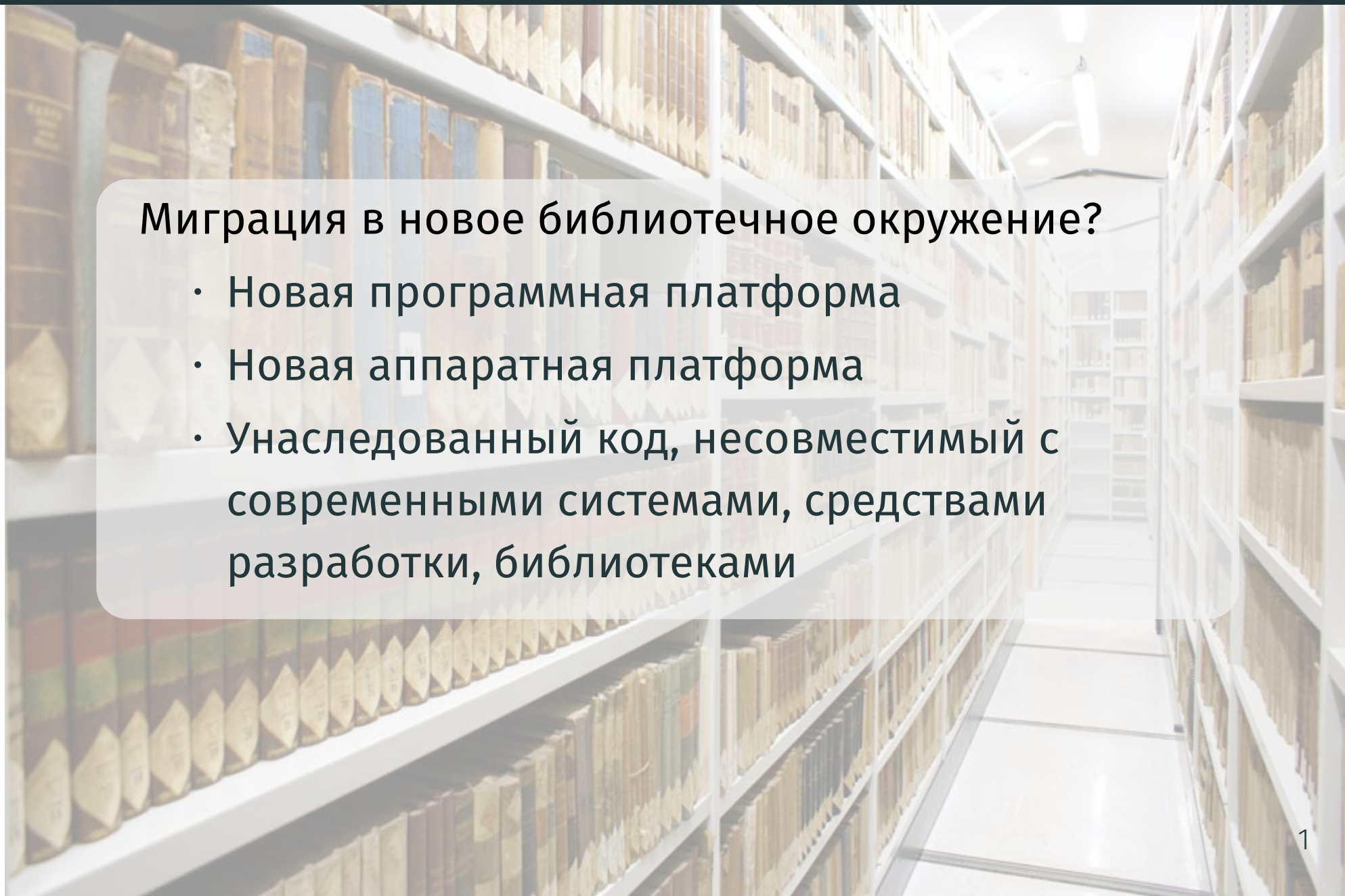
Автоматизация миграции программного кода на новый набор библиотек

Артем Алексюк

13 октября 2016 г.

