test's requirements, the client is created as a sub module of the test's module of the ATF framework. Because most of the modules are implemented in Java, which make it easier to use Java as the programming language being used to implement the client module.

The client is a RESTful consumer, which have the duty of gathering and sending the requirements data to the server. All the tests are defined as a Java class which describe all the requirements and devices as well as the connections needed for the test.

The requirements of a test class are defined using custom Java reflection annotation **@Require*** and **@Required***. Each **@Require** annotation is representing a required device and each **@Required** annotation is serving as list of devices. **@Required** is an interface which implements corresponding **@Require** interface.

To keep it simple, the client has only one class with two methods, one is to fetch the HW requirements from the test and the other is to send the request to server. The client has only one purpose and it is carried out independently from the server, which follows the practice of the RESTful web service protocol.

4.2.1 Methods

Method fetchRequirements take a class as argument and users must pass their test class when apply in use. To serve the demonstration purpose, this method does not read all the **@Require** and **@Required** annotations. It only read some obvious annotation that most of the test contains. These annotations are being described as following:

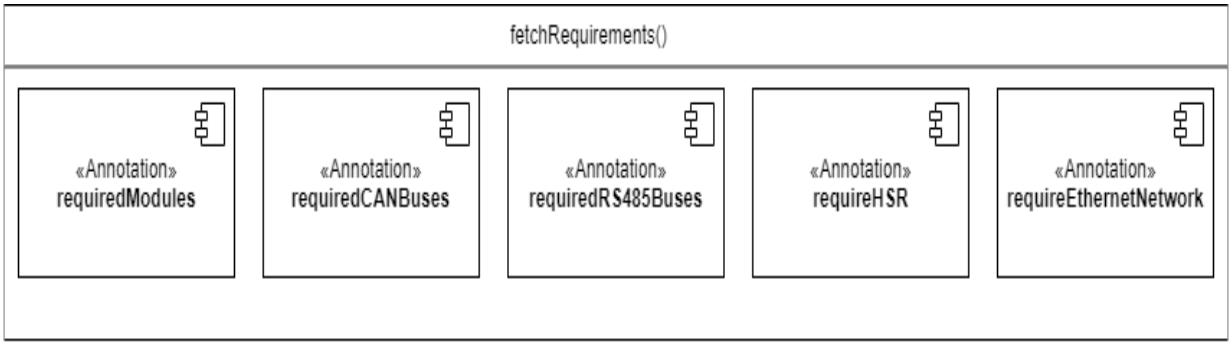

Figure 18. Fetch requirements method

The method fetches all needed requirements listed by **@Require** and **@Required** annotations in a test class and combine into one list. This list is later used in the main method and send to the server using Jersey web service client method.

The Jersey web service client is being used to send the request as a string to the server.

4.2.2 The client source code examples

```
for (Annotation annotation : annotationRequiredRS485Buses) {
    if (annotation instanceof RequireRS485Bus) {
        RequireRS485Bus rrs = (RequireRS485Bus) annotation;
        listOfRS485Devices.addAll(Arrays.asList(rrs.devices()));
    }
}
```

Figure 19. Extract requirements of RS485 Bus

Java reflection was used to address the requirements of the test, as being shown in **Figure 19**, a for loop was used to extract the required information of RS485 Bus. After being extracted, the information of the required bus will be added to a list to send to the server.

4.2.3 Testing

After the implementation phase has finished, two test classes (Test A and Test B) were used to check if the implementation has met the requirements. Test A required multiple devices, which none of the available HW sets can perform it. On the other hand, Test B is expected to be executed by at least one HW sets.

As a result, when the program runs on Test A, none of the HW sets input can perform it, and server sends a message to tell the client that no sets is qualified. As expected, when Test B is being examined, the server sends a message to tell the client the ID of the HW sets that fit with the description of test requirements.

5 ADVANCED IMPLEMENTATION PLAN

Other than all the functionalities implemented in the prototype, an advanced version of the application is planned to implement with better inputs and outputs.

