UNIVERSITY OF GOTHENBURG

Automated System Testing and Evaluation of Fieldbus Interfaces

Bachelor of Science Thesis in Software Engineering

LEE TORRES
ROBIN MURÉN

Automated System Testing and Evaluation of a Fieldbus Interface

LEE TORRES
ROBIN MURÉN

Examiner: MATTHIAS TICHY

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2014

Abstract: In order to create better solutions for automated system testing of fieldbus interfaces, we created simulated system environments that can interact with multiple hardware devices. A suite of automated system tests were then run against these simulated environments in order verify against the formal specifications of the fieldbus interface. After five iterations of testing and verification of specifications, results indicate that automated system testing that rely on simulated environments can be used to test functionality and stability of fieldbus interfaces.

Keywords: fieldbus interface, fieldbus interface testing, automated system testing, simulated network device

1. Introduction

Fieldbus is a computer network protocol family that is used for sending data between different devices. The fieldbus family became popular in the 1990s, when production plants began to see an increasing need to establish communication paths between production devices, control centers, maintenance components, management stock and other plant components [1]. While the fieldbus provides the communication protocol between devices, it is the fieldbus interface that is responsible for the regulating of messages sent through the fieldbus, making sure that messages are sent to the correct hardware recipients and that the data being sent can actually be handled by the intended recipients.

In a system that makes use of a fieldbus, the fieldbus is used to facilitate communication between master and slave devices. Master devices are those that monitor the activity of slave devices. Master devices can also request that the slave device should initiative a new set of behaviors. Slave devices are those that perform a specific function and relay information back to the master when requested by the master. In the example of a production plant, the master devices can be scanners or those that relay information back to the control centers, while the slave devices can be production devices.

In a system that makes use of a fieldbus, a fieldbus interface can be created to access and control the functionality of the fieldbus. This is done by interfacing already existing code which facilitates communication between hardware devices. This fieldbus interface can be lines of code or an entire application.

1.1. Problem Description

Since fieldbuses are responsible for the intercommunication between devices, it becomes vitally important that they are 'interoperable' [1]. This means that the fieldbuses are vendor independent, or are compatible with many devices with various hardware profiles. As Patoni et al mentioned, a customer should be able to use the fieldbus to connect their devices, with little configuration needed from the side of the customer [2].

However, the challenge for the fieldbus interfaces, or the control logic for the fieldbuses, is that it must actually be used to interact with a wide range of devices. A fieldbus is able to work with many devices because it supports many connection types between devices, and it is able to send and receive data payloads of different sizes. The type of connections and data payload sizes that can be utilized between devices depend directly on the devices themselves.

The challenge of creating a fieldbus interface, or the control logic of the fieldbus, is that it must be able to use the appropriate connections and data payload sizes when communicating with the devices connected to its system. Through the fieldbus interface, different devices of specific hardware profiles, can carry out simple actions such as opening a connection path, and sending and receiving data on these connection paths. Each of these

actions, in relation to specific devices, becomes a specification that the fieldbus interface must meet. As the range of devices that the fieldbus interface can work with increases, so does the amount of specifications it must verify to be able to work with. A fieldbus interface must also maintain its functionality under complex network conditions, and working under these conditions are also specifications that need to be met.

Being able to test that a fieldbus interface is compatible with many devices in various system conditions is time consuming. One often has to have multiple devices available to test on. One has to also create many variations of testing systems that are comprised of numerous combinations of hardware devices. The testing of these complex conditions is also often happening within the context of time-boxed work cycles.

To efficiently resolve this problem, one solution would involve simulating multiple hardware devices, which frees one from actually having to have the devices be available and integrated into a computer network. By having multiple simulated devices, one can take any combination of simulated devices and arrange them in a way to simulate complex systems that would be otherwise be time consuming to put together through real hardware. The advantage of this approach is that one can combine simulated hardware devices with real hardware devices. This allows one to test the capability of the fieldbus interface to interact with a wide range of devices, and its ability to interact in a stable manner under complex system conditions.

This study is done to help advance the state of art of verifying fieldbus interface specifications. This concerns making the verification of fieldbus interfaces more reliable by exposing the interface to a systematic testing approach.

1.2. Purpose

The purpose of this study is to advance the state of art of verifying fieldbus interface specifications. This concerns the making verification of fieldbus interfaces more reliable by exposing the interface to a systematic testing approach. This systematic approach will be aided by the simulation of hardware components in the system.

1.3. Aims and Objectives

The main objective and goal of this to create automated system tests that evaluate the functionality and stability of a fieldbus interfaces. Functionality relates to the ability of the fieldbus interface to pass messages between numerous devices in the system. Reliability relates to the ability of the system to perform this basic functionality, even under complex network conditions that involve simulated devices and concurrency.

In order to be able to achieve our main objective we will have to:

- Produce a fieldbus interface for a simulated master device that we can run automated system tests on.
- Evaluate the effectiveness of a suite of automated system tests in its ability to verify specifications of the fieldbus interface

- Make automated system tests to check if there are specifications that have not yet been implemented or are implemented incorrectly.

1.4. Scope and Limitations

This study will be limited to the testing and verification of specifications for a fieldbus interface. These specifications are those that are necessary to perform the expected functionality of the fieldbus interface, as defined by the company for whom the fieldbus interface is created for. Due to a non-disclosure agreement, we can not explain in depth specifics about the fieldbus interface or the expected functionality.

During 12 weeks, we developed a fieldbus interface for a company. The study then is concerned with how 5 weeks of automated system tests were able to aid in reducing errors in the fieldbus interface. To meet acceptable standards of low errors, the fieldbus interface had to pass the company's acceptance tests.

Testing and verification is not done on all the formal specifications associated with the particular fieldbus that the fieldbus interface is built upon. As the formal specifications of the communication protocol are extensive, we are able to verify less than 5% of these formal specifications.

Automated system tests will be limited to systems containing of, at most, 2 slave devices of different hardware profiles and which contain at one time, no more than 6 simulated master devices.

1.5. Research Questions

From the aforementioned research goals we derived the following research questions.

Q1: *How can automated system tests be used to verify that a fieldbus interface is correctly sending and receiving specified data?*

Q2: How can automated system testing support stability evaluation of a fieldbus interface?

2. Background

2.1. Company Info

The company we will be creating a fieldbus interface for is located in Sweden, and specializes in the field of industrial networking devices. These devices are intermediary components, which help facilitate the communication between many types of devices.

This company is continually producing new intermediary network devices, as well as acquiring ones from other companies. To test these intermediary devices, this requires that the company has many teams of test developers who are producing the fieldbus interfaces, and the testers that are verifying specifications. Their products are then released to a global list of clients.

2.2. Fieldbus Interface

Fieldbuses are communication protocols that are used for communication between different hardware devices. According to Stohl et al [3], the demand for fieldbuses increased when production plants in the 1990s needed to establish communication paths between production devices, control centers, maintenance components, management, stock, and other plant components.

What arose was a move towards standardized protocols. The aim of these standardized protocols was to set up mature protocols that could be used by a wide range of devices.

A common practice in a fieldbus system is to have an intermediary device that facilitates fieldbus communication between the numerous master and slave devices in the network. However, to test whether an intermediary device is actually able to facilitate correct communication between devices, a fieldbus interface is required. In the context of our study, this is how the fieldbus interface will be used.

The fieldbus interface is used to generate network traffic between master and slave devices via the intermediary device. To test how the intermediary device is able to respond to heightened network activity, master devices can be simulated.

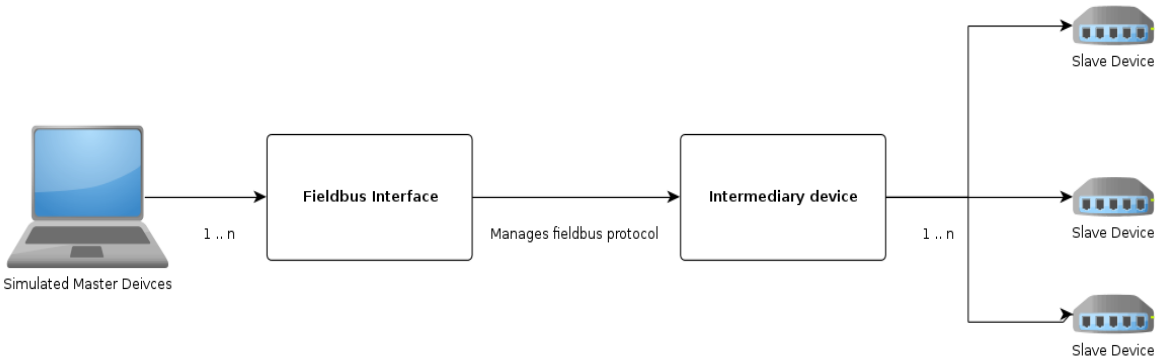

Illustration 1: Fieldbus Interface Integrated in a Test System

Illustration 1 shows how the fieldbus interface would be used within the work flow of testing specifications. There can be any number of simulated master devices, which are generated from a computer. These simulated masters interact with the fieldbus interface, which is responsible for managing the sending and receiving of messages through an

intermediary device. Information sent by the fieldbus interface is packaged according to the specifications of the fieldbus. This intermediary device is then responsible for sending packets of data to the slave devices. Once the slave devices receive messages, the slave device replies with response messages, which are sent to the intermediary device and then to the fieldbus interface. These response messages are then routed back to the simulated master device that originated the original message that generated a response.

