



Автоматизация миграции программного кода на новый набор библиотек

Артем Алексюк

14 октября 2016 г.

Санкт-Петербургский политехнический университет Петра Великого
JetBrains Research

Миграция в новое библиотечное окружение:

- Новая программная платформа
- Новая аппаратная платформа
- Новая версия библиотеки
- Унаследованный (несовместимый) код
- ...

Сложности

- Практически всегда миграция идет вручную
- Много рутинной работы \implies высокая вероятность внесения ошибок
- Однотипные действия при миграции нескольких проектов

Требуется автоматизация

Пример задачи миграции

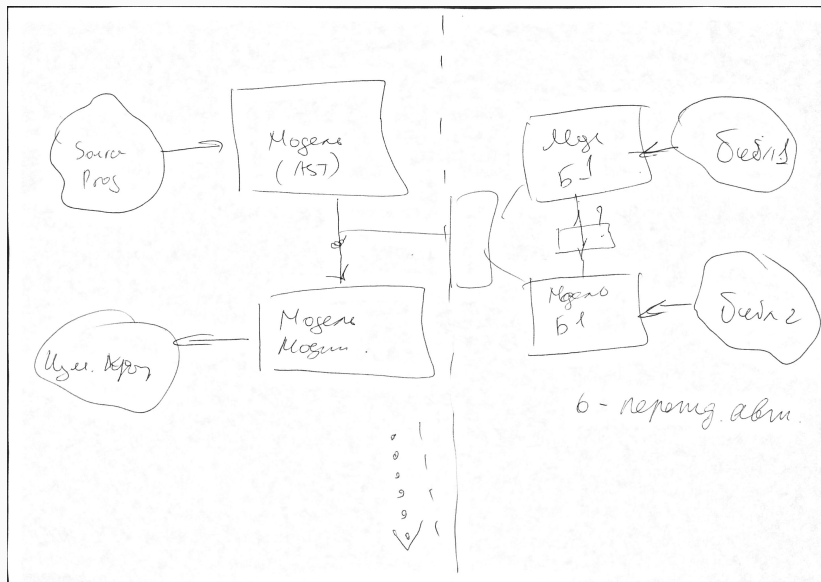
Программа, использующая использующая библиотеку `java.net.URLConnection`:

```
URL url = new URL("http://api.ipify.org/");  
URLConnection conn = url.openConnection();
```

Программа, использующая использующая библиотеку `Apache HTTP Client`:

```
CloseableHttpClient httpClient =  
    ↪ HttpClient.createDefault();  
HttpGet httpget = new  
    ↪ HttpGet("http://api.ipify.org/");  
CloseableHttpResponse httpResponse =  
    ↪ httpClient.execute(httpget);
```

Общий подход к решению задачи

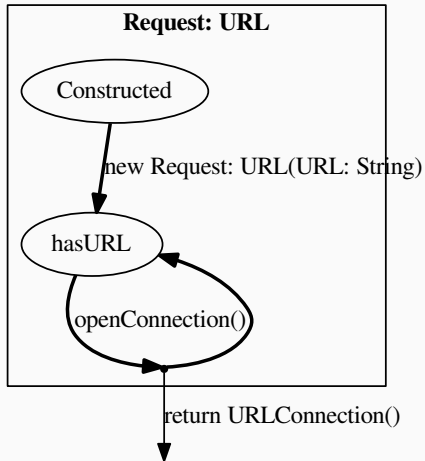


Модель библиотеки

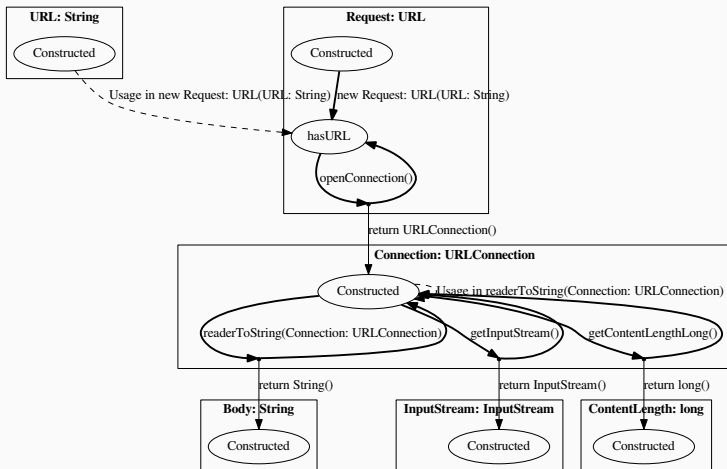
Для описания моделей используется набор расширенных конечных автоматов

