



# Автоматизация миграции программного кода на новый набор библиотек

---

Артем Алексюк

15 октября 2016 г.

Санкт-Петербургский политехнический университет Петра Великого  
JetBrains Research

Миграция (портирование) 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

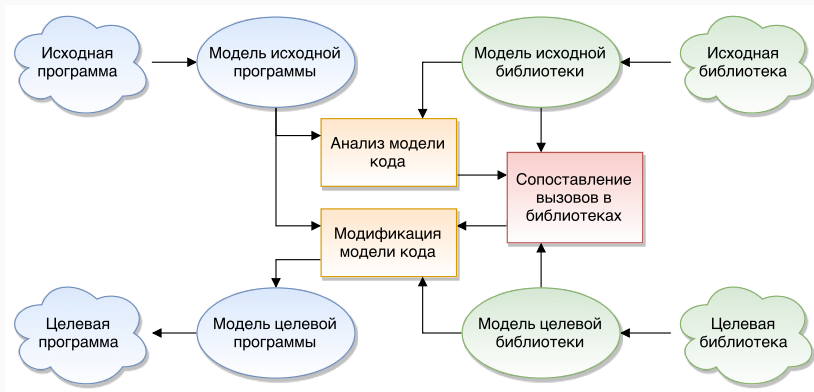
Программа, использующая библиотеку  
`java.net.URLConnection`:

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

Программа, использующая библиотеку Apache  
`HttpClient`:

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

# Общая схема решения задачи



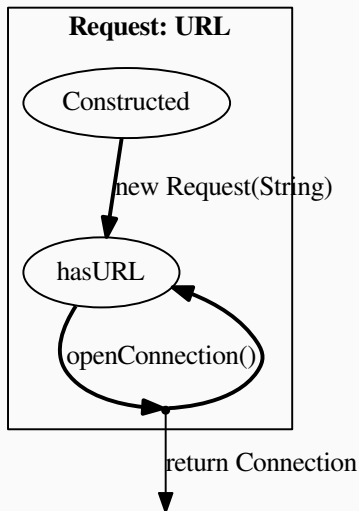
# Модель библиотеки

Для описания моделей используется набор расширенных конечных автоматов

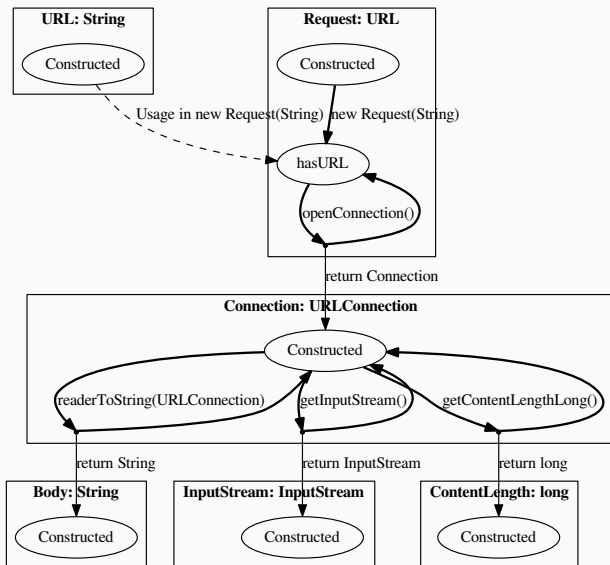
- Каждый переход в автомате — действие над библиотекой
- В процессе перехода могут порождаться новые автоматы
- Автомат может иметь атрибуты, и переходы могут зависеть от них
- Для переходов можно указать их побочные эффекты