However, to see if the intermediary device is functionality correctly, one has to be certain that fieldbus interface that controls it is functioning correctly. In this situation, the aim is to use the automated system tests as an aid in producing a correctly functioning fieldbus interface. With this confirmed, it becomes easier to test the intermediary device with numerous types of master and slave devices.

2.3. Related Work

Research was done on related work, in order to determine the current State-of-the-Art of testing of fieldbuses and fieldbus interfaces. This information could be used to help influence our study in a positive way by indicating which areas to focus on. Studying related work can also be used to determine what is missing in the current State of Art, and give us an indication as to which area to focus on so that we can make a meaningful contribution to this domain.

In order to find related work, we focused on searching through the databases of IEEE Xplore, as well as the ACM Digital Library. We have three major search terms: “fieldbus interface test,” “fieldbus verification,” and “fieldbus simulator.” These encompass the search words relevant to our two main research questions. We also used variations of these keywords. Instead of only using “fieldbus,” we substituted “fieldbus” with established fieldbus protocol names in the search string. This lead to additionally using these search terms: “devicenet,” “common industrial protocol simulation,” and “profibus test.”

It was discovered that dynamic verification of a fieldbus interface is dependent on having integrated hardware and software components. An important precondition is having the hardware components connected through networking cables and intermediary devices. When the fieldbus interface is implemented to interact with these intermediary devices, then a full system test can be run in order to verify that the fieldbus interface meets the specified requirements. A study done by Hong et al [5] suggested that the traditional system test of having all the hardware and software components integrated is the ideal way of doing a system test. By automating the sending and receiving of data between the devices on the network, one can have extensive automated system tests that aid in verifying system specifications. In their study, the primary concern of their fully integrated system was to systematically test the functionality and limitations of specific hardware components in the system. In contrast to this, our work will be focusing more heavily on systematically testing the software aspects of the fully integrated system.

When a system test is based off a fully integrated hardware and software solution, the problem arises of having system tests that are dependent on specific hardware being

available. If a network is supposed to handle the communication between hundreds of devices with many different hardware profiles, it can be difficult to gather all those varied devices in order to perform a system test. An alternative proposed by Maturana et al [6] is to make use of simulations. The devices that are being communicated to in the network can be a series of micro-controllers. The study presented by Maturana et al involved taking simulated hardware profiles, and then installing them on the micro-controllers. This way, they could test different hardware profiles without having to actually possess them. On this basis, they could run system tests that actually send data across a bus, but the end devices that received the data could be simulated.

Work done by Mossin et al [7] also involved the use of simulations when testing a system relying on a fieldbus interface. In this particular study, the network itself and the hardware devices on it are purely simulated through software, which removes the dependency of needing the intended hardware devices. Each network device that is being simulated is represented by independent applications that are able to communicate to each other using a simulated fieldbus. The fieldbus has its control algorithms managed by the fieldbus interface. With this large, simulated network, they are able to verify that the fieldbus interface is able to manage important functions like working with event-driven communications, and timed communications that execute in a cyclical manner.

The work we are conducting will also involve the use of simulations to mock the behavior of hardware components in a fully integrated fieldbus system. Maturana [6] takes the approach of taking microelectronics, and installing one simulation of a device on each micro-controller. Through this method, he simulated multiple slave devices. Each micro-controller could mimic the behavior of a chosen slave device through software, but still was physically connected to the system. We will not take the approach of using multiple micro-controllers, but will instead limit ourselves to running our simulations on a company issued laptop computer. The advantage of our approach is that the number of simulations we run will not be constrained by having the same amount of micro-controllers available. Our constraint would be the laptops available memory. Since each simulated master is a programmatically created object, creating an instance doesn't consume many system resources. Additionally, we will be working mainly with simulations of master devices, whereas most other studies focus on the simulation of slave devices.

Similar to Mossin et al [7], we will use purely a software solution to simulate different devices on the network. However, their system was more concerned with getting an overview of how data flows through a fieldbus system and is used as a way to evaluate the architecture of systems. Our study is concerned with verification of specifications.

Work has also been put forth to indicate the importance of using automated system tests to verify the specifications of a system that networks multiple devices. In a study by T. Syed et al [8], they made use of automated system testing to test a fully integrated system made up of computers that are connected by Asymmetric Digital Subscriber Line (ASDL) equipment. A fieldbus interface system is similar in that several devices are networked together, and numerous asynchronous requests and responses are being sent simultaneously. To test the numerous variations in this system, automated system testing is also required to produced numerous, complex situations that are reproducible.

Another study by M. Kantona [9], shows that automated system tests can be used to quickly verify the specifications of embedded systems, therefore increasing production. In this study, the target are television systems in an assembly line environment. Their study shows that in the television production industry, the manual process of verification creates a bottleneck within their production line. Their study shows that by using automated system testing, produced televisions can be quickly verified and certified for deployment. Taking this as a point of interest, we will be using automated system testing to quickly verify that the fieldbus interface is working properly with the company's devices and any other devices that the fieldbus may interact with.

There is also research on using unified modeling language (UML) to model the slaves that can appear on the network, and on that basis, tools can generate test cases [10]. These test cases mainly focus on functionality that is related to slave devices, such as setting and getting a slave devices' attributes. These basic tests, however, unlike the automated systems tests that we created, do not verify more detailed specifications such as the order in which network commands need to be sent. These tests also do not verify system stability during complex network activity. This includes the presence of multiple devices on the network and the reactions to concurrency.

3. Research Method

3.1. Design Science Overview

The research is guided by the principles of design science. As A. Hevner et al mentioned, "Design science, as the other side of the IS research cycle, creates and evaluates IT artifacts intended to solve identified organizational problems [11]." In the context of our project, the problem is the slow, manual process of verifying specifications of a fieldbus interface. There is also the issue of being able to verify that fieldbus interface meets the specifications even under complex network conditions.

Our solution, or our main IT artifact, is the suite of automated system tests that speed up the verification process. These tests will focus on testing three main aspects of the system, mainly the aspects relating to "Communication", "Payload", and "Stability". These will be further discussed in the Data Collection section. In order to achieve these tests, we will simulate numerous possible system environments that could be used by the fieldbus. This includes the simulation of multiple master devices, and its interaction with actual hardware devices. On the basis of numerous combinations of potential system environments, we perform the tests that examine the system on the basis of the aspects of "Communication", "Payload", and "Stability". These tests are being done on the basis of multiple concurrent operations, so the system must also meet the specifications while running concurrent activities with a wide range of devices. The suite of automated system tests can verify that the system is able to execute concurrent activity, without breaking due to deadlocks and coding error.

The aim of the tests are to evaluate the aspects of "Communication", "Payload", and "Stability" with a system running, at minimum, 3 simulated master devices and working with the slave devices given by the company. A fieldbus interface that is tested against these system conditions and is able to perform at the specifications defined by the company would then be considered one that meets their requirements. One can then analyze if meeting these requirements were aided by the use of automated system tests. Prior to running automated system tests, the fieldbus interface is subjected to static code review and manual testing. If there are errors not detected by doing using these methods, but were later discovered by the automated system tests, then finding these errors could be attributed to the use of automated system tests.

