KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Technology

First cycle, 15 credits

Developing Guidelines for Structured Process Data Transfer

PONTUS AMGREN
EMIL OLAUSSON

Stockholm, Sweden, 2023

Developing Guidelines for Structured Process Data Transfer

PONTUS AMGREN

EMIL OLAUSSON

Bachelor's Programme in Information and Communication Technology
Date: August 31, 2023

Supervisor: Mira Kajko-Mattsson
Examiner: Thomas Sjöland
School of Electrical Engineering and Computer Science
Swedish title: Utvecklande av riktlinjer för strukturerad process dataöverföring

Abstract

Today, society is ever-increasing in its use of technology and computers. The increase in technology creates a need for different programming languages with unique properties. The creation of a system may require multiple languages for multiple processes that need to transfer data between one another. There are several solutions for sharing data between processes with their respective strengths and weaknesses. The differences create a problem of needing to understand the data transfer solutions to use them effectively.

This thesis addresses the problem of there not existing any guidelines for data transfer solutions. The purpose is to create guidelines for choosing a data transfer solution. The goal is to help software developers find a data transfer solution that fits their needs.

The thesis is meant to inform and contribute to the understanding of possible solutions for sharing data between processes. A literature study and practical study were needed to get that understanding. The literature study was conducted to understand the solutions and to be able to compare them. After that, a practical study was performed to work with the solutions and gain experience. The study was meant to gain measurements for later comparisons of data transfer solutions. The measurements followed the comparative criteria of *speed*, *resource usage*, and *language support*.

The result was the creation of guidelines that displayed different scenarios based on the comparative criteria. For each situation, there was a recommendation of solutions that would help in the given situation. These results accomplished the goal and purpose by providing guidelines that could help software developers choose a data transfer solution.

Keywords

Inter-Process Communication, Data Interchange Format, Performance Evaluation

Sammanfattning

Användningen av olika teknologier och datorer ökar konstant i dagens samhälle. Detta skapar ett behov av olika programmeringsspråk med olika egenskaper. Ett projekt kan kräver flera språk för olika processer, så kan programmen behöva kommunicera genom att överföra data sinsemellan. Det finns olika dataöverföringslösningar för att dela data och de har sina svagheter och styrkor. Skillnaderna skapar problemet att en användare behöver förstå dataöverföringslösningar för att använda dem effektivt.

Avhandlingen tar upp problemet att det inte finns några riktlinjer för dataöverföringslösningar. Syftet med avhandlingen är att skapa riktlinjer för att välja en dataöverföringslösning. Målet med avhandlingen är att hjälpa mjukvaruutvecklare att välja en dataöverföringslösning som passar deras behov.

Avhandlingen är menad att informera och att bidra till förståelsen av dataöverföringslösningar. Därför krävdes det både en litteraturstudie och en praktisk studie. Litteraturstudien utfördes för att få en förståelse för de olika lösningarna och kunna jämföra dem. Den praktiska studien utfördes för att arbeta med lösningarna och lära sig om dem. Arbetet var menat att ta fram mätvärden för att kunna jämföra dataöverföringslösningar. Mätvärdena följde jämnförelsekriterierna *hastighet*, *resursanvändning*, och *tillgängliga språk*.

Resultatet av avhandlingen var skapandet av riktlinjerna. Riktlinjerna visar olika situationer baserade på jämförelsekriterierna. För varje situation rekommenderas det en dataöverföringslösning som hjälpte i situationen. Resultatet uppnådde syftet och målet med avhandlingen genom att skapa riktlinjer som hjälper mjukvaruutvecklare att välja dataöverföringslösningar.

Nyckelord

Interprocesskommunikation, Dataöverföringsformat, Utvärdering av Prestanda

Acknowledgements

First of all, we would like to thank our supervisor, Mira Kajko-Mattson, for her valuable feedback and insight throughout the project. Her support and feedback has helped shape the thesis to what it has become. With her knowledge she helped guide the thesis in the proper direction when we were unsure of how to proceed further. We would like to thank Mira Kajko-Matsson for being available to answer our constant questions throughout the project.

We would also like to thank our examiner, Thomas Sjöland, for essential feedback that allowed us to focus our efforts on the correct path. His insightful comments helped improve the thesis and provide a clear direction. We want to thank Thomas Sjöland for being attentive to detail to help with problems that we did not think about.

We would like to thank our fellow students who have been a part of the whole creation process. They have given hard and valuable critique that has helped in the creation process. They have also allowed for discussion, which led to a better thinking process for solving problems.

Lastly, we would like to thank our families and friends for their unending support during the creation of the thesis. Their support was crucial for making us able to write the thesis even in times of doubt and stress.

Contents

1. Introduction

Today we live in a technology-dominated society where almost everything is in some way connected to the Internet (Statistics Sweden, 2022). To be able to use the internet one needs some kind of computer. Then to use the computers in the way that we want them, we need some kind of way to give instructions. Those instructions are called programming. As there are many different spoken languages there are different programming languages. All of these programming languages function in different ways, much like spoken languages and their different grammar. With how spoken languages are affected by their location on earth, programming languages are affected by what purpose they are meant to serve. This leads to programming languages having different strengths and weaknesses and are good at different things (Nanz and Furia, 2015).

The differences in strengths and weaknesses creates situations where one might want to use multiple programming languages, to utilise their respective strengths. This is supported by a study that found that the average number of programming languages used on projects hosted on GitHub was five (Mayer and Bauer, 2015). The problem with using multiple languages is that it involves working with multiple programs. The problem with working with multiple programs is that they are designed not to affect one another. This is done to prevent programs from getting involved with one another and creating problems. However, allowing programs to cooperate can lead to multiple benefits. Some benefits from cooperation is an increase in speeds and a decrease in the need for copying data. Therefore, there was a need to develop solutions for transferring or sharing data between programs.

1.1 Background

When writing a program the user writes what it is supposed to do in a programming language. The written program can either be started directly after writing, or the program needs to be converted into a form that the computer can read, before starting. Whether the program needs to be converted depends on the programming language. Once the program finally starts a process is created. The process is a way for the computer to recognize the running program. Therefore, when transferring data between programs one needs to transfer it between their respective processes. Solutions exist for the transfer of data between processes.

There are multiple solutions that transfer data between processes that have already been created. The solutions were created with different goals in mind. These different goals gave the solutions different strengths and weaknesses.

Due to the strengths and weaknesses of the different solutions, they should be used in different situations. This is because some situations may require the solutions to be as fast as possible, but other situations may require that the solution is resource efficient. That then creates the problem that one needs to have a good understanding of the different solutions, in order to implement the most suitable one.

All developers may not have a sufficient understanding of the solutions, required to choose the most suitable one. They then need to search for information to make an educated decision. The problem is that the information is spread out and there are no easy guides or guidelines to help choose a solution. Therefore there is a need for such guidelines that can help the developer make the decision. This will result in the developer not having to spend time and effort searching for hard-to-find information, or choosing a suboptimal solution.

1.2 Problem

The problem that this thesis addresses is that there are no guidelines for data transfer solutions.

1.3 Purpose

The purpose of this thesis is to create guidelines for choosing a solution to transfer data between processes.

1.4 Goal

The main goal of this thesis is to help software developers find the data transfer solution that best fits their needs.

1.5 Research Method

The thesis is about giving information and understanding different solutions for communication between processes. This requires a good

understanding of the solutions and the inner workings of them. To gain information on the different solutions a pre-study is conducted which includes both a literature study and a practical study. To compare the solutions with the information gained from the pre-study, a comparison model is created with important criteria.

The type of the research is qualitative in how the thesis works with different case studies and observations. This resulted in new insights that came from both the studies and our own measurements. A comparative approach was chosen, as the different solutions needed to be compared to one another.

The literature studies were used to gather information on the solutions from different perspectives. The practical study was used to gather our own experience and measurements to compare with others. The measurements were later used to compare the solutions to one another. The research followed a qualitative structure with the support of comparative research. The research method is further explained in Chapter 3.

1.6 Target Audience

The target audience of this thesis is the software development industry and the academic field. The thesis could help the industry by presenting some of the available options and when to use what solution. This could help by allowing inexperienced software developers to choose the optimal solution for their needs. For the academic side, the thesis could be used as educational material for programming students. The thesis could also be used as educational material to help students understand the different solutions and how to evaluate different solutions. They could learn how to evaluate solutions by understanding how the thesis evaluates communication methods. For the research community of the academic side then the thesis will provide information on the solutions, both through literature studies and some measurements.

1.7 Scope and Limitations

The focus of the thesis was to evaluate the different solutions for transferring data between processes. The data, in this case, was limited to two structures: tree-structured and tabular. The data was chosen to be represented by the following formats: Extensible Markup Language (XML), JavaScript Object Notation (JSON), Binary JSON (BSON), and Comma-Separated Values (CSV). With the large number of different languages and their differences, not all languages can be tested. This resulted in the thesis mainly evaluating the communication solutions for the

languages Javascript, Python and C. The thesis focuses on the solutions that the operating systems Windows and Linux provide.

1.8 Benefits, Ethics, Sustainability

The result of the thesis could benefit academic students that study software development. This is due to the fact that they could get a better understanding of the data transfer solutions from the thesis. The students could also use the thesis as a way to learn how to evaluate different solutions by using measurement and understanding of the solution. The result could positively contribute to companies that work with different programs and processes. The thesis could have a positive impact due to the company being able to understand what method they should use for their needs.

The thesis could help software professionals choose a solution that fits their needs. This could help save time and resources for the developers when choosing a solution. It could also help by choosing a solution that better fits the system. This could make the system run better and save time, resources and money.

With the perspective of sustainability the thesis can in one way help. The thesis compares different methods of transferring structured data between processes. The comparison could help understand which method is better in resource usage on the used device. This information can be used when creating a system with low resource usage, in order to achieve lower energy consumption. Lower energy consumption can lead to lower usage of non-renewable energy sources. During the creation of this thesis, importance was held on keeping to the IEEE Code of Ethics (Institute of Electrical and Electronics Engineers. 2020).

1.9 Thesis Outline

The thesis is structured in the following manner:

- *Chapter 2: Data transfer solutions:* This chapter provides information on the subject of data transfer solutions. It first gives a historical background and then goes into more detail about different solutions.
- *Chapter 3: Research Method:* This chapter describes the research method of the thesis. It contains the different aspects of the research

such as research phases, research strategies and more. This chapter also includes the comparison model that is used in the thesis.

- *Chapter 4: Pre-study Results:* This chapter presents the measurements from different methods in accordance with the comparison criteria.
- *Chapter 5: Guidelines for Use of Structured Data Transfer Solutions:* This chapter presents the guidelines that are created from the results in Chapter 5.
- *Chapter 6: Analysis and Discussion:* This chapter summarises the results of Chapters 5 and 6. The results from the chapters are then analysed. Lastly in the chapter, there is a discussion of the results and what they mean.
- *Chapter 7: Conclusions and Future Work:* This chapter describes the conclusions that are drawn from the analysis. It is then further extended with potential future works, outside of the scope of this thesis.

2. Data Transfer Solutions

Chapter 2 contains information regarding data transfer solutions. It includes why they were created, what they are called and how they work. The chapter goes through a historical introduction to data transfer solutions within Section 2.1. This is to give an understanding of how and why the data transfer solutions exist. Section 2.2 provides a more theoretical background on the subject of data transfer solutions and computers. The section also includes some information on why data transfer solutions are needed, and how the data can be structured. Section 2.3 contains information on existing solutions for how to transfer data between programs. The last section is Section 2.4 which discusses related works.

2.1 Historical Background

When looking into the historical aspect of computers then in the time of the 1940s were the start of the computers we know today. The first commercial computers could not run multiple programs or processes at a time. But in 1961 the computer called LEO-III was created and was the first computer that allowed for multitasking (*Leo Computers Society*, n.d.). Multitasking allowed for multiple programs and processes to be run at the same time. This was achieved by running a lower prioritised process when higher prioritised processes waited for something (*Leo III User Manual Vol IV MASTER PROGRAMME and PROGRAMME TRIALS SYSTEM*, n.d.). Multitasking improved computation speed but allowed for the problem of processes affecting one another whilst they were running.

Potential defects could occur from processes freely affecting one another's memory. Therefore memory protection was created to limit a process to only affect its own memory. But having cooperating processes can increase computation speed, modularity, and information sharing (Silberschatz, Galvin, and Gagne, 2018). This led to the creation of inter-process communication (IPC), which allowed for safer ways of affecting processes memory.

The first IPC solution was the use of shared memory that multiple processes can access. Shared memory was implemented first with the operating system XDS-940 in the early 1960s (Silberschatz, Galvin, and Gagne, 2018). It allowed for the cooperation of processes. However, with it came the problem of multiple processes writing over one anothers information. This led to the creation of semaphores in the THE operating system in the mid 1960s (Silberschatz, Galvin, and Gagne, 2018). Semaphores allow for the decision of

which process can access the memory at a given moment. Later, in the late 1960s came the creation of message passing with the RC 4000 operating system (Silberschatz, Galvin, and Gagne, 2018). Message passing allowed the sharing of messages of memory, instead of whole memory spaces.

The evolution of computers led to diversity in how the computers functioned and handled data. This created a need for a more standardised way of organising data that could be shared amongst computers that differ in their inner workings. There were many different standards that were created and those standards were called data interchange formats.

One early data interchange format, originating in 1972, is CSV (IBM, 1972). Later, along with the rise of the internet in the 1990s, another data interchange format called XML was developed (Hemmendinger, 2023). Soon after XML rose to popularity, another data format called JSON, was developed in 2001 (Florescu and Fourny, 2013). JSON was designed as a lightweight alternative to XML, also being able to represent complex structures, but with a simpler syntax. As internet traffic grew and more data was being transferred, lightweight, fast, and efficient formats were developed. One such format is BSON (MongoDB, n.d.).

2.2 Computer Fundamentals

Section 2.2 contains information that is needed to understand the technical side of the problem and solutions. The section brings forth information regarding how computers work and how data can be organised in Section 2.2.1. It also showcases a subset of the different formats used for data storage in Section 2.2.2. Finally, Sections 2.2.3 presents the different fundamentals to inter-process communication solutions.

2.2.1 Computer Systems

The data that is used in a process comes in many different forms called data types. Examples of data types include Booleans for representing true/false values, Integers for representing whole numbers, and Strings for representing a sequence of characters. It is possible to organise data into structures called data structures. There are many types of data structures, two common ones are trees and lists. Lists store data in a linear manner, while tree data structures allow for storing non-linear hierarchical data. Figure 1 depicts the differences between these structures.

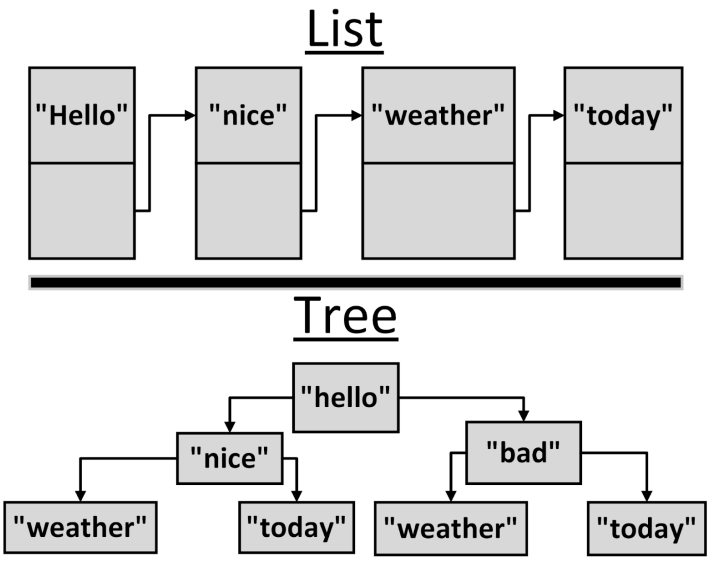

Figure 1. List (above) and tree (below).

Figure 1 shows that the list is linear with its data where one value, in this case a word, points to the next value. This is in contrast to the tree-structure illustrated below the list, which has multiple paths for each value. This results in the tree being able to have different values in a hierarchical structure. Or in the case of Figure 1, it can present different ways a sentence can be structured.

When working with computers it may be desirable to have a computer that is only intended for a single purpose. This can be attained using a virtual machine (VM). Oracle describes a VM as a "computer made from software" that can run any software on a physical computer (*What Is a Virtual Machine?*, n.d.). With a VM, resources can be allocated to create a testing environment that is unaffected by programs outside of the VM.

Out of the available resources in a computer, the thesis focuses on the following: CPU, storage, and memory. The CPU is the central part of the computer and follows all instructions. The storage is where all of the persistent information on the computer is kept. The memory or random access memory (RAM) acts as smaller and faster temporary storage for the CPU.

