Санкт-Петербургский государственный университет
Кафедра системного программирования

Расширение функциональности отладчика в IDE для языков C/C++ на платформе Eclipse

Друмов Максим Андреевич

Научный руководитель: к.ф.-м.н. Д. В. Луцив, доцент кафедры системного программирования
Консультант: П. А. Бабанов, старший программист-разработчик ПО I категории, ООО «Софтком»

Рецензент: В. В. Оносовский, исполнительный директор, ООО «Софтком»

Санкт-Петербург
2025

Введение

- **Отладка** является важным этапом разработки программного обеспечения, так как позволяет своевременно выявлять и устранять ошибки
- **CLDT(Clang Development Tool)** — IDE для C/C++ на базе Eclipse Platform, разрабатываемая в компании ООО «Софтком»
- **Ограниченность** текущей реализации
  - Поддерживается только архитектура MIPS64
  - Отсутствует удалённая отладка
  - Ограниченная расширяемость отладчика
- **Расширение** функциональности отладчика
  - Адаптация базовой функциональность отладчика
  - Расширяемость под разные архитектуры и форматы
  - Удалённая отладка

Постановка задачи

Целью работы является расширение функциональности отладчика в *IDE* для языков *C/C++* на платформе *Eclipse*. Для её выполнения были поставлены следующие задачи:

1. Провести обзор текущей реализации отладчика и существующих решений для выявления требований
2. Доработать архитектуру отладчика с учётом требований расширяемости и поддержки удалённой отладки
3. Реализовать:
   - обработку форматов исполняемых файлов и отладочной информации(ELF/DWARF)
   - отображение регистров
   - отображение стека вызовов
   - программные точки останова
   - пошаговое выполнение (с заходом, выходом и обходом)
   - отображение переменных
   - промежуточный отладочный формат
4. Провести тестирование

