CLOSURE Toolchain User Manual for Java Language

Release version 2.0

Peraton Labs

August 23, 2022

Contents

1 CLOSURE Toolchain Overview

1.1 What is CLOSURE?

DARPA's Guaranteed Architecture for Physical Systems (GAPS) is a research program that addresses software and hardware for compartmentalized applications where multiple parties with strong physical isolation of their computational environment, have specific constraints on data sharing (possibly with redaction requirements) with other parties, and any data exchange between the parties is mediated through a guard that enforces the security requirements.

Peraton Labs' Cross-domain Language extensions for Optimal SecUre Refactoring and Execution (CLOSURE) project is building a toolchain to support the development, refactoring, and correct-by-construction partitioning of applications and configuration of the guards. Using the CLOSURE approach and toolchain, developers will express security intent through annotations applied to the program, which drive the program analysis, partitioning, and code autogeneration required by a GAPS application.

Problem: The machinery required to verifiable and securely establish communication between cross-domain systems (CDS) without jeopardizing data spillage is too complex to implement for many software platforms where such communication would otherwise be desired. To regulate data exchanges between domains, network architects rely on several risk mitigation strategies including human fusion of data, diodes, and hypervisors which are insufficient for future commercial and government needs as they are high overhead, customized to specific setups, prone to misconfiguration, and vulnerable to software/hardware security flaws. To streamline the design, development, and deployment of provably secure CDSs, new hardware and software co-design tools are needed to more effectively build cross-domain support directly into applications and associated hardware early in the development lifecycle.

Solution: Peraton Labs is developing CLOSURE (Cross-domain Language-extensions for Optimal SecUre Refactoring and Execution) to address the challenges associated with building cross-domain applications in software. CLOSURE extends existing programming languages by enabling developers the ability to express security intent through

overlay annotations and security policies such that an application can be compiled to separable binaries for concurrent execution on physically isolated platforms.

The CLOSURE compiler toolchain interprets annotation directives and performs program analysis of the annotated program and produces a correct-by-construction partition if feasible. CLOSURE automatically generates and inserts serialization, marshaling, and remote-procedure call code for cross-domain interactions between the program partitions.

In this document, we describe the CLOSURE toolchain for Java programs.

1.2 Architecture

The CLOSURE architecture for Java is shown in the figure below. The architecture builds on existing open-source ecosystems including the Java Development Kit and the AspectJ compiler.

There are three main layers in the architecture:

- MULES: This layer includes support for annotating source code with CLOSURE language extensions (CLE). The annotated code added by the developer is then compiled using an unmodified Java language compiler to produce JVM bytecode.
- CAPO: This layer deals with program analysis and partitioning. A System Dependency Graph model of the compiled Java program is constructed, and analyzed by the CLOSURE conflict analyzer based on a constraint solver. If a feasible partitioning is found, aspect-oriented program code to handle the cross-domain isolation and communication concerns is generated, and these aspects are woven into the application code, one for each enclave.
- HAL: This layer provides applications with a 0MQ-based interface for cross-domain communications and abstracts out the details of heterogeneous cross-domain guards (GAPS hardware) that it manages.

Some key differences between the C toolchain [1] and the Java toolchain are: (i) the use of aspects for partitioning rather than physically dividing and modifying the application source code; (ii) the use of reflection in the serialization and marshaling; (iii) lack of a multi-target binary generation (MBIG) layer, as the Java VM supports a write-once run anywhere paradigm; and (iv) autogeneration of HAL interface code as part of the aspects rather than the use of a separate XDCOMMS API library.

1.3 Workflow

The CLOSURE workflow for building cross-domain applications in Java is shown in the figure below.

