Berner Fachhochschule
Wirtschaft

Proof of Concept Microkernel Architecture

Proof of Concept for a Text Processing Tool Using Microkernel Architecture

Assignment 2

Course of study	**Bachelor of Science in Wirtschaftsinformatik**
Submitted by	**Marius Hägele, Nicola Bolt, Alper Simsek, Donjet Dzemaili**
Module	**Software Architecture - SDA2**
Experts	**Siddhartha Singh** **Prof. Dr. Sebastian Höhn**
Date of submission	**16.12.2024**

Contents

1. Introduction

Text processing is widely used in many areas, such as content creation and machine learning. It’s an essential tool for professionals and researchers. With this project, we aimed to create a flexible and efficient text processing tool that is both powerful and easy to extend and maintain.

To achieve this, we used a microkernel architecture. This approach separates the core system functions from additional features, making the tool modular. The core handles basic tasks, while the actual text processing is done by plugins. This setup made the tool easier to maintain and expand. We developed several plugins that can perform tasks like word counting, readability analysis, and keyword searching. By allowing plugins to be dynamically integrated, the tool offers users a customizable and versatile experience. We also added file selection, which made the tool more user-friendly.

Throughout the project, we faced challenges related to plugin management, performance, and fault tolerance. Solving these problems helped us better understand the microkernel architecture and how it can be used to build scalable and reliable systems. This document explains how we designed the architecture, the plugins we developed, and the lessons we learned. Thanks to the modular design, new features can be added without changing the core system. This means the tool can easily be enhanced with more text processing functions in the future.

2. Microkernel Architecture

The Microkernel Architecture is a design pattern that separates the core functions of a system from its additional features. At the center is a minimal core, called the "kernel," which handles only the essential operations needed to keep the system running. Extra features are added as plugins that connect to the core using a defined set of rules.

One of the biggest advantages of the microkernel approach is its modularity. Each part of the system can be developed, tested, and maintained separately. This makes debugging and maintenance much easier. It also means that new features can be added without having to modify the core system. For example, in our text processing tool, plugins like the Word Counter, Case Converter, or Readability Checker can be added or updated without affecting the core.

This architecture also makes the tool very flexible. It’s easy to add new plugins, so the tool can grow and adapt to meet different user needs. Another benefit is that errors in one plugin don’t affect the rest of the system, which improves stability and reliability.

The microkernel architecture is used in many fields. For example, in operating systems like Windows NT or Mach, the core manages essential operations, while device drivers are added as extensions. Similarly, in database systems or game engines, features like indexing or AI modules are implemented as plugins. For our project, this architecture was a great choice because it’s simple to understand and highly extendable.

Overall, the microkernel architecture made our text processing tool robust, easy to maintain, and very customizable. Users can choose the plugins that fit their needs. The modular design lets the tool perform tasks like counting words, analyzing readability, and even reversing text. This project really showed us how practical and elegant the microkernel approach can be.

2.1. Principles of Microkernel Architecture

Minimal Core:

The core of the system includes only the basic features needed to keep it working. The core of the Text Processing Tool handles file input, plugin management and interaction flow.

Dynamic Plugins/Extensions:

Additional features are plugins or extensions that interact with the core via a defined interface. You can add, remove or update these plugins without affecting the core system.

Separation of Concerns:

The architecture separates the core and plugins. The core is lightweight and maintainable, while the plugins add specific functionalities.

Scalability and Flexibility:
The architecture allows developers to easily extend the system without affecting the overall stability of the core.

Fault Isolation:
Plugins work independently, so if one fails, the system keeps going.

2.2. Advantages of Microkernel Architecture

The Microkernel Architecture is ideal for modular systems. Modules can be developed, tested and maintained separately, reducing complexity. It also allows new plugins to be added without changing the core, making the system adaptable to changing needs.

The lightweight core makes it easy to maintain and debug and allows changes to plugins without affecting the core's stability. The architecture can cope with errors in one plugin without crashing the whole system. Plugins can often be reused in other projects with similar architectures. The Microkernel Architecture is a powerful framework for building scalable and robust systems.

3. Plugins Selection

The Text Processing Tool has a set of plugins that each do a specific text processing job. These plugins are loaded at runtime, allowing easy customization and extension. Here is a list of all the plugins and what they do.

3.1. Case Converter

Purpose: Converts the text to a specific case format.

The Case Converter plugin in case_converter.py changes text to different case formats. It offers three text conversion options. The first option, Uppercase, makes all letters uppercase. The second option, Lowercase, changes all letters to lowercase. The third option, Capitalized, capitalizes the first letter of every word and leaves the rest lowercase.

Example:

Input: "hello world"

Option: Capitalized

Output: "Hello World"

3.2. Character Counter

Purpose: Counts the number of characters in the text.

The Character Counter plugin in character_counter.py analyzes character usage in text. This plugin does two things. It counts the total number of characters, including spaces and punctuation. It also counts the number of characters without spaces, showing the text content without whitespace.