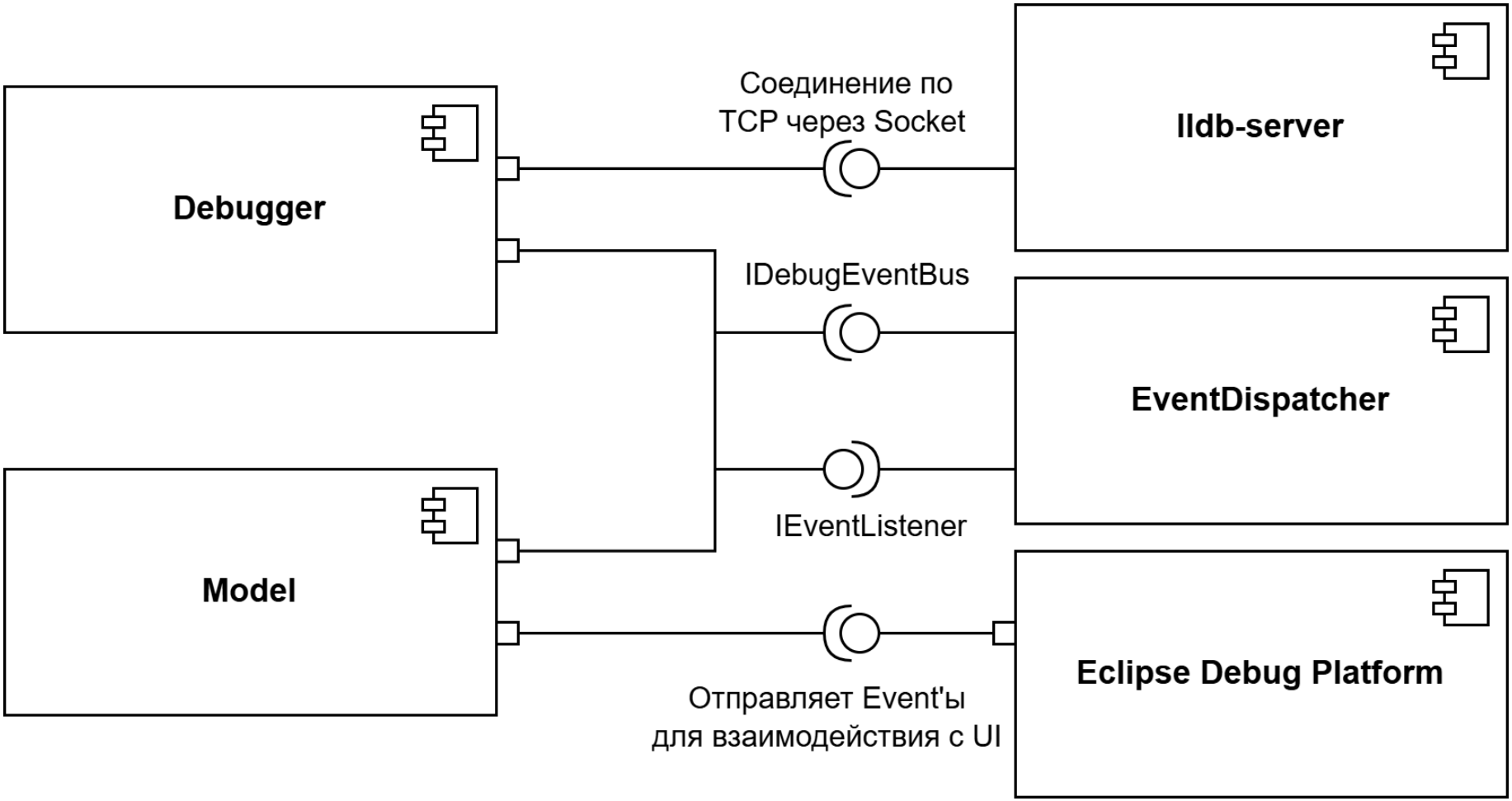
Обзор

Рассмотрены IDE, которые используют LLVM и LLDB для отладки:

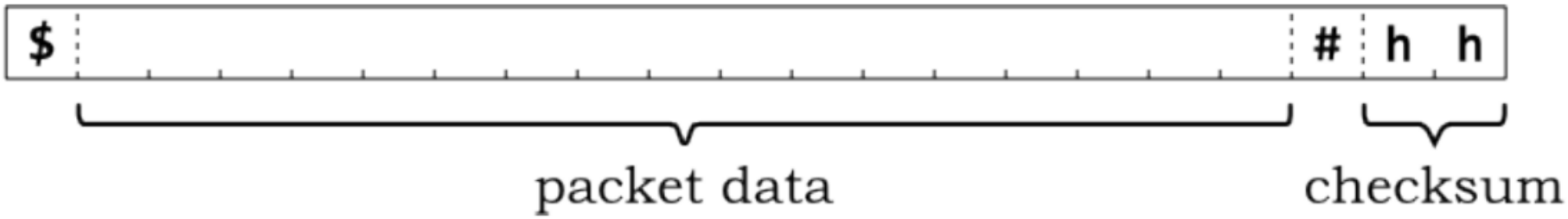
IDE	LLDB	lldb-server	Кроссплатформ.	Тип лицензии	Доступность
Visual Studio	–	+	–	Коммерческая (Microsoft)	+
Eclipse CDT	+/–	–	+	Открытая (EPL-2.0)	+
VS Code	+/–	+/–	+/–	Открытая (MIT) / Комм. (Microsoft)	+
Xcode	+	+	–	Коммерческая (Apple)	+
CLion	+	+	+	Коммерческая (JetBrains)	–

Проанализирована текущая реализация отладчика.