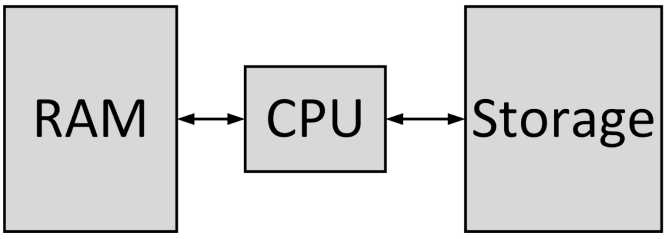

Figure 2. The structure of RAM, CPU and storage

Figure 2 presents the structure of how the CPU, RAM and storage are structured. The CPU acts similar to an accountant working at their desk. The storage is similar to a filing cabinet, in how it contains lots of information and the CPU can take information from the storage. This is similar to the accountant taking forth a new document from their cabinet. RAM is a place where information from the storage can be placed temporarily. RAM is more limited in size compared to regular storage but is faster. This results in CPU placing often used information in the RAM so it can get the information faster than getting it from the storage. This is similar to how the accountant can keep important documents on their desk so that they do not need to stand up and get a new document from the filing cabinet.

2.2.2 Data Interchange Formats

There are many data interchange formats. Some of the most common ones are XML, JSON, BSON, and CSV. These formats were developed for different purposes. Thus, the performance of a program implementing a format depends on the properties of the data.

Before storing the formatted data in a file or transferring it using an ICP solution, it needs to be serialised. Serialisation is the process of converting the data in the programming language to the standardised structure of the format. To then use the variables stored in the serialised structure, it must first be deserialised. Deserialisation simply returns the stored data back to its original state. Serialisation and deserialisation are sometimes referred to as encoding and decoding, respectively.

Extensible Markup Language

XML is one of the most commonly used text-based formats for storing structured data (World Wide Web Consortium, 2019). XML is designed to be readable by both humans and machines. Figure 3 shows a simple example of the XML syntax.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <car>
3       <brand>Volvo</brand>
4       <body_type>SUV</body_type>
5       <color>Black</color>
6       <year>2018</year>
7   </car>
```

Figure 3. XML syntax

The data in an XML file consists of elements. Each element has a start and an end tag, such as the tag “brand” in Figure 3. Between these tags, goes either a string of alphanumeric characters, or other elements. This type of nesting allows for representing more complex data structures, such as trees.

JavaScript Object Notation

JSON is a text-based data-interchange format (JSON, n.d.). It is, like XML, designed to be readable by both humans and computers. The structure of a JSON file consists of objects called attribute-value pairs, where the value can be either a string, number, boolean, array or another JSON object. The ability to nest objects allows for the creation of a complex structure for data storage. Figure 4 shows a simple example of the JSON syntax.

Figure 4. JSON syntax

Binary JavaScript Object Notation

BSON is a binary file format, meaning that the contents of the file are stored as ones and zeros. BSON files are thus readable to a computer, but not to a human. Other than being a binary file format, BSON is similar to JSON in many ways. Some key differences though, are that BSON supports a few more data types than JSON, it requires additional overhead, and it can traverse the content more easily as the data is indexed (MongoDB, n.d.).

Comma Separated Values

CSV is a text-based format for storing values in a structured way. The values are stored in plain text, separated by commas (Shafranovich, 2005), thus making it human-readable. Due to this simple structure, CSV files are very space efficient. However, the simple structure also limits the complexity of the data stored. Figure 5 shows a simple example of the CSV syntax.