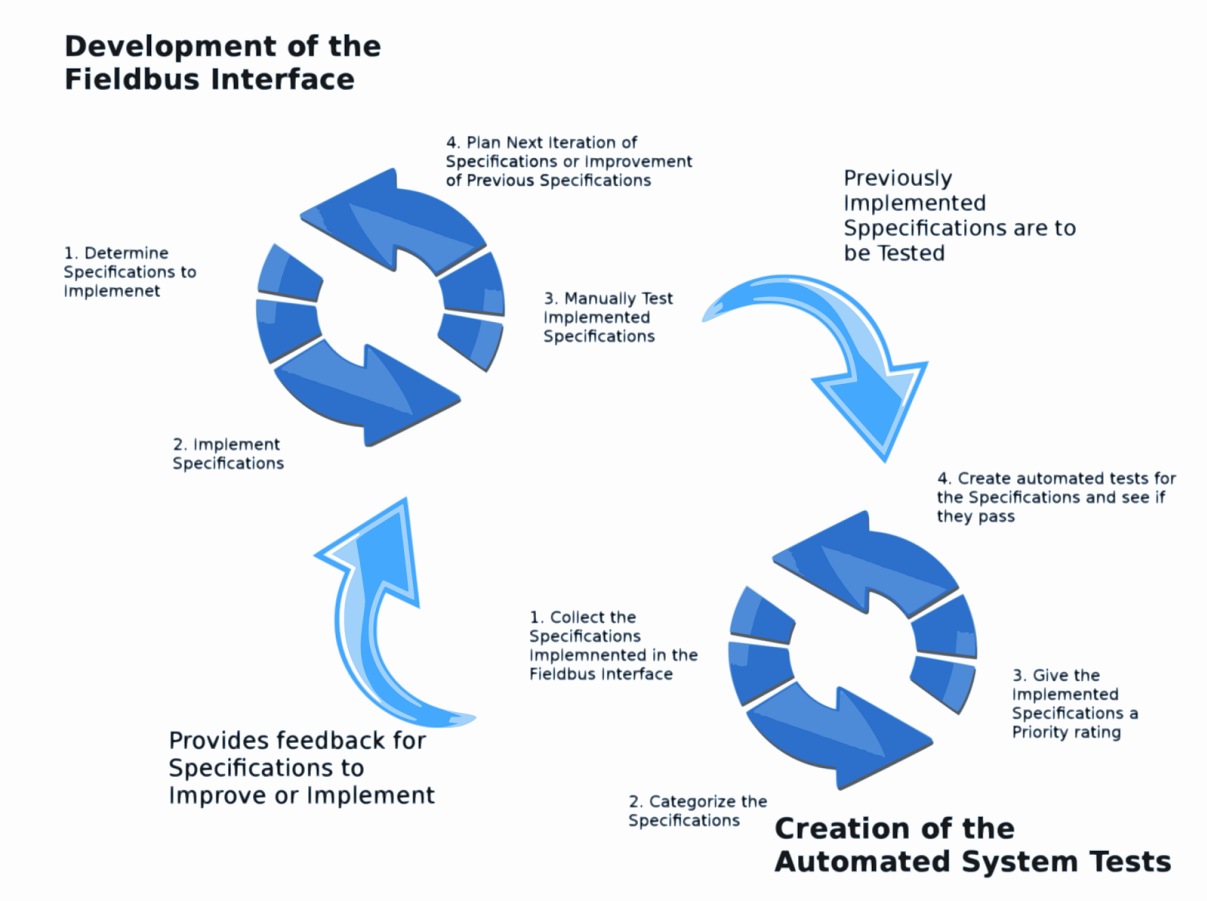

Illustration 2: Automated System Tests Influencing Development of the Fieldbus Interface

Design Science takes on an iterative approach, in which the artifact we create (the suite of automated system tests) will be iteratively improved. **Illustration 2** shows this process. As the fieldbus interface is iteratively developed, the results of each iteration become the basis of creating a new iteration of automated system tests. The results of these automated systems tests then influence how the next iteration of the fieldbus interface is carried out. This process of mutual influence continued until the end of development.

3.2. Data Collection

In order to understand which specifications we had to verify, we dedicated time to compiling a list of specifications that were important to the fieldbus interface. During the 3 weeks of research prior to development and 9 weeks during development, we came to understand the domain that we worked in. The company had provided us with a list of high-level requirements that were to be implemented, and during these 12 weeks, we broke down these high-level requirements into distinct specifications.

Our data collection process can be summarized into these 3 points:

1. Collecting of the technical fieldbus interface specifications that we need to implement for the company.
2. Verifying that the fieldbus interface meets these specifications mentioned in the above point, through the use of automated system tests.
3. Verify that the system is able to execute specifications without running into deadlocks and system crashes due to use of concurrency.

Specifications are put into three categories, mainly “Communication,” “Data Payloads,” and “Stability.” These categories were created by us for the purpose of this study, and represent the important aspects of our fieldbus interface. Since these are the aspects that the fieldbus interface needs to excel in, these are the areas that our automated system tests were focused on.

Specifications labeled as “Communication” relate to the ability of our interface to send and receive data with devices that have a wide range of hardware profiles. Each device in the fieldbus interface’s network has a specific level of functionality with regard to its ability to send and receive messages. How robust its ability to communicate depends on its hardware profile because devices with less powerful processors cannot handle certain types of messages. The automated system test are used to monitor whether devices in the fieldbus interface’s network are able to send and receive messages that it is actually capable of handling.

Specifications labeled as “Data Payloads” relate to whether devices on the network are sending the correct encoded data, and decoding the correct received data, according to the appropriate situations.

Specifications labeled as “Stability” refer to whether our fieldbus interface is able to maintain an acceptable level of stability as the network complexity scales up. We will be able to increase the network complexity by simulating multiple hardware devices. In addition, the specifications must execute correctly without failing due to deadlocks. This becomes a concern because the system is executing many actions concurrently.

In addition, specifications will also be given the ratings of “High,” “Medium,” and “Low.” “High” specifications include the opening and closing of connections, managing of message passing between devices, and encoding and decoding of payloads. “Medium” ratings involve the getting and setting of the attributes of slave devices. Specifications

labeled as “Low” are specifications that are not critical to the final delivery, but that the company adviser requests if we have a surplus in development time.

Specifications given a category of “Communication” or “Payload” are based off specific functional requirements. For example, here are sample two specifications belonging to the categories of “Communication” and “Payload”:

Specification Name	Category	Priority	Behavior
Opening a connection	Communication	High	A simulated master opens a connection type with a slave device, allowing the master and slave to communicate with each other
Retrieving a slave device’s baudrate	Payload	Medium	A simulated master has an open connection with a slave device. The master requests information about the rate at which the slave device is sending cyclical messages to the master

Table 1: Example Specifications

In the above example, in **Table 1**, “Opening a connection” is the attempt to open one of two possible connection types between a master and slave device. It is given a priority of “High” because all communication capabilities is based on an open connection. Failure to correctly open a connection will prevent other specifications to be carried out.

“Retrieving a slave device’s baudrate” belongs to the category of “Payload” because the master device is requesting packets of data from the slave device. This specification is given a priority of “Medium” because a failure to carry out this activity does not prevent other specifications from being carried out. It is not categorized as “Low” because this is a specific functionality that the company expects to be delivered.

Specifications were converted into tests through the use of Python 2's built in *unittest* library. This library was chosen in order to reduce dependencies to external libraries. This made the automated system tests easier to port to other systems.

When a testing system was created, which can consist of any number of simulated master devices and 1-2 slave devices, system tests were created to test how the fieldbus interface was able to interact with a testing system. These system tests were created as python *unittest* classes. By adding *unittest* functions to the *unittest* classes, each *unittest*

function would contain a series of *unittest* assertions. Each *unittest* function would pass only if all of their *unittest* assertions would pass. Each *unittest* class would pass only if all its *unittest* functions passed.

For example, in **Table 1**, one specification is called "opening a connection." In this case, the *unittest* class would contain a *unittest* method, where:

1. A simulated master device would be created. This simulated device would need to pass a *unittest* assertion, showing that it was properly created.
2. Then a function call would be made that created a connection object, which connects the simulated master device object with code that interfaces the slave device. A *unittest* assertion would also need to be run on this function call, to see if the returned connection object was in an active state.

If the *unittest* assertions involving the creation of the simulated master, and the connection object passed, then this portion of the system test would pass.

The next step would be to increase the complexity, to see if the specification could be achieved with added concurrency and simulated masters. A new *unittest* function would be added to this *unittest* class. The main focus again would be on testing the opening of connections, except:

1. 2 simulated masters would be created. A *unittest* assertion would be made for each corresponding simulated master, to check if they were successfully created.
2. The 2 simulated masters would attempt to create a connection object with a slave device. Here, 2 *unittest* assertions must pass, indicating that each of the simulated masters was able to create an active connection.

For this portion of the system test to pass, 4 *unittest* assertions in total would need to pass. Following this, then a new *unittest* function would be created that involves 3 simulated masters being created. Each simulated master would attempt to create a connection object with a slave device. All these steps would also need to pass all it's *unittest* assertions. More *unittest* functions can be added to the *unittest* class, by creating a *unittest* function where 3 simulated masters would each try to connection to 2 slave devices.

The *unittest* class would then be run, with all its *unittest* functions attempting to test the "opening connection" specification, but under different system conditions.