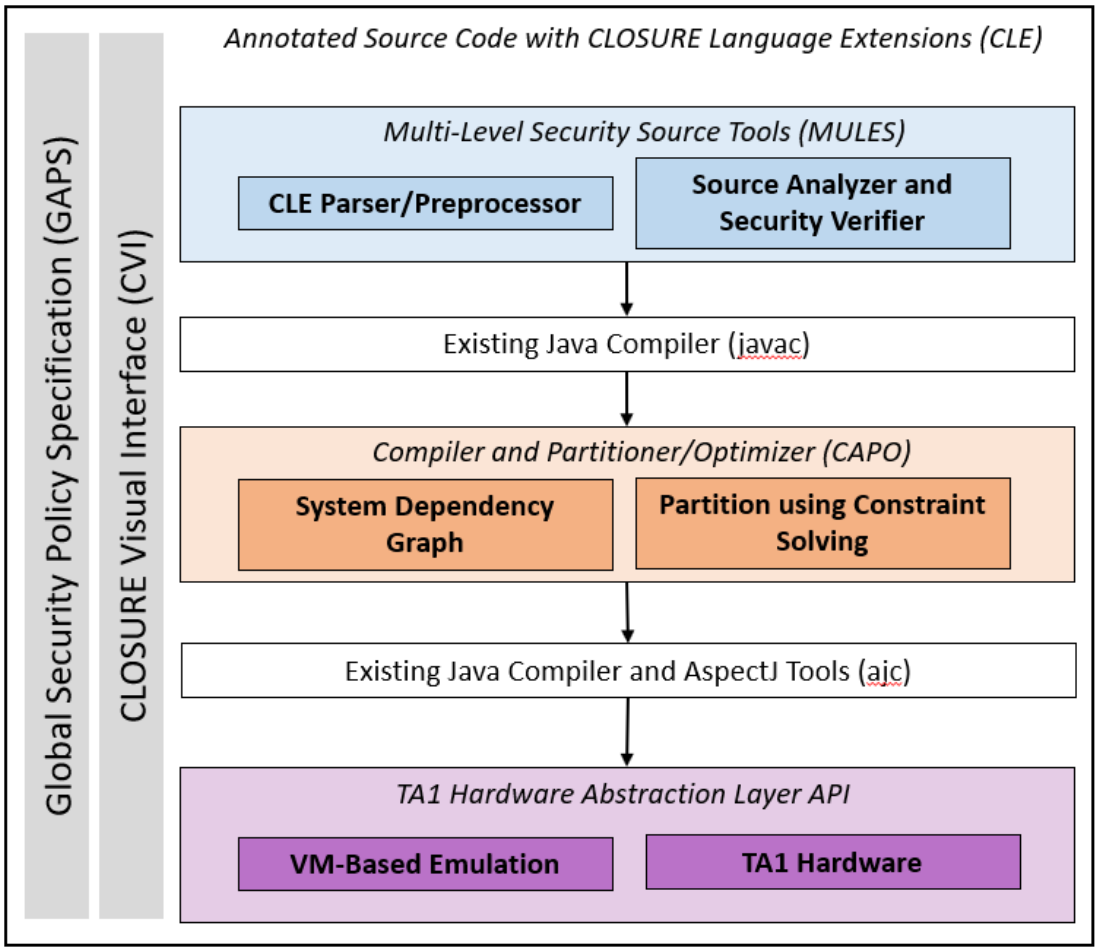


Figure 1: CLOSURE architecture

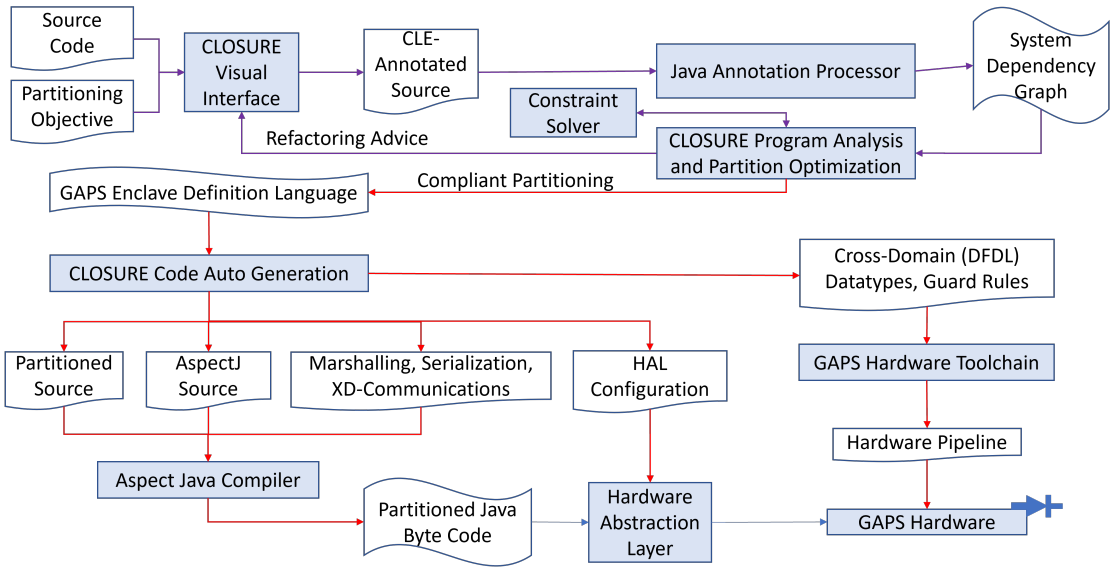


Figure 2: CLOSURE Workflow for Java

In the first stage, the developer either writes a new application or imports an existing source which must be tooled for cross-domain operation. The developer must have knowledge of the intended cross-domain policy. While CLOSURE provides means to express this policy in code, the requirements analyst/developer determines the actual cross-domain data sharing policy. The developer then uses CLE to annotate the program as such. The developer can use the CLOSURE Visual Interface (CVI) based on Visual Studio Code [2]. Additional plugins to provide syntax hints (similar to the C toolchain [1]) are planned in future work.

From there, we use the Java compiler to generate a jar file that we can feed to a tool called JOANA [3], which builds a system dependency graph (SDG) model of the annotated program. Using the model produced from the SDG, our conflict analysis based on the CLOSURE constraint model for Java (implemented using MiniZinc) determines if the partitioning of the annotated program is feasible. If not, feedback is provided back to the developer for refactoring needed to get a compliant program. Once the program is deemed compliant (via the conflict analyzer), CLOSURE proceeds with automated tooling in which CAPO and associated tools divide the code, generate code for cross-domain remote procedure calls (RPCs), describe the formats of the cross-domain data types via DFDL [4] and codec/serialization code, and generate all required configurations for interfacing to the GAPS hardware via the Hardware Abstraction Layer (HAL).

1.4 Document Roadmap

In the rest of this document, we first present a quick start guide followed by a detailed usage of the toolchain components. For each component, we describe what it does, provide some insight into how it works, discuss inputs and outputs and provide invocation syntax for usage. We conclude with a discussion of the limitations of our current toolchain and a roadmap for future work. We provide samples of significant input and output files in the appendices and provide a list of bibliographic references at the end.