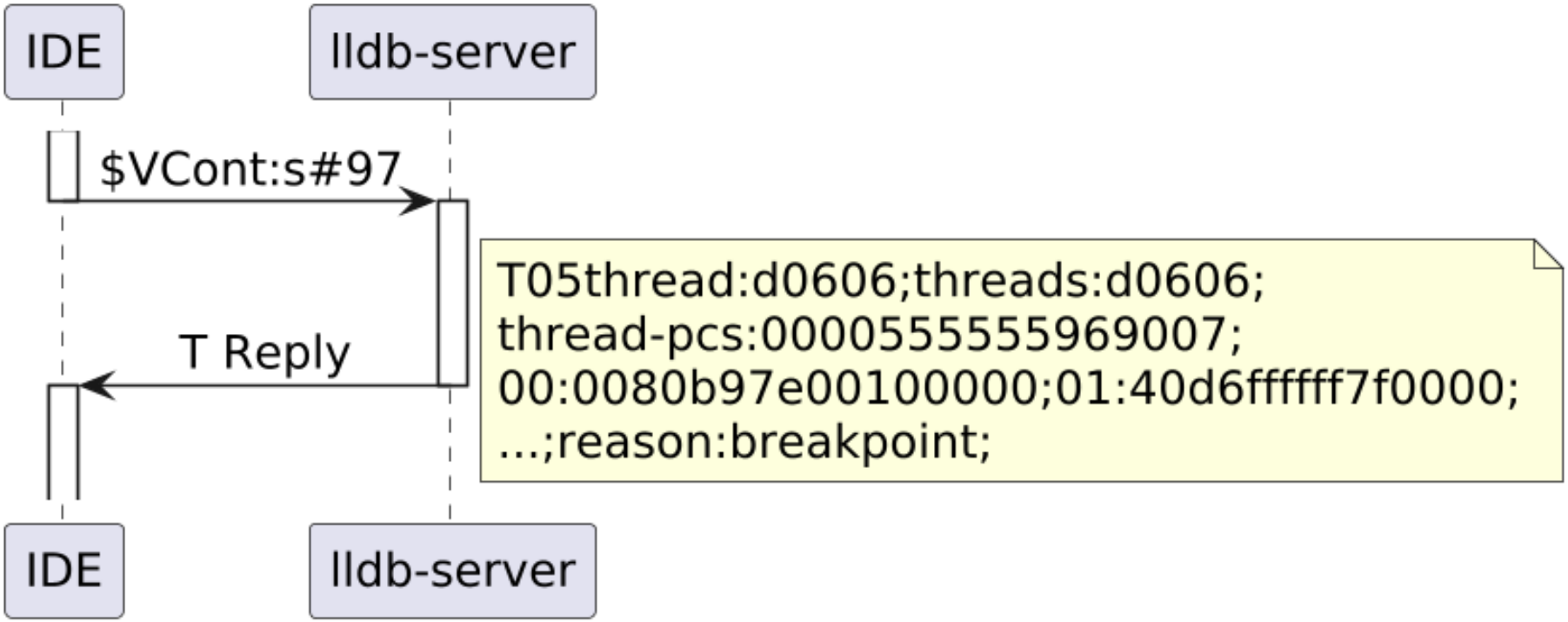
Архитектура: Отладчик



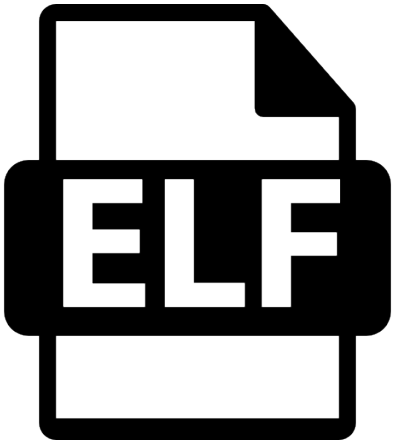
Архитектура: Remote Serial Protocol



Архитектура: Remote Serial Protocol



Реализация: Базовая функциональность



Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 11
  - New_configuration (4) [Debug process]

main.c ×

```c
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 11
  - New_configuration (4) [Debug process]

main.c ×

```c
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 13
  - New_configuration (4) [Debug process]

main.c ×

```c
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 4
      - main.c, line 13
  - New_configuration (4) [Debug process]

main.c ×

```
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 4
      - main.c, line 13
  - New_configuration (4) [Debug process]

main.c ×

```c
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 5
      - main.c, line 13
  - New_configuration (4) [Debug process]

main.c ×

```
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 6
      - main.c, line 13
  - New_configuration (4) [Debug process]

main.c ×

```
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 14
  - New_configuration (4) [Debug process]

main.c ×

```
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Debug × Project Explorer

- New_configuration (4) [Локальный запуск]
  - New_configuration (4) [Debug configuration]
    - Thread ID:84f3
      - main.c, line 16
  - New_configuration (4) [Debug process]

main.c ×

```
1  #include <stdio.h>
2
3  int sum_array(int* arr, int size) {
4      int sum = 0;
5      for (int i = 0; i < size; i++) {
6          sum += arr[i];
7      }
8      return sum;
9  }
10
11 int main() {
12     int data[] = {1, 2, 3, 4, 5};
13     int total = sum_array(data, 5);
14     printf("Сумма: %d\n", total);
15
16     return 0;
17 }
```