Example output:

Total Characters: 1200

Characters Without Spaces: 1050

3.3. Longest Word Finder

Purpose: Identifies the longest word in the text and its length.

The Longest Word Finder plugin identifies the longest word in a text and shows its length. It splits the text into words and removes punctuation. It then finds the longest word and displays it.

Example:

Input: "This is an example of a microkernel architecture."

Output:

Longest Word: "architecture "

Length: 12

3.4. Readability Checker

Purpose: Measures how easy or difficult the text is to read.

The Readability Checker plugin evaluates how easy or difficult a text is to read. It uses the Flesch Reading Ease formula to calculate a readability score. This score shows how complex a text is by looking at things like sentence length and word structure.

Example:

```
Applying plugin: Readability Checker

Output File Content:

Readability Checker:
Readability Score (Flesch): 21.46
Readability Level: Very Confusing
```

3.5. Text Search

Purpose: Searches for specific words or phrases in the text.

The Text Search plugin lets users search for words or phrases in text. The plugin asks the user to enter a word or phrase to search for. The system scans the text and counts how many times the specified term appears.

Example:

```
Applying plugin: Text Search
Enter the word or phrase to search for: Ordnung

Output File Content:

Text Search:
'Ordnung' found 2 times.
```

3.6. Word Counter

Purpose: Counts the total number of words in the text.

The Word Counter plugin calculates the number of words in a text. This plugin splits text into words using spaces. It counts the words to give you the total word count.

Example:

Input: "This is an example text file."

Output: Word Count: 6

3.7. Word Frequency

Purpose: Displays a histogram of the most frequently used words.

The Word Frequency plugin analyses text to find the most common words and shows this information in a histogram. The plugin cleans and processes the text, removing punctuation and converting all words to lowercase.

Example:

the: *****

system: ***

user: **

tool: **

3.8. Word Reverser

Purpose: Reverses every word in the text while maintaining the original order of the words.

The Word Reverser plugin reverses each word in a sentence while keeping the original order. The plugin splits the text into words and reverses them.

Example:

Input: "Hello World"

Output: "olleH dlroW"

4. Challenges we faced

Developing a system for plugin integration turned out to be quite challenging. One of the main tasks was designing a strong interface between the core and the plugins to make sure everything worked smoothly and without compatibility issues. Even though the architecture was meant to isolate plugin errors, we still experienced situations where faulty plugins unexpectedly affected the data flow and user experience. This showed us how important it is to have effective error isolation in place.

Another challenge was the communication between the core and the dynamically loaded plugins, which created additional overhead. This became particularly noticeable during operations that involved a lot of data, as it impacted the system's performance. On top of that, testing and debugging took a lot of effort because we had to carefully validate how each plugin interacted with the core and with other plugins. Finding and fixing the causes of errors was often more difficult than we initially expected.

5. Conclusion

In this project, we were able to demonstrate how a text processing tool can be implemented using the microkernel architecture. This architecture proved to be flexible and scalable. By separating the core and the plugins, we developed a modular system that can easily adapt to different text processing requirements. The core handled essential tasks such as file input/output and loading plugins, while the plugins provided functionalities like word counting, readability analysis, and term searching.

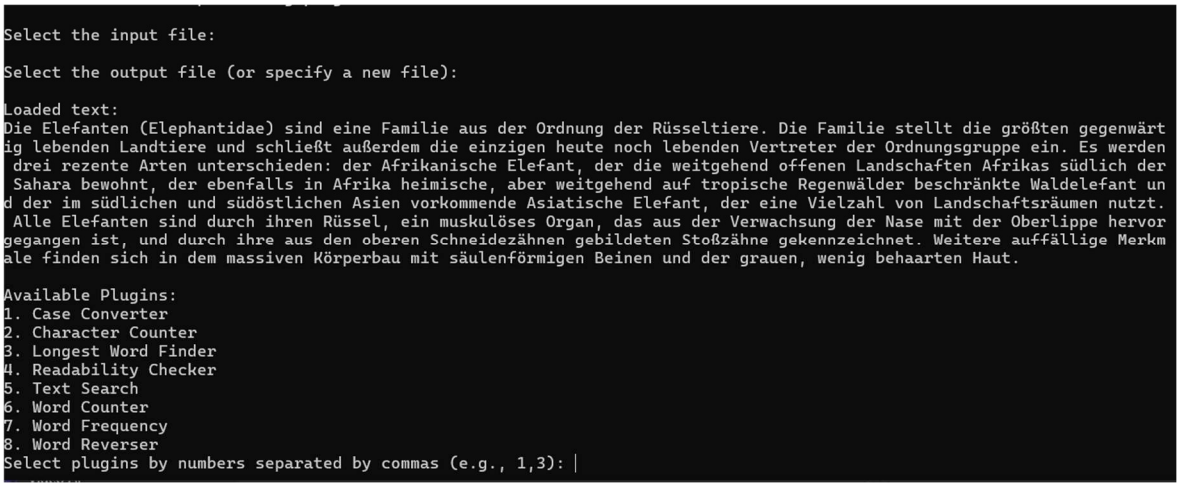
However, during the implementation, we faced several challenges that showed us the practical limitations of this architecture. Designing a stable interface between the core and the plugins was particularly challenging, as it required ensuring compatibility and smooth interaction. Despite the planned separation between the core and plugins, faulty plugins occasionally disrupted the data flow and user experience. Additionally, communication between the core and dynamically loaded plugins sometimes led to performance issues, especially during data-heavy operations.