2 Installation and Quick Start For CLOSURE Java Toolchain

2.1 Prerequisite

The CLOSURE Java toolchain is released as a Docker [5] container based on Ubuntu 20.04. See Docker Installation for instructions on installing Docker on a Ubuntu Linux system.

2.2 Pre-built Releases

A pre-built source release and binary release are available from our repository. Alternatively, for convenience, one can pull the corresponding docker releases:

```
docker pull gapsclosure/closure-java-src:latest
docker pull gapsclosure/closure-java-bin:latest
```

Using the source release docker, one can skip the next step (Build the Source Container) and proceed to the rest, which builds a docker image equivalent to the binary release when completes successfully.

2.3 Build the Source Container

Save the dockerfile in the appendix to a file, e.g. the default Dockerfile, and build the container as follows.

```
docker build -f Dockerfile -t closure:src  .
```

2.4 Start the Docker Image

```
docker run -ti --device /dev/video0 closure:src
```

where `closure:src` is the docker repository and tag of the image and `/dev/video0` is the device file for the camera on the host.

2.4.1 Build JOANA

You can build JOANA as follows:

```
rm -rf /tmp/smoke_main
cd $CAPO/joana
./setup_deps
ant
ant doc-wala
```

2.4.2 Compile the Demo Application

Next, compile the annotated source code for the unpartitioned demo application as follows:

```
cd $CAPO/examples/eop2-demo/
ant
```

2.4.3 Run the Conflict Analyzer

Run the CLOSURE conflict analyzer to check for feasible partitions.

```
cd $CAPO
java -cp $CLASSPATH org.python.util.jython zincOutput.jy -m
 ↪  ./examples/eop2-demo/src/com/peratonlabs/closure/eop2/video/manager/VideoManager.java
 ↪  -c ./examples/eop2-demo/dist/TESTPROGRAM.jar -e
 ↪  com.peratonlabs.closure.eop2.video.manager.VideoManager -b
 ↪  com.peratonlabs.closure.eop2.
```