```
1    brand,body type,color,year
2    Volvo,SUV,Black,2018
```

Figure 5. CSV syntax

Notice how there is no “car” element in the CSV example shown in Figure 5, as there is for the other examples. This is because the limited complexity does not allow for storing hierarchical data. Thus there is no way to show that the attributes of the car belong to a specific car object.

2.2.3 Inter-process Communication

IPC allows two or multiple running processes on a computer to share data. The sharing of data allows processes to cooperate and it can lead to a lower need for copying data, increase calculation speed, and create a modular system (Silberschatz, Galvin, and Gagne, 2018). To share the data between multiple processes there are two main models and those are shared memory and message passing (Silberschatz, Galvin, and Gagne, 2018). The main models are depicted in Figure 6.

There are differences between shared memory and message passing in how they allow processes to access the data. The shared memory model uses a memory space on the computer that multiple processes can access and that is how they share data. This is seen in Figure 6 where the two processes look at a memory segment that is between the processes. Message passing works by having processes send data as messages between the processes. This is depicted in Figure 6 where the messages share a queue and can access the short messages numbered M1 to Mn. For those different models, there are different ways of implementing them and some of those will be described in Section 2.3.

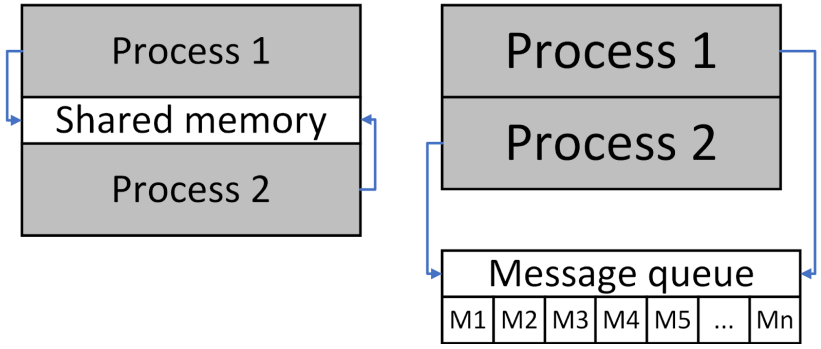

Figure 6. Shared memory (left) and message passing (right)

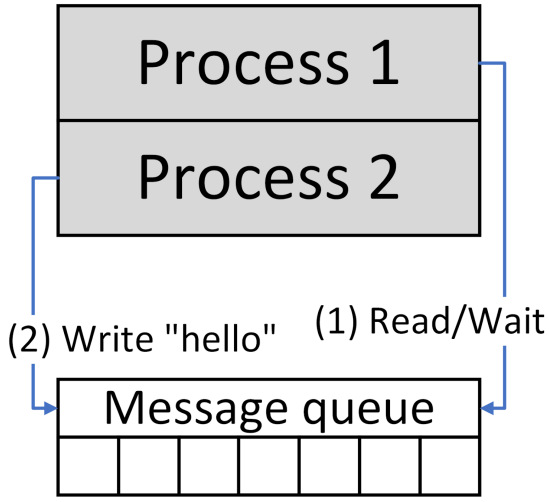

Figure 7. Two processes communicating through synchronous message passing

When it comes to message passing there is the problem of synchronisation and whether they are synchronous or asynchronous, also known as blocking and non-blocking (Silberschatz, Galvin, and Gagne, 2018). The difference between them is that one requires some waiting whilst the other does not. Synchronous is the one that requires waiting and that is both for sending and receiving messages. So if a message is sent the process will wait until the message has been received. If receiving a message then the process will wait until it can read a message. An example of synchronous message passing is illustrated in Figure 7, where process 1 will wait for a message until process 2 writes hello. Asynchronous does not require any waiting for sending or retrieving messages. So when sending a message it will simply send the message and resume other operations, without checking if the message has been received. Then when reading a message it will try to read and if there is nothing to read it will simply read nothing and resume other operations. If the message passing in Figure 7 is asynchronous and process 1 reads first, then it will not wait for process 2 to write hello. This results in process 1 reading nothing.

2.3 Existing Solutions

Multiple IPC solutions are used and available on different operating systems. Therefore this section describes some of the solutions for IPC. Section 2.3.1 describes the more common packages of IPC solutions whilst Sections 2.3.2-2.3.4 describe types of solutions.

2.3.1 Packages

One family of solutions are from the portable operating system interface (POSIX). POSIX works on most operating systems. This is not applicable for Windows as they only use POSIX in C libraries. The thing with POSIX is that it works with operating systems and is used in C and C++ languages. The POSIX library in C allows the software developer to use methods such as Shared memory, message queues, pipes, and a specific type of socket called Unix sockets.

When wanting to use the IPC solutions on a Windows system a software developer can use the Windows Application program Interface (API). This allows for setting up IPC solutions, but it is not something that is for certain programming languages. One language that allows for the use of Windows API is the C programming language.

2.3.2 Shared Memory

POSIX has a command that is called shm_open that creates a shared memory space that multiple processes can utilise (Kerrisk, 2010). In addition to this, there is a POSIX command called mapped memory file and that allows the process to access the shared memory as if it was a file. This allows the processes to easily read and write data to the so-called file to be able to transfer the data between processes.

2.3.3 Message Queues

POSIX has a command that is called mq_open that creates a message queue that multiple processes can utilise (Kerrisk, 2010). To that the users can specify what the maximum message size can be. The maximum default message size in the Linux operating system is 8 kb (Mq_overview, 2023). To read and write messages to the queue one can simply read and write with mq_send() and mq_recieve. A process can access the message queue as long as it knows the name of the queue.

2.3.4 Pipes

The pipe solution follows the message passing model and is one of the POSIX solutions. There are commonly two types of pipes: ordinary pipes, also called anonymous pipes, and named pipes (Silberschatz, Galvin, and Gagne, 2018). An ordinary pipe allows for two processes to send messages in a one-way system so one process sends and one receives. If a user wants to have the two

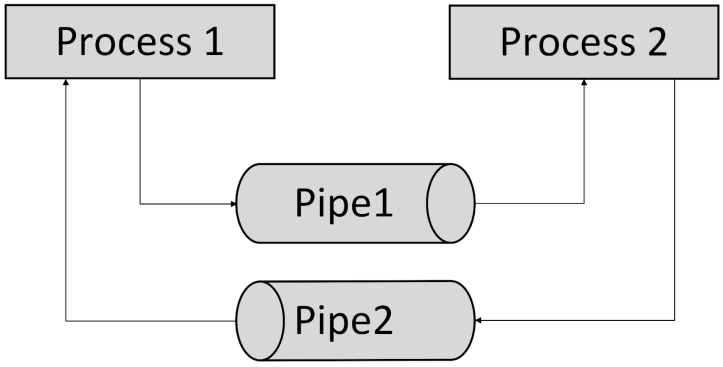

Figure 8. Two processes communicating through two pipes

processes send messages back and forth, then there needs to be two pipes. One pipe goes from Process One to Process Two and the other goes from Process Two to Process One. This is depicted in Figure 8 where Pipe 1 is for messages from Process 1 to Process 2. Then Pipe 2 is the other way around.

There is a problem with ordinary pipes and that is that they only work for a process that was created by another process. This means that everything needs to be in the same program and that does not allow for different programs to communicate. The solution to this problem is the pipes called named pipes.

Named pipes or also referred to as FIFOs in the UNIX operating systems are pipes that look like typical files (Silberschatz, Galvin, and Gagne, 2018). The FIFO solutions allow multiple processes to use the pipe to transfer data and they allow for processes from different programs to use them. The pipes will also remain after the processes stop using them, as opposed to ordinary pipes that are removed once no processes are using them. Then the FIFOs also allow for processes to read and write from the same pipe but there can only be one direction at a time. This means that if one wants to have processes send and receive at the same time there should be two FIFO pipes created.

2.3.5 Sockets

The way to allow computers to communicate over a network is by the use of sockets. There are standards for sockets and those are called protocols and they describe how the data transfer is supposed to happen. This allows for information to be sent over a network to the right computer, and to the right process of the receiving computer. There can however be differences between programming languages for how the data is sent, so that is something to keep in mind.

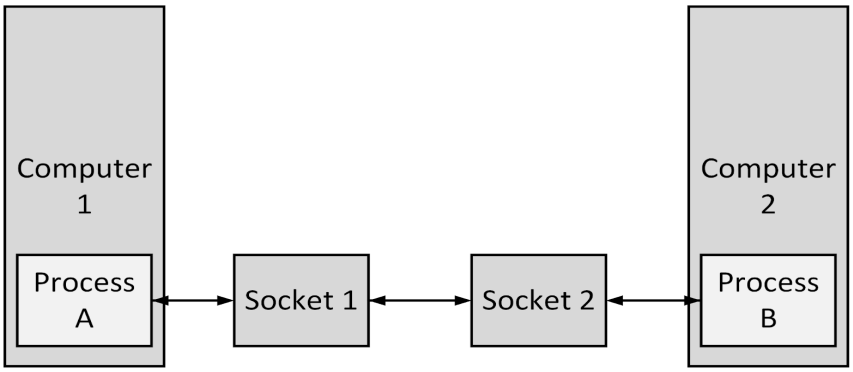

Figure 9. Two processes on different computers communicating through sockets

Figure 9 depicts how sockets work for communication between processes on different computers. In Figure 9 Process A exchanges data with Socket 1 which is attached to Computer 1. This is the same for Process B which exchanges data with socket 2 which is attached to Computer 2. Then the communication can be exchanged through the sockets so that data from Process B can be received by Process A.

Regular sockets can be used to send data between processes on the same computer. Therefore it works more like other IPC solutions. This is achieved by having the socket for each process and sending data between those sockets. This is illustrated in Figure 10. Figure 10 depicts how Process A can exchange data with Process B by the use of their Socket 1 and 2.

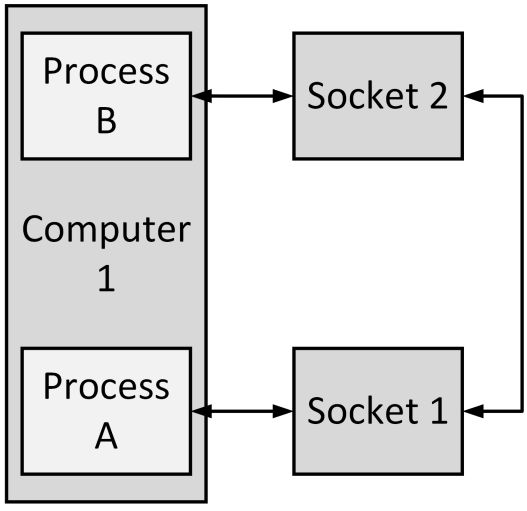

Figure 10. Two processes on the same computer communicate through a socket network.

Regular sockets work for transfers over both a network and in a computer. There is also a solution called Unix domain sockets which only allows for transfers within a single computer. Unix domain sockets work similar to a regular socket but instead of binding a socket to an address then a socket is bound to a file name. The file name can be a to a name that does not appear in the file system (Kerrisk, 2010).

2.4 Related Work

When investigating data interchange formats, a journal article similar to the thesis was found: *A Literature Review on Device-to-Device Data Exchange Formats for IoT Applications*, by Kaur, Ayyagari, Mishra, and Thukral (2020). The article compares ten data formats, including the ones covered in this thesis. The article references nine sources including theses, journal articles, and conference proceedings, that have evaluated and compared different combinations of the introduced data formats. For each source, relevant results are brought up and a conclusion regarding the involved formats are drawn. Finally the article summarises the strengths, weaknesses, and suitable applications for all ten formats.

One report that is related to the subject of inter-process communication is a report by Zoran Spasov and Ana Madevska Bogdanova (2010). The study was called *Inter-process communication, analysis, guidelines and its impact on computer security*. The report is about the security issues of IPC solutions, both in general and for specific solutions. The solutions are only for windows programs and regards named pipes, ordinary pipes, shared memory, microsoft message queues, and microsoft remoting. The report notes on the solutions properties and their strengths and weaknesses in regards to security. It also provides some suggestions on when to use a certain method. The problem with the source is that the recommendations are all in written flowing text, which makes it hard to find certain suggestions. The report does not give recommendations depending on what properties that a developer wants from their solution.

There is an ebook by Marty Kalin that was called *A guide to inter-process communication in Linux* (n.d). The ebook included information on shared memory, pipes, message queues, sockets, and signals. The information is both how the solutions work, how to implement the solutions, and a short summary of their uses. The ebook comes with suggestions of when to use certain solutions and when not to use them. The ebook notes that shared memory would be suitable for smaller data stream sizes and not suitable for larger sizes. For pipes and message queues it notes that those are simple and easy to use solutions. For sockets and signals there were no suggestions of when to use them. Then lastly the report noted some overall suggestions and it was

repetitive as it does not recommend shared memory for larger data and recommends pipes or sockets instead. The report also brings up that no solution is the perfect solution and each has their use cases with a trade off for performance and simplicity.

3. Research Method

Chapter 3 presents the research methodology that was followed during the thesis. Section 3.1 describes the overall research strategy that was followed. The rest of the sections describe the different parts of the research strategy in more depth. The research phase is described in Section 3.2 and lists the different stages in the research process. Section 3.3 presents and motivates the research methods that were used. Section 3.4 describes the comparison model used during the research. Section 3.5 describes the instruments that were used to collect and evaluate data. Lastly Section 3.6 presents potential threats to the validity and 3.7 describes ethical requirements.

3.1 Research Strategy

The research is extensive so there was a need for a methodical way to find and compile information. This methodology was also needed as the information on the subject was spread out over many sources. To help the gathering of information efficiently, a research strategy was chosen. This was to gather as much information as possible within the limited time frame of the project.

Table 1. Overview of the research strategy

Research phases	Research methods	Research instruments	Validity threats	Ethical requirements
4 main phases: Pre-study Creation of comparison model Creation of guidelines Finalisation	Qualitative and Comparative	Literature study Comparison model Github Visual Studio Code Oracle VirtualBox Google scholar Tests	Credibility Transferability Dependability Confirmability	Information Consent Confidentiality Usufruct

Table 1 illustrates the research strategy that was used in the thesis. The research strategy consisted of six components as can be seen in Table 1. The components were the *research phase, research methods, research instruments, respondents, validity threats* and *ethical requirements*. All components are described in the coming sections, in the order that they are presented in Table 1.

3.2 Research Phases

The research phases present the different parts of the research process in this thesis. Each phase corresponds to an activity that was adopted to follow a structured approach to the thesis. The research phases and how they are connected are presented in Figure 11. Figure 11 depicts all of the phases: pre-study including literature study and practical study, creation of the comparison model, implementation including the creation and improvements of the guidelines, and lastly finalisation and evaluation of the guidelines.

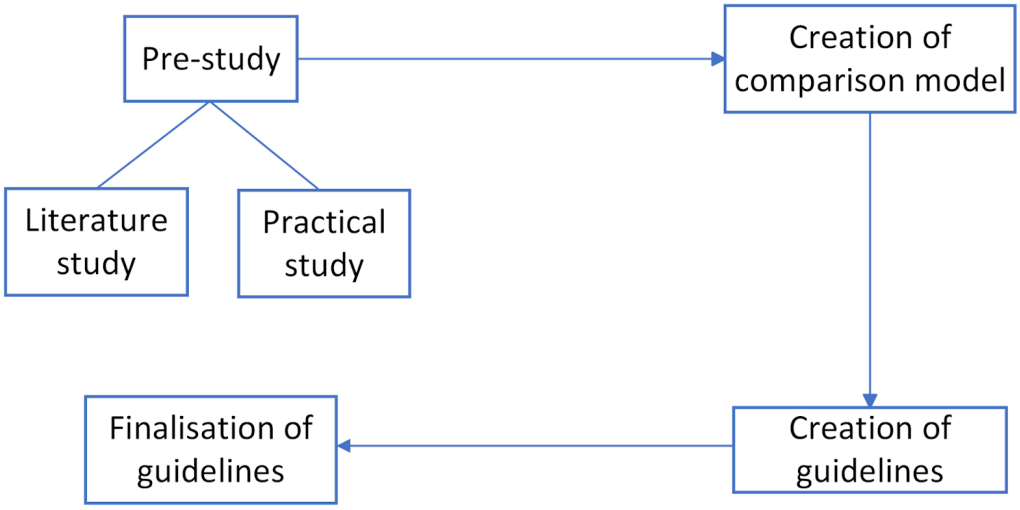

Figure 11. The research phases of the thesis

3.2.1 Pre-study

The goal of the pre-study phase was to gather information on the area and potentially find existing guidelines. The pre-study was conducted in two steps. The first was a literature study where information was searched on the internet through different search engines. Then the practical study was carried out, in which our tests were conducted on the data transfer solutions. This was done in order to gather further information and complement the information

found in the literature study. Tests for IPC solutions and data interchange formats were conducted independently of each other.

The way that literature study was performed was by the use of search engines such as Google and Google Scholar. Google Scholar is a search engine where the results primarily contain scientific articles and journals with the searched keywords. This was used to find information on specific areas and get technical specifications. The Google search engine results in more general information such as new articles, online discussions and websites. The use of Google was to find more general information such as online coding platforms to understand concepts, partially through examples.

With both search engines, keywords were used and they were words such as “data transfer”, “data formats”, “guidelines”, “IPC” and “XML”. These words gather the necessary information and possibly find other guidelines. The searches resulted in information both on different solutions, and how the data can be structured. It provided a good understanding of keywords that would later be used in the implementation phase to create the guidelines. The information also provided information on key aspects of the solutions that were important so that those aspects could be provided through criteria in the comparison model.

3.2.2 Creation of Comparison Model

The goal of the *creation of the comparison model* phase was to use the information from the pre-study to create a comparison model. The comparison model was then used in the *implementation* phase to compare the different solutions to one another. Therefore the comparison model needed to include criteria that were seen as important for the different solutions.

The comparison model was created to have three different criteria for the solutions. The criteria were *speed of transfer*, *resource usage* and *language support*. The criteria were then used to compare the different solutions with the information that was gathered in the *pre-study* phase. The result of the comparisons was then used for the *implementation* phase when creating the guidelines. The comparison model is further explained in Section 3.4.

3.2.3 Creation of guidelines

The *creation of guidelines* phase involved taking the information from the pre-study to gather the results of the solutions. The results were a collection of information that regarded the different criteria that were presented in the comparison model. From those results, a guideline was created that was meant to provide information and suggestions for the different solutions. After the

guidelines were created they were the input for the next phase *finalisation of guidelines*.

3.2.4 Finalisation of guidelines

The last phase was the *finalisation of guidelines* phase and it included no sub-phases. This phase would result in the final guidelines that were created from the results provided from the *creation of guidelines* phase. Once the final guidelines were created then a final correction was done. The final corrections were to correct small errors that occurred throughout the *creation of guidelines* phase. The small corrections could be things such as grammatical errors or odd phrasing.

3.3 Research Methods

Qualitative research is used to gather non-numerical data, such as case studies and observations, in order to understand the experiences and attitudes of individual people or groups (Bhandari, 2023). The qualitative research type was chosen for this thesis, since the aim was to discover new insights, through reviewing case studies and gathering measurements of our own.

Comparative research is a method for comparing related things, based on some criteria. The comparison of entities allows for similarities and differences to be discovered and further studied. Comparative research was chosen as an appropriate method because the research phase included many types of data transfer methods that needed to be compared. This comparison was conducted in accordance with the criteria presented in Section 3.4.

Quantitative research, unlike qualitative, uses numerical data to gain a concrete and objective answer to a specific question. The numerical data would also be used to reach a generalised answer for a larger population. This makes it suitable for doing experiments on a subject group of the populace. This thesis works with gaining in-depth knowledge on the subject of data transfer solutions and is therefore not giving a concrete answer to a concrete question. This made it not suitable to use a quantitative research method for the thesis.

3.4 Comparison Model

Section 3.4.1 introduces the comparison model and its criteria. Each criterion is first described in detail and how it was measured. The purpose of the

criterion is motivated by explaining what the expected result is, and how that contributes to the final results. Section 3.4.2-3.4.4 describes the three criteria: *speed*, *resource usage* and *language support*, respectively.

3.4.1 Introduction

The comparison model consisted of 3 comparison criteria. The 3 criteria were *speed*, *resource usage*, and *language support*. The reason for the criteria was to evaluate and compare the different solutions to each other. Thus, the criteria needed to be easy to measure and to be used in a comparison between the solutions. Although the main comparison criteria were the same for IPC methods and formats, their inherent differences required minor alterations to the metrics used.