To ensure reliability, an extensive phase of testing and debugging was necessary. This made us realize how important clear structures and thorough validation are to identify and fix problems early on.

Overall, this project helped us gain a deeper understanding of the microkernel architecture. We saw how it can be used to build flexible and expandable systems, but also learned about the trade-offs that come with it. New plugins can be added seamlessly without impacting the core, which makes the system future-proof. This project showed us the importance of thoughtful design and careful testing in building stable and adaptable systems that can meet both current and future requirements.

6. Examples in/output:

6.1. Input



6.2. Output

```
Output File Content:

Case Converter:
DIE ELEFANTEN (ELEPHANTIDAE) SIND EINE FAMILIE AUS DER ORDNUNG DER RÜSSELTIERE. DIE FAMILIE STELLT DIE GRÖSSTEN GEGENWÄR
TIG LEBENDEN LANDTIERE UND SCHLIESST AUSSERDEM DIE EINZIGEN HEUTE NOCH LEBENDEN VERTRETER DER ORDNUNGSGRUPPE EIN. ES WER
DEN DREI REZENTE ARTEN UNTERSCHIEDEN: DER AFRIKANISCHE ELEFANT, DER DIE WEITGEHEND OFFENEN LANDSCHAFTEN AFRIKAS SÜDLICH
DER SAHARA BEWOHNT, DER EBENFALLS IN AFRIKA HEIMISCHE, ABER WEITGEHEND AUF TROPISCHE REGENWÄLDER BESCHRÄNKTE WALDELEFANT
 UND DER IM SÜDLICHEN UND SÜDÖSTLICHEN ASIEN VORKOMMENDE ASIATISCHE ELEFANT, DER EINE VIELZAHL VON LANDSCHAFTSRÄUMEN NUT
ZT. ALLE ELEFANTEN SIND DURCH IHREN RÜSSEL, EIN MUSKULÖSES ORGAN, DAS AUS DER VERWACHSUNG DER NASE MIT DER OBERLIPPE HER
VORGEGANGEN IST, UND DURCH IHRE AUS DEN OBEREN SCHNEIDEZÄHNEN GEBILDETEN STOSSZÄHNE GEKENNZEICHNET. WEITERE AUFFÄLLIGE M
ERKMALE FINDEN SICH IN DEM MASSIVEN KÖRPERBAU MIT SÄULENFÖRMIGEN BEINEN UND DER GRAUEN, WENIG BEHAARTEN HAUT.

Character Counter:
Total Characters: 945
Characters Without Spaces: 820

Longest Word Finder:
Longest Word: Landschaftsräumen
Length: 17

Readability Checker:
Readability Score (Flesch): 21.46
Readability Level: Very Confusing

Text Search:
'Rüssel' found 2 times.

Word Counter:
Word Count: 126
```

```
Word Counter:
Word Count: 126

Word Frequency:
der: **********
die: *****
und: *****
aus: ***
elefanten: **
sind: **
eine: **
familie: **
lebenden: **
ein: **


Word Reverser:
eiD netnafelE )eaditnahpelE( dnis enie eilimaF sua red gnundrO red .ereitlessüR eiD eilimaF tllets eid netßörg giträwneg
eg nednebel ereitdnaL dnu tßeilhcs medreßua eid negiznie etueh hcon nednebel retertreV red eppurgsgnundrO .nie sE nedrew
 ierd etnezer netrA :nedeihcsretnu red ehcsinakirfA ,tnafelE red eid dnehegtiew neneffo netfahcsdnaL sakirfA hcildüs red
 arahaS ,tnhoweb red sllafnebe ni akirfA ,ehcsimieh reba dnehegtiew fua ehcsiport redläwnegeR etknärhcseb tnafeledlaW dn
u red mi nehcildüs dnu nehciltsödüs neisA ednemmokrov ehcsitaisA ,tnafelE red enie lhazleiV nov nemuärstfahcsdnaL .tztun
 ellA netnafelE dnis hcrud nerhi ,lessüR nie sesöluksum ,nagrO sad sua red gnushcawreV red esaN tim red eppilrebO negnag
egrovreh ,tsi dnu hcrud erhi sua ned nerebo nenhäzedienhcS netedlibeg enhäzßotS .tenhcieznnekeg eretieW egilläffua elamk
reM nednif hcis ni med nevissam uabreprök tim negimröfneluäs nenieB dnu red ,neuarg ginew netraaheb .tuaH


Do you want to process another file? (yes/no):
```