5.1 The server functionalities

Currently, the server has some functions that can be improved to increase the automation process. The following table shows the comparison between some of the current server and the advanced, better server:

Basic server	Advanced server
Static three available XML configuration files at any given time	Dynamic number of XML configuration file
Read only five types of devices/modules for the comparisons	Read all the required devices/modules from both test's requirements and HW sets configuration
Return only the ID of the chosen HW sets after the comparisons	Return more information of the chosen HW sets
	Reserved HW sets should be excluded from the next test examination
No wait time history of HW sets	Consider wait time history of each HW sets

Figure 20. Comparison of basic and advanced server version

At this moment, the server has three available XML configuration files and this number is static, which means that even though a HW sets is reserved for the previous test, it is

not excluded from the next test examination. This is a waste of server resource as the reserved HW sets should not be considered.

The basic server read only five certain defined types of devices/modules from the available HW sets, which is not an ideal way of gathering data and decrease the accuracy of the comparison. In the advanced version, it should be able to read all types listed the available HW sets.

The advanced server should return more information of the chosen HW sets to the client after it has processed the comparisons than only the ID of it, which is the ability of the basic server.

Each HW sets spend a certain amount of time to process the test. This wait time should be considered in the advanced server when it makes the choice for the next incoming test. It means that if a HW sets has the ideal setup with minimal number of devices, but it is reserved at the moment when the next test is examined and the wait time of it is not too long, it should be choosing to process the test instead of other HW sets.

5.2 The client functionalities

In the basic implementation, the client gathers only five types of required devices/modules. In the advanced version, it should be able to read all the types of required devices.

6 DISCUSSION AND CONSLUSION

The purpose of this thesis work was to study the functionalities of ATF framework and identify a better way to schedule the test to run on the HW sets to later determine if it could bring significance value to the software development process.

Furthermore, it is important at the end of the thesis work period for a personal evaluation of what have been done and what not to be repeated, on top of everything is what can be enhanced.

Generally, the pre-study process has been completed comparatively well with considerable amount of the knowledges and feasibility study about the early stage of the thesis work.

In the scope of the thesis, basic factors have been taken into consideration and after thoroughly calculated, a decision was made based on the benefits that each of different proposed approaches possibly bring. Therefore, a simple prototype of implementation was created to support the initial development idea of the project.

Nonetheless, there are many flaws during the development phase of this thesis work and most of it could have been handled in a better way. Most of the defects come from human nature perspective. Despite of the fact that the pre-study phase is the most important phase, it could be cut back and shortened, which means more time would have been given to the implementation phase.

In the beginning, the inadequacy of experience and knowledge led to difficulty and later become burdensome in the initial planning of preparing the tasks.

Additionally, time management and personal task prioritization have not been completed properly. Many development tasks could have been put on a higher priority than other work of the thesis writing process. Consequently, a considerably valuable lessons learnt from this thesis work and will be applied in the future projects to avoid making the same mistakes.

In conclusion, the goals of this thesis have been fulfilled, the work has built an essential key structure and open an opportunity for further application of the scheduler to the testing phase of the software development process and increase the automation operation.

REFERENCES

/1/ This is Wärtsilä. (Accessed 08/09/2019)

https://www.wartsila.com/about

/2/ TestNG. (Accessed 11/09/2019)

https://testng.org/doc/

/3/ Java SE in a glance. (Accessed 11/09/2019)

https://www.oracle.com/java/technologies/java-se-glance.html

/4/ Why Git? (Accessed 11/09/2019)

https://www.atlassian.com/git/tutorials/why-git

/5/ Configuring ATF. (Accessed 12/09/2019)

https://confluence.devops.wartsila.com/display/WKB/Configuring+ATF

/6/ ATF overview. (Accessed 12/09/2019)

https://confluence.devops.wartsila.com/display/WKB/ATF+framework+overview

/7/ Build Agent. (Accessed 18/09/2019)

https://www.jetbrains.com/help/teamcity/build-agent.html

/8/ Backtracking explained. (Accessed 19/09/2019)

https://medium.com/@andreaiacono/backtracking-explained-7450d6ef9e1a

/9/ Recursive functions. (Accessed 21/09/2019)

https://www.geeksforgeeks.org/recursive-functions/

/10/ Spark Java. (Accessed 30/09/2019)

http://sparkjava.com/

/11/ Reflection in Java. (Accessed 10/10/2019)

https://www.geeksforgeeks.org/reflection-in-java/

/12/ GSON in Java. (Accessed 20/10/2019)

https://github.com/google/gson