# Пример автомата

\* Иллюстрации сгенерированы автоматически

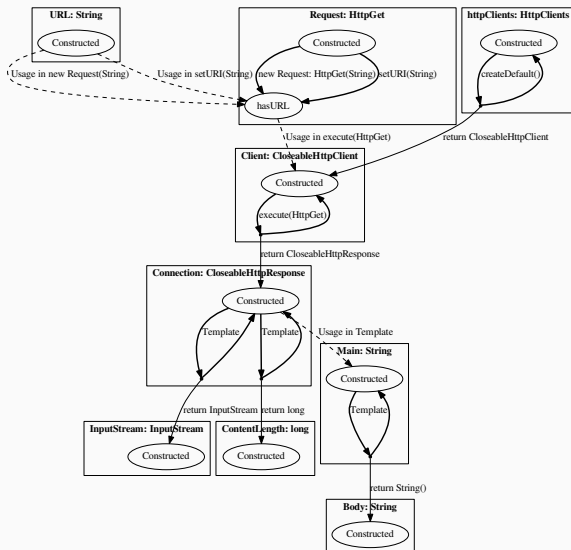


# Фрагмент модели библиотеки HttpURLConnection

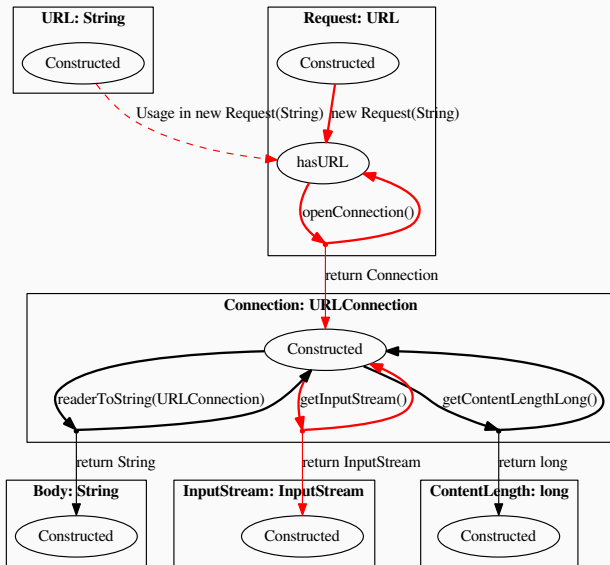




# Фрагмент модели библиотеки Apache HttpClient



# Задействованные переходы в автоматах



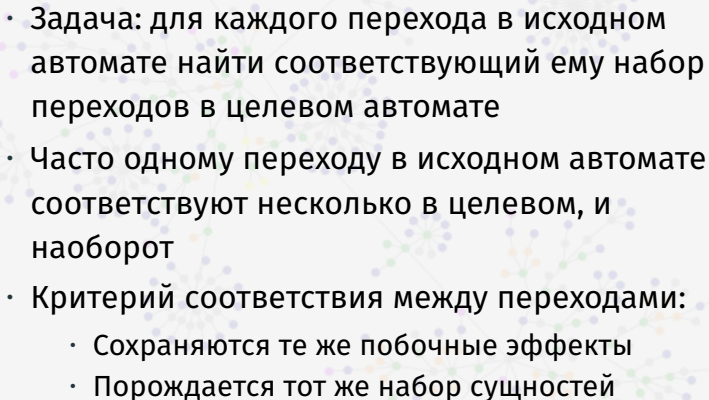
# Анализ портируемой программы

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

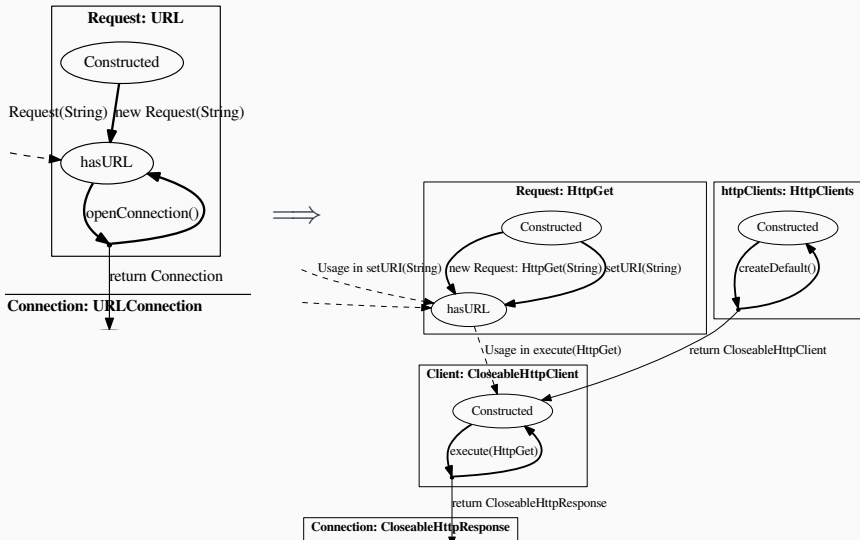
# Анализ портируемой программы

- В дополнение можно использовать динамический анализ: программа инструментируется, составляется трасса выполнения, в которой фиксируется необходимая информация.
- Подразумевается, что у портируемой программы имеется достаточный набор тестов

# Поиск соответствия между переходами

- 
- Задача: для каждого перехода в исходном автомате найти соответствующий ему набор переходов в целевом автомате
  - Часто одному переходу в исходном автомате соответствуют несколько в целевом, и наоборот
  - Критерий соответствия между переходами:
    - Сохраняются те же побочные эффекты
    - Порождается тот же набор сущностей

# Поиск соответствия между переходами



# Поиск соответствия между переходами

- Изначальный подход
  - модель переводится в граф и задача сводится к поиску пути на графе
  - ! не учитываются побочные эффекты и условия переходов
- Текущий подход:
  - из-за требования сохранить побочные эффекты, обычный граф не подходит
  - сеть автоматов сводится к графу с атрибутами
  - используется модифицированный волновой алгоритм (учитывающий побочные эффекты)

# Прототип инструмента

Рассмотренные идеи реализуются в прототипе инструмента миграции для языка Java:

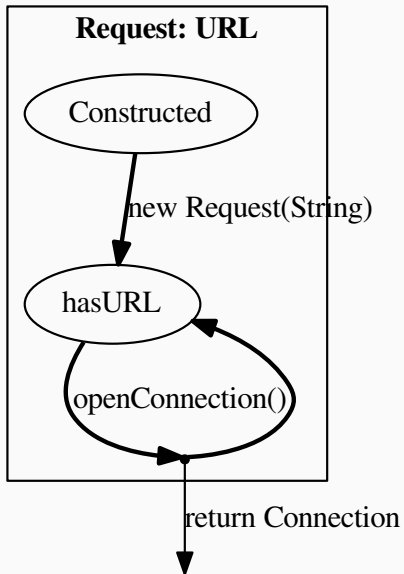
- сформирован DSL на Kotlin для описания моделей библиотек
- реализовано получение трассы выполнения мигрируемой программы (с использованием AspectJ)
- производится портирование простых искусственных программ
- создается система проверки корректности преобразования



## Фрагмент DSL-описания модели библиотеки

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

## Пример автомата



## 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 =
    ↪ HttpClientClients.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.joining("\n"));
    System.out.println(response);
} else {
    System.out.println("Error!");
}
```

# Дальнейшее развитие подхода

- Развитие метода поиска соответствия между автоматами
- Разработка метода сопоставления побочных эффектов
- ...

---

\* 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

# Контакты

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

Github:

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

Спасибо за внимание!