Automated Semantics-Driven Source Code Migration: a Pilot Prototype

Artyom Aleksyuk, Vladimir Itsykson

June 26, 2017

Peter the Great St. Petersburg Polytechnic University
JetBrains Research

Introduction

Migration to a new library environment

- Transition to a new software platform
  - Mobile $\Longleftrightarrow$ Desktop
  - Server $\Longleftrightarrow$ PC
- Addition of a new library in the project
- Upgrade of an existing library
- Replacement of the library
- ...

Migration sample

Code fragment which uses java.net.URLConnection class from the Java Class Library:

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

Code fragment which uses Apache HttpClient library:

```
HttpClient httpclient =
  ↪  HttpClients.createDefault();
HttpGet httpget = new
  ↪  HttpGet("http://api.ipify.org/");
HttpResponse httpResponse =
  ↪  httpclient.execute(httpget);
```

Difficulties

- Usually the code is migrated manually
- Migration requires a lot of tedious work $\Longrightarrow$ has a tendency to introduce new defects in the code
- Identical actions while migrating several projects

Automation is neccesary

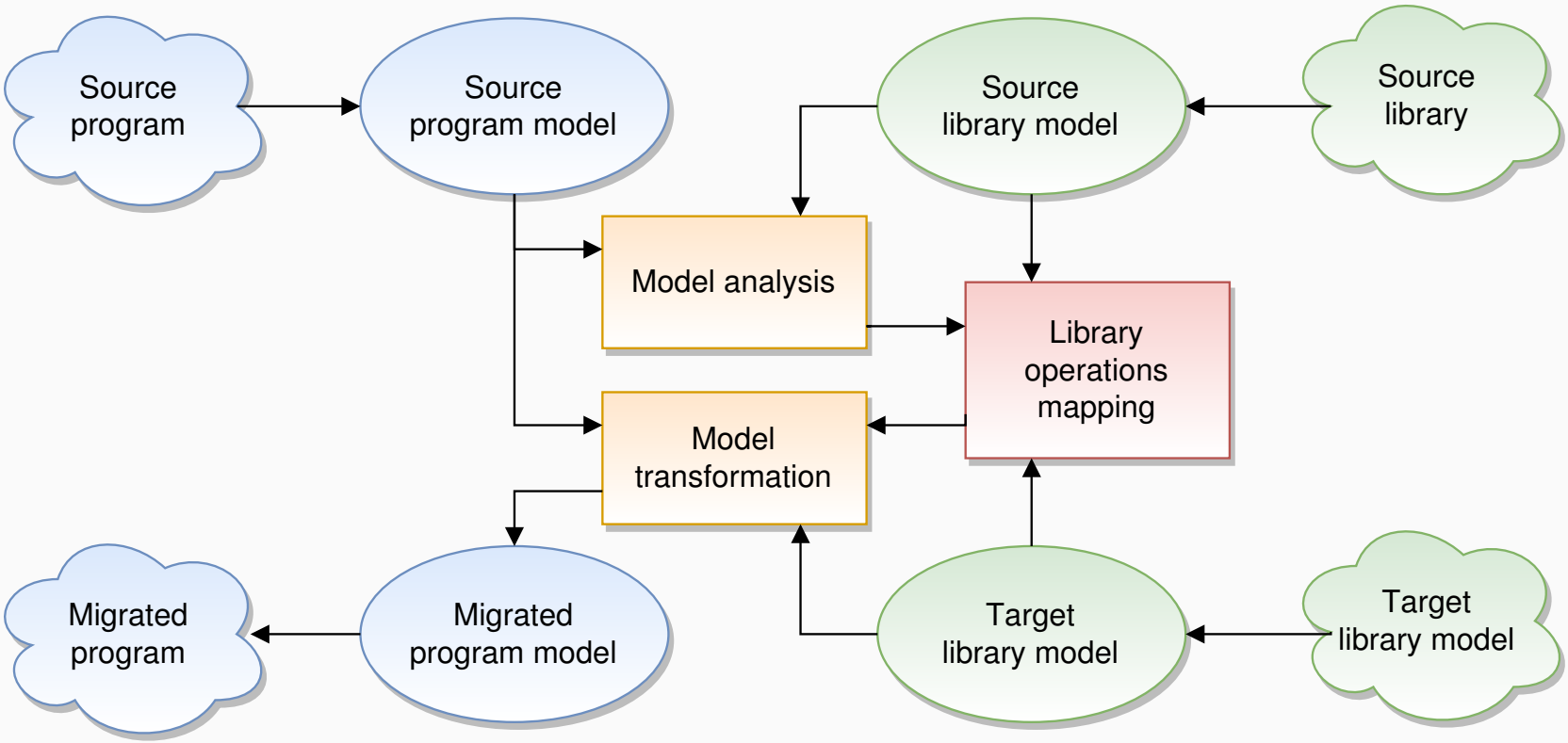