Санкт-Петербургский политехнический университет Петра Великого

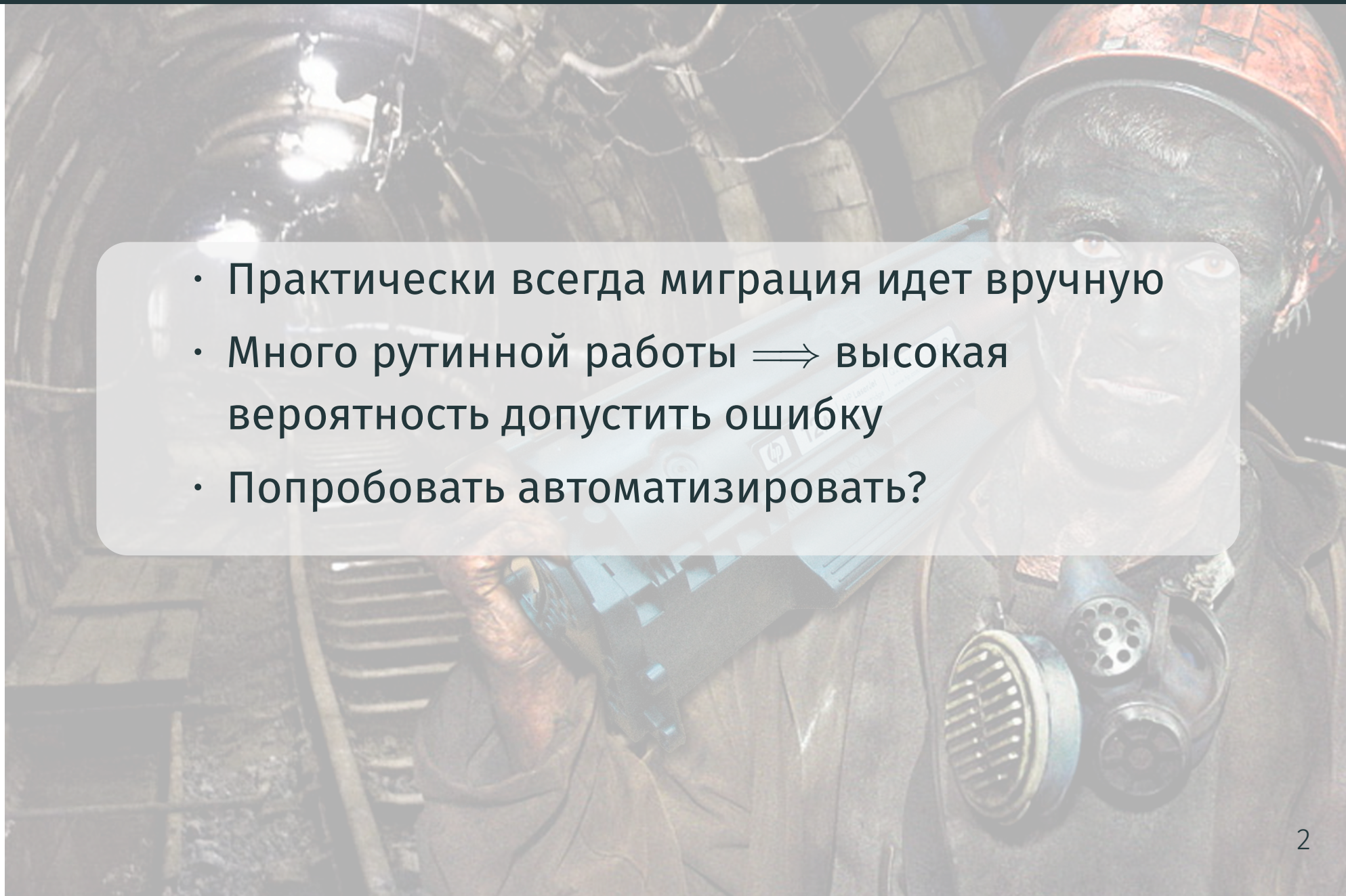
Актуальность



Миграция в новое библиотечное окружение?