Since different data structures and sizes of data affect the performance, different scenarios were tested. The different scenarios were chosen to represent real-life situations. For example, one situation includes transferring a small amount of data in a nested structure, which could represent a client sending a request to a server for some information. In another example, a large amount of data in a simple tabular format is transferred, possibly representing the server responding to the client by sending many rows from a database. For the testing, a combination of data structure complexity and data size will be evaluated.

3.4.2 Speed

The first criterion was *speed*. For the IPC methods, this refers to the speed of transfer of data. That is, how fast data can be transferred from one process to another. For the formats, the speed criterion refers to the serialisation or deserialisation time. That is, the time it takes to convert an object in the programming language to the specified format, or vice versa.

The reason why the speed was included was that different solutions had different transfer speeds and that could be impactful for a user. So for the criterion, it was argued that a higher speed was better for most situations. This criterion was expanded upon with how the speed was affected by the size of the data. This was due to how the speed of the solutions could depend on if the data was small or large. The metric for this criterion when comparing IPC methods, was the time it took to transfer data of a predetermined size, with the base unit seconds. For formats, the metric was the time it took to serialise and deserialise data of a predetermined size, with the base unit seconds.

3.4.3 Resource Usage

Resource usage was the second criterion of the comparison model. It aimed to measure the amount of resources used on the machine. For the IPC methods, this refers to the resources used while transferring the data. For the formats, it refers to the resources used when data is serialised or deserialised.

The specific resources that were measured for IPC methods were: *processor usage* and *memory usage*. These resources were measured with the metrics: CPU time and number of bytes allocated for the process, respectively.

When comparing the data interchange formats, the specific resources measured were: *processor usage*, *memory usage*, and *space requirement*. The first two resources were measured the same way as with IPC solutions. The third resource, space requirement, was defined as the size of the serialised format, in bytes. The space requirement is interesting as it directly determines the size of data, and thus affects the transfer time, since more data needs to be transferred.

3.4.4 Language Support

To make the IPC method or format easy to implement it is preferable if a language has official libraries and support for easier use. If there are libraries for the solutions then the user can simply use them instead of trying to create support for the solutions themselves. This helps save development time and makes it easier to work with the methods and formats.

Here we chose to look at some of the most popular programming languages, and for each IPC method and format, check if the programming language provides official, external, or no library support. The included programming languages were: C, C++, C#, Python, Java, Javascript, Rust, and Ruby. The C language provides information on a low-level language. C++ and C# are included to provide information on compiled and object-oriented languages. Java and Python are included as they are high-level interpreted languages. Javascript is included as it provides information for a high-level language that is just-in-time compiled. Rust was included as it is a general-purpose language that is relatively new as the official version came out in 2015 (*Announcing Rust 1.0 | Rust Blog*, 2015). Lastly Ruby was included as it was designed with programming productivity in mind.

3.5 Research Instruments

The instruments used in the thesis were a *Literature study*, *Comparison model,* and *Tests*. Then there were tools that were used during the thesis and such tools were GitHub, Visual Studio Code and Oracle VirtualBox. What the instruments were and why they were used is explained in the following points:

- ***Literature study:*** this was an essential part of the research as most of the information on how the solutions worked came from it. Also due to the time limitation of the project, not all tests could be done by us. That meant that many measurements came from existing related work.
- ***Comparison model:*** was used when comparing the different solutions to one another. This was essential to the project as the guidelines relied on there being comparisons between solutions. This was required to understand which solution was best for a situation. The comparison model is explained in Section 3.4.
- ***Tests:*** were used to gain measurements in the practical study. There were two types of test for IPC solutions, which were large transfers and small transfers. Large transfers involved sending the contents of a file from one process to another. The file sizes ranged from 1 MB to 1 GB. The small transfers regarded sending a message from a process to another and then back again. The small message size ranged from 1 kB to 512 kB. The small test represents a system that needs messaging between processes and the larger test is when multiple processes work with files. For the data interchange formats, tests were conducted with varying size and structure of the data. These tests were conducted in both JavaScript and Python. The structure and results of the tests are described in detail in Chapter 4.
- ***GitHub:*** is a tool that allows for the organisation and sharing of code. The tool is used in the practical study phase to organise the tests and share the test programs over multiple computers. It helps with saving the information in an easy-to-find location (*GitHub: Let's Build From Here*, n.d.).
- ***Visual Studio Code:*** is a computer program that allows for editing and writing programs. The program allows for the creation of tests in different languages and the ability to execute them (*Visual Studio Code - Code Editing. Redefined*, 2021).
- ***Oracle VirtualBox:*** is a tool that allows for virtualization on a computer. What virtualization does is it allows for a separate machine or a VM to run on a regular machine. This VM does not need to be of the same operating system as the original machine. This allows for tests

to be both run on Windows and Linux systems on the same machine (*Oracle VM VirtualBox*, n.d.).

- ***Google scholar:*** is a search engine that provides sources from journals, books, reports, and more. The search engine allows for the efficient search of scientific papers within a subject. When searching for information, certain key phrases were used such as *IPC, data interchange formats,* and *resource usage* (*Google Scholar*, n.d.).

3.6 Validity Threats

The criteria for validity threats are used to test the strength and soundness of a research method. Therefore they are presented in this section to test the chosen research method. The validity threats for qualitative research are *credibility*, *transferability*, *dependability* and *confirmability* (Shenton, 2004). All of the criteria are further explained in the following points and how they were addressed:

- **Credibility:** Credibility deals with the trustworthiness of the project and the claims that are made. It is important that when stating that x leads to y that it is credible and well-founded. That means that there needs to be trust in the results of the final guidelines.

- **Transferability:** This threat is in accordance with how generalised the results are to other situations. The thesis aims to create guidelines that could help in multiple situations and therefore the results should be generalised.

- **Dependability:** Dependability regards if the results would be replicable if the research was done again and in a different context. It can be hard to provide dependability with a qualitative research method as the process can heavily depend on context.

- **Confirmability:** This threat is the potential for the research to be biassed. This confirmability issue can come from the writer's bias affecting the study.

3.7 Ethical Requirements

With any type of research, it is important that the ethical part is accounted for. To account for the ethical part there were some requirements that needed to be followed in a qualitative research method. The four requirements for

qualitative research were *information requirement*, *consent requirement*, *confidential requirement* and *usufruct* (Vetenskapsrådet, 2002). A description of the requirements is provided below:

- **Information requirement:** This requirement was to ensure that all parties in the research were informed about the purpose and participation. It was important that those that participated in the research understood the purpose of the research and understood their right to participation. This was ensured by providing the participants notice that they could resign their participation at any point. If the participants were to resign after their provided information was used, but before publication, then the provided information was not used. It was also provided by notifying the participants of the research and its purpose before beginning to work together.

- **Consent requirement:** This regards well-founded for the participants to know about their rights for participating and their consent. It was to ensure that the participants were willing to work together and not be forced. This requirement was met by receiving verbal confirmation from participants that they were willing to participate. It was also provided by ensuring the participants that they were free to resign at any point before publication.

- **Confidential requirement:** The confidential requirement was in regards to allowing the participants to remain anonymous. To provide confidentiality the names and workplaces of the participants were not disclosed. This was further extended by asking if the participants wanted further anonymity and to provide it.

- **Usufruct:** This requirement states that the results attained should not be used for anything other than the thesis' predetermined purposes. The specific purpose of this thesis was to create guidelines for data transfer. The results from the literary study, and practical study were used for this purpose and this purpose only.

4. Pre-study Results

The first phase of the research strategy was the pre-study and which contained the literature study and practical study. The results from the pre-study are presented in different sections. Sections 4.1 and 4.2 introduces the results by presenting IPC and data interchange formats results respectively. The results from Sections 4.1 and 4.2 are from both literature studies and practical studies.

4.1 IPCs

Section 4.1 presents the results of both the literature and the practical study. For each criterion of the comparison model, there is a subsection where the results of each criterion are presented. Subsection 4.1.1 presents the comparison of the different solutions in regard to their speed of transfer. Subsection 4.1.2 presents the solutions in regard to their resource usage. Subsection 4.1.3 presents the language support of the solutions.

4.1.1 Speed of Transfer

From the literature study three articles were found. These three articles note the transfer speeds of different IPC solutions. The first article notes the transfer speeds of pipes, shared memory, and sockets (Venkataraman and Jagadeesha, 2015). The second article notes the transfer speeds of message queues, pipes, shared memory, and sockets (Smith and Wells, 2017). The last article compares the transfer speeds of pipes, Unix sockets, and regular sockets (Wright, Gopalan, and Kang, 2007).

Literature Study

To present the results from the literature study the maximum speeds of the IPC solutions are presented. The maximum speed was the highest speed that was recorded in an article. The articles were done on different machines and therefore the speeds were different between articles. The maximum speeds were presented in Table 2.

Table 2 depicts the results from the literature study of transfer speeds of IPC solutions. The table contains the maximum speeds that were found for the solutions in Megabytes per second. The table, therefore, notes the solution, the speed of the solution and the source where the data came from.

Table 2. Maximum transfer speeds for IPC solutions

	Speed (MB/s)	Source
FIFO Pipes	3,000	(Venkataraman and Jagadeesha, 2015)
Socket	1,700	(Venkataraman and Jagadeesha, 2015)
Shared memory	6,500	(Venkataraman and Jagadeesha, 2015)
Message queues	567	(Smith and Wells, 2017)
Unix Socket	1,500	(Wright, Gopalan, and Kang, 2007)

The article by Aditya Venkataraman and Kishore Kumar Jagadeesha tested the IPC solutions of pipes, sockets and shared memory (2015). Therefore they gave no transfer speeds for message queues or Unix Sockets. The article by Kwame Wright, Kartik Gopalan and Hui Kang did a study on pipes, Unix sockets and regular sockets (2007). The results from the article were the Unix sockets had the best transfer speeds whilst pipes were second with around 40% of the top-speed Unix sockets. The last was the regular sockets with 28% of the Unix max speed. The source was the only source found that tested Unix sockets.

The result of the message queues came from an article by Dylan Smith and George Wells (2017). The article was tested on a Windows system. To use the libraries in the article they created their own libraries with the use of C code. This resulted in the fastest IPC solution being the message queues. After that came the pipes which had a speed that was 68% of the message queues. Next was a solution called file mapping which is very similar to shared memory and it was 30% the speed of message queues. Lastly was Java's implementation of sockets and it was 1.2% of message queues. The authors noted that this was due to the larger overhead of the messages. The messages had a larger overhead due to the protocols involved with socket transfers.

Practical Study

After the literature study came the results from the practical study and those results are shown in Table 4. The table describes the results from a program that consisted of two processes. One process read a file with a specific buffer size and sent it to the other process with the use of the IPC solution. The second process read the data and then wrote the data into a new file. This created a program that would copy the contents of a file. The tests were done with different sizes of files and with different buffer sizes for the solutions. The file sizes ranged from 1 MB to 5 GB and the buffer was 16 kB, 64 kB, and 256 kB. There was a problem with the message queue. The problem was that message queues had a maximum buffer size of 8 kB.

The results from the practical study of IPC solutions all came from a VM running Ubuntu on a Windows machine. The original windows machine was running a 6-core CPU and 16 GB of RAM. The allocated resources of the Ubuntu VM are noted in Table 3. The VM was allocated 4 of the 6 available cores and 5 GB of RAM.

There was a test that was for IPC solutions when sending a shorter message and getting it back. So Process 1 sent a message to Process 2 and then Process 2 sent a new message to Process 1. The results from the test are noted in Figure 12. The results in Figure 12 display the transfer speeds of the IPC solutions in MB/s with different message sizes. The vertical axis is organised in a logarithmic scale whilst the horizontal axis is in accordance with the message size. Figure 12 showed that pipes were the best for message sizes 1 kB, 4 kB, 8 kB, 16 kB, and 32 kB. The graph in Figure 12 presented that sockets provided the fastest speed of transfer for messages of 2 kB. Lastly was shared memory which was the fastest solution for message sizes 64 kB, 256 kB, and 512 kB.

Table 3. Specification of VM

Number of cores	4
CPU frequency	3.6 Ghz
RAM	5 GB

Table 4. Maximum transfer speeds for IPC solutions when sending contents of a file

	Speed (MB/s)	File (MB)	Buffer (kB)
FIFO Pipes	587	64	64
Socket	1,476	16	256
Shared memory	574	64	256
Message queues	378	64	16
Unix Socket	918	16	256

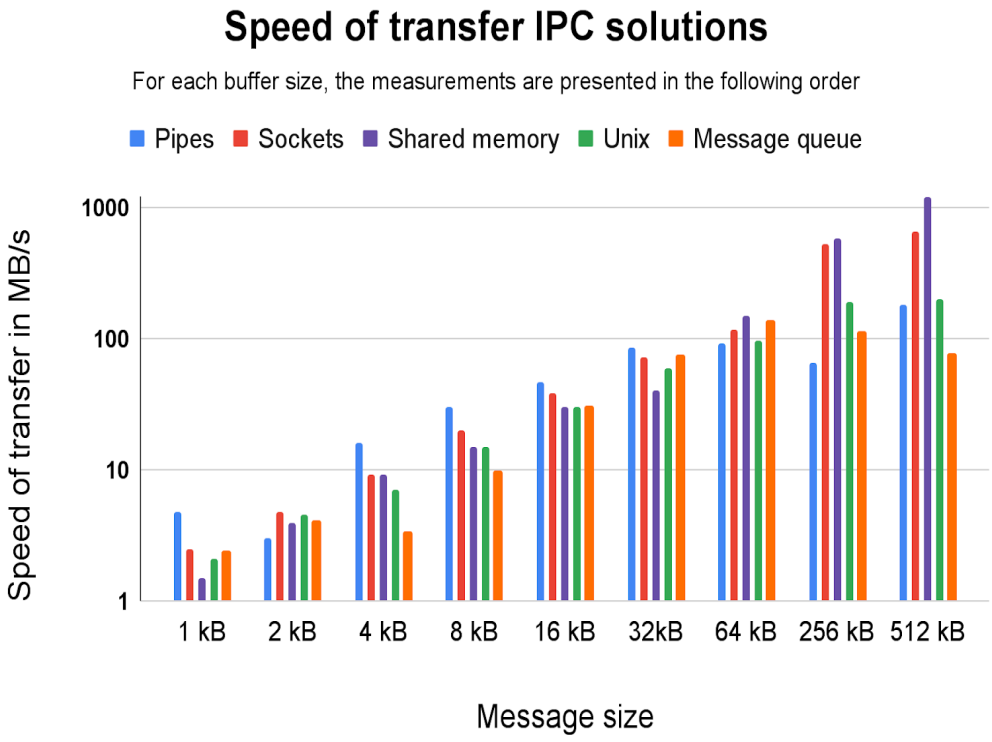

Figure 12. Transfer speeds for IPC solutions when sending one message

4.1.2 Resource Usage

During the literature study no sources on the resource usage of the solutions were found. Therefore only tests from the practical study noted the resource usage of the IPC solutions. The resource usage that was measured was the amount of memory used in the RAM and the amount of CPU time used. The result of the memory usage is displayed in Table 5.

The test for Table 5 was to send information from a file to another process that wrote it into a new file, similar to Table 4. The tests were done with file sizes ranging from a 1 MB file to a 5 GB file. Table 5 shows the average memory usage, by the process and shared memory, over four different file sizes for the solutions for specific buffer sizes. The average was from running the test 20 times. The buffer sizes were 16 kB, 64 kB and 256 kB.

The result from Table 5 showed that pipes had the lowest memory usage when working with smaller files. Then for files larger than 16 MB message queues took over the lowest memory usage. The worst solution was the regular sockets and it was the worst for all file sizes. Shared memory had the second lowest memory usage at 1 MB files but is second worst or worst for other sizes. The solution in the middle was Unix sockets which were often the second worst or the middle solution in memory usage.

Table 5. Average memory usage in megabytes of IPC solution depending on file size

	1 MB	**16 MB**	**64 MB**	**1 GB**
FIFO Pipes	0.44	4.47	4.51	4.44
Socket	4.81	4.81	4.92	4.92
Shared memory	2.05	4.82	4.79	4.77
Message queues	4.04	4.04	4.03	4.08
Unix Socket	4.75	4.75	4.75	4.77

Table 6. Average CPU usage time in milliseconds of IPC solution depending on file size

	1 MB	16 MB	64 MB	1 GB	5 GB
FIFO Pipes	4	39	145	7133	44007
Socket	0	1	47	1612	3717
Shared memory	3	42	352	5160	15728
Message queues	1	24	109	2593	7083
Unix Socket	0	40	161	3563	7909

The second test was the measure of CPU usage and it is portrayed in Tables 6 and 7. The tables were for the same tests as with memory usage in Table 5 where the contents of a file were transferred. The tests were for the data sizes 1 MB, 16 MB, 64 MB, 1 GB, and 5 GB. All of these tests were also with different buffer sizes of 16 kB, 64 kB and 256 kB. The results were the average from running the tests 10 times.

Table 7. Average CPU usage time in milliseconds of IPC solution depending on buffer size

	16 kB	64 kB	256 kB
FIFO Pipes	7262	12404	11128
Socket	1200	988	1039
Shared memory	8366	3157	1248
Message queues	2577	2229	2097
Unix Socket	3046	1568	1271

The result in Table 6 depicts the average CPU usage of the two processes in CPU time of milliseconds. Some sizes were noted as zero due to the monitoring program not being able to measure lower times. The result in Table 6 was the usage when copying files of different sizes. The averages were the averages from each buffer size that could be seen in Table 7.

The results in Table 7 shows the average CPU usage time depending on the buffer size for the solution, in milliseconds. The table depicted the buffer sizes in the upper horizontal line and the solutions in a vertical line. These results were from all of the file sizes that could be seen in Table 6.