Existing approaches

- Call translation
- OS-level virtualization
- Use of wrappers
- Syntax-based approach
- Semantics-based approach

Comparison of approaches

- Use of wrappers
  - Wrapper is a dummy library which provides the interface of the source library and the implementation of a target one
  - Is not scalable
- Syntax-based approach
  - Implemented in tools like TXL and IDEA Search and Replace
  - Is able to perform only basic code changes
- Semantics-based approach
  - The most scalable and powerful option

Semantics-based approach scheme



The previously proposed formalism (in a nutshell)

Specifies library behavior using a set of extended finite state machines (EFSMs)

- Each transition in the automaton refers to some kind of interaction with the library
- A new EFSM may be created during a transition
- Semantically important operations are described using actions
- EFSMs may have attributes

Itsykson V. The formalism for semantics specification of software libraries (PSSV, 2016).

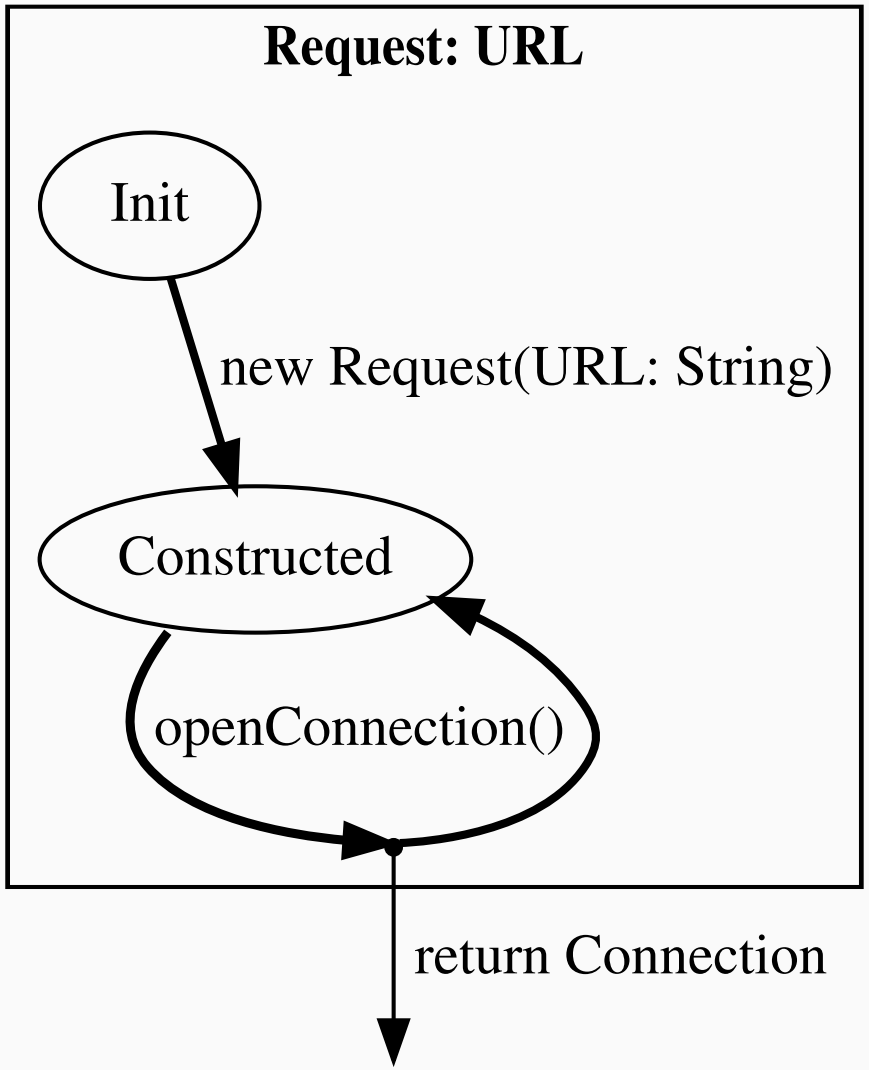
Library metamodel

Designed for Java language and intended for object-oriented libraries

Defines:

- Methods and constructors from the library
- Possible arguments of methods and constructors
- Classes, interfaces and their relations

Example of the EFSM (graph visualization)



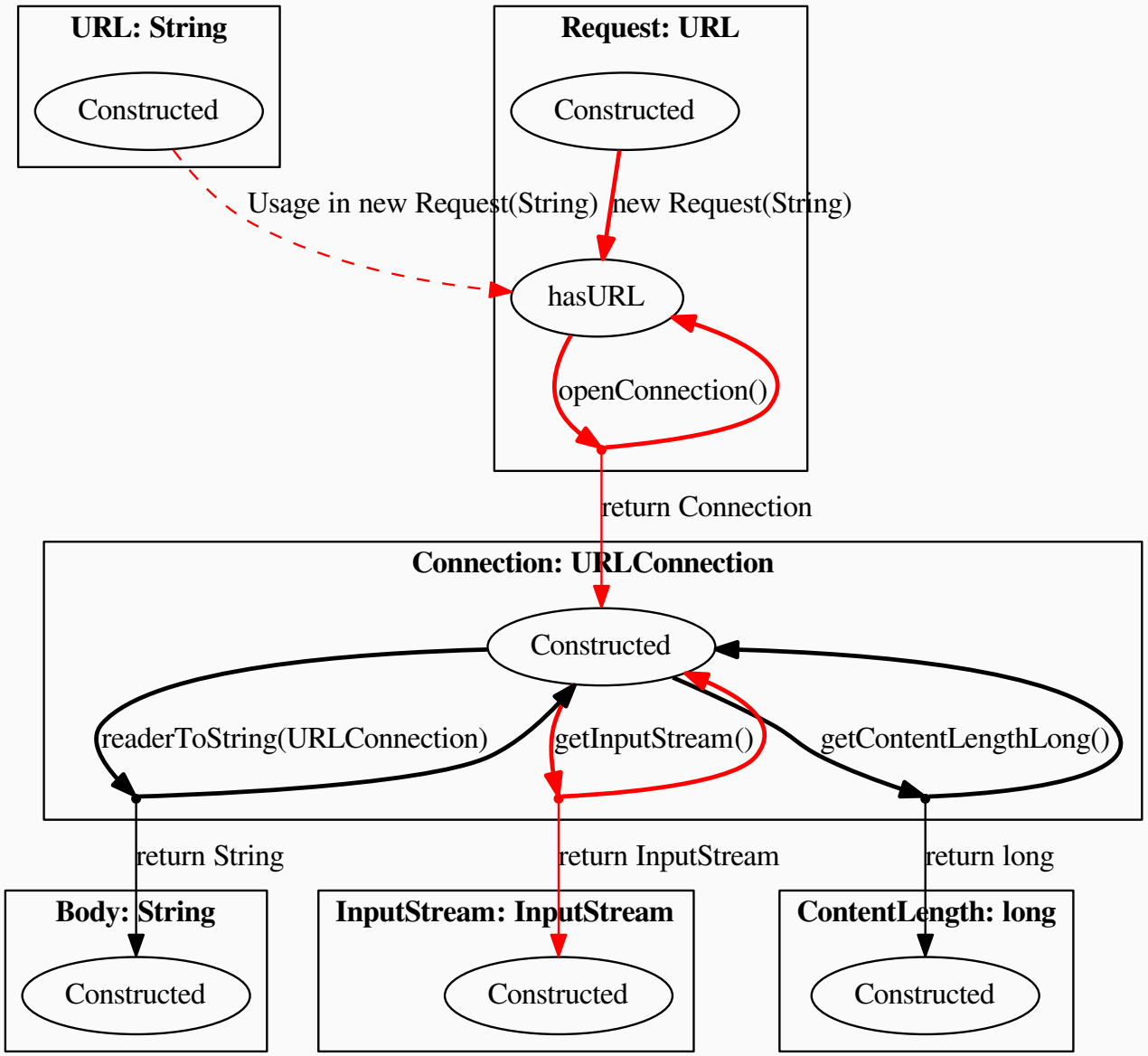
Migration procedure

- Trace extraction
- Trace mapping
- Equivalent trace calculation
- Mapping of a new trace back into the code model (reverse mapping)
- Program transformation