- Новая программная платформа
- Новая аппаратная платформа
- Унаследованный код, несовместимый с современными системами, средствами разработки, библиотеками

Сложности



- Практически всегда миграция идет вручную
- Много рутинной работы $\Longrightarrow$ высокая вероятность допустить ошибку
- Попробовать автоматизировать?

Общий подход к решению задачи

- Создать модели используемых библиотек
- Модель - частная спецификация, описывающая внешнее поведение библиотеки
- Проанализировать модели, рассчитать для них изменения
- Преобразовать изменения обратно в код

Почему модели, а не сам код?

Удобнее анализировать декларативные модели, чем императивный код

Спецификация библиотек

Уже есть языки для описания библиотек (например, IDL для RPC), но они не содержат семантическую информацию. Модель библиотеки должна:

- Детально описывать внешний интерфейс библиотеки;
- Задавать возможные протоколы использования библиотеки;
- Специфицировать побочные действия библиотеки – влияние ее на окружение;
- Явно вводить семантические описания поведения библиотек.

Формализм

- Для описания моделей используются расширенные конечные автоматы:
  $A =< Q, Q_0, X, V, C, T >$
  C - множество стимулов (например, вызовов функций), $_i$ – появление i-ой стимула
  C_i^A - множество семантических действий, инициируемых запуском функции при условии истинности C_i^{CondA}
  C_i^D - множество дочерних автоматов, запускаемых функцией при условии C_i^{CondD};

Формализм

Для описания моделей используется набор расширенных конечных автоматов:

- L – основной расширенный конечный автомат, описывающий поведение всей библиотеки;
- Si – i-ый дочерний расширенный конечный автомат, запускаемый при достижении определенных условий;

Формализм

Дочерний конечный автомат:

$A =< Q, Q_0, X, V, C, T >$

C - множество стимулов (например, вызовов функций), $_i$ – появление i-ой стимула

C_i^A - множество семантических действий, инициируемых запуском функции при условии истинности C_i^{CondA}

C_i^D - множество дочерних автоматов, описывающих сущности и их экземпляры (создаются при истинном C_i^{CondD})

Формализм

Дочерний конечный автомат:

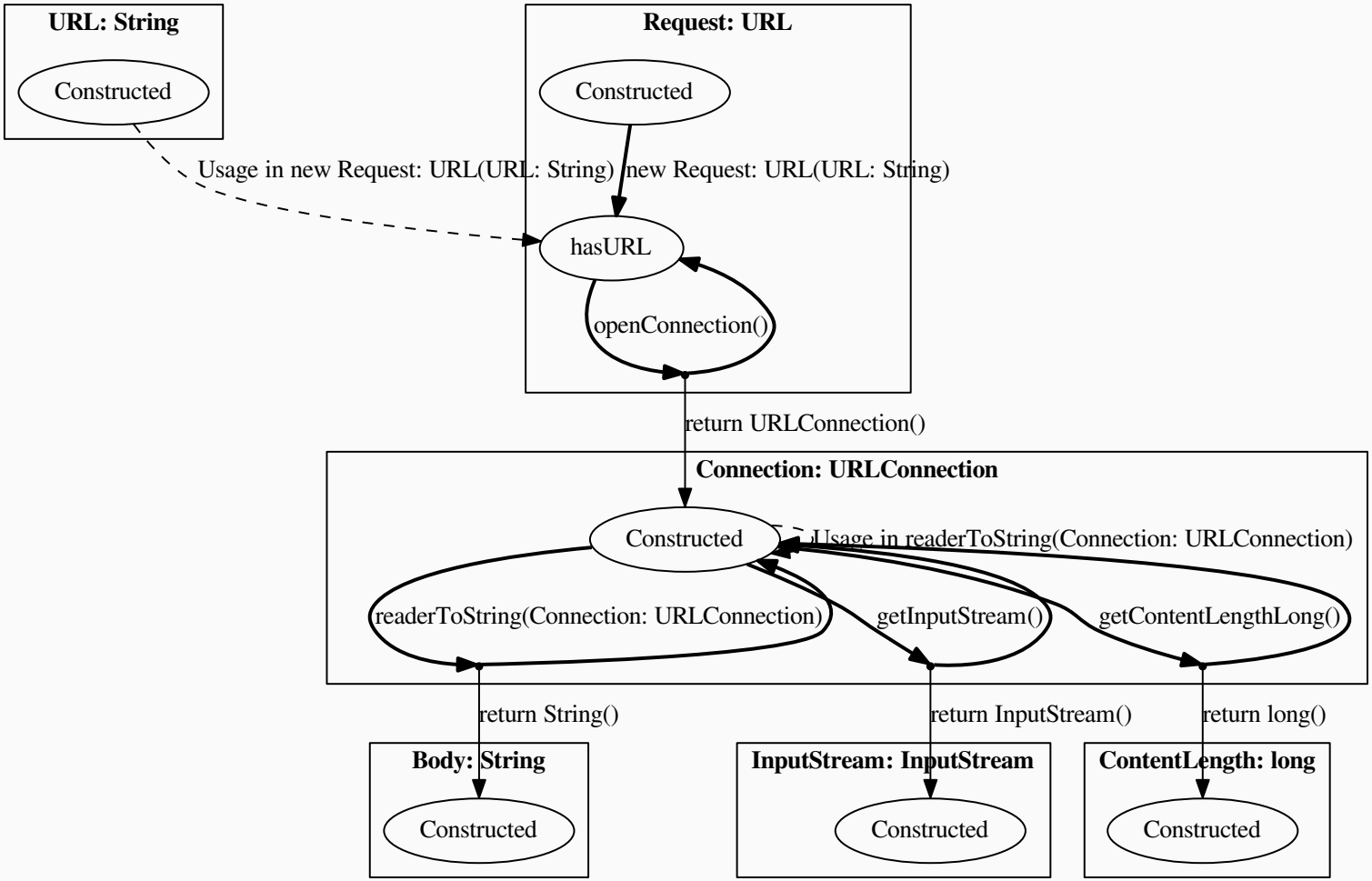
$A =< Q, Q_0, X, V, C, T >$

C - множество стимулов (например, вызовов функций), $_i$ – появление i-ой стимула

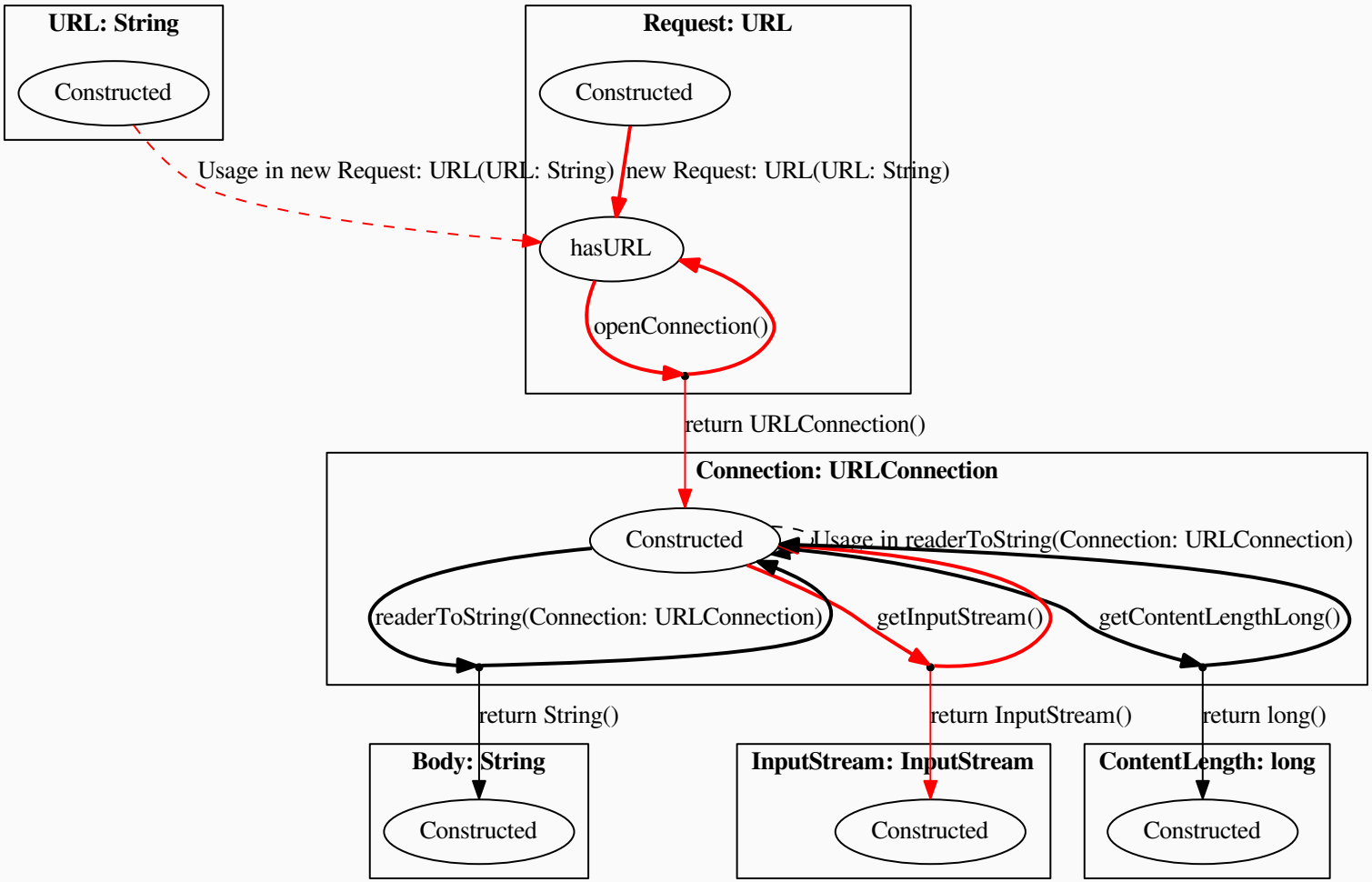
C_i^A - множество семантических действий, инициируемых запуском функции при условии истинности C_i^{CondA}