Another example of creating system tests involves the second specification in **Table 1**, labeled "Retrieving a slave device's baudrate." The first *unittest* function would involve:

1. One simulated master would be created, with a *unittest* assertion being made to check that the simulated master object was properly created.
2. The simulated master would need to create an open connection with a slave device, and a *unittest* assertion would be made to check that the connection was active.
3. The simulated master would make a function call to request the baud rate of the slave device. A *unittest* assertion would be made to check if this was successful. Each data request returns a response code to indicate if the request was successful, and it

has to return a data packet that is within a particular data range. For the *unittest* assertion to pass, a response code must indicate success and a data packet within the expected data range would need to be returned.

Once this is done, more *unittest* methods would be created. Attempts to get a slave's baudrate would be done, with more instances of simulated masters and slave devices being used, similar to the incremental testing done on the "opening connection" specification.

Testing of specifications were carried out during a 5 week period, with each week representing an iteration in the testing. Each specification that was tested was given a classification, and a priority. Tests were run on a weekly basis, and would be given either a pass or fail status. Failure to pass any unittest assertions would result in a fail status.

Since the fieldbus interface works with concurrency, a suite of automated system tests would have to be run at least 30 times in succession. This is because due to the use of concurrency, an action could theoretically be successful 29 times, but could fail once in 30 tries. Failure could occur anywhere during the beginning, middle or end of 30 test runs. This meant that repeated test executions would be necessary in order to detect errors. If a test failed at least once in 30 tries, it would be considered a failed test for that iteration. Based off our observations, executing a test suite 30 times was sufficient to detect concurrency issues.

The automated system tests are aimed at testing three main areas:

1. The fieldbus interface is correctly passing messages between the master and slave devices
2. The simulated masters are correctly storing information about its open connections and providing correct access to its open connections
3. The fieldbus interface and the simulated masters are able to maintain functionality even under complex network activity. Complex network activity is defined as at least 2 simulated master devices being present, and the reliance on concurrent connections and message passing.

During each test iteration, we collect data on how many of the specifications need to be verified, and in relation to how many were actually verified by our automated system tests. This gives us data on also how many specifications have yet to be verified. We also collect data on how many "High," "Medium," and "Low" specifications are verified by our automated system testing on a weekly basis. The aim of collection this data, from the perspective of design science, is to prove that our IT artifact (the suite of automated tests), is useful in verifying the most important specifications [11].

3.3. Data Analysis

The analyzing of our data can be categorized into three parts:

1. Reflecting on the effectiveness of our automated system tests to verify specifications by executing the automated system tests. In Design science, this is referred to functional testing [11]. This relates to research question 1, which is observing how the tests can be used to verify that the correct sending and receiving of data is achieved.
2. Reflecting on the ability of our automated system tests to the observe stability of our fieldbus interface under increasingly complex conditions. This relates to research question 2, namely how we can use the automated tests to evaluate stability of the system.
3. Looking for themes that can help improve our automated system tests. This relates to improving the automated system tests that answer both research questions 1 and 2.

Checking the effectiveness of our system tests is merely analyzing for each particular week, how many specifications were verified, related to how many are left to verify. Also, we should also be able to detect how many of our previously verified specifications are no longer passing, in these situations, we can conclude that such a phenomena is being produced by a coding error. In weeks where our automated system tests are not correctly verifying "High" priority specifications, we use that information to increase our automated system tests that target "high priority specifications." The statistical information we generate are made into charts. This gives us a quick overview of our progress, giving us an indicator of where to focus during the next testing iteration

We will be simulating many master devices on our network, and we will need to observe if these simulated devices can co-exist on a computer network and are able to perform multiple concurrent actions. We will observe if the automated system tests are able to produce a wide range of stress tests, showing where we may have potential bottlenecks and deadlocks. When we identify areas in our fieldbus interface that are having stability issues, we use this information to decide on which areas of the fieldbus interface to improve.

By knowing which areas we are most likely to generate deadlocks, more automated system tests are focused on the problem spots.

4. Results

4.1. Results of the Automated System Tests

The automated test suite was created over a period of 5 weeks, with each week representing an iteration of the test suite. The test suites were not started on until the fieldbus interface was able to perform a minimum level of functionality. Namely, this meant

opening and closing connections, simulation of multiple master devices, and sending and receiving messages between multiple simulated master devices and actually existing slave devices.

Referring to **Table 2**, one can see that during 5 iterations, we were able to test, at minimum, 5 specifications per week, with at maximum 15. During this five-week iteration period, we were able to test 60 specifications. Most of the specifications were verified during iterations 1 and 5.

This can be attributed to the changes in customer demands. Before iteration 1 had started, we had produced a functioning fieldbus interface. Due to having a functioning system available, we had many specifications to verify through our automated system tests. After iteration 1, we received a request to add more concurrency into the system. This effectively caused us restructure the architecture and create new code to include more concurrency. Due to this, during iterations 2-4, a new fieldbus interface was created, and our automated system tests were meant to test its incremental development.

Week 5 had the most verification of specifications because at this point, our new fieldbus interface was completed and everything was integrated. We were able to do more testing on complex, concurrent situations that were previously unavailable.

Iteration	Tested Specifications	Passed Specifications	Passing Percentage
1	14	9	64%
2	9	7	78%
3	13	12	92%
4	9	9	100%
5	15	11	73%

Table 2: Passed and Tested Specifications During 5 Iterations

Referring to **Illustration 3**, one can see a chart depicting how many specifications were tested during each iteration, in relation to how many specifications actually passed our tests, on a weekly basis. We depict the contents of **Table 2**, but in a chart form. While **Table 2** is meant to show the actual numerical results, **Illustration 3** gives a holistic view of the

results by way of a visual. For the purpose of showing which iterations that the automated system tests were most successful in detecting failures in verification, **Illustration 3** is presented.

Illustration 3 shows that many of failures to pass verification tests were detected in iterations 1 and 5, while iterations 2, 3, and 4 had relatively few failures in verifying specifications. The reason that fewer specifications were successfully verified in iteration 1 was because this was the first time that we tried to verify specifications of a fieldbus interface with minimum functionality. When everything was integrated, problems arose regarding to the system reacting to multiple simulated master devices. Our previous manual testing was done on only one simulated master, and it worked well in this situation.

In iteration 5, there were noticeably more failed attempts at verification. This is because at this point, we had integrated a new fieldbus interface that included significantly more concurrency. Failures to pass verifications can be attributed to problems that relate to concurrency, mainly race conditions that lead to deadlocks. Other concurrency related issues relate to multiple threads operating and data being incorrectly handled while being shared between the threads.

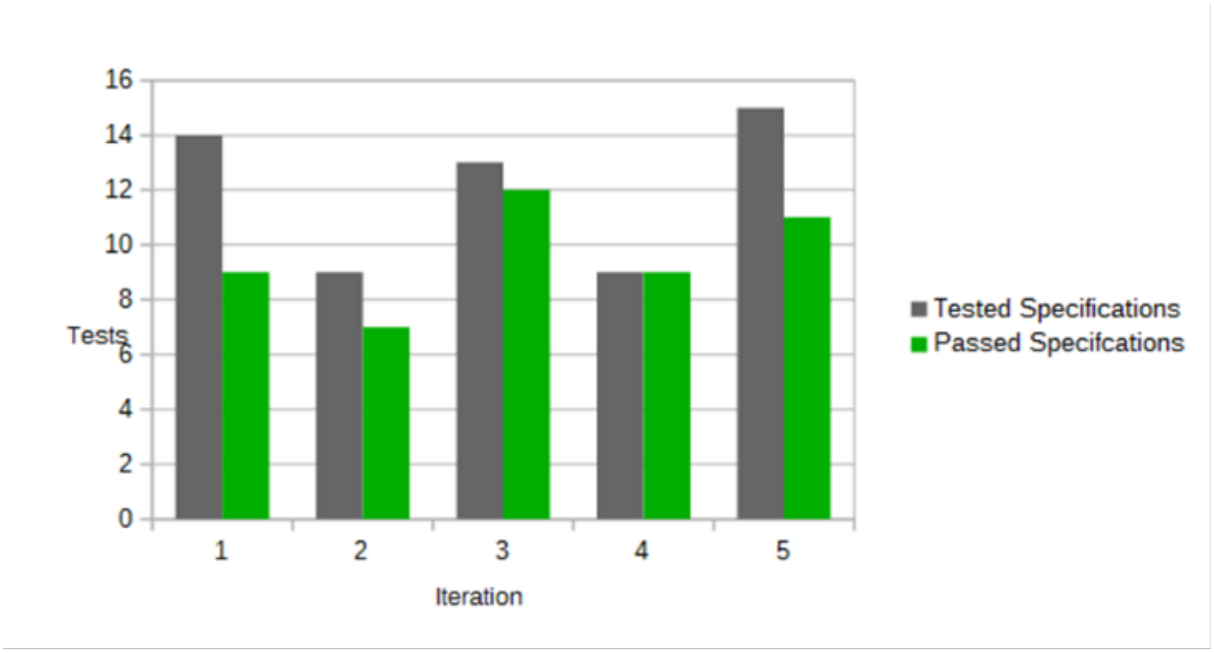

Illustration 3: Passed and Tested Specifications during 5 iterations

Over the five week period that we ran tests, all specifications were successfully tested and passed. **Illustration 4** is meant to illustrate the ability of the automated tests to find faults in general. In other words it gives an idea of the system's error margin. To be more precise, the system successfully passes all tests 80 percent of the time and finds errors 20 percent of the time.