Trace extraction & mapping

- The trace is needed to understand which parts of the code should be changed
- A sequence of method/constructor calls
- Trace mapping transforms the code trace into the model trace, i.e. a sequence of EFSM transitions

Model trace example



Equivalent trace calculation

- Equivalence criteria:
  - The new trace carries out the same set of semantic actions
  - The new trace creates the same set of entities (represented by EFSMs)
- The task is reduced to finding a path on the graph
- To perform the search, we use an algorithm based on the BFS
- *And what about actions and attributes?*

Equivalent trace calculation

- A new graph is created that reflects the possible search states
  - The vertices of the graph correspond to the traces
  - The edge of the graph represents the ability to append a transition to the trace
  - Each vertex stores a context, i.e. a set of available EFSMs
- To handle argument requirements, we extended BFS with a separate queue for vertices with missing dependencies
  - Such vertices are processed when the dependency is resolved

Equivalent trace calculation

- Each transition from the source trace is usually handled separately
- But sometimes we are able to combine several transitions and process them together
  - This trick allows to apply complex transformations on the source code, such as reordering
- The procedure also includes several steps to operate with entities from the context

Reverse mapping & program transformation

- During the reverse mapping step, a set of EFSM's transitions is transformed back to the code model form
- Often it is needed to add new variables to the code
- Program transformation step includes the removal of unnecessary statements, replacement of expressions, addition of new statements

Prototype of the migration tool

- Processes Java 8 code
- Provides an easy-to-use DSL for describing libraries
- Includes modules for visualization, user interaction, trace extraction and migration itself
- Written in Kotlin[1] language

---

1. https://kotlinlang.org/

The fragment of library description

```
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(EntityParam(machine = url))
)
LinkedEdge(
        dst = request.getDefaultState(),
        edge = CallEdge(
                machine = urlData,
                src = hasURL,
                methodName = "openConnection"
        )
)
```

Trace extraction

- The trace includes an order of execution, argument's values, etc.
- Extraction may be done dynamically or statically
- The tool prototype employs the dynamic method
- We use aspect-oriented programming to instrument the code