The results of Table 6 show that both pipes and shared memory used the most CPU time when working with all file sizes. The shared memory was worse for sizes 16 MB and 64 MB. Pipes were the worst for the rest of the file sizes. The solution that had the least CPU usage time was sockets and it was for all file sizes. The second best was message queues and the third best was Unix domain sockets.

The results from Table 7 showed that sockets had the lowest CPU usage for all buffer sizes. Then shared memory had the most for 16 kB buffer and pipes had the most for the rest of the buffer sizes. Unix domain sockets were the third best solution for every data size, except for 64 kB. For message queues it was shown that for buffer sizes of 16 kB, it had the second lowest usage time. The message queues had the third lowest CPU time usage when the buffer size was 64 kB. For the buffer size of 256 kB it is shown in Table 7 that the message queues had the second highest CPU usage time.

4.1.3 Language Support

The language support of each of the solutions is presented in Table 8. The table has three text types: **bold**, *italic* and empty. The different text types represent what kind of library the solution needed. If the test is bold, that was an official library for the programming language. The italic text represents an external library that a user can add to their project. The empty text boxes represent that no library was available for a solution in a given programming language. There are some notes in Table 8. The notes are explained after the table. The notes regard the definition of what is an official library for solutions. The first note is about the problem of an official library being available on one operating system but not another. The second note regards the definition of whether a library is not strictly called the same as the IPC solution. But the library provides the same functionality as the IPC solution.

Table 8. Language support for IPC solutions

	FIFO pipes	Sockets	Shared memory	Message queues	Unix Sockets
C	**mkfifo**	**socket.h**	**shm.h**	**mqueue.h**	**socket.h**
C++	**mkfifo**	**socket.h**	**shm.h**	**mqueue.h**	**socket.h**
C#	*Mono.Posix.NETStandard*	**System.Net.Sockets**	**System.IO.MemoryMappedFiles**	**System.Messaging** **(Note 1)**	**System.Net.Socket**
Python	**os**	**socket**	*posix-ipc*	*posix-ipc*	**socket**
Java		**java.net.Socket**	**Java.nio (Note 2)**		**java.net.UnixDomainSocketAddres**
Javascript	*fifo-js*	**WebSocket**		*PosixMQ*	*unix-dgram-socket*
Rust	*nix*	**std::net**	*nix*	*IPC-Channel*	**std::os::unix::net**
Ruby	**FileUtils**	**socket**	*sysvipc*	*sysvipc*	**socket**

- *Note 1*: The problem with message queues for C# was that it did have an official library for message queues on Windows, but not on Linux. To be able to use message queues on Linux then one needs to either use a library called RabbitMQ or create a library. The RabbitMQ library allows for message queues that are based on sockets and may have complications when trying to access them without the RabbitMQ library. The final option was to create a new library that relies on C code to access the message queue system by the Linux operating system.
- *Note 2*: The note about the Java shared memory is that it is not called shared memory. With Java three libraries from java.nio can be used and it will act like shared memory. If the users want to use the shared memory that the operating systems provide then one needs to create one's own library, which can be done in C.

4.2 Formats

Section 4.2 presents results from both literature and practical study, regarding the compared data interchange formats. Each subsection corresponds to a criterion from the comparison model. Sections 4.2.1-4.2.3 present the criteria for serialisation and deserialisation time, resource usage, and language support respectively. Sections 4.2.1 and 4.2.2 begin by presenting the results of the literature study, which is then followed by results from the practical study.

4.2.1 Serialisation and Deserialisation Time

In total, three relevant articles and two web pages were found on the topic of time comparisons between data interchange formats. Articles by Šimec and Magličić (2014) and Nurseitov, Paulson, Reynolds, and Izurieta (2009) included JSON and XML performance comparisons. Both articles find that JSON is significantly faster than XML. The third article by Vahdati, Karim, Huang, and Lange (2015) compares XML and CSV, and finds that CSV is faster. The web pages, (*JSON vs BSON*, n.d.) and (GeeksForGeeks, 2023), both note BSON as being faster than JSON.

Literature Study

The article by Šimec and Magličić (2014) found that decoding a simple data example in PHP was more than four times faster with JSON compared to XML. Although JSON significantly outperformed XML in terms of speed, it was noted in the conclusion that XML is still a valuable format thanks to its broader support for different data types.

The comparison study by Nurseitov et al. (2009) investigated the difference in performance between JSON and XML when transferring data from a client to a server where the data was decoded. Measurements were taken on CPU utilisation for both client and server, system memory utilisation, and transmission times. The results show that JSON was more than fifty-eight times faster than XML when transferring and decoding many objects at once. When transferring and decoding a lower amount of objects in intervals, JSON was approximately forty times faster on average.

In the paper by Vahdati et al. (2015) the performance of a mapping from formats including XML and CSV, to the Resource Description Framework (RDF) format, was measured. The results showed that the mapping time from XML to RDF was more than nine times slower than the mapping from CSV to RDF.

GeeksForGeeks, (2023) presents differences between the JSON and BSON formats. Among the listed differences, a claim for BSON is that “It is faster than JSON”. This claim is supported by another webpage, (*JSON vs BSON*, n.d.), stating for JSON that “It is comparatively less faster than BSON.”

Practical Study

Our practical study compared the serialisation and deserialisation time of JSON, BSON, XML, and CSV. The time here, refers to the total time of serialising and then deserialising the data to and from a given format. The benchmarks were written and conducted using two programming languages: Python and JavaScript. The serialisation and deserialisation were conducted under six scenarios. Scenarios 1-3 involved tabular structured data of increasing sizes. Scenarios 4-6 involved tree-structured data of increasing sizes. The simple data consisted of items with seven attributes of different data types. The complex data consisted of items with fifteen attributes of different data types, at different depths in the tree-structure. The specific structures used in the practical study can be found in the Appendix. The scenarios of different sizes consisted of 1, 100, and 10,000 items respectively. Since CSV does not support non-linear data, it was not included in the complex-data scenarios.

The results from the Python benchmarks, as shown in Table 9, was that JSON was the fastest, for all scenarios. For the medium and large data size scenarios with simple structured data (scenarios 2 and 3), the order of fastest serialisation and deserialisation was: JSON, CSV, XML, BSON. For the small data size scenario (scenario 1), it went: JSON, CSV, BSON, XML. When running the benchmarks with the complex structured data (scenarios 4-6), the order was constant, regardless of data size: JSON, XML, BSON.

Table 9. Serialisation and deserialisation time in Python in milliseconds

Scenario	JSON	BSON	CSV	XML
1	0.004	0.016	0.007	0.022
2	0.154	1.218	0.191	0.985
3	18.197	133.400	19.169	115.389
4	0.006	0.034	-	0.038
5	0.346	3.089	-	2.142
6	50.104	330.066	-	284.902

Table 10. Serialisation and deserialisation time in JavaScript in milliseconds

Scenario	JSON	BSON	CSV	XML
1	0.002	0.02	0.152	0.062
2	0.089	0.514	0.333	2.717
3	8.635	49.990	17.369	284.297
4	0.003	0.03	-	0.088
5	0.197	1.137	-	5.316
6	19.607	117.557	-	583.843

When running the benchmarks in JavaScript, JSON achieved the fastest time for all scenarios. Table 10 shows the results from all scenarios in JavaScript. For the scenario with simple structured data of small size (scenario 1), the order of fastest execution went: JSON, BSON, XML, CSV. For the medium and large scenarios (scenarios 2 and 3), the execution order was: JSON, CSV, BSON, XML. For the scenarios with complex structured data (scenarios 4-6), the order of fastest execution was the same for all data sizes: JSON, BSON, XML.

4.2.2 Resource Usage

Three relevant articles were found that compared resource usage between data interchange formats. The paper by Nursevoit et al. (2009) found that XML had higher memory usage than JSON. It also found that JSON had higher user CPU utilisation, but lower system CPU utilisation. The paper by Vahdati et al. (2015) comparing XML and CSV found that memory usage for XML was higher. The third paper by Popić, Pezer, Mrazovac, and Teslić (2016) included a comparison of file size between JSON and BSON. The results show that the JSON files were larger.

Literature Study

The paper by Nurseitov et al. (2009) showed for the test where many objects were transferred and decoded at once, XML system memory usage was approximately 8% higher than JSON. In the same test, JSON had 58% higher user CPU utilisation, but 71% lower system CPU utilisation. In the tests where fewer objects were sent in intervals, JSON's user CPU utilisation was, on average, 13% higher than XML's. On the system side, JSON CPU utilisation was 61% lower. The average memory utilisation for these tests was 0.3% higher for XML than JSON.

The comparison by Vahdati et al. (2015) also included memory usage as a comparison metric and found that CSV outperformed XML in this category. Recorded XML memory usage was more than 26% higher than the memory usage for CSV. The paper also recorded the number of generated RDF triples resulting from the mapping. The results show that the mapping from XML to RDF generated approximately 20% more triples than the CSV to RDF mapping.

The difference in file size between JSON and BSON formats was shown in the paper by Popić et al. (2016). When converting 51690 messages into JSON and BSON formats and summarising the file sizes, the JSON files were approximately 17% larger than the BSON files.

Practical Study

Our practical study measured the CPU and memory usage during the serialisation and deserialisation of the formats. The formats were then written to file, which allowed their file size to be measured for the space requirement criterion. The scenarios measured were the same as in the serialisation and deserialisation time benchmark.

The results from the space requirement benchmark show that for simple tabular data, CSV takes up the least amount of space. Following CSV, JSON and BSON perform similarly. XML files were the largest. The specific measurements are displayed in Table 11.

For the CPU usage benchmarking scenarios with simple structured data, the results depended heavily on input size. For the small test case (scenario 1), JSON performed best, followed by BSON, XML, and CSV. For the medium-sized input (scenario 2), the order of best performance was JSON, CSV, BSON and XML. Finally, for the large test case (scenario 3), CSV had the lowest CPU usage, followed by JSON, BSON, and XML. When benchmarking the complex structured data (scenarios 4-6), the order of best performance was constant: JSON, BSON, then XML.

The scenarios where CPU usage was measured were each run between one hundred to ten thousand iterations, and then the average was calculated, in order to achieve a consistent value. The values for average CPU usage for one iteration run in JavaScript are displayed in Table 12. The columns in the table represent a data interchange format. The rows represent a combination of data size and structure. The specific scenarios for the rows are given in the leftmost column.

Table 11. File sizes

Scenario	JSON	BSON	CSV	XML
1	162 B	164 B	107 B	205 B
2	14.336 kB	13.969 kB	6.064 kB	17.568 kB
3	1.396 MB	1.379 MB	0.586 MB	1.711 MB
4	325 B	330 B	-	428 B
5	30.373 kB	30.233 kB	-	39.368 kB
6	2.969 MB	2.973 MB	-	3.846 MB

Table 12. CPU time in microseconds

Scenario	JSON	BSON	CSV	XML
1	15	30	91	77
2	60	291	138	1 011
3	2 746	18 118	2 306	66 994
4	1	31	-	107
5	81	527	-	2 101
6	7 810	36 994	-	144 212

Table 13. Memory usage in megabytes

Scenario	JSON	BSON	CSV	XML
1	34.499	35.652	45.683	38.638
2	36.476	39.153	40.764	43.184
3	72.19	92.528	76.858	153.764
4	34.844	36.055	-	39.01
5	37.849	41.215	-	52.5
6	78.332	104.25	-	182.988

In the same benchmarks as for the CPU usage, memory usage was also measured. Also here, the results for the simple data depended on input size. For the small test case (scenario 1), JSON had the lowest memory usage, followed by BSON, XML, and CSV. For the medium-sized test case (scenario 2), JSON again performed the best, followed by BSON, CSV, and XML. For the large test case (scenario 3), the order of lowest memory usage was: JSON, CSV, BSON, and lastly XML. The order when benchmarking the complex structured data (scenarios 4-6) was constant: JSON, BSON, XML. Table 13 shows the memory allocated for the process while running one hundred iterations in JavaScript.

4.2.3 Language Support

The language support criteria for the data interchange formats were based on whether the language had an official or external library for encoding and decoding of the respective formats.

Table 14. Language support for formats

	JSON	BSON	CSV	XML
C	*cJSON*	*libbson*	*libcsv*	*libxm12*
C++	*RapidJSON*	*mongo-cxx-driver*	**standard C++ streams**	**libxml++**
C#	**System.Text.Json**	**MongoDB C#/.NET driver**	**System.Data**	**System.Xml**
Python	**json**	*PyMongo*	**csv**	**xml**
Java	**javax.json**	**MongoDB Java driver**	**java.io**	**javax.xml**
Javascript	**JSON**	*bson*	*csv-parser*	*xml-js*
Rust	**serde_json**	**bson**	*csv*	*xml-rs*
Ruby	**json**	**MongoDB Ruby driver**	**CSV**	**Nokogiri**

The results were chosen to be displayed in Table 14. **Bold** characters in a cell represent that the format is official and built-in to the language. *Italic* characters in a cell represent that the language does not have an official or built-in library, but that third-party libraries are available. The text in the cells represents the name of the official, or possible third-party, resource.

5. Guidelines for Use of Structured Data Transfer Solutions

Chapter 5 presents the guidelines that were created based on the results from the pre-study. Section 5.1 summarises the structure of the chapter and what components it contains. Section 5.2 describes how to use the guidelines. The purpose of the guidelines are described in Section 5.3. After that comes Section 5.4 which provides the limitations of the created guidelines. Section 5.5 contains the list of created guidelines. To improve the use of the guidelines there is visualisation of the guidelines as flowcharts in Section 5.6. After that comes Section 5.7 which is the summary of the literature and practical study that contains information that was used to motivate the guidelines. Lastly are the validity threats and how the thesis addresses them in Section 5.8.

5.1 Introduction

A visual introduction to the structure of the guidelines is presented in Figure 13. The figure illustrates the different sections and how they are organised. The

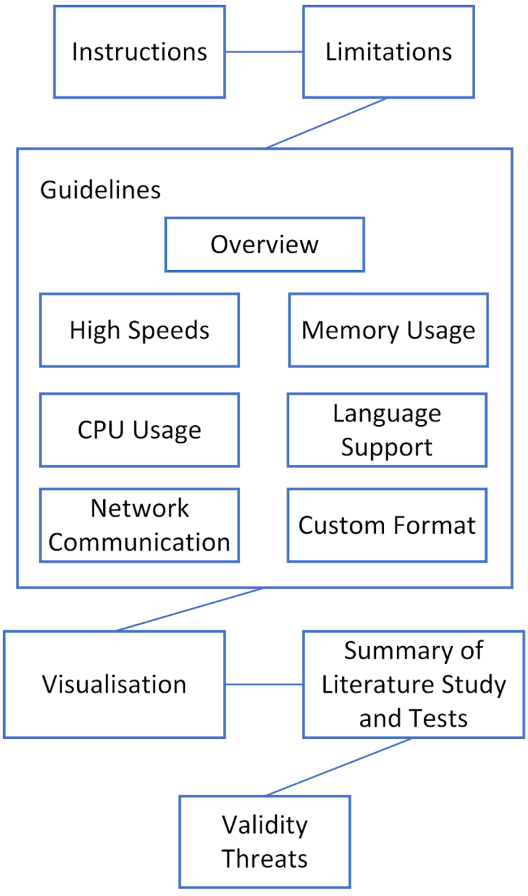

Figure 13. Overview of guidelines structure

instructions section contains information on how to read and use the guidelines. The limitations section describes the limitations of the guidelines. Third is the guidelines section, which is introduced with an overview, followed by the guidelines themselves. After that comes a visualised version of the guidelines, in the form of flowcharts. Second to last is the summary of the literature study and tests which contains information regarding the motivation for each recommendation. The last section describes how the thesis' validity threats were dealt with.

5.2 Instructions

The guidelines bring up information regarding six areas of concern. The areas of concern the solution should work with are **high speeds** (HS), achieve the lowest **memory usage** (MU), or achieve the lowest **CPU usage** (CU). The last three areas are that the solutions have good **language support** (LS), **network communication** (NC), or use a **custom format** (CF). The areas of concern are visualised in Figure 14. To use the guidelines the user selects an area that they think is important. With the selected area of concern they check the subcategories.

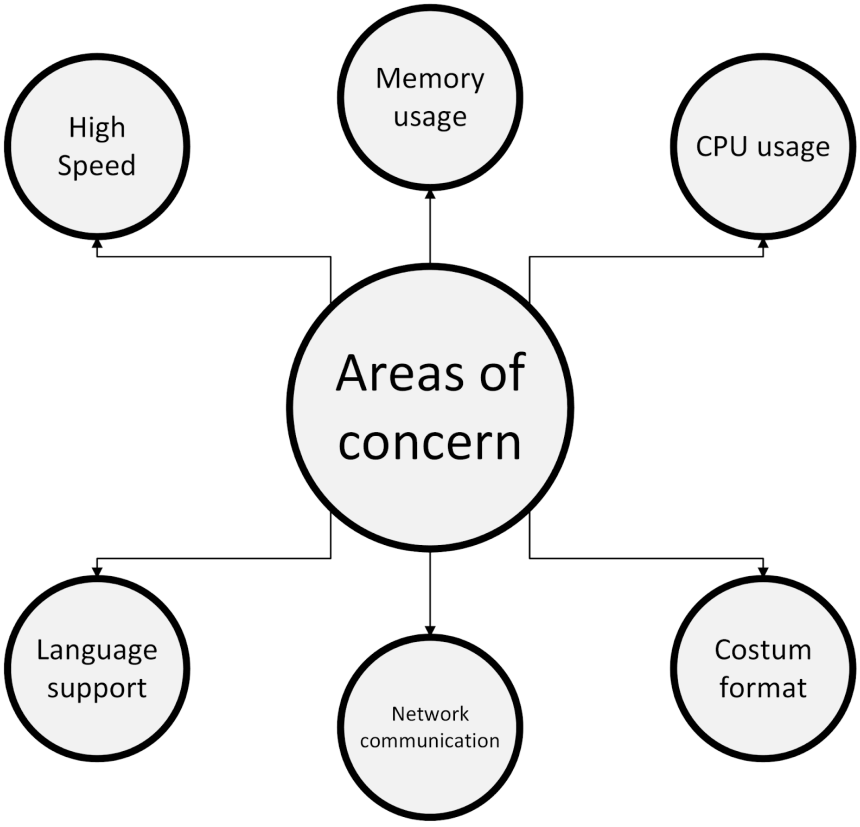

Figure 14. The areas of concern

In each of the six different areas of concern there are three subcategories that can change the recommendation. The three subcategories are the **size of data**, **structure of data**, and the **programming language**. The recommendation can change depending on the size of data so the guidelines bring forth the data size ranges. The ranges are for data less than 6kB, between 6kB and 40kB, between 40kB and 1MB, or larger than 1MB. The structure of the data can change the recommendation so the guidelines bring up two different structures, simple or complex. Lastly the programming language can impact the recommendation therefore the guidelines bring up Javascript and Python.

Figure 15 presents how the subcategories work. The figure starts on the left with an area of concern, in this case high speed. With the area of concern chosen it goes to the subcategories, illustrated as a table. In the table there are the different sizes of data, structures, and programming languages. There can be different recommendations depending on the values of the different subcategories. After the values of the subcategories are determined there is a recommendation of an IPC solution and data interchange format.

When using the guidelines the user will select an area of concern that they think is important for their situation. Once they have selected an area then they can look at the recommendations. The recommendations are divided into three subcategories to provide a suitable solution. So with the area of concern the user will look at the recommendation with their needs in mind. So if the user chooses the area of wanting high speeds then they will look for guidelines with the ID starting with HS. To get a recommendation the user will need to understand what data they will use. For example, a user knows that they will use data larger than 1MB with a simple structure, written in Python. The user can then see that a recommendation matches that situation: HS-1. In this case the user would be recommended to use sockets as an IPC solution and CSV as a data interchange format.

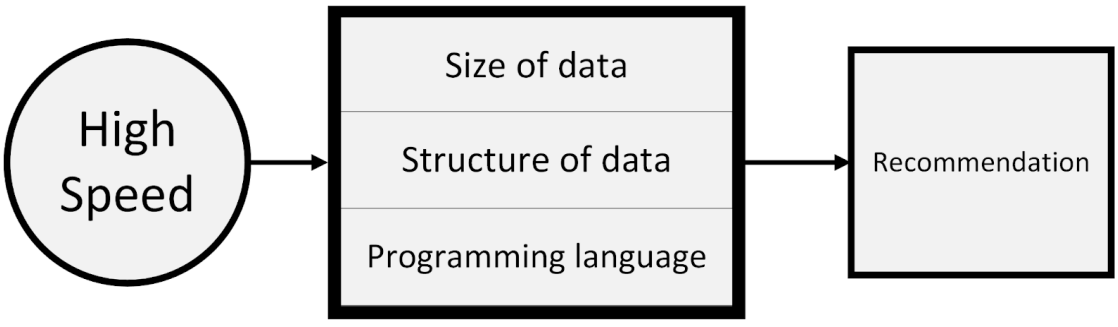

Figure 15. How the subcategories work

The guidelines are given in two different forms. The main format is the text based format that is described in Section 5.5. The second format is the visualised format which is presented in Section 5.6. The text based format provides the areas of concern and subcategories in the shape of section and subsection. This allows the user to go to a section and then further to a subsection to find their recommendation. The text format also provides detailed information on the recommendation and the motivation behind it. This is in contrast to the simpler response of the visualisation in Section 5.6.