As explained earlier the reason for the error margin is due to concurrency issues where multiple threads create race conditions. This leads to deadlocks as well as problems

with multiple threads sharing common resources.

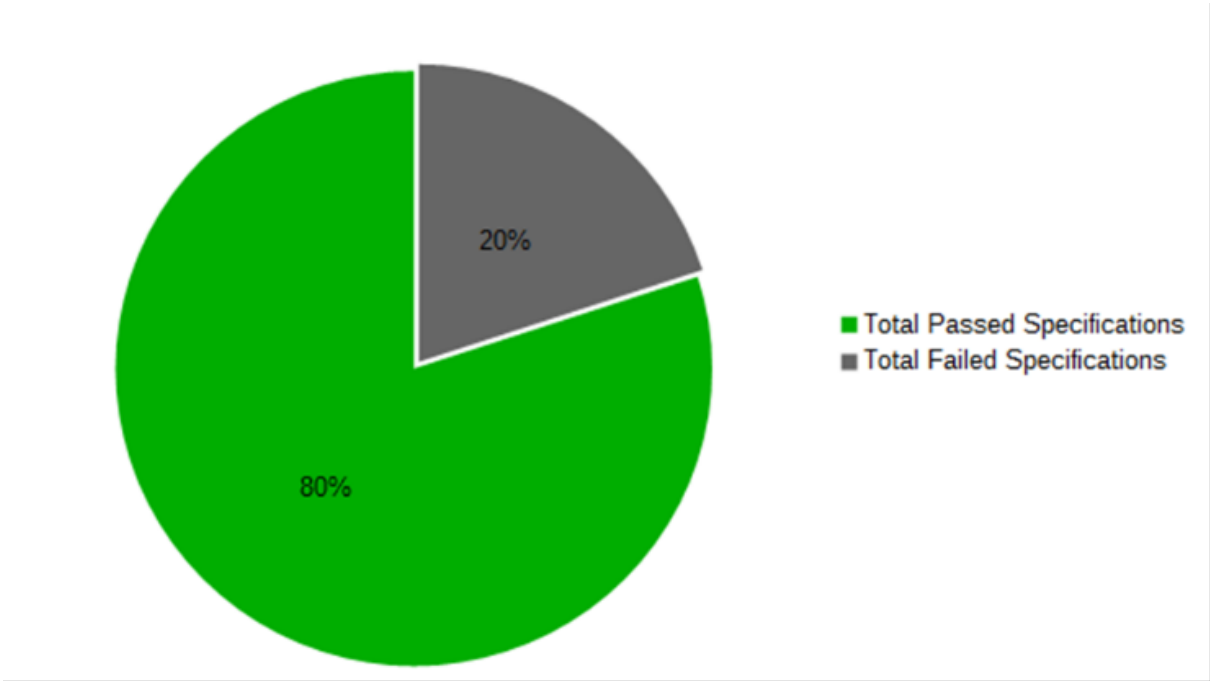

Illustration 4: Total Passed and Tested specifications during 5 iterations

As described earlier each specification that was included in the verification was also categorized into one of three different categories, as well as given a priority group. **Table 3** shows how many from each category, as well as from each priority group, were tested during each iteration.

Early on in the verification phase, the system only contained a minimum amount of functionality. However, the functionality that was included was of high importance for the system to work in general and hence being of a high priority. This lead to the amount of high priority or critical specifications incorporated into the testing to decline as more general and less important specifications were included. As can be seen in **Table 3,** the number of critical specifications drops from being twelve in the first iteration to seven in the second.

Table 3 shows how the number of specifications in the "Communication" category also drops after the first two iterations. This is because initially the communication had to be working according to specifications in order for us to be able to verify other specifications during later iterations.

	Priority			Category		
Iteration	Low	Medium	High	Communication	Stability	Payload
1	0	2	12	4	8	2
2	0	2	7	7	2	0
3	0	5	8	0	8	5
4	0	9	0	0	0	9
5	0	7	8	0	8	7
Total	0	25	35	11	26	23

Table 3: Categories and Priorities of Specifications during 5 iterations

In **Illustration 5**, we depict the contents of **Table 3**, but in a chart form. While **Table 3** is meant to show the actual numerical results, **Illustration 5** gives a holistic view of the results by way of a chart. **Illustration 5** show how the focus was on communication specifications early in the testing period. Towards the end of the testing period we went more towards focusing on ensuring that the "Payload" specifications were verified correctly and worked as intended.

In order to ensure a stable and working solution, the "Stability" specifications were continuously considered during the five iterations. The "Stability" specifications had more focus every other week in order to accommodate for any changes that were made between iterations.

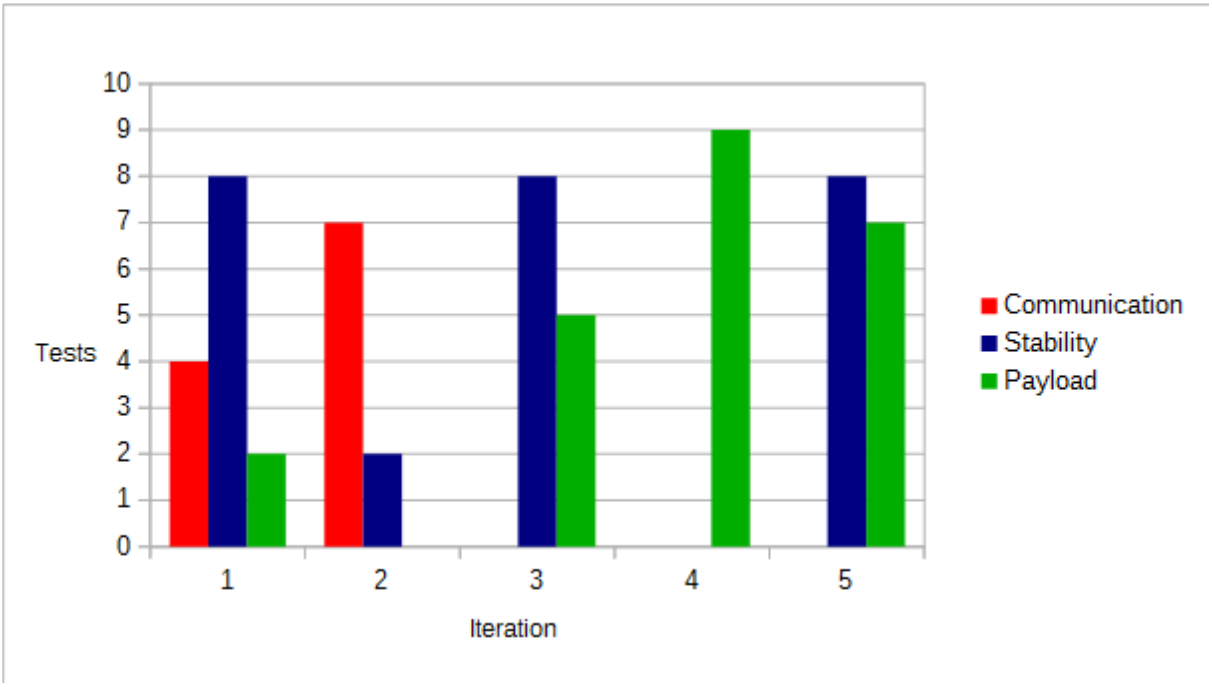

Illustration 5: Tested specifications, listed by categories, during 5 iterations

In **Illustration 6** we illustrate how the prioritization of specifications changes over the course of the testing period. First of all, there were no specifications with a low priority categorization. The reason behind their not being any low priority specifications is because we had to be able to produce a system with at least a certain amount of functionality and all of the specifications that were needed were at least to some extent important.

Illustration 6 shows that that the high priority specifications were mostly implemented early in the testing period. As we progressed through the testing period we implemented more general specifications that weren't as important for the system to work

as intended hence the increase in lower, medium priority specifications.