Реализация: Базовая функциональность

Variables | Breakpoints | Registers | Expressions

Name	Value
data	int [] <0x7fffffffe4a0>
int	1
int	2
int	3
int	4
int	5
total	15

Реализация: Базовая функциональность

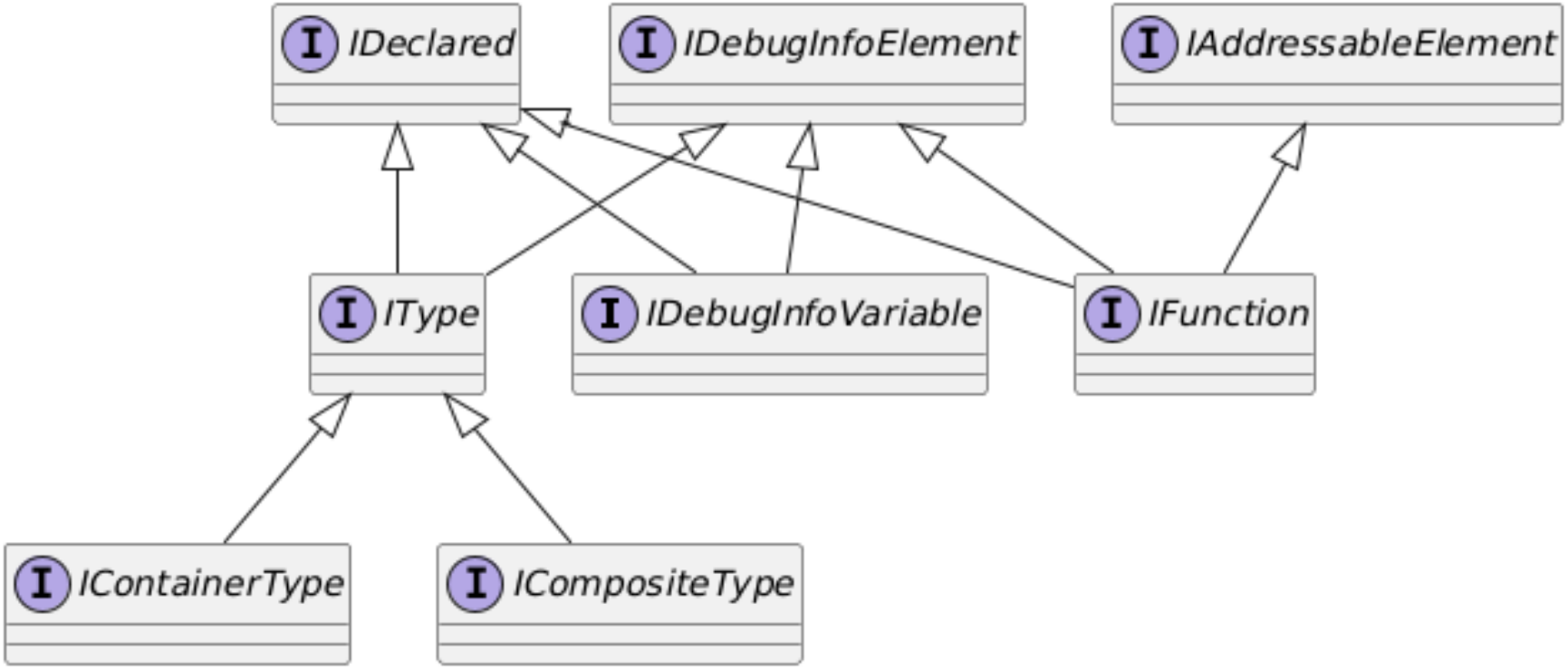
Variables | Breakpoints | Registers | Expressions

Name	Value
Advanced Vector Extensions	
General Purpose Registers	
rax	0x555555555190
rbx	0x7fffffffe5e8
rcx	0x555555557dc8
rdx	0x7fffffffe5f8
rdi	0x1
rsi	0x7fffffffe5e8
rbp	0x7fffffffe560
rsp	0x7fffffffe4c8
r8	0x0
r9	0x7ffff7fca380
r10	0x7fffffffe1e0
r11	0x203
r12	0x1
r13	0x0
r14	0x555555557dc8
r15	0x7ffff7ffd000
rip	0x555555555190
rflags	0x246
cs	0x33