The second presentation format was the visualised format that is presented in Section 5.6. The visualisation is presented in the form of flowcharts that the user can follow to get their recommendation. There are two flowcharts with one presenting the recommendation for IPC solution and the other for the recommendation of data interchange format. So the user is intended to use both flowcharts to get their recommended solution.

5.3 Purpose

The purpose of the guidelines is to help software developers use a good data transfer solution for their situation. This was needed as there are many different solutions for this problem but each has their strengths and weaknesses. This requires developers to have an understanding of the solutions to use an appropriate solution. However, gaining the understanding can be time consuming which could be spent elsewhere. Therefore some guidelines that would ease the choice of solution would help save time for developers and companies.

5.4 Limitations

One of the limitations of the guidelines is that they do not bring up the recommendations for the mix of areas of concern. This makes it so that the guidelines do not provide recommendations for situations where high speeds and low CPU usage are wanted.

A limitation of the guidelines is that it only looks into a small number of languages. For the IPC solutions it only looks at the solutions with evidence based on tests written in C. For data interchange formats it only regards Python or Javascript. This is a limiting factor as seen with high speed data interchange formats where the language affected the recommendation with a file size of larger than 1 MB.

For the area of concern of language support there was a limited amount of languages investigated. There was a total of eight languages and those were C, C++, C#, Python, Java, Javascript, Rust, and Ruby. For each language if there was an official library, external library, or no library for the solution.

The guidelines only regard a handful of IPC solutions and data interchange formats. The IPC solutions that the guidelines worked with were pipes, sockets, shared memory, message queues, and Unix sockets. The solutions were all compared against one another. All solutions except Unix sockets were included in at least one guideline. For data interchange formats the formats that were compared were JSON, BSON, CSV, and XML. The formats were tested and compared, and all formats except BSON were recommended in one or more guidelines.

5.5 Guidelines

Section 5.5 presents the guidelines as headings followed by motivation and evidence for the chosen IPC solution and data interchange format. Section 5.5.1 summarises the guidelines in a table to give a simple overview. Sections 5.5.2-5.5.7 are divided into the areas of concern: high speeds, low memory usage, low CPU usage, language support, network communication, and custom format, respectively.

5.5.1 Overview

Each of the areas of concern, introduced in Section 5.2, brings forth a possible important aspect for the user. It is therefore important for the user to easily find the recommendation for their situation. That is why an overview of the guidelines are presented in this section. To increase the readability of the guidelines each recommendation has an ID. The ID is based on the area of concern that the recommendation is a part of.

Table 15 presents the overview of the guidelines. The table presents each of the guidelines ID in the left column and their respective recommendation title in the right column. Therefore the user can choose an area of concern that they think is important. The user then looks up the abbreviation of the area of concern. With the abbreviation the user can then look at the rows associated with the area of concern. The user looks at the recommendation in the right column to find a recommendation for their needs. With the complete ID in the left column the user can look up the full description for the guidelines in Sections 5.5.2-5.5.7.

Table 15. Overview of guidelines

ID	Title
High Speeds	
HS-1	Use Sockets and CSV for Simple Data larger than 1 MB in Python
HS-2	Use Sockets and JSON for Simple Data larger than 1 MB in JavaScript
HS-3	Use Shared Memory and JSON for Simple Data between 40 kB and 1 MB
HS-4	Use Pipes and JSON for Simple Data smaller than 40 kB
HS-5	Use Sockets and JSON for Complex Data larger than 1 MB
HS-6	Use Shared Memory and JSON for Complex Data between 40 kB and 1 MB
HS-7	Use Pipes and JSON for Complex Data smaller than 40 kB
Memory Usage	
MU-1	Use Message Queues and JSON for Data larger than 1 MB
MU-2	Use Pipes and JSON for Data smaller than 1 MB
CPU Usage	
CU-1	Use Sockets and CSV for Simple Data larger than 1 MB
CU-2	Use Sockets and JSON for Simple Data smaller than 1 MB
CU-3	Use Sockets and JSON for Complex Data

Language Support	
LS-1	Use Sockets and JSON for Language Support
Network Communication	
NC-1	Use Sockets for Network Communication
Custom Format	
CF-1	Use XML for All Custom Format Data

5.5.2 High speeds

When developing systems it may be important that the solutions can transfer as much data as possible in a limited time frame. This is to lower possible waiting time or stutters in the system. Therefore a high speed of transferring the data can lead to a smoother experience for the user.

When it comes to the higher speed solutions then there were three areas that impacted the recommendations. The areas were the size of the data that was to be transferred, the structure of the data, and the choice of programming language. The size of data would impact both the IPC solution and the data format solution. The structure of the data and programming language however, only impacted the recommendation for data interchange format.

HS-1: Use Sockets and CSV for Simple Data larger than 1 MB in Python

For transferring simple structured data larger than 1 MB using Python, the guidelines recommends sockets as IPC solution, and CSV as data interchange format.

Sockets were the recommended solution for files larger than 1 MB mostly due to the results in Section 5.7. The results showed that sockets was the fastest IPC solution, for files larger than 1 MB. One of the tests in Section 5.7 had the lowest size of 1 MB and therefore could show that sockets had better performance for files larger than 1 MB. This is supported by the articles as sockets had the third maximum transfer speed. The maximum transfer speeds was 1 700 MB/s in the article by Venkataraman and Jagadeesha (2015). This would point to sockets being one of the faster solutions.

CSV was the chosen data interchange format for large, simple data when programming in Python. Even though JSON serialisation and deserialisation is slightly faster, as can be seen in Section 5.7, the resulting CSV object is smaller. The smaller file size here is enough for the difference in IPC transfer speed to be greater than the difference in serialisation and deserialisation time. Thus, even though the serialisation and deserialisation phase is faster with JSON, the complete transfer becomes faster with CSV.

HS-2: Use Sockets and JSON for Simple Data larger than 1 MB in JavaScript

For transferring simple structured data larger than 1 MB using JavaScript, the guidelines recommends sockets as IPC solution, and JSON as data interchange format.

Sockets were the recommended solution for files larger than 1MB mostly due to the results in Section 5.7. The results showed that sockets was the fastest IPC solution, for files larger than 1 MB. One of the tests in Section 5.7 had the lowest size of 1MB and therefore could show that sockets had better performance for files larger than 1 MB. This is supported by the articles as sockets had the third maximum transfer speed. The maximum transfer speeds was 1 700 MB/s in the article by Venkataraman and Jagadeesha (2015). This would point to sockets being one of the faster solutions.

JSON was the chosen data interchange format for large, simple data when programming in JavaScript. This is due to the fact that JSON had the fastest serialisation and deserialisation for all sizes of data as described in Section 5.7. Section 5.7 describes that JSON is the fastest with it taking half of the time as the second fastest format CSV.

HS-3: Use Shared Memory and JSON for Simple Data between 40 kB and 1 MB

For transferring simple structured data between 40 kB and 1 MB, the guidelines recommends shared memory as IPC solution, and JSON as data interchange format.

For files between the sizes of 40 kB and 1 MB then shared memory was the recommended solution. This was due to the results in Section 5.7 where shared memory was one of the fastest solutions in the small message test. The results of the test showed that shared memory was the fastest for data between the sizes of 64 kB and 512 kB. Shared memory is regarded as fastest for 40 kB due

to it being the fastest for 64 kB. It was also supported by shared memory being the fastest solution from the articles. The shared memory achieved a max transfer speed of 6,500 MB/s, in the article by Venkataraman and Jagadeesha (2015). Therefore pipes were not recommended at 40 kB even when it performed better at 32 kB message sizes.

With files with a size between 40 kB and 1 MB, JSON was the fastest solution. This was regardless whether the programming language was JavaScript or Python. It was supported by the results in Section 5.7, which describes that JSON is the fastest. Section 5.7 describes that JSON is the fastest solution with CSV being the second fastest solution. Section 5.7 describes that the fastest solution was JSON for all measuring scenarios.

HS-4: Use Pipes and JSON for Simple Data smaller than 40 kB

For transferring simple structured data smaller than 40 kB, the guidelines recommends pipes as IPC solution, and JSON as data interchange format.

The reason that pipes were recommended for file sizes less than 40 kB is due to the result of the test described in Section 5.7. The results showed that pipes were the fastest solution for file sizes of 32 kB or smaller. This resulted in pipes being the recommended solution for files less than 40kB. The recommendation was also reinforced with the pipes being the second fastest solution from the articles. Pipes achieved a max transfer speed of 3,000 MB/s in the article by Venkataraman and Jagadeesha (2015).

With files smaller than 40 kB, JSON was the fastest solution. This was regardless whether the programming language was JavaScript or Python. It was supported by the results in Section 5.7, which describes that JSON is the fastest. Section 5.7 describes that JSON is the fastest solution with CSV being the second fastest solution. Section 5.7 describes that the fastest solution was JSON for all measuring scenarios.

HS-5: Use Sockets and JSON for Complex Data larger than 1 MB

For transferring complex structured data larger than 1 MB, the guidelines recommends sockets as IPC solution, and JSON as data interchange format.

Sockets were the recommended solution for files larger than 1MB mostly due to the results in Section 5.7. The results showed that sockets was the fastest IPC solution, for files larger than 1 MB. One of the tests in Section 5.7 had the lowest size of 1MB and therefore could show that sockets had better performance for files larger than 1 MB. This is supported by the articles as

sockets had the third maximum transfer speed. The maximum transfer speeds was 1 700 MB/s in the article by Venkataraman and Jagadeesha (2015). This would point to sockets being one of the faster solutions.

JSON was recommended because it achieved the lowest serialisation and deserialisation time measured for all tests, as described in Section 5.7. Although CSV was almost as fast as JSON for large data in Python and generated a smaller file size, it is disqualified due to its inability to represent complex data. This is supported by Šimec and Magličić (2014) and Nurseitov et al. (2009), where JSON was found to be the fastest out of the measured formats.

HS-6: Use Shared Memory and JSON for Complex Data between 40 kB and 1 MB

For transferring complex structured data between 40 kB and 1 MB, the guidelines recommends shared memory as IPC solution, and JSON as data interchange format.

For files between the sizes of 40 kB and 1 MB then shared memory was the recommended solution. This was due to the results in Section 5.7 where shared memory was one of the fastest solutions in the small message test. The results of the test showed that shared memory was the fastest for data between the sizes of 64 kB and 512 kB. Shared memory is regarded as fastest for 40 kB due to it being the fastest for 64 kB. It was also supported by shared memory being the fastest solution from the articles. The shared memory achieved a max transfer speed of 6,500 MB/s, in the article by Venkataraman and Jagadeesha (2015). Therefore pipes were not recommended at 40 kB even when it performed better at 32 kB message sizes.

JSON was recommended because it achieved the lowest serialisation and deserialisation time measured for all tests, as described in Section 5.7. Although CSV was almost as fast as JSON for large data in Python and generated a smaller file size, it is disqualified due to its inability to represent complex data. This is supported by Šimec and Magličić (2014) and Nurseitov et al. (2009), where JSON was found to be the fastest out of the measured formats.

HS-7: Use Pipes and JSON for Complex Data smaller than 40 kB

For transferring complex structured data smaller than 40 kB, the guidelines recommends pipes as IPC solution, and JSON as data interchange format.

The reason that pipes were recommended for file sizes less than 40 kB is due to the result of the test described in Section 5.7. The results showed that pipes were the fastest solution for file sizes of 32 kB or smaller. This resulted in pipes being the recommended solution for files less than 40kB. The recommendation was also reinforced with the pipes being the second fastest solution from the articles. Pipes achieved a max transfer speed of 3,000 MB/s in the article by Venkataraman and Jagadeesha (2015).

JSON was recommended because it achieved the lowest serialisation and deserialisation time measured for all tests, as described in Section 5.7. Although CSV was almost as fast as JSON for large data in Python and generated a smaller file size, it is disqualified due to its inability to represent complex data. This is supported by Šimec and Magličić (2014) and Nurseitov et al. (2009), where JSON was found to be the fastest out of the measured formats.

5.5.3 Memory Usage

Memory usage can be important for a user as they do not want their data transfer solution to take precious memory from other important programs. If the solutions use too much memory then it could result in the complete system slowing down. This would be as a result from processes and solutions fighting over memory space.

Memory usage can be especially important in computers with a low amount of memory and lots of processes running. An example of this would be a mobile phone as they have many processes running in the background, such as social media apps, checking if someone is calling, or a game the phone is running. With the processes each needs some memory to have good performance and if that memory is used by the solutions then it can slow the phone.

MU-1: Use Message Queues and JSON for Data larger than 1 MB

For transferring data larger than 1 MB, the guidelines recommends message queues as IPC solution, and JSON as data interchange format.

When looking at the results in Section 5.7 it showed that for all sizes except 1 MB message queue had the lowest memory usage. The memory usage was consistent for the file sizes. This resulted in the recommendation that for files larger than 1 MB message queues would give lower memory usage.

JSON was recommended because it achieved the lowest memory usage for all test scenarios, as shown in Section 5.7.

MU-2: Use Pipes and JSON for Data smaller than 1 MB

For transferring data smaller than 1 MB, the guidelines recommends pipes as IPC solution, and JSON as data interchange format.

The recommendation for pipes with smaller files was due to the results of the test that were done, as described in Section 5.7. The results described that pipes had the lowest memory usage for files of 1 MB. This suggested that pipes worked well with smaller files. Therefore pipes were the recommended solution for files less than 1 MB.

JSON was recommended because it achieved the lowest memory usage for all test scenarios, as shown in Section 5.7.

5.5.4 CPU Usage

CPU usage can be important for a user as they do not want their data transfer solution to impact the processing speed of another program. If a system relies on many parts that does a large amount of calculations then CPU time can be limited. Then a solution that also wants CPU time can slow down the whole system.

Low CPU usage can help not impact the performance of other parts of a system. This would be helpful in a system that deals with large amounts of data that is calculated. The calculation can take time and therefore needs time to be calculated by the CPU. Then it would be a problem if the calculation program was fighting for CPU time with the solution that provides the data from one process to another.

CU-1: Use Sockets and CSV for Simple Data larger than 1 MB

For transferring simple data larger than 1 MB, the guidelines recommends socket as IPC solution, and CSV as data interchange format. The IPC format

did not change the choice of IPC solution and the same solution was for all file sizes.

For the IPC part of the solution then sockets were the recommended solution. This came from the results that are described in Section 5.7. Section 5.7 describes that sockets have the lowest CPU usage out of all the solutions. Therefore sockets is the recommended solution for files larger than 1 MB.

CSV was the recommended data interchange format, because it achieved the lowest CPU time for the large data test scenario. The results of the test scenarios are described in Section 5.7.

CU-2 Use Sockets and JSON for Simple Data smaller than 1 MB

For transferring simple data smaller than 1 MB, the guidelines recommends socket as IPC solution, and JSON as data interchange format. The IPC format did not change the choice of IPC solution and the same solution was for all file sizes.

For the IPC part of the solution then sockets were the recommended solution. This came from the results of the test as described in Section 5.7. Section 5.7 shows that sockets has the lowest CPU usage out of all the solutions. The tests did not involve file sizes less than 1 MB but the test involved buffer sizes less than 1 MB. Sockets had the lowest CPU usage for all buffer sizes. This would indicate that if a message was the size of the buffer, less than 1 MB, that sockets would still have the lowest CPU usage. Therefore sockets are the recommended solution for data smaller than 1 MB.

JSON was chosen as the recommended data interchange format because it achieved the lowest CPU time for the small and medium simple structured data. This result was obtained from test scenarios, and is described in Section 5.7.

CU-3 Use Sockets and JSON for Complex Data

For transferring complex data, the guidelines recommends socket as IPC solution, and JSON as data interchange format. The IPC format did not change the choice of IPC solution and the same solution was for all file sizes.

For the IPC part of the solution then sockets were the recommended solution. This came from the results of the test as described in Section 5.7. Section 5.7 shows that sockets has the lowest CPU usage out of all the solutions. The tests did not involve file sizes less than 1 MB but the test involved buffer sizes less