- Каждый переход в автомате - действие над библиотекой (вызов функции, конструктора и так далее)
- В процессе перехода могут порождаться новые автоматы
- Автомат может иметь атрибуты, и переходы могут зависеть от них (иметь условия перехода)

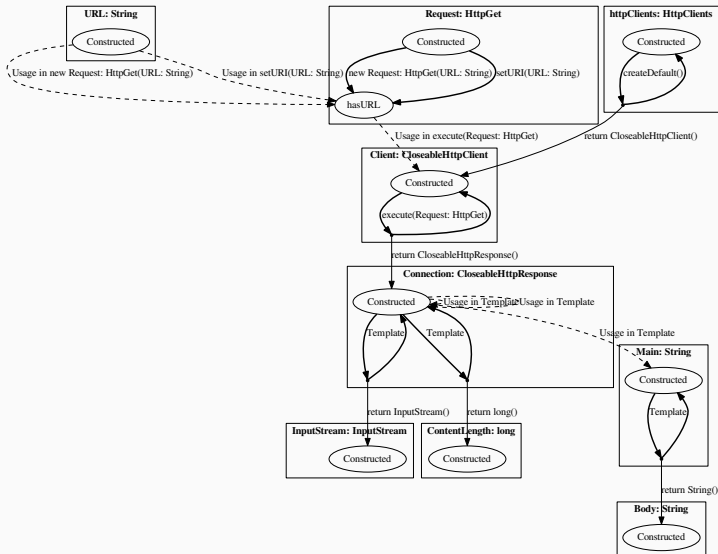
Пример автомата



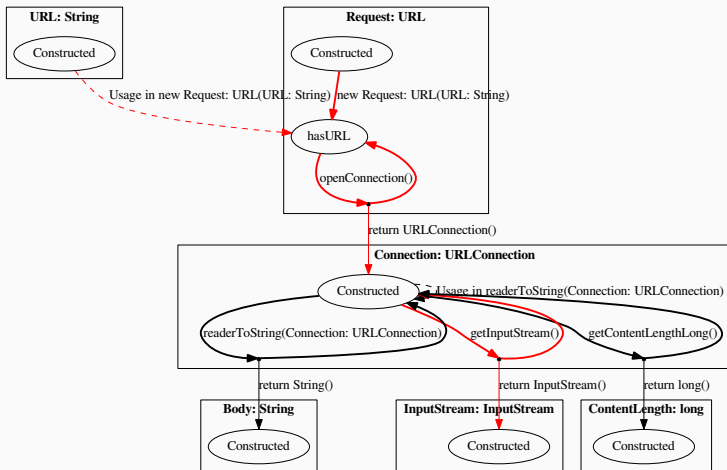
Фрагмент модели библиотеки HttpURLConnection



Фрагмент модели библиотеки Apache HTTP Client



Задействованные переходы в автоматах



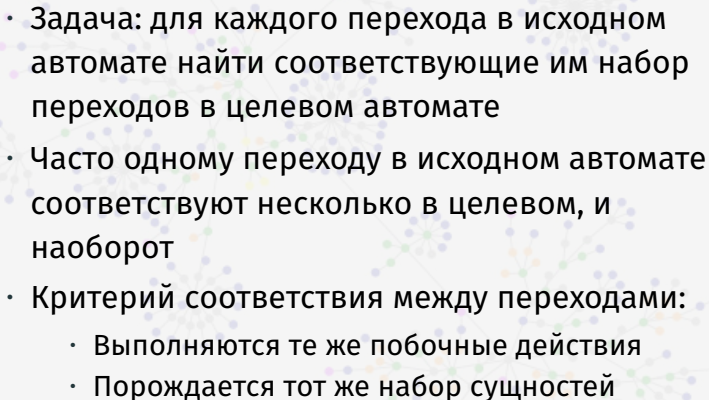
Анализ портируемой программы

- Задача: необходимо связать вызовы API библиотеки в портируемой программе и переходы в автомате
- Простейший вариант решения: получаем AST портируемой программы, ищем в нем все вызовы библиотечных методов и по имени метода выбираем переход в автомате
- Проблема: некоторую информацию практически невозможно получить из исходного кода