Implementation of the migration procedure

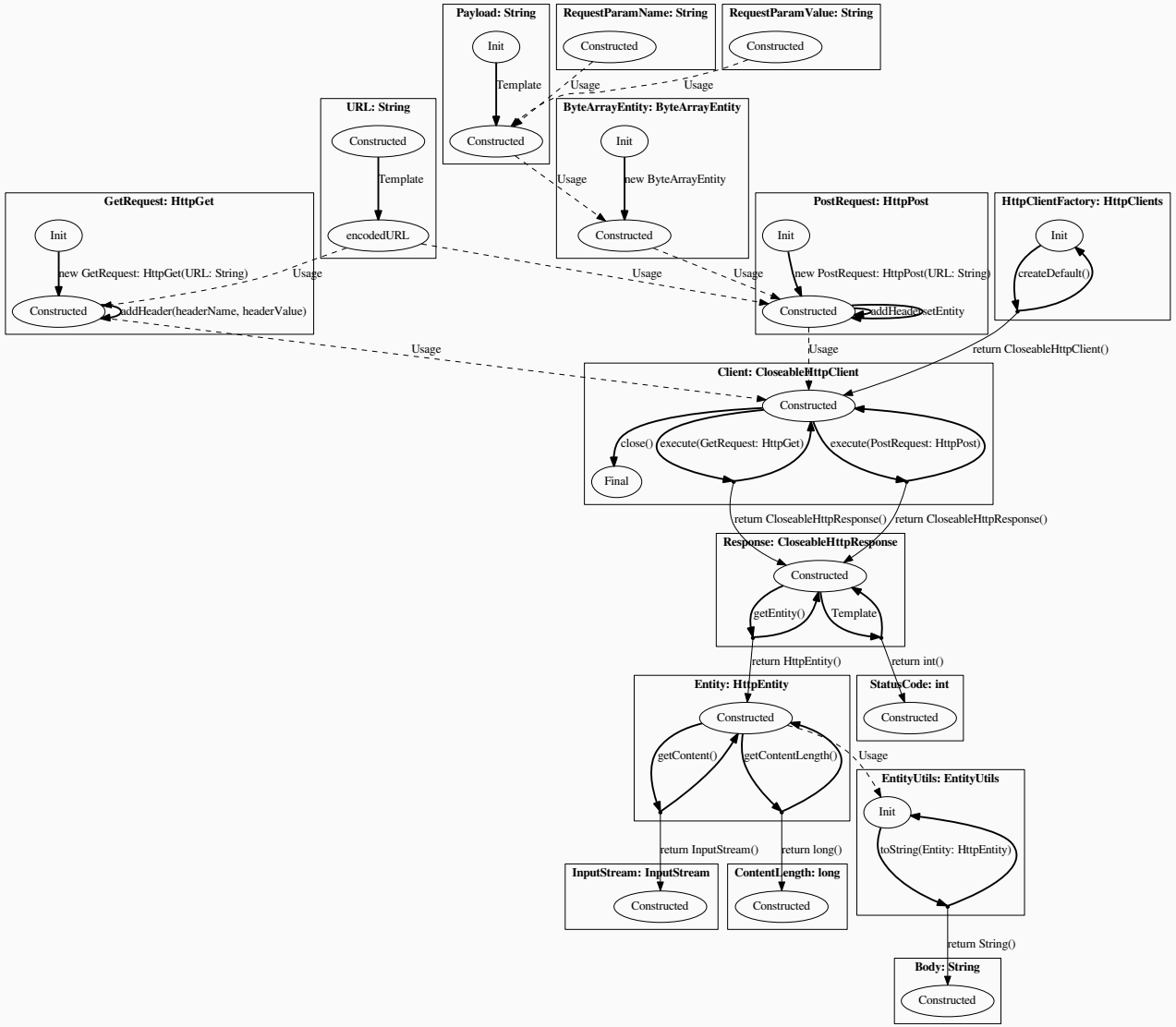
- The tool prototype uses AST as a code model
  - CFG, SSA add no benefits, as we already have an execution trace
- If the tool prototype is unable to automatically calculate the equivalent trace, it asks for help from the user
- The transformation process keeps comments and code formatting unchanged
  - Thanks to a JavaParser library

Evaluation

During the evaluation we:

- Created models for 3 libraries: Apache HttpClient, OkHttp, HttpURLConnection
- Prepared a set of test examples
  - Several artificial examples
  - A real-world project
- Successfully migrated all test examples to new libraries

Library model visualization



Test example (source code)

```
RequestBody formBody = new FormBody.Builder()
        .add("q", parameters)
        .build();
Request request = new Request.Builder()
        .url(Endpoint.INSTAGRAM_QUERY_URL)
        .post(formBody)
        .header("Cookie",
  ↪  String.format("csrftoken=%s;", random))
        .header("Referer",
  ↪  "https://www.instagram.com/")
        .build();
Response response =
  ↪  this.httpClient.newCall(request).execute();
```

Test example (migrated code)

```
URLConnection migration_JavaRequest_1 = new
  ↪  URL(Endpoint.INSTAGRAM_QUERY_URL).openConnection();
migration_JavaRequest_1.setDoOutput(true);
migration_JavaRequest_1.setRequestProperty("Cookie",
  ↪  String.format("csrftoken=%s;", random));
migration_JavaRequest_1.setRequestProperty("Referer",
  ↪  "https://www.instagram.com/");
migration_JavaRequest_1.setDoOutput(true);
migration_JavaRequest_1.getOutputStream().write(("q"
  ↪  + "=" + URLEncoder.encode(parameters,
  ↪  "UTF-8")).getBytes());
```

Conclusion

- The semantics-driven migration procedure was created
- An easy-to-use DSL was constructed
- The tool prototype which is able to migrate Java 8 programs was developed
- The feasibility of automated code migration was demonstrated
- The applicability of the proposed metamodel and procedure was confirmed

Directions of the future research

- Refinement of library specification formalism
- Development and extension of library model specification language
- Increase the possibilities of user control on the migration process
- Development of a more reliable and feature-rich migration tool

Contacts

Email: aleksyuk@kspt.icc.spbstu.ru

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

Thank you for your attention!