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

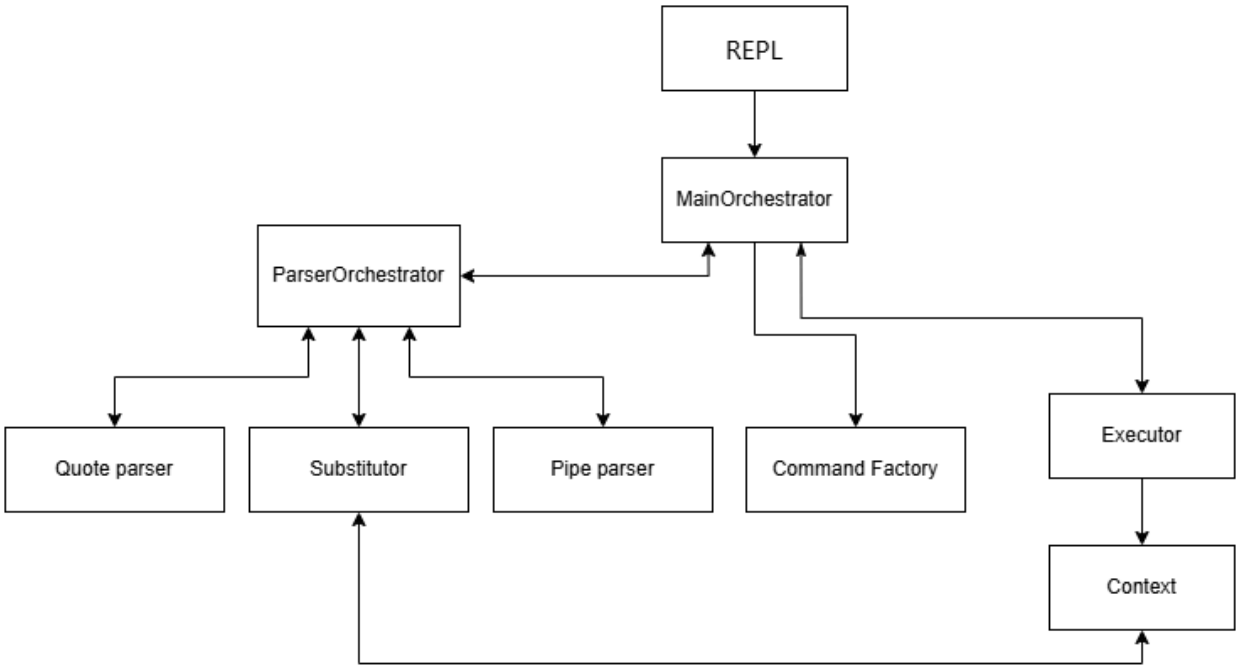
Стейкхолдер	Интересы и требования
Пользователи CLI	Удобный, предсказуемый интерфейс, аналогичный стандартному shell.
Разработчики	Простота добавления команд, модульность, четкое разделение ответственности компонентов.

3. Ключевые архитектурные решения

- **Модульная архитектура**: Система разделена на независимые компоненты (REPL, ParserOrchestrator, Executor и др.).
- **Расширяемость**: Легко добавить новые команды через реализацию интерфейса `Command`.

- **REL-цикл**: Реализован цикл "Ввод – Исполнение".
- **Оркестрация исполнения**: Используется `ParserOrchestrator` для парсинга и `Executor` для исполнения последовательностей команд.
- **Оркестрация парсинга**: `ParserOrchestrator` координирует работу парсеров (токенизация, подстановка переменных, разделение команд).
- **Потоковая обработка**: Команды связаны через стандартные потоки (input/output), поддерживается конвейер (`|`).
- **Контекст выполнения**: Управление переменными окружения через `Context`.

4. Архитектурные представления



4.1 Логическое представление

Основные компоненты и их взаимодействие:

- **REPL**: Управляет циклом ввода-исполнения-ввода, передает данные в `MainOrchestrator`.
- **MainOrchestrator**: Координирует парсинг и исполнение команд.
- **ParserOrchestrator**:
    - Токенизирует ввод через `Queue parser`.
    - Выполняет подстановку переменных через `Substitutor`.
    - Разделяет команды по пайпам через `Pipe parser`.
- **CommandFactory**: Создает объекты команд на основе токенов.
- **Executor**: Последовательно исполняет команды.

- **Context**: Хранит переменные окружения.

4.2 Взаимодействие компонентов (последовательность выполнения)

Пример команды: `echo $PATH | wc`

1. **Пользователь** → `REPL`: Ввод строки `echo $PATH | wc`.
2. **REPL** → `MainOrchestrator`: Передача строки для обработки.
3. **MainOrchestrator** → `ParserOrchestrator`:
    - `Queue parser`: Разбивает строку на токены (`echo`, `$PATH`, `|`, `wc`).
    - `Substitutor`: Заменяет `$PATH` на значение из `Context`.
    - `Pipe parser`: Разделяет токены на список команд: `[echo, wc]`.
4. **MainOrchestrator** → `CommandFactory`: Создает объекты `EchoCommand` и `WcCommand`.
5. **MainOrchestrator** → `Executor`: Передает список команд.
6. **Executor**:
    - Запускает `EchoCommand`, передает его вывод в `WcCommand`.
    - `WcCommand` выводит результат и возвращает код возврата, который кладется в окружение.

5. Описание архитектурных компонентов

5.1 REPL (Read-Execute-Print Loop)

Назначение: Главный цикл приложения.

Функции:

- Чтение ввода пользователя.
- Передача данных в `MainOrchestrator`.
- Вывод ошибок.

5.2 MainOrchestrator

Назначение: Координация парсинга и исполнения команд.

Функции:

- Вызов `ParserOrchestrator` для преобразования строки в список списков токенов.
- Создание списка команд с помощью `CommandFactory`.
- Передача подготовленных команд в `Executor`.

5.3 Парсинг ввода: ParserOrchestrator

Назначение: Преобразует строку ввода в списки списков токенов.

Функции:

- **Токенизация** с учетом кавычек (`QuoteParser`). Разделяет строку по трём типам токенов: строка в двойных кавычках, строка в одинарных кавычках, остальное.
- **Подстановка переменных (`Substitutor`)**. Подставляет переменные контекста в токены (нерекурсивно).
- **Разделение по `|` на команды (`PipeParser`)**.

5.4 Исполнение команд: Executor

Назначение: Последовательно выполняет команды, связывает их потоки.

Функции:

- Последовательное исполнение команд.
- Возврат кода завершения.

5.5 Создание команд: CommandFactory

Назначение: Создание объектов команд на основе токенов.

Функции:

- Определение типа команды (встроенная, внешняя, присваивание).
- Инициализация соответствующих классов (`EchoCommand`, `AssignmentCommand`).

5.6 Интерфейс команд: Command

Назначение: Базовый интерфейс для всех команд.

Поля:

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

Методы:

- `execute()`: выполнение логики команды.
- `setInputStream()`, `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.

5.7 Контекст выполнения: Context

Назначение: Хранение переменных окружения (включая код возврата).

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

- Поддерживается только текстовый ввод/вывод.
- Внешние команды исполняются через системные вызовы.
- Переменные окружения не сохраняются между сессиями.