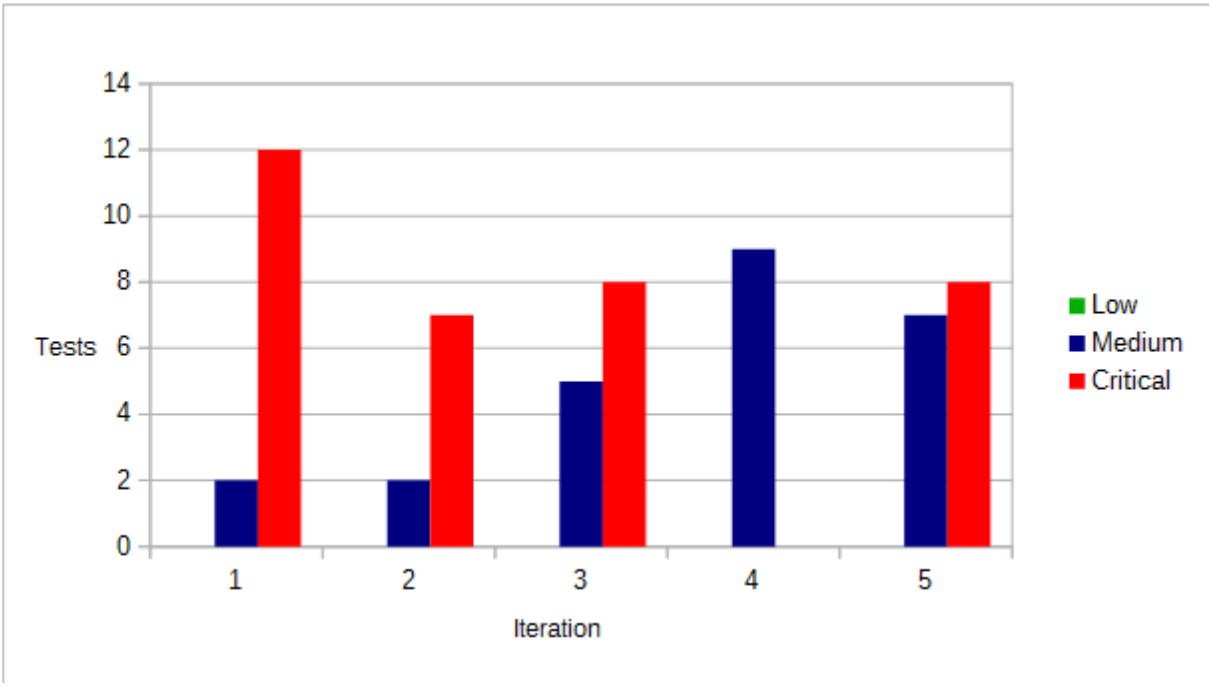

Illustration 6: Tested Specifications, Listed by Priority, During 5 Weeks of Iterations

In **Table 4** is an overview regarding the categories of specifications that we tested, and their priorities. Based off this, it shows how many of each combination of category and priority were tested. It shows that we were not able to test any specifications that were prioritized as "Low". A specification given the prioritization of "Low" indicates an implementation that is outside of the initially promised scope to be delivered to the company. These were mainly specifications with priorities of "High" and "Medium".

Category	**Priority**	**Total Times Tested**
Communication	High	9
Communication	Medium	2
Communication	Low	0
Payload	High	0
Payload	Medium	23
Payload	Low	0
Stability	High	26
Stability	Medium	0
Stability	Low	0

Table 4: Total Tested Specifications, Listed by Categories and Priority

Specifications classified as "Communication" were directed at correct opening and closing of network connections. These were often prioritized as "High" because without being able to establish connections, sending and receiving of messages would be impossible.

Specifications marked as "Payload" related to the verifying that when messages were sent to target devices, an appropriate response was generated, and that the response message contained the expected data payload. These specifications were often marked as "Medium." This was because even if the payloads didn't return correct information, it wouldn't cause the system to crash or prevent further communication between devices in the system. These medium specifications account for 38% of our specifications that we verified.

Specifications marked as "Stability" related to the ability of the system to continue working during increased network complexity. This network complexity could be caused by things like multiple simulated master devices, and reliance primarily on concurrent activity. These specifications were all found to be prioritized as "High" because failure of these activities could cause the entire system or crash, or prevent message communication between the devices within the network. Overall, 58% of our specifications that were verified were considered high priority.

4.2. Overall Design of the Fieldbus Interface that we Created

The system was developed iteratively with changes occurring with each iteration. These changes were usually in response to bugs and in order to accommodate new functionality such as concurrency or fault-tolerance.

The system was essentially built in two modules. A module that was taking care of simulating a master device and one that held utility functions as well as some extra information.

The master module consisted of a number of classes with a wide range of different purposes. Below are a few examples.

MasterClass

The MasterClass is used to simulate a master device. An object of the MasterClass is able to, for example, open connections and send messages. Any number of MasterClass objects can be instantiated.

- MasterClass
    - Represents an instance of a simulated master.
    - Uses the OpenConnectionClass to interface the slave devices.
    - Is the main object that is used to do most operations in the system.

OpenConnectionClass

The OpenConnectionClass is used to open 1 of 2 possible connection types. This is

done by sending a message from a simulated master device to a slave device. The message that is sent contains information about the connections communication protocol. This includes information such as the size of the data fields, connection identifier, etc. After the initial message is sent, the simulated master device expects a response from the slave device. The response will include either the accepted connection identifier or an error. Whenever a connection is requested, a new thread is setup for that specific connection object in order for the system to be able to work concurrently.

- OpenConnectionClass
    - Sends open connection request message to slave device.
    - Saves connection identifier returned in the response message.
    - Is used by CloseConnectionClass to terminate connection.

MessageClass

Through the use of the OpenConnectionClasses, information can be exchanged between instances of the MasterClass and the various slave devices. The packets of data being sent and received contains information specific to that transmission, such as the identity of the master device, the identity of the slave device, and the data payloads being sent and received. Various message classes are used to store the information within the data payloads.

- MessageClass
    - Takes data payloads of bytes, and breaks the payloads down into it's bit components so that they can be stored as retrievable variables
    - Works with data payloads used for sending and receiving messages
    - Many different message classes due to many messages types and because messages use data payloads of varying sizes

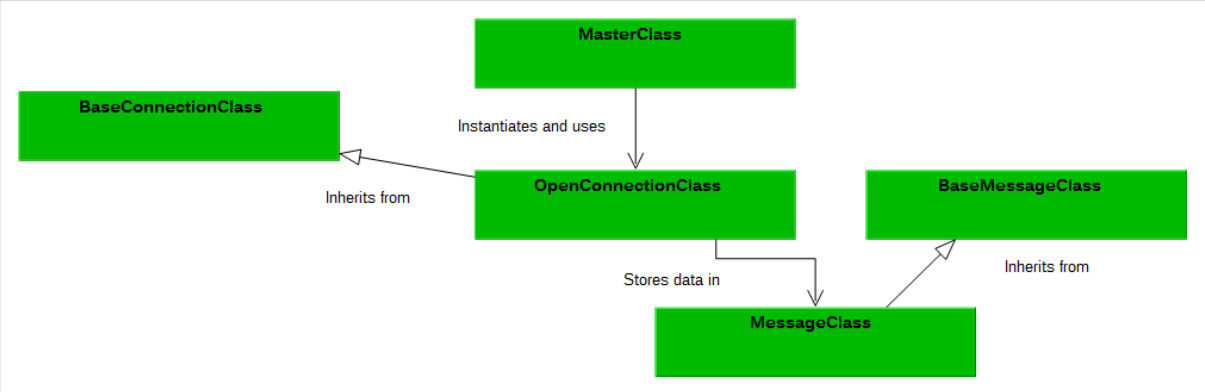

Illustration 7: Overall Class Diagram

Above is Illustration 7, which is a high-level class diagram of our system. It highlights that the "OpenConnectionClass" and the "MessageClass" described above are abstractions of our

system. They are actually base classes, from which we inherited from and created specific connection classes and message classes.

External Libraries

In order to be able communicate over the network, we were provided with an interface that was used to gain access to the network. This interface worked with drivers that allowed communication between the slave devices and the intermediary device given by the company. This interface was an external library provided by the company.

