Muğla Sıtkı Koçman University
Department of Computer Engineering

Senior Design Project I

Implementation Of Domain-Specific Language Using Rust

Analysis & Progress Report

Ali DİKME , Mehmet Reşit ÇAĞAN

Supervisor: Burak EKİCİ

January 10, 2024

Contents

Implementation of DSL

1. Introduction

In this comprehensive report, we will delve into the world of a programming language named Rust. Rust is gaining attention in the technology community for its unique qualities, its safety and efficiency. This growing interest makes Rust an important subject for us to explore and understand. Our primary objective for first senior design is to gain a deeper understanding of Rust and to examine its practical applications in various projects.

We will begin our exploration by introducing Rust. We will explain what Rust is, its key features, and why it is becoming a language of choice for many programmers. This introduction to Rust aims to provide a clear and simple understanding of the language's basics.

Following the introduction, we will compare Rust with another well-known programming language, C. This comparison is crucial as it will help us understand the advantages and differences of Rust in contrast to other languages. We will focus on simple and easy-to-understand examples to make this comparison clear and informative.

In our report, we have included practical examples of projects developed in Rust. Specifically, we will look at two different versions of the Tic-Tac-Toe game. One version uses a method known as shared memory, while the other employs message passing for communication between different parts of the program. These projects are not just for illustration; they are pivotal in helping us grasp how Rust works in real-world applications.

Lastly, we will look towards the future and discuss our plans for using Rust. We are particularly excited about the prospect of creating our own small, specialized language using Rust. This ambitious project, while challenging, promises significant learning experience.

In conclusion, our journey through this report will take us from the basic principles of Rust to its practical applications and future potential.

2. Motivation

In this part of our report, we want to share with you why we are excited about learning Rust. Rust is not just another programming language; it has special features that make it very interesting and useful. Our motivation for studying Rust comes from several reasons.

Firstly, Rust is known for being very safe. In programming, making mistakes is common. These mistakes can cause big problems. Rust is designed to stop many of these mistakes. This makes programs written in Rust more reliable. For us, learning a language that helps reduce errors is very exciting.

Secondly, Rust is fast. Speed is important in programming. Programs that run fast are better. Rust helps make programs that can run very quickly. This is good for many kinds of projects, especially when we need to handle a lot of information or do things quickly.

Another reason we are motivated to learn Rust is that it is becoming more popular. Many companies and programmers are starting to use Rust. This means that knowing Rust can be very useful for jobs in the future. It is always good to learn skills that are in demand.

We are also interested in Rust because it can be used for many different things. Some people use Rust to make websites. Others use it for more complex tasks, like operating systems or

scientific calculations. This shows that Rust is a very flexible language. Learning Rust means we can work on many different kinds of projects.

Lastly, our big project is to create our own small language using Rust. This is a challenging task. But it is a great way to use all the skills we learn. This project motivates us to understand Rust deeply.

In summary, our motivation for learning Rust comes from its safety, speed, popularity, flexibility, and the exciting opportunity to create something new with it. We hope that by sharing our motivations, you too will find interest and excitement in learning Rust.

3. Literature Review

In this part of our report, we discuss how Rust can be used for creating parsers and domain-specific languages (DSLs). This topic is very interesting because Rust offers unique features that make it a good choice for these kinds of projects.

Parsers are important in programming because they help computers understand and process languages. From the articles and books we read, we learned that Rust is a great tool for building parsers. The reason is that Rust is very careful and exact. This means it can handle the rules of languages very well. When we use Rust to make a parser, it can work quickly and without mistakes. This is good because parsers need to be fast and accurate.

Then, we explored how Rust can be used to make domain-specific languages. A domain-specific language is a special kind of programming language that is made for a specific purpose. For example, a language just for making websites, or a language for doing scientific research. Rust is very helpful for creating these languages. The reason is that Rust lets us write code that is both clear and powerful. This is important when making a new language because we want it to be easy to understand and use.

The books and articles also talked about the safety of Rust. This is a big reason why people like to use Rust for these projects. When we write code in Rust, it helps us avoid common mistakes that can cause problems. This is very important when working with languages and parsers because even a small error can cause big problems.

Another point we found in our research is that Rust is becoming more popular for these kinds of projects. Many programmers and companies are using Rust to build parsers and domain-specific languages. They choose Rust because it is reliable, fast, and safe. This popularity also means there is a lot of support and resources available. This is very helpful for anyone learning Rust or starting a project in Rust.

Finally, the real-world examples of projects done in Rust were very inspiring. We read about different projects where programmers used Rust to create parsers and domain-specific languages. These examples show us how powerful and flexible Rust is. They also give us ideas for our own projects.

In conclusion, our research shows that Rust is an excellent choice for building parsers and domain-specific languages. Its precision, speed, safety, and growing popularity make it stand out. We are excited to use Rust for our projects and explore its capabilities further.

4. Rust Examples

Our journey with Rust began with learning the basics. Rust is different from other languages because it focuses a lot on safety. This means Rust tries to prevent errors before the program

runs. We read books and watched tutorials to understand Rust's unique features. The more we learned, the more we realized how powerful Rust is for building complex programs.