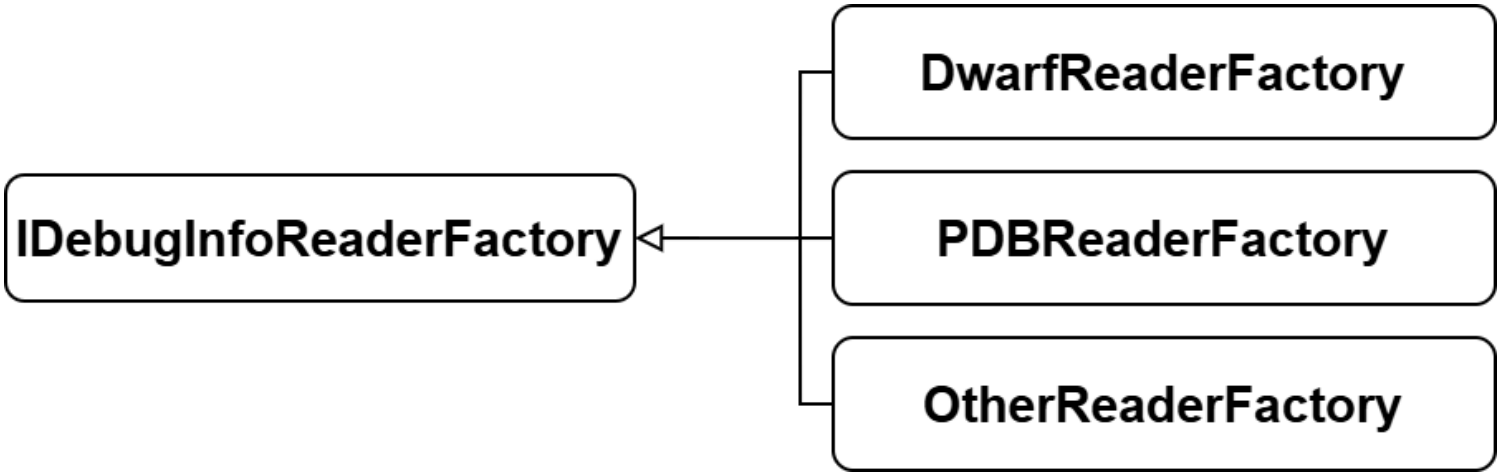
Реализация: Промежуточный отладочный формат

- **Типы данных:** примитивные типы, указатели, массивы, структуры, перечисления, шаблонные типы
- **Переменные:** параметры функций, члены классов и структур
- **Функции:** адреса, аргументы, возвращаемые типы, inline
- **Файлы и строки:** соответствие между адресами инструкций и исходным кодом
- **Контексты:** области видимости, пространства имён, классы и блоки

Реализация: Промежуточный отладочный формат



Реализация: Промежуточный отладочный формат



Тестирование

Инструменты: для модульного тестирования использовались JUnit 5 и Mockito. Эталонные данные для проверки формировались с помощью Python и библиотеки pyelftools в формате YAML.

Проверяемая функциональность:

- Преобразование DWARF в промежуточный формат
- Интерпретация выражений IDF[1]
- Формирование и отправка RSP-команд
- Обработка ответов сервера и генерация событий
- Покрытие более 80%

[1] IDF — промежуточный отладочный формат

Апробация

- Взаимодействие с заказчиком и проведение демонстраций осуществлялись на протяжении всей разработки
- Демонстрации проводились на реальных проектах компании с участием представителей заказчика
- Подтверждена корректная работа базовой функциональности на нескольких дистрибутивах Linux
- Были обнаружены ограничения, связанные с объёмом отладочной информации на крупных проектах
- Собрана обратная связь от заказчика и зафиксированы приоритетные требования к доработке системы

Результаты

1. Проведен обзор текущей реализации и анализ существующих решений
2. Доработана архитектура отладчика с учётом требований расширяемости и поддержки удаленной отладки
3. Выполнена реализация:
   - доработана обработка форматов ELF/DWARF
   - программные точки останова ставятся через выпадающего меню обзорной линейки, а также по двойному нажатию
   - поддержано пошаговое выполнение (с заходом, выходом и обходом)
   - реализовано отображение регистров, стека вызовов и переменных
   - реализованы промежуточный отладочный формат и возможность поддержки новых форматов через Eclipse Extension Point
4. Проведено модульное тестирование фабрики отладочного формата DWARF и клиента протокола RSP. Покрытие составляет более 80%. Проведена апробация

Исходный код закрыт.