Threading

In order to employ concurrency in the fieldbus interface, multiple threads have to be created. In short, these threads are:

1. The main thread that processes the simulated masters
2. Threads for each open connection, allowing for messages to be concurrently sent and received
3. Threads for interfacing and using the external libraries

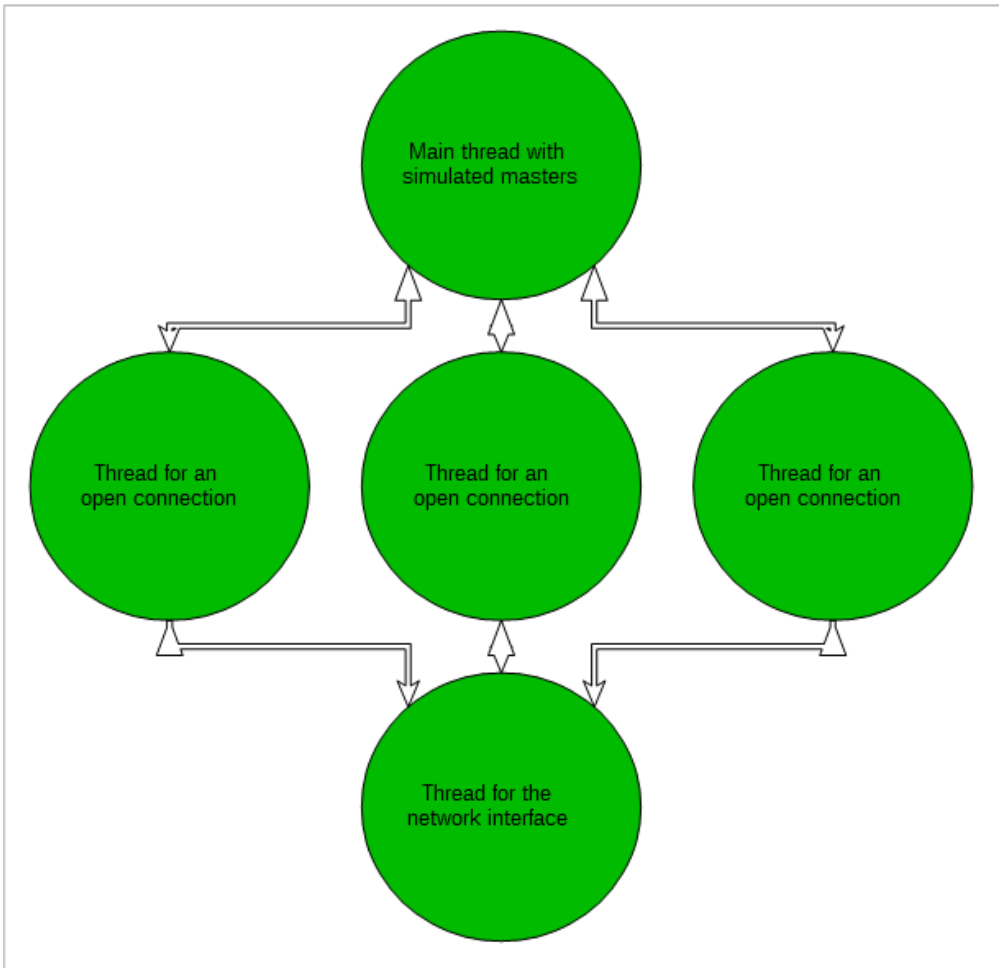

Illustration 8: Threads

Above is **Illustration 8**, there is depiction of how the threads worked with each other. The multiple simulated masters were on a main thread, and the "Thread for the network interface," was the thread that accessed the external libraries. These two threads were fixed, were as the number of threads for open connections could vary in number. This is because each open connection instance had its own thread, and there could be multiple open connections.

There are then two main challenges when working with the threads. The first is passing data between the threads. The second is making sure that when a simulated master requests information, that the the response message is directed back to the correct simulated master instance. As is seen in **Illustration 8**, a response message would have to travel through at minimum, three threads.

The solution was then to employ a solution with queues, which could encapsulate data and send it between threads. We also employed a registry to keep track of which simulated masters had sent data to another thread. This ensured that when a response was generated, the simulated master instance would receive the correct the response message from another thread.

4.3. The Overall Result

From the side of the company, we were given external libraries used for interfacing the slave devices. We also received 2 slave devices that were accessed through the external libraries, and an intermediary device that facilitated communication between the slave devices and our company issued computers. From our side, our main focus was mainly on software. We programmed a fieldbus interface that used the external libraries that communicated with the slave devices. This programmatic solution included simulations of master devices, and the communication between the simulated master devices and the external libraries.

Since our main task was to focus on software, we searched for a way to improve the fieldbus interface that we would deliver. In order to improve our fieldbus interface, we programmed a suite of automated system tests that could test the functionality of the fieldbus interface. Through testing, we discovered problem areas within the fieldbus interface that either needed to be fixed or improved. The suite of automated system tests were created and improved iteratively, within the framework of design science.

We met with our company advisor on a weekly basis to show the progress of our fieldbus interface. During these meetings, the fieldbus interface was often subjected to an acceptance test, which we continually passed. As a result, the company has a fieldbus interface that is being used for live testing. In these testings, the intermediary device is connected to different slave devices, and the behavior of the external libraries in relation to the different slave devices can be observed.

5. Analysis and Discussion

What follows is a discussion of how through the incremental development and improvement of our suite of automated system tests, we were able to answer research questions 1 and 2. Namely, how the automated system tests were able to help us test messaging capabilities of the fieldbus interface, and how we were able to evaluate its stability under complex system conditions. Once the findings have been discussed, what follows is a section mentioning possible threats to validity.

5.1. Findings in Relation to Research Question 1

The bulk of what a fieldbus interface executes is done through sending and receiving of messages. Through sending messages, a master device can make requests to open and close a variety of network connections with slave devices, to request the attributes of a slave device, to set the attributes of a slave device, and to make requests to receive a

stream of data that is sent repeatedly at determined intervals from numerous slave devices. Since the main functionality of a fieldbus interface is done through sending and receiving of messages, the initial and large portion of the job of the automated system tests is to verify that sending and receiving of messages is being correctly done.

We have classified specifications into the three categories of "Communication", "Payload", and "Stability". Of these three, the correct sending and receiving of data relates itself to specifications that fall into the categories of "Communication" and "Payload".

Of the 60 total specifications that we were able to verify in the fieldbus interface, 37 of the specifications belonged to the categories of "Communication" and "Payload" (see **Table 3**), meaning that 61% of automated system tests were dedicated to verifying specifications related to sending and receiving messages. At the beginning of the research, one of our main targets was to analyze how we could use automated system tests to evaluate how well the fieldbus interface was able to correctly send and receive messages. Since 61% of our automated tests were targeted at these important areas of “Communication” and “Payload”, our data shows that we were successful in targeting this important area.

Illustration 5 shows that during iterations 1 and 2, a concentrated effort was made to test “Communication” specifications. In iterations 3, 4, and 5, a concentrated effort was made to test “Payload” specifications. The common pattern is that testing these important areas were concentrated for a few iterations, and once testing had reached satisfactory levels, they ended. This shows that through the aid of automated system testing, we were able to verify the fieldbus interface specifications that related to sending and receiving messages.

In **Table 4**, it shows that 9 of the 11 "Communication" specifications that were verified were prioritized as “High.” They were prioritized this way because if the connections were not correctly opened and closed, the entire fieldbus interface would not even be able to pass messages. Of the 3 categories of specifications that we tested for, "Communication" was the most important, and of the 11 "Communication" specifications that we tested for, 9 were of high priority. This shows that automated testing can be used to target the most important categories of specifications, and even further concentrate on their most critical parts. While increased targeting is a general function of testing, from the context of fieldbus interface testing, it helped us evaluate if message passing was successful with different slave devices.

23 of the 23 "Payload" specifications were marked as medium (see **Table 4**). The "Payload" category is more related to whether the messages being sent are getting the correct responses from target devices. Out of the 60 specifications we tested, 23 belonged to this category. **Illustration 5** indicates that most of these specifications were verified in the last 3 iterations. This is due to more focus being put on them during this period, and each iteration provided insight into which areas need to be tested more in depth in the later iterations. In relation to the fieldbus interface, the tests gave us insight into if requests to the slave devices were generating the appropriate response codes and if the data payloads were within an expected range. In cases where the data payloads were incorrect, they were often caused by an incorrect or incomplete chain of events being executed in order to generate a response from a slave device.

5.2. Findings in Relation to Research Question 2

Referring to **Table 3**, 26 of the 60 verified specifications belonged to the "Stability" category. “Stability” dealt with how the system behaves when multiple master devices are simulated and therefore increase the network traffic. “Stability” also relates to concurrency specifications, an important feature towards the later parts of development. 43% of verifications were dedicated to this area, and this indicates that the suite of automated system tests were used to make a deliberate concentration in this area.

The bulk of the "Stability" verifications were conducted during iterations 1, 3, and 5 (see **Illustration 5**). This is because during the period before iteration 1, and during iterations 2 and 5, new code was being integrated into the fieldbus interface. Once code was iteratively integrated, this allowed the "Stability" verifications during iterations 1, 3, and 5 to happen. By use of progressive testing, this indicates that the automated suites were used to systematically improve the system, and that the testing suites were used to complement the incremental development of the fieldbus interface. So if concurrency were the focus during a development week, then automated testing of concurrency would be created to support development.