Our first project was a Tic-Tac-Toe game. This is a simple game where two players put 'X' or 'O' in a grid. The challenge was to make this game in Rust using threads. We used something called a Mutex. A Mutex is like a lock. It helps to make sure that only one thread can use a part of the code at a time. This is important to prevent problems when two threads want to change the same thing. In our game, we created a shared board that both players could access. We used threads to allow each player to take turns. The Mutex made sure that only one player could move at a time. This project taught us how to manage shared resources in Rust. After finishing our first project, we started a second version of the Tic-Tac-Toe game. This time, we used message passing instead of a Mutex. Message passing is like sending notes between threads. Each thread has a mailbox. One thread sends a message to the mailbox of another thread. This way, they can communicate without using the same memory. In this version, each player's move was sent as a message. The game board was not shared like in the first project. Instead, each thread had its own copy of the board. They updated it based on the messages received. This project helped us understand a different way of managing threads in Rust.

Through these projects, we learned a lot about Rust and threads. The first project with the Mutex taught us how to protect shared data. The second project with message passing showed us how to communicate between threads without sharing data. The main difference between the two projects is how they handle the game board. In the first project, there is one board, and threads take turns using it. In the second project, each thread has its own board, and they update it by sending messages. Both projects were great for learning Rust. We understood how to use threads, Mutex, and message passing. These are powerful tools in Rust that help make programs safe and efficient.

5. Rust vs. C: A Comparative Analysis

In the world of programming, Rust and C are important languages. They are similar in some ways but also very different in others. In this section, we compare Rust and C, focusing on their key differences, such as references and ownership, and type safety.

One big difference between Rust and C is how they handle references and ownership. In Rust, there is a system called 'ownership'. This system makes sure that each piece of data in the program is owned by a specific part of the code. This helps prevent errors and makes the program safer. When you want to use data in different parts of the program, Rust uses something called 'borrowing'. This is like asking permission to use the data for a while. In C, there is no such system. You can use data anywhere in your program. This freedom is good in some ways but can lead to mistakes, like using data that has already been deleted. This can cause serious errors in your program.

Type safety is another important area where Rust and C differ. Rust is very strict about types. This means that it checks very carefully to make sure you are using data in the right way. If you try to use data of one type as if it were another type, Rust will give you an error. This strictness helps prevent many common programming mistakes. C is less strict about types. It allows you to do more things with different types of data. While this can be useful, it also means it's easier to make mistakes. These mistakes can be hard to find and can cause your program to fail in unexpected ways.

Memory safety is a big advantage of Rust over C. Rust is designed to be memory safe. This means it prevents many common errors related to memory, like accessing memory that the program should not use. Rust's ownership system plays a big part in this. It makes sure that memory is used correctly and safely. In C, memory safety is a common problem. Because C gives the programmer a lot of control over memory, it's easy to make mistakes. These mistakes can lead to serious problems like crashes or security vulnerabilities.

Both Rust and C are known for being fast. C has been used for many years to write fast, efficient programs. Rust also offers excellent performance. The big difference is that Rust provides this performance while also adding more safety checks. This means you can write fast code in Rust without worrying so much about making mistakes.

In conclusion, Rust and C are powerful languages, but they have different strengths. Rust is more focused on safety, preventing errors, and making sure your program behaves correctly. C offers more flexibility and control but requires the programmer to be very careful to avoid errors. Depending on what you need for your project, either Rust or C could be the right choice.

6. Future Work: Implementing A Domain-Specific Language in Rust

After learning a lot about Rust, we have a new goal. We want to create our own language using Rust. This language will be a 'domain-specific language' (DSL). A DSL is a special kind of programming language made for a specific purpose.

Rust is a good choice for making a DSL for several reasons. First, Rust is very safe. It helps us avoid mistakes in our code. This is important when creating a new language because we want it to work well and not cause errors. Second, Rust is very clear. This means we can write code that is easy to understand. When making a new language, we want it to be simple for people to use. Lastly, Rust is powerful. It can handle complex tasks. This is good because sometimes we need to do complicated things in our new language.

Our plan is to make a small DSL that is easy to use. We want to focus on a specific area, like web development or data analysis. This way, our language can be very good at one thing. We will start by deciding what features we want in our language. Then, we will use Rust to write the code for these features.

Creating a DSL is not easy. There are some challenges we might face. One challenge is making sure our language is easy to understand and use. We don't want it to be too complicated. Another challenge is testing our language to make sure it works well. We need to check for any errors and fix them.

As we work on this project, we will learn a lot. We will get better at using Rust. We will also learn about language design and how to solve problems.

7. Conclusion

As we come to the end of our report, we want to look back at what we have learned and shared about Rust. Rust is a programming language that is both powerful and safe. It is good for many different kinds of projects.

We talked about how we learned Rust. It was not always easy, but it was worth it. Rust's systems like ownership and type safety make it special. They help us write better and safer programs.

Our projects, the two versions of the Tic-Tac-Toe game, showed us how Rust works with threads. We saw how Mutex and message passing are used in Rust. These projects were a great way to practice what we learned.

We also compared Rust with C. Both languages are good, but they are different. Rust is more about safety and preventing errors, while C gives more control but with higher risk of mistakes.

Finally, we are excited about our future work. We plan to create a domain-specific language using Rust. This will be a big challenge, but we are ready to learn more and grow as programmers.

In conclusion, our journey with Rust has been very educational. We have seen how Rust can be used in different ways and for different projects. We are excited to keep using Rust and to discover more about what we can do with it.