Анализ портируемой программы

- В дополнение можно использовать динамический анализ: программа инструментируется, составляется трасса выполнения, в которой фиксируются значения аргументов.
- Подразумевается, что у портируемой программы имеется достаточный(?) набор тестов

Поиск соответствия между переходами

- 
- Задача: для каждого перехода в исходном автомате найти соответствующие им набор переходов в целевом автомате
 - Часто одному переходу в исходном автомате соответствуют несколько в целевом, и наоборот
 - Критерий соответствия между переходами:
 - Выполняются те же побочные действия
 - Порождается тот же набор сущностей

Поиск соответствия между переходами

- Изначальный подход
 - модель переводится в граф и задача сводится к поиску пути на графе
 - - не учитываются побочные действия и условия переходов
- Текущий подход:
 - из-за наличия условий, обычный граф не подходит
 - сеть автоматов сводится к динамическому графу
 - используется модифицированный волновой алгоритм (учитывающий побочные действия)

Прототип инструмента

Рассмотренные идеи реализуются в прототипе инструмента миграции для языка Java:

- сформирован DSL на Kotlin для описания моделей библиотек
- реализовано получение трассы выполнения мигрируемой программы (с использованием AOP)
- производится портирование простых искусственных программ
- создается система проверки корректности преобразования

Фрагмент DSL-описания модели библиотеки

```
val url = StateMachine(entity = HTTPEntities.url)
val request = StateMachine(entity = HTTPEntities.request)
val hasURL = State(name = "hasURL", machine = request)
ConstructorEdge(
    machine = request,
    src = request.getDefaultState(),
    dst = hasURL,
    param = listOf(Param(machine = url))
)
```


Пример миграции. Программа «До»

```
URL url = new URL("http://api.ipify.org/");
URLConnection conn = url.openConnection();
if (conn.getContentLengthLong() > 0) {
    String response = new BufferedReader(new
    ↪ InputStreamReader(conn.getInputStream()))
        .lines().collect(Collectors.joining("\n"));
    System.out.println(response);
} else {
    System.out.println("Error!");
}
```

Результат миграции. Программа «После»

```
HttpGet url = new HttpGet("http://api.ipify.org/");
CloseableHttpClient newMachine_Client_0 =
    ↪ HttpClient.createDefault();
CloseableHttpResponse conn = newMachine_Client_0.execute(url);
long linkedEdge_ContentLength_1 =
    ↪ conn.getEntity().getContentLength();
if (linkedEdge_ContentLength_1 > 0) {
    InputStream linkedEdge_InputStream_2 =
    ↪ conn.getEntity().getContent();
    String response = new BufferedReader(new
    ↪ InputStreamReader(linkedEdge_InputStream_2)).lines().collect(Collectors.toList());
    System.out.println(response);
} else {
    System.out.println("Error!");
}
```

Дальнейшее развитие подхода

- Развитие метода поиска соответствия между автоматами
- Разработка метода сопоставления побочных действий
- TBD 1 *
- TBD 2 *
- ...
- TBD n *

* В процессе обсуждения

Дальнейшее развитие прототипа инструмента

- Проектирование языка для описания моделей (сейчас это ad-hoc решение)
- Расширение поддержки модели в прототипе
- (Существенное) пополнение репозитория моделей библиотек
- Создание репозитория тестовых программ для миграции
- Тестирование инструмента на реальных проектах

Email: `aleksyuk@kspt.icc.spbstu.ru`

Github:

`https://github.com/h31/LibraryMigration`