In reference to **Illustration 3**, one can see that many of the failed tests occurred during iterations 1 and 5. Many of these tests were due to messages failing to be routed back to the correct simulated master. Additionally, many errors were generated due to concurrency. Since fieldbus interface employed concurrency, the automated system tests were able to detect failures caused by deadlocks. Since there were multiple threads and many simulated masters that were interacting with these threads, the fieldbus interface was also responsible for passing data between these threads and between the simulated masters. Our solution to this involved a message passing mechanism that allowed the data to be thread safe and which could occasionally block execution until certain responses were generated. The automated system tests helped us find failing areas in this process.

5.3. Overall Effectiveness of the Test Suites

Illustration 2 shows that during the course of verifying 60 specifications, 20% of the total specifications, or 12 of the specifications, were found to not have initially passed our tests. By having 12 tests fail at attempts to verification, this gave us the opportunity to fix 12 problem areas. These were problem areas that were not initially detected during the course of static code review, paired programming review, and manual black box testing. Had we not run our suite of automated tests, these failures at verification would have likely gone unnoticed.

5.4. Threats to Validity

During implementing the fieldbus interface, we actually created two entirely different fieldbus interfaces. We implemented one that didn’t include concurrency, but focused mainly on carrying functionality through the use of multiple, simulated master devices. Based off this fieldbus interface, we created our first iteration of the automated system tests. After this iteration, the company requested that we add concurrency and to restructure the architecture. Due to this, we had re-implemented many of our features within the context of an entirely new framework. On this basis, we completed the remaining 4 iterations of the suite of automated tests.

This can become a threat to validity through the process of maturation. Test iterations 2-4 had noticeably fewer errors detected when compared to iteration 1. This could be because iterations 2-4 were tests on a fieldbus interface which was a re-implementation of a fieldbus interface that was tested during iteration 1. Even though the architecture was different and there was added concurrency, many of the basic functionality was the same. This could have led to unintentionally having more error free source code to test for iterations 2-4. Having less errors could affect our data regarding the efficiency of our tests in finding errors.

Before we began each iteration of automated system tests, we also performed static code review and several manual tests on the fieldbus interface. On this basis of these testing procedures, we would run an iteration of automated system tests. However, we didn't implement a standard in terms of how much manual testing and static code review we performed prior to doing the automated system tests. We did what we considered was enough to have the system run correctly in a few trial runs. How many trial runs and how exhaustive the manual testing was wasn't standardized. Therefore more exhaustive manual testing and code review one week could affect how many errors our automated system tests were able to discover during a test iteration.

6. Conclusion

The research paper was meant to evaluate whether automated system tests could be used to evaluate whether a fieldbus interface is able to perform its core functionality. This functionality would need to be stable even during complex network activity, such as during interactions with multiple, simulated hardware devices and during the execution of multiple concurrent activities.

Prior to running the automated system tests on our system, we engaged in many other forms of testing. We did static, peer code review in order to find errors in code, and to also detect the times where we had misinterpreted specifications and implemented incorrectly. In addition, we also engaged in the practice of pair programming in order to reduce errors. During development, we also ran manual tests to see if the fieldbus interface source code was executing correctly. Despite all these measures, the suite of automated

system tests were still able to fail at a rate 20%. This indicates that automated system tests can be used to complement other testing methods in order to create a correctly functioning fieldbus interface. This is because we were able to find errors that we were not able to detect using other testing methods.

Due to the results presented by our automated system tests, we can conclude that the tests can be used to deliberately verify areas of a fieldbus interface which are of vital importance. 62% of the system tests were directed at answering the Research Question 1, which was the testing of sending and receive of messages. The remaining 38% of the system tests were directed at answering Research Question 2, which was using the automated suite to evaluate the stability of the system.

Not only can the suite of automated tests be used to target areas of the highest interest, they can be used to target the most critical areas of the system. Of the 60 specifications we verified, 58% of them were targeted at specifications which we deemed as being high priority to the operation of the fieldbus interface. The remaining 42% were targeted at specifications prioritized as "Medium".

The automated system tests is an IT artifact that can also be created incrementally, in parallel with development. Many of the categories that the automated tests focused on an iteration-by-iteration basis, were focused on because these were the areas being developed in the fieldbus interface.

At the same time, the suite of tests were also used to influence development. Whichever areas were spotted as not passing the tests, we focused on fixing during development. When we discovered that our suite of tests were not focusing on certain areas enough, this could be due to two reasons. Either we didn't create enough automated system tests to explore those areas, or we didn't have enough code to test on. If it was due to the later condition, this influenced us in building up that part of the system which was lacking.

Of particular interest was using the automated test suites to detect failures in complex, concurrent conditions. The automated test suites could produce situations involving numerous concurrent actions. Since these were in the form of test suites, these tests were reproducible, and could be run many times. Due to the concurrent nature of the system, the tests would need to be run several times to find conditions that could fail, but were not guaranteed to fail 100% of the time. The value of the automated systems tests was having reproducible system tests that could be run multiple times in succession, and which could detect these occasional, concurrency caused errors.

7. Future Work

To test whether our automated system tests has aided in creating a reliable fieldbus interface that can scale up in complexity, future work could include working with more complex systems.

At present, fieldbus interface is set up to work with 2 slave devices and a series of simulated master devices. The next step would be to test the fieldbus interface with a wider range of slave devices, and to observe how they respond to the simulated masters. Naturally, the interactions with new slave devices could be tested with new automated system tests.

Additionally, we could test increasingly more complex levels of network activity. By having the fieldbus interface interact with several more slave devices at once, we could also increase the amount of simulated master devices on the network. By expanding our suite of automated system tests, we could evaluate how the fieldbus interface is able to respond to this heightened network activity.

Since the fieldbus interface is only able to maintain a limited amount of connection types with the slave devices, one obvious next step would be to expand the amount of connection types that the fieldbus interface can manage. To test how these additional connection types respond to various slave devices, the use of automated system tests could be employed to help discover problem areas within the system.

8. References

1. R. Stohl and K. Stibor, “Safety Through Common Industrial Protocol,” in *14th International Carpathian Control Conference*, Rytro, 2013, pp. 442-445.
2. R. P. Pantoni, N. M. Torrisi, D. Brandao, E.A. Mossin, “Integration of an Open and Non-prorietary Device Description Technology in a Foundation Fieldbus Simulator,” in *IEEE International Workshop Communication Systems*, Dresden, 2008, pp. 435-444.
3. R. Stohl and K. Stibor, “Safety Through Common Industrial Protocol,” in *14th International Carpathian Control Conference*, Rytro, 2013, pp. 442-445.
4. R. P. Pantoni, N. M. Torrisi, D. Brandao, E.A. Mossin, “Integration of an Open and Non-prorietary Device Description Technology in a Foundation Fieldbus Simulator,” in *IEEE International Workshop on Factory Communication Systems*, Dresden, 2008, pp. 435-444.
5. J. Hong, C. Tsai and C. Wu, “Hierarchical System Test by an IEEE 1149.5 MTM-Bus Slave-Module Interface Core,” *Very Large Scale Integration (VLSI) Systems*, IEEE Transactions, vol.8, no. 5, pp. 503-516, 2000.
6. F. Maturana, R. Ambre, R. Staron and D. Carnahan, “Simulation-based Environment for Modeling Distributed Agents for Smart Grid Energy Management,” *IEEE 16th Conference on Emerging Technologies & Factory Automation*, Toulouse, 2011, pp. 1-7.
7. E.A. Mossin, R.P. Pantoni, D. Brandao and N.M. Torrisi, “Fieldbus Simulator: Architecture, Typical Experiment and Tool Evaluation,” in *34th Annual Conference of IEEE*, Orlando, Fl, pp. 3512 – 3517.

8. T. Syed, S.R. Das, S.N. Biswas, and E.M. Petriu, “Developing Automated Test System for ADSL Equipment” in *IEEE Instrumentation and Measurement Technology Conference Proceedings*, Victoria, B.C., 2008, pp. 1833-1838.
9. M. Kantona, I. Kastelan, V. Pekovic and N. Teslic, “Automatic black box testing of television systems on the final production line,” *IEEE Transactions on Consumer Electronics*, vol. 57, no. 1, 2011, pp. 224-231.
10. S. Magnus and J. Krause, “Test Generation for Model Based Fieldbus Profiles,” *IEEE International Conference on Industrial Technology*, Athens, pp. 682-687.
11. A. Hevner, S. March, and S. Ram, “Design Science in Information Systems Research,” *MIS Quartlery*, 2004, vol. 28, no. 1, pp. 75-105.