than 1 MB. Sockets had the lowest CPU usage for all buffer sizes. This would indicate that if a message was the size of the buffer, less than 1 MB, that sockets would still have the lowest CPU usage. Therefore sockets are the recommended solution for data smaller than 1 MB.

JSON was chosen as the recommended data interchange format because it achieved the lowest CPU time for the small and medium simple structured data. This result was obtained from test scenarios, and is described in Section 5.7.

5.5.5 Language Support

Language support can improve how easy it can be to implement a solution in a system. This is because a developer does not need to search for libraries or create their own to get something working. Therefore high language support would make it easier to work with the solution.

When a solution is easy to implement then that could lead to multiple benefits. One benefit would be that developers have to spend less time implementing the solution. This can lead to more time working on bigger aspects of the system, which can help make deadlines and the same costs. If a solution is easy to implement with official libraries then it can be more easily maintained in the future of the system. Lastly, if a solution has official libraries in many languages then it will be easier to add new languages to the system which could work with the old system.

LS-1 Use Sockets and JSON for Language Support

For language support the sockets and JSON was the recommendation. This recommendation was not divided by data structures and file formats as they do not impact the language support. The only aspect that affects the language support is the programming languages that are included and how many there were.

For the IPC solution sockets were the recommended solution as it was the only solution with eight official libraries. This resulted in it being the solution with best language support, as the second best was Unix sockets with seven official libraries and one external. The data interchange format with the best language support was JSON with official libraries in six out of eight languages. This was the best as the second best solutions CSV and XML had five official libraries out of eight.

5.5.6 Network Communication

Some systems may run over a cluster of different computers. In this situation multiple cooperative processes can run on multiple different computers. These processes in different computers need to transfer data between one another to complete the whole system. This requires some solutions that can transfer data between the computers.

The use of multiple separate computers can increase things such as reliability, performance, and modularity. The reliability is increased with how if one computer turns off then another can take its place. An example of this is the google file system described by Ghemawat, Gobioff, and Leung (2003). The google file systems worked by storing information of a master computer and if it stopped working then another computer could take its place. The performance could be increased by having two computers do different operations. This would help as a single computer would do operation one and then the second operation. But with two computers then a computer can do one of the operations and both are done at the same time. The last is an increase in modularity which would be a result of if one wants more performance or adds a new part of the system, then one can add a new computer.

NC-1 Use Sockets for Network Communication

With sockets being the only solution that can transfer data over a network, it should be used over other solutions. The communication over the network has no impact on the data interchange format. The format is only affected by the size of the data and the structure of the data. Therefore for the use of data interchange format, check the other guidelines.

5.5.7 Custom Format

There are some restrictions when working with formats such as JSON, BSON and CSV in terms of flexibility. One such restriction is the inability to write comments in the format structure, without having to create an attribute specifically for it. If flexibility is desired for the structure of the data, then a more extensible data format should be used.

CF-1 Use XML for All Custom Format Data

Regardless of IPC solution, if flexibility with the structure of the data is desirable, then XML is the recommended data interchange format. Much of the flexibility of the XML format comes from the ability to define namespaces, include attributes in element tags, and include comments. Unfortunately,

XML achieved the worst performance on almost all tests. Therefore it can only be recommended if all of the comparison criteria are irrelevant.

The custom format does not have any influence on the IPC solution. Therefore the choice of IPC solution is decided by going through the other guidelines and choosing a solution in accordance with user criteria.

5.6 Visualisation

To improve the ease of using the guidelines, some visualisations were created. The visualised flowcharts depict the recommendations depending on the need of the software developer. The flowcharts are designed after the recommendations in Section 5.5. Figure 16 represents the IPC solution part of the guidelines, while Figure 17 represents the data interchange part. Splitting the guidelines into two separate flowcharts was done to improve readability.

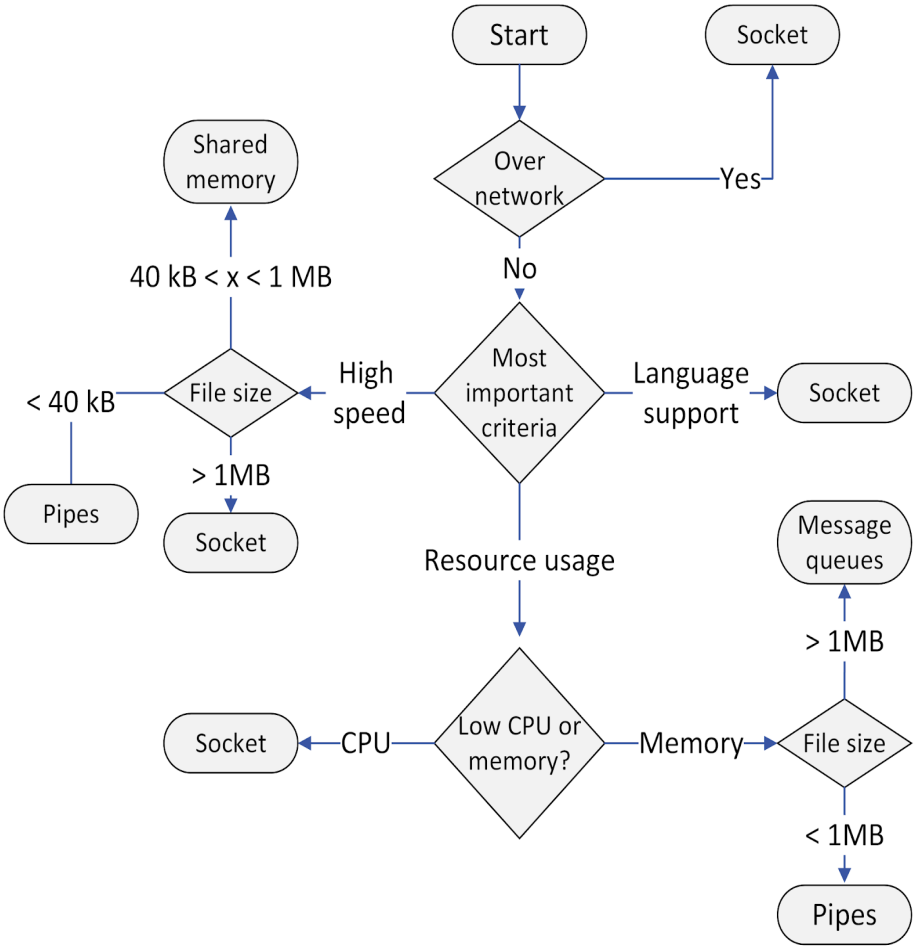

Figure 16. Visualisation of guidelines for IPC recommendations

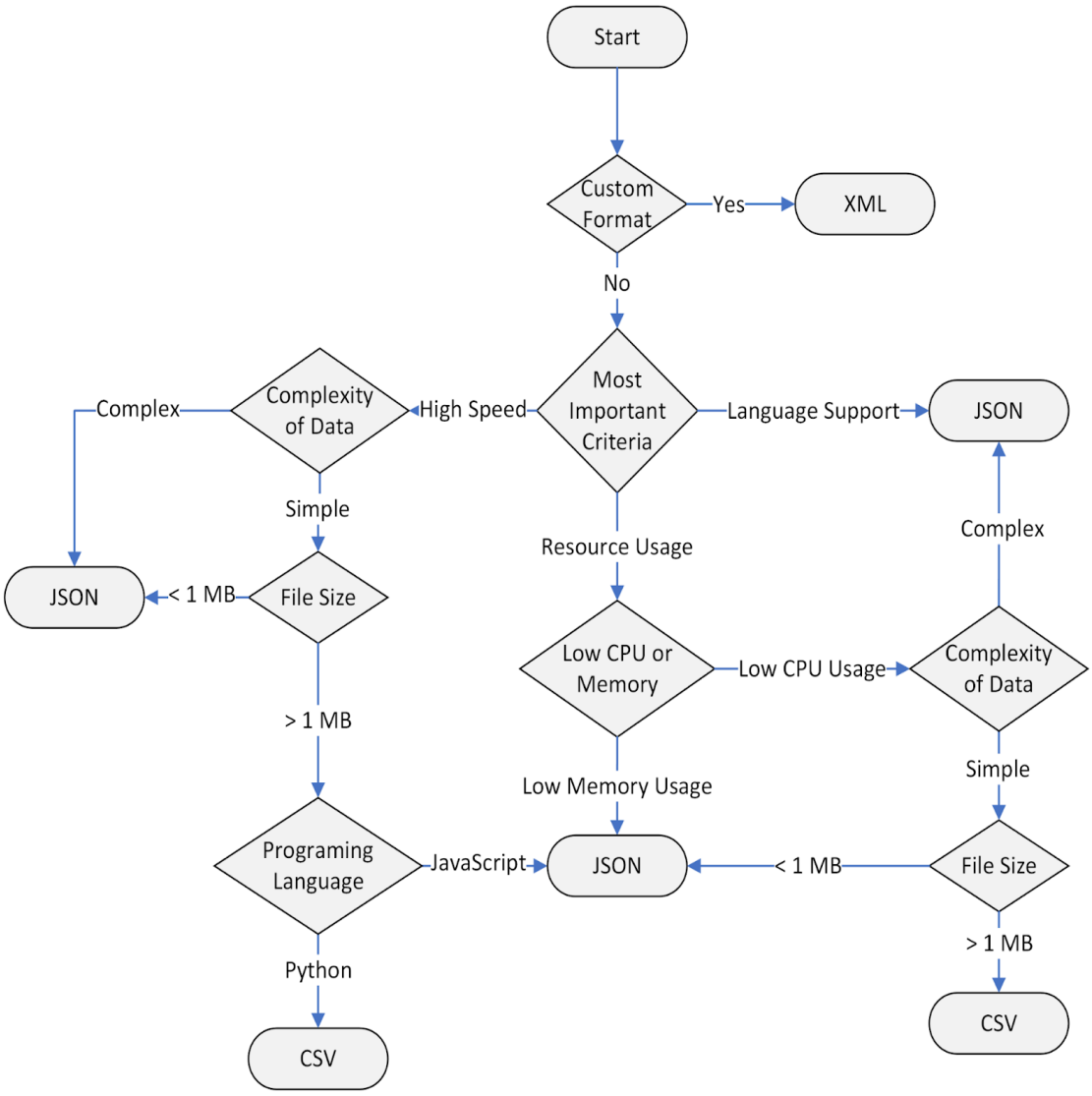

Figure 17. Visualisation of guidelines for data interchange format recommendations

The user uses the flowchart by starting in the square called *start*, at the top of the figure. The user then follows the arrows from the diamond-shaped decision boxes, depending on their answer. This step is repeated until an oval conclusion box is reached. Once the user reaches an oval box then they have their recommended solution. Guidelines consist of a combination of an IPC solution and data interchange format. Therefore, both flowcharts need to be consulted in order to reach a complete result.

The following is an example of a user consulting Figure 16 to determine the IPC solution for their use case: The user starts in the *start* box at the top of the figure. They then follow the arrow to the *over network* decision box and can either choose to follow the *yes* or *no* arrow. This user does not need their solution to transfer data over networks, and thus chooses *no*. The user then

follows the *no* arrow to the *most important criteria* decision box. This user wants the data transfer to have a low resource impact on the system, and therefore chooses to follow the *resource usage* arrow. This leads the user to the *low CPU or memory* decision box. The user is more concerned with low CPU usage and therefore follows the *CPU* arrow. The user has now arrived at the *Socket* conclusion box, and is done.

5.7 Summary of Literature Study and Tests

The summary contains information regarding the literature that was found on the subject and the tests that were done. Section 5.7.1 describes the articles that were found and used in the motivation for the different solutions. The first paragraph is the IPC solutions articles and the second paragraph is the data interchange format articles. Section 5.7.2 contains information on the IPC tests that were conducted and their results. The last Section 5.7.3 contains the test for data interchange formats and their respective results.

5.7.1 Literature

There are three articles that were used in the guidelines for high speed for IPC solution motivation. One article is *Interprocess communication with Java in a Microsoft Windows Environment* by Dylan Gregory Smith and George Wells (2017). The second article is *Evaluation of inter-process communication mechanisms* by Aditya Venkataraman and Kishore Kumar Jagadeesha (2015). The last article was *Performance analysis of various mechanisms for inter-process communication* by Kwame Wright, Kartik Gopalan, and Hui Kang (2007).

For the articles regarding high speed for data interchange formats there were a total of three. The first article is *Comparison of JSON and XML Data Formats* by Alen Šimec and Magdalena Magličić (2014). The second article is *Comparison of JSON and XML Data Interchange Formats: A Case Study* by Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta (2009). The last article is *Mapping Large Scale Research Metadata to Linked Data: A Performance Comparison of HBase, CSV and XML* by Sahar Vahdati, Farah Karim, Jyun-Yao Huang, and Christoph Lange (2015).

5.7.2 IPC Tests

Tests were done to get numerical data on the speed of transfer of IPC solutions. There were two different tests, one that sent a message back and

forth with the IPC solution. The other test read the contents of a file, sent all of the data through the solution to another process and then wrote the data to a new file. The messages in the first test ranged from 1 kB to 512 kB. The files in the second test ranged from 1MB to 5 GB. The result of the first test was that pipes were the fastest solution for messages less than 64 kB. The test showed that shared memory was the fastest solution for messages between 64 kB and up to 512 kB. The results from the second test showed that sockets were the fastest solution for the different file sizes.

The memory usage of the IPC solutions were measured by the use of a file copying test. The test involved reading the data from a file, transferring the data to another process with the IPC solution, and writing the data to a new file. The data sizes of the file ranged from 1 MB to 1 GB. The results of the tests were that pipes had the lowest memory usage for the 1 MB files. Message queues proved to have the lowest memory usage for files larger than 1 MB. The tests were also done with measuring the CPU usage of the solution. The results of those tests were that sockets had the lowest CPU usage for all file sizes.

The test for measuring the CPU usage of the solutions was the same as for memory usage. The test involved sending the contents of a file between two processes with an IPC solution. The file sizes ranged from 1 MB to 5 GB. To the test there was also a varying buffer size for the solutions. The buffer sizes were 16 kB, 64 kB, and 256 kB. The results of the test showed that sockets had the lowest CPU usage for all file sizes and all buffer sizes.

5.7.3 Data Interchange Format Test

Measurements for serialisation and deserialisation time for the data interchange formats were conducted twice. The first round of measurements were written in Python, and the second in JavaScript. The Python measurements showed that JSON serialisation and deserialisation time was consistently the fastest, for all measuring scenarios. For the scenarios where the data was restricted to tabular structure, CSV was consistently the second fastest. The JavaScript measurements also resulted in JSON achieving the fastest serialisation and deserialisation time for all measuring scenarios.

Memory usage was measured for all formats during serialisation and deserialisation. Regardless of data complexity and size, memory usage for JSON was consistently the lowest.

Like memory usage, CPU usage was also measured during serialisation and deserialisation. When data was restricted to a tabular structure, JSON CPU usage was lowest for the small and medium size tests. For the test with large

data size, CSV CPU usage was the lowest. For the tests with complex structured data, JSON achieved the lowest CPU usage regardless of data size.

After serialising a data object to a file, the size of the file was measured and recorded. For the tests with tabular structured data, CSV achieved the smallest file size, for all scenarios. For the tests with complex structured data, JSON file size was the smallest for the small and large test case. For the medium test case, BSON file size was the smallest.

5.8 Validity Threats

The criteria for validity threats are used to test the strength and soundness of a research method. Therefore they are presented in this section to test the chosen research method. The validity threats for qualitative research are *credibility*, *transferability*, *dependability* and *confirmability*. All of the criteria are further explained in how they were addressed:

- **Credibility:** To solve this there was a mixture of both our practical research and literary studies. Most of the work was based on multiple studies and therefore if all of them said the same thing then the conclusion is most likely trustworthy. In addition to that, some measurements were done in order to confirm and bolster claims done in some studies.

- **Transferability:** In order to help with keeping the results generalised then multiple sources were used. By using multiple sources it showed results from different situations.

- **Dependability:** To provide dependability the thesis uses both literature studies and some measurements. This would help provide dependability as the measurements would be able to be done again in different contexts. In addition, the measurements were done multiple times and then the average was taken from those. This would increase dependability as the measurements would be done in different contexts of the system.

- **Confirmability:** To help prevent bias in the results the research process was thoroughly described to help note how the research was done. Then to further improve confirmability then the motivation for each decision was made so that other researchers could understand the reasoning.

6. Analysis and Discussion

It is important to analyse the results and bring forth a discussion. Therefore this chapter is about analysing and discussing the results from the pre-study and the guidelines. In Section 6.1 the results of the pre-study phase are analysed. It focuses on comparing results from the literature study with the results from the practical study. Section 6.2 presents an analysis on the guidelines. Lastly is Section 6.3, which is a discussion of the results. The discussion is meant to bring forth new questions, answers, and perspectives.

6.1 Analysis of Pre-study

The literature study from the IPC section of the pre-study showed that shared memory was the fastest. It was followed by pipes, sockets, Unix sockets and message queues in order of fastest to slowest. In the practical study, it was shown that for larger files, sockets were the fastest option. Pipes were shown to be the fastest solution for messages sizes smaller than 64 kB. For message sizes between 64 kB and 1 MB it was shown that shared memory was the fastest solution.

In regards to resource usage there were two predominant solutions: FIFO pipes and sockets. When transferring 1 MB files, FIFO pipes showed the lowest memory usage, but the highest CPU usage. This was in contrast to sockets which showed the lowest CPU usage but the highest memory usage. The solution that had the best average of both low memory usage and low CPU usage was message queues. This was due to message queues having one of the lower memory usages and often the second-lowest CPU usage.

One thing to note about the results is the message sizes used in the test. For the single message test there were messages larger than 8 kB. Hoewer as noted in Section 2.3.3 message queues have a maximum message size of 8 kb in the Linux operating system. This required the message queues to send multiple messages, instead of one, for tests with messages larger than 8 kB. The small message sizes were seen as a property of message queues and could be the reason for message queues low memory usage. This would be due to message queues not needing as much message buffer size in memory for the message.

For IPC solutions, only speed of transfer criterion included both results from the practical study and literature study. Therefore that was the only criterion with the possibility of a comparison between the studies. When looking at the max speed of transfer then there could be a difference between all solutions. All solutions were slower in the practical study than in the literature study. The closest was for sockets, where the practical study came close to achieving

the same transfer speeds as the literature study. The results were also different when sending the contents of a file. The main difference was that sockets proved to be one of the fastest solutions in the practical study. This was in contrast to the literature study where sockets were a sixth of the speed of the fastest solution, which was shared memory.

Regarding the performance of the data interchange formats, BSON performed worse than expected in the practical study. During the literature study, no research-based articles studying BSON serialisation and deserialisation time were found. However, the web pages that were found, both noted that BSON is faster than JSON. This was not supported by the practical study, where BSON was several times slower than JSON, for all measuring scenarios.

6.2 Analysis of Guidelines

The results from Chapter 5 presented the guidelines that were the result of the information from the pre-study in Chapter 4. The guidelines were structured in a way that provided six main areas of concern. Those areas were high speed, low memory usage, low CPU usage, language support, network communication, and custom formats. Each represented a main property that the user would want the solution to focus on. Then for each area of concern, there were subcategories that could change the recommendation. The subcategories were the size of data, the format of the data, and the programming language. The sizes could be less than 40 kB, between 40 kB and 1 MB, and larger than 1 MB. For the data formats, they could be simple, complex, or custom. The programming languages could either be Python and Javascript.

For the data formats there were different recommendations when using simple formats. Sometimes CSV was recommended and sometimes JSON. For complex data formats, the JSON dominated and was the recommended solution for all areas of concern. One of the areas of concern was custom formats. XML was the solution that provided the most amount of flexibility, and was thus the recommended format.

Regarding the IPC solution, sockets were the recommended solution for network communication, language support, and low CPU usage. The high-speed area of concern had three different solutions depending on the file size. Those solutions were pipes, sockets, and shared memory. For the memory usage area of concern, either pipes or message queues were recommended, depending on file size. Pipes were recommended for files smaller than 1 MB, whilst message queues were recommended for files larger than 1 MB.

Unix sockets and BSON were not recommended for any of the guidelines. The reason why Unix sockets were not recommended was because they were not great in any aspect, but average in many. The guidelines focused on one area of concern at a time. This resulted in the recommendations focusing more on solutions with the best performance in one area. Regarding BSON, the serialisation and deserialisation were noted in the literature study to be faster than JSON. However, this was not reliably shown in the practical study. This caused the guidelines to not confidently recommend the format.

6.3 Discussion

One interesting note from the result was the literature study of IPC solutions noted that sockets would be one of the slower solutions. This was not true for the practical study where sockets were one of the faster solutions. This created the question of why sockets were better in the practical study than in the literature study. This could be due to how the tests were conducted, because the tests were running on a VM, because of the computer hardware, or a combination of these factors. An understanding of the reason could lead to better guidelines. The guidelines would improve by including more relevant areas of concerns and subcategories which would be more likely to give accurate recommendations.

