Архитектурная документация CLI-приложения (по IEEE 42010:2011)

---

1. Общее описание системы

Название системы: Интерпретатор командной строки (CLI)
Цель: Предоставить удобный интерфейс для выполнения команд в стиле Unix с поддержкой пайпов, переменных окружения и пользовательских команд.
Заинтересованные стороны:

- **Пользователи** — используют CLI для выполнения команд и автоматизации задач.
- **Разработчики** — расширяют функциональность, добавляют новые команды.

---

2. Стейкхолдеры и их интересы

Стейкхолдер	**Интересы и требования**
Пользователи CLI	Удобный, предсказуемый интерфейс, аналогичный стандартному shell.
Разработчики	Простота добавления новых команд, удобная архитектура.

---

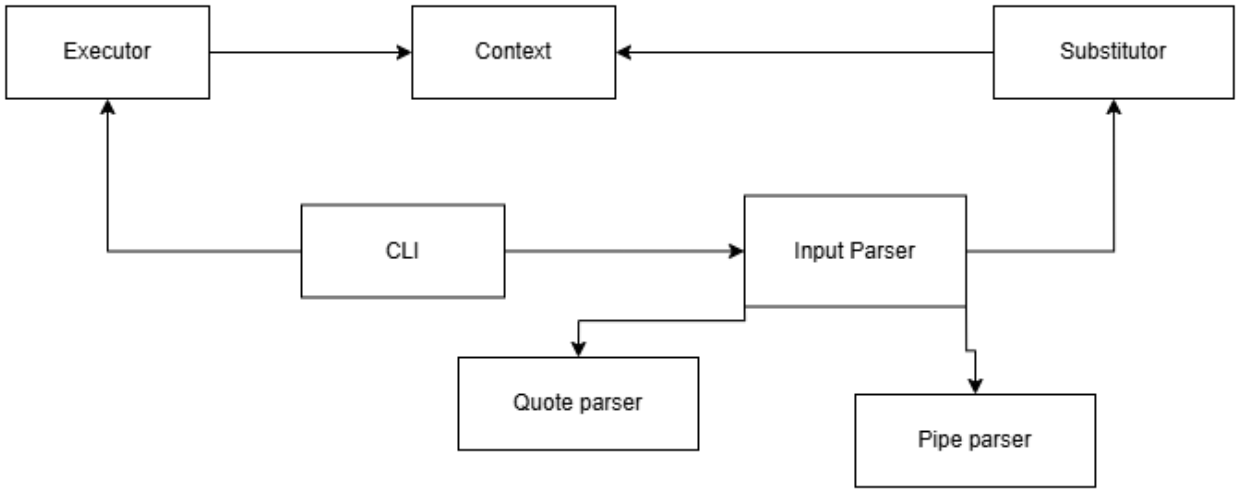
3. Ключевые архитектурные решения

1. **Модульная архитектура:** система разбита на независимые компоненты (CLI, InputParser, Executor и др.).
2. **Расширяемость:** легко добавить новые команды через реализацию интерфейса `Command`.
3. **REPL-цикл:** реализован стандартный цикл "Ввод – Исполнение – Вывод".
4. **Потоковая обработка:** команды связаны через стандартные потоки (input/output), поддерживается конвейер (`|`).
5. **Контекст выполнения:** управление переменными окружения через `Context`.

4. Архитектурные представления

4.1 Логическое представление

Основные компоненты и их взаимодействие:



4.2 Взаимодействие компонентов (последовательность выполнения)

Пример команды: `echo $PATH | wc`

Пользователь → CLI:
Ввод команды "echo $PATH | wc"

CLI → InputParser:
Считывает строку и вызывает парсинг.

InputParser:
- QuoteParser: токенизирует строку.
- Substitutor: заменяет $PATH на значение из Context.
- PipeParser: разбивает на команды [EchoCommand, WcCommand].

CLI → Executor:
Передает список команд.

Executor:
- Запускает EchoCommand → передает output в WcCommand.
- Выполняет WcCommand

---

5. Описание архитектурных компонентов

5.1 Главный модуль: `CLI`

Назначение: Главная точка входа, реализует REPL-цикл.
Функции:

- Читает ввод пользователя.
- Делегирует разбор строки модулю `InputParser`.
- Выполняет команды через `Executor`.
- Обрабатывает исключения.

5.2 Парсинг ввода: `InputParser`

Назначение: Преобразует строку ввода в объекты команд.
Функции:

- Токенизация с учетом кавычек (`QuoteParser`).
- Подстановка переменных (`Substitutor`).
- Разделение по `|` на команды (`PipeParser`).
- Создание объектов `Command`.

5.3 Вспомогательные парсеры

- **QuoteParser:** выделяет строки, заключенные в кавычки.
- **Substitutor:** заменяет переменные окружения.
- **PipeParser:** разделяет команды по `|`.

5.4 Исполнение команд: `Executor`

Назначение: Последовательно выполняет команды, связывает их потоки.
Функции:

- Управляет запуском команд.

- Связывает output одной команды с input следующей.

5.5 Интерфейс команд: `Command`

Назначение: Базовый интерфейс для всех команд.

Поля:

- `args`: аргументы команды.
- `input`, `output`: потоки ввода/вывода.

Методы:

- `execute()`: выполнение логики команды.
- `setInputStream()` и `setOutputStream()`: настройка потоков.

Реализации:

- `CatCommand` — вывод файла.
- `EchoCommand` — вывод текста.
- `WcCommand` — подсчет строк, слов, байтов.
- `PwdCommand` — текущая директория.
- `ExitCommand` — завершение работы.
- `AssignmentCommand` — присваивание переменной.
- `UnknownCommand` — выполнение внешних команд.

5.6 Контекст выполнения: `Context`

Назначение: Хранение переменных окружения.

Методы:

- `getVar(name)`: получение значения переменной.
- `setVar(name, value)`: установка переменной.

---

6. Качественные характеристики

- **Расширяемость:** добавление новых команд через реализацию `Command`.
- **Модульность:** компоненты слабо связаны.
- **Надежность:** обработка исключений в REPL-цикле.
- **Производительность:** использование потоков для пайпов.

---

7. Ограничения и предположения

- Ввод ограничен текстовыми командами.
- Внешние команды выполняются через системный вызов.