C_i^D - множество дочерних автоматов, запускаемых функцией при условии C_i^{CondD};

Пример автомата



Пример автомата



Извлечение информации из кода

- Статический анализ: используя одно из подходящий представлений программы (CFG, SSA, Def-Use chains), попытаться отобразить порядок действий на модель
- Проблема - иногда нужно знать, какое значение хранится в переменной. Статический анализ не всегда может дать ответ.

Извлечение информации из кода

- Динамический анализ: программа инструментируется, на основе трассы выполнения составляется путь в модели
- Для инструментирования используется AOP (аспекты) в реализации AspectJ
- Преимущества: проще в реализации, выше точность
- Бонус: проверка корректности использования исходной библиотеки

Поиск соответствия между моделями

При переносе необходимо сохранить:

- Набор действий, совершенных над библиотекой
- Зависимости по данным

Поиск соответствия между моделями

- Модель переводится в граф и задача сводится к поиску кратчайшего пути на графе
- Первые версии инструмента: использовался алгоритм Дейкстры. Недостаток подхода: не все связи можно представить с помощью графа
- Сейчас: используется модификация волнового алгоритма
- В поисках более оптимального подхода

Поиск соответствия между моделями

Зависимости по данным:

- Сущности, которые необходимы для осуществления перехода (например, аргументы функции), отмечены в модели специальными связями
- Если для перехода требуется сущность, которая отсутствует в текущем контексте, ищем способ создания этой сущности

Текущее состояние

- Модель библиотеки, где можно описать состояния, связи и прототипы функций.
- Описание моделей с помощью DSL на Kotlin
- Инструмент для миграции программ на языке Java
- Система проверки корректности преобразования

Пример миграции

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!");
}
```

Пример миграции

После:

```
HttpGet url = new HttpGet("http://api.ipify.org/");
CloseableHttpClient newMachine_Client_0 = HttpClients.
    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

- Улучшение алгоритма поиска соответствия, повышение производительности + увеличение возможностей
- Проектирование языка для описания моделей (сейчас это ad-hoc решение)
- Поиск способа задания соответствия между семантическими доменами
- Увеличение количества информации, содержащейся в моделях