One possible answer to why sockets were fast in the practical study, but slow in the literature study, is the difference in computer hardware and operating system. The tests were done on a VM and it may have had an impact on the results. This would be due to IPC solutions working with the operating systems. Adding a VM could potentially add complexity to the operating system functionality. This could possibly slow down solutions that are more heavily reliant on the operating system.

The type of tests proved to have an impact on the IPC solutions. This was seen in how pipes and shared memory were better in the back-and-forth message test, in Figure 12. The file copying tests in Table 4 showed that sockets were the faster solution. This would indicate that a generalised answer to the choice of IPC solution could be hard to make. Therefore it can be important to include a variety of areas of concern and subcategories to provide accurate guidelines.

The reason for the surprising result on BSON serialisation and deserialisation time is not clear, and could depend on several factors. One such factor is the choice of programming language. It is possible that BSON would have performed relatively better than JSON, if the same tests were run in, for example, C. This is because C is a low-level programming language, more

optimised for binary operations. This behaviour might be beneficial for binary-based formats, such as BSON.

7. Conclusions and Future Work

The amount of technology and computers we as humans use today are ever increasing. With that increase comes an increase in the number of different programming languages. Those languages have different strengths and weaknesses. Using different languages for different parts of a system is one reason for needing data transfer solutions. The problem is that there are no guidelines for data transfer solutions. Therefore the purpose of the thesis is to create guidelines that would help in choosing a suitable data transfer solution. The goal of the thesis is to help inexperienced software developers to find a data transfer solution that fits their needs.

The thesis uses a qualitative research method with the support of a comparative method. The comparative method is supported by a comparison model. The results of the thesis is information on the solutions from a literature study and a practical study. The results are also the guidelines that help software developers choose a suitable data transfer solution for their situation.

7.1 Conclusions

The research was conducted in four phases which were *Pre-study*, *Creation of comparison model*, *Creation of Guidelines*, and *Finalisation of Guidelines*. The *Pre-study* was about gaining information on the solutions through a literature study and a practical study. The *Creation of comparison model* phase involved creating a comparison model to further compare the solutions. The *Creation of Guidelines* phase involved creating the guidelines and continually improving them. The last phase, *Finalisation of Guidelines,* was the creation of the final version of the guidelines. The final version was created by correcting problems from the latest version of the guidelines.

To be able to compare the different solutions a comparison model was created. The model contained criteria that were deemed to be of importance for the IPC solutions and data interchange formats. The different criteria were *speed*, *resource usage*, and *language support*. The *speed* criterion was included to determine which solution was the fastest, as it would be important for some use cases. The *resource usage* criterion was included as a low impact on the system can be desirable. A low system impact allows for more resources to be assigned to more essential programs. The last criterion was *language support*. With higher language support, the solution would be easier to implement in different systems. An easier implementation can lead to a decrease in development time.

The results from the literature study showed that shared memory is the fastest IPC solution. In the practical study, it was shown that it depends on the data size. The pre-study shows that depending on the data size either pipes or message queues have the lowest memory usage. Sockets had, for all data sizes, the lowest CPU usage, and the best language support.

For data interchange formats, the literature study showed that JSON was faster than both XML and CSV. In regards to resource usage it was shown that XML had higher memory usage and higher system CPU utilisation than JSON. However, JSON had higher user CPU utilisation. Additionally, the literature study showed that XML also had higher memory usage than CSV. All of these findings were supported by the practical study with benchmarks of format system impact. One of the statements from the literature study which was not supported by the practical study, was a difference in file size between JSON and BSON. The reason for this contradiction would have to be explored in future research.

The result of the thesis are guidelines that are structured to help choose a combination of IPC solution and data interchange format. The guidelines provide recommendations for solutions depending on what area of concern the user prioritises. For each area of concern, there is a recommendation depending on the data sizes, structure of the data, and programing language.

The thesis provides the research domain with use cases for both IPC solutions and data interchange formats. For example, the literature study for IPC solutions showed that shared memory was the fastest solution. However, in the practical study, it was noted that shared memory was only the fastest for larger single message transfers. If a developer wanted constant transfers of larger messages than sockets would potentially be the best solution. Another example is that BSON was expected to perform better due to the information from the literature study. But in the practical study, it was found that BSON did not perform as well. This may have been due to the programming languages in which the tests were implemented.

The guidelines can help the research domain by being a potential starting point for further guidelines. The problem of the thesis is that there were no guidelines. However, the thesis has now created guidelines. The guidelines can be used to further the research domain. This would be done by further improving or expanding on the existing guidelines. The current results could also be used as an inspiration for similar reports in other research areas.

7.2 Future Work

There is an opportunity to expand on this thesis by conducting future work. With the comparison model already created, the future work could be expanding the number of solutions and programming languages. The practical study of the thesis focuses on IPC solutions from the POSIX library. This could be expanded to include other libraries like system V, or the use of other message queues, such as RabbitMQ. There could also be an expansion of the data interchange formats by including formats such as Protocol Buffers, YAML, and MessagePack.

There is an opportunity to expand the tests in the pre-study by including more programming languages. The tests were only run on C, Python, or JavaScript. The pre-study can be expanded to include an investigation for other programming languages.

An increase of languages in the pre-study can lead to improvements in the guidelines. The problem is that only two programming languages are represented in the guidelines. The guidelines show that language has an impact on the recommended solution. Thus, an increase in languages can lead to more accurate recommendations.

The guidelines could also be extended to factor in multiple areas of concern in one recommendation. The current guidelines only focus on one area of concern at a time. For example, if the user wants high speeds or low memory usage. By including multiple areas of concern in every recommendation, the user can prioritise certain areas of concern, for example, wanting low memory usage as number one priority, and good language support as second priority. This would result in the guidelines recommending a solution that has one of the lowest memory usage and good language support.

One of the main aspects that the thesis was missing was the use of interviews. The interviews could be with both software professionals, and possible users of the guidelines. New information and perspectives could be obtained by interviewing software professionals knowledgeable in the field. The professionals could give feedback on the guidelines to note what they feel is important when choosing a data transfer solution. By interviewing possible users, such as inexperienced software developers, the guidelines could be improved to help usability. The interviews would give a better insight into what is needed for the guidelines to be more readable.

7.3 Reflection

The guidelines from this report addresses the problem that there are no guidelines for data transfer solutions. During the creation of the guidelines, problems occurred that we have learned from. Most of these problems occurred during the testing and evaluation of the solutions. It was difficult to design tests that simulated real-life scenarios. Finding the potential criteria for the comparison model proved to be difficult. Another challenge was how to measure the properties of the solutions. This was especially hard due to us being inexperienced in working with the solutions. If a similar project would be undertaken, seeking advice from someone knowledgeable in the field would greatly help.

There is a belief that the guidelines could prove helpful for users when choosing a data transfer solution. The guidelines provide information on what solutions there are, and when to use them. This would help software developers that are either working with data transfer solutions or getting started. These individuals can be either academic students, who need to understand the different solutions, or inexperienced software developers. The software developers would use the guidelines to easily choose a solution. This would allow them to focus more on implementation, rather than choosing a solution. This would help save both development time and money for the developers and companies. This is especially important since we live in a technology-dominated society.

References

Announcing Rust 1.0 | Rust blog. (2015, May 15). Retrieved August 30, from https://blog.rust-lang.org/2015/05/15/Rust-1.0.html

Bhandari, P. (2023, January 30). *What Is Qualitative Research? | Methods & Examples*. Scribbr. Retrieved June 25, 2023, from https://www.scribbr.com/methodology/qualitative-research/

Computer History Museum. (n.d.). *Timeline of Computer History*. Retrieved June 07, 2023, from https://www.computerhistory.org/timeline/computers/

Florescu, D., & Fourny, G. (2013). JSONiq: The History of a Query Language. *IEEE Internet Computing*, *17*(5), 86–90. https://doi.org/10.1109/mic.2013.97

GeeksforGeeks. (2023). *Difference Between JSON and BSON*. GeeksforGeeks. Retrieved June 26, 2023. https://www.geeksforgeeks.org/difference-between-json-and-bson/

Ghemawat, S., Gobioff, H., & Leung, S. A. (2003). *The Google file system*. https://doi.org/10.1145/945445.945450

GitHub: Let's build from here. (n.d.). GitHub. Retrieved August 30, 2023. https://github.com/

Google Scholar. (n.d.). Retrieved August 30, 2023. https://scholar.google.com/

Hemmendinger, D. (2023, May 8). *XML*. *Encyclopedia Britannica*. https://www.britannica.com/technology/XML

IBM. (1972). IBM FORTRAN Program Products for OS and the CMS Component of VM/370 General Information (Document No. GC28-6884-0).

IEEE. (2020) *"7.8 IEEE Code of Ethics"* IEEE Policies, Section 7 - Professional Activities (Part A - IEEE Policies).

JSON. (n.d.). *The JSON Data Interchange Format*. Retrieved May 4, 2023, from https://www.json.org/json-en.html

JSON vs BSON. (n.d.). Javatpoint. Retrieved June 26, 2023, from https://www.javatpoint.com/json-vs-bson

Kalin, M. (n.d) *A guide to inter-process communication in Linux*. OPENSOURCE.COM. Received on 2023-05-30. Taken from https://opensource.com/downloads/guide-inter-process-communication-linux

Kaur, A., Ayyagari, S., Mishra, M., & Thukral, R. (2020). A Literature Review on Device-to-Device Data Exchange Formats for IoT Applications. *JOURNAL OF INTELLIGENT SYSTEMS AND COMPUTING*, *1*(1), 1–10. https://doi.org/10.51682/jiscom.00101001.2020

Kerrisk, M. (2010). *The Linux Programming Interface*(1st ed.). San Francisco, No Starch Press.

Krishnaveni, S., & Ruby, D. (2016). Comparing and evaluating the performance of inter process communication models in Linux environment. *International Journal of Trend in Research and Development*, 51-55. 2023-05-03, Received from http://www.ijtrd.com/papers/IJTRD4246.pdf

Leo Computers Society. (n.d.). *Leo III Installations*. Retrieved June 18, 2023, from http://www.leo-computers.org.uk/leo-3s.html

Leo III User Manual Vol IV MASTER PROGRAMME and PROGRAMME TRIALS SYSTEM. (n.d.). Retrieved June 17, 2023, from http://settle.ddns.net/LeoMan/Vol4P1.htm#s6

Mayer, P. & Bauer, A. (2015). An empirical analysis of the utilization of multiple programming languages in open source projects. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering* (pp.1-10). Association for Computing Machinery, New York, NY, USA, Article 4, Received from https://doi.org/10.1145/2745802.2745805

MongoDB. (n.d.). *Explaining BSON With Examples*. Retrieved May 4, 2023, from https://www.mongodb.com/basics/bson

Mq_overview(7) - Linux manual page. (2023). Linux Man-Pages. Retrieved June 26, 2023, from https://man7.org/linux/man-pages/man7/mq_overview.7.html

Nanz, S., & Furia, C. A. (2014). A Comparative Study of Programming Languages in Rosetta Code. In *2015 IEEE/ACM 37th IEEE*

International Conference on Software Engineering,(pp. 778-788) Florence, Italy, https://doi.org/10.1109/icse.2015.90

Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). Comparison of JSON and XML Data Interchange Formats: A Case Study. In *Computer Applications in Industry and Engineering* (pp. 157–162). https://www.cs.montana.edu/izurieta/pubs/caine2009.pdf

Oracle VM VirtualBox. (n.d.). Retrieved August 30, 2023, from https://www.virtualbox.org/

Popić, S., Pezer, D., Mrazovac, B., & Teslić, N. (2016). *Performance evaluation of using Protocol Buffers in the Internet of Things communication*. https://doi.org/10.1109/sst.2016.7765670

Shafranovich, Y. (2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. https://doi.org/10.17487/rfc4180

Shenton, A. K. (2004). Strategies for ensuring trustworthiness in qualitative research projects. *Education for Information*, *22*(2), 63–75. https://doi.org/10.3233/efi-2004-22201

Silberschatz, A. Galvin, P., B. & Gagne, G. (2018). *Operating System Concepts* (10th ed.) John Wiley & Sons.

Šimec, A., & Magličić, M. (2014). Comparison of JSON and XML Data Formats. Received from

https://www.researchgate.net/publication/329707959_Comparison_of_JSON_and_XML_Data_Formats

Smith, D. G., & Wells, G. C. (2017). Interprocess communication with Java in a Microsoft Windows Environment. *South African Computer Journal*, *29*(3), 198-214. https://doi.org/10.18489/sacj.v29i3.500

Spasov, Z. & Madevska Bogdanova, A. (2010). *Inter-process communication, analysis, guidelines and its impact on computer security*. Institute of Informatics, Faculty of Natural Sciences and Mathematics, Ss. Cyril and Methodius University in Skopje, Macedonia.

Statistics Sweden, (2022), *Share of persons who have used the Internet several times per day*, Statistics Sweden, Received 02-02-2023 from https://www.scb.se/en/finding-statistics/statistics-by-subject-area/living-conditions/living-conditions/ict-usage-in-households-and-by-individuals/pong/tables-and-graphs/share-of-persons-who-have-used-the-internet-several-times-per-day/

Vahdati, S., Karim, F., Huang, J., & Lange, C. (2015). Mapping Large Scale Research Metadata to Linked Data: A Performance Comparison of HBase, CSV and XML. In *Communications in computer and information science* (pp. 261–273). Springer Science+Business Media. https://doi.org/10.1007/978-3-319-24129-6_23

Venkataraman, A. and Jagadeesha, K, K. (2015). Evaluation of inter-process communication mechanisms. 2023-05-02 Received from https://pages.cs.wisc.edu/~adityav/Evaluation_of_Inter_Process_Communication_Mechanisms.pdf

Vetenskapsrådet. (2002). *Forskningsetiska principer inom humanistisk-samhällsvetenskaplig forskning*. Stockholm: Vetenskapsrådet.

Visual Studio Code - Code editing. Redefined. (2021, November 3). Retrieved August 30, 2023. https://code.visualstudio.com/

What is a Virtual Machine? (n.d.). Oracle Sverige. https://www.oracle.com/se/cloud/compute/virtual-machines/what-is-virtual-machine/

World Wide Web Consortium. (2019). *XML Core (Second Edition)*. Retrieved May 4, 2023, from https://www.w3.org/standards/xml/core

Wright, K., Gopalan, K., & Kang, H. (2007). Performance analysis of various mechanisms for inter-process communication. *Operating Systems and Networks Lab, Dept. of Computer Science, Binghamton University*. 2023-05-03, Received from https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d523500aa27c65c9e76215ea2aa59e87b9c5760c

Appendix

This appendix includes figures that illustrate the structure of the data that was used for the benchmarking in Section 4.2. The benchmarks included two different structures of the data. These were tabular structured data, and nested structure data. Figures 18 and 19 show the structure of the data in the JSON format, as to display the differences between tabular and nested data.

The structure of the tabular data is illustrated in Figure 18. It contains one item, consisting of seven attributes: *id*, *name*, *price*, *description*, *color*, *size*, and *material*. When benchmarking the larger scenarios with multiple items, these were all generated and put inside of the *Simple_data* structure.

The structure of the nested data is illustrated in Figure 19. This structure also represents one item. Here, two of the attributes, *properties* and *reviews*, represent a list of other attributes, or collection of attributes. Similar to the tabular structure, when benchmarking the larger scenarios, multiple items were placed inside the *Complex_data* structure.


```
{
    "Simple_data": [
        {
            "id": 27363,
            "name": "item 17",
            "price": 82.44,
            "description": "This is a random item",
            "color": "red",
            "size": "large",
            "material": "wool"
        }
    ]
}
```


Figure 18. Tabular data in JSON

```json
{
    "Complex_data": [
        {
            "id": 98226,
            "name": "item 56",
            "price": 91.97,
            "description": "This is a random item",
            "properties": {
                "color": "green",
                "size": "large",
                "material": "nylon"
            },
            "reviews": [
                {
                    "id": 9117,
                    "text": "This is a great product!",
                    "rating": 5
                },
                {
                    "id": 6129,
                    "text": "I would not recommend this product",
                    "rating": 5
                }
            ]
        }
    ]
}
```


Figure 19. Nested data in JSON

TRITA-EECS-EX- 2023:699