If the program is properly annotated, a `cut.json` file is produced showing the class assignments to each enclave and the methods in the cut.

```
cp cut.json $HOME/gaps/CodeGenJava/test
```

2.4.4 Build HAL

Build the HAL daemon as follows.

```
cd $HOME/gaps/hal
make
```

2.4.5 Build Code Generator

Build the code generator tool as follows.

```
cd $HOME/gaps/CodeGenJava
ant
```

2.4.6 Partition the Demo Application

Generate aspects and weave them into the demo application to generate a partitioned executable for each enclave.

```
cd $HOME/gaps/CodeGenJava
java -jar code-gen/code-gen.jar
```

2.4.7 Run the Demo Application

Run the demonstration application.

```
cd $HOME/gaps
./run.sh
```

Once started, there will be three sets of terminals, from left to right, one for each of the Purple, Orange and Green partitions. Within each partition, the top terminal is the output for HAL and the bottom one for the Java app.
Wait until the Purple enclave (the leftmost one) is ready and sending messages to the other enclaves. Then on the host of the container, start a browser and go to the URL `http://172.17.0.2:8080/`.

```
# On the host
firefox http://172.17.0.2:8080/
```

Click on the Play button. The camera image should appear in the browser at this point.

2.5 Additional Notes for CLOSURE Developers

Developers who wish to extend the CLOSURE Java partitioner or visualize the generated system dependency graph (SDG) will find the following steps useful.

Generate SDG and Dot files for the test program as follows:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
export
 ↪  CLASSPATH="joana/dist/*:testprog/dist/*:jython-standalone-2.7.2.jar:jscheme-7.2.jar:'
java -cp $CLASSPATH org.python.util.jython JoanaUsageExample.jy \
  -c './testprog/dist/TESTPROGRAM.jar' \
  -e 'com.peratonlabs.closure.testprog.example1.Example1' \
  -p -P 'out.pdg' \
  -d -D 'out.dot' \
  -j -J 'out.clemap.json'
```

You can launch the SDG viewer, and visualize the SDG interactively:

```
java -cp $CLASSPATH edu.kit.joana.ui.ifc.sdg.graphviewer.GraphViewer
```

You can produce the program partition next:

```
java -cp $CLASSPATH org.python.util.jython zincOutput.jy
 -m './example1/src/example1/Example1.java'
 -c './example1/dist/TESTPROGRAM.jar'
 -e 'com.peratonlabs.closure.testprog.example1.Example1'
 -b 'com.peratonlabs.closure.testprog'
```

The command line parameters are as follows:

```
  -m option indicates what java file has the main class to analyze
  -c option indicates the jar file to analyze
  -e option indicates the class with the entry method
  -b option indicates the prefix for the classes that are of interest
```

Running this command will result in generating the following artifacts

- enclave_instance.mzn
- pdg_instance.mzn
- cle_instance.mzn
- cut.json
- dbg_edge.csv
- dbg_node.csv

- `dbg_classinfo.csv`

The `dbg_edge.csv` and `dbg_node.csv` files report useful information about all of the nodes and edges in the SDG being analyzed that can be useful to debug and find issues with annotations.

The `dbg_classinfo.csv` file contains the class name, field, and method name to ID relationships.

The three `.mzn` files are what get fed to MiniZinc along with the fixed `.mzn` files in the `constraints/` directory to run the constraint solver. If the program is properly annotated, a `cut.json` file is produced showing the class assignments to each enclave and the methods in the cut.

Since the output of the constraint solver reports edge IDs, useful scripts are available in the `capo/Java/scripts` directory. The `edgeDbg.py` script takes an edge ID as input and produces the debug information for the associated source and destination nodes. Similarly, `getclassName.py` takes a class ID and produces the corresponding class name for the ID. Note that these scripts assume the `dbg_*.csv` files are in the same directory as the scripts.

Set classpath and java location and build the application to be partitioned.
These commands assume you are in the `capo/Java` directory.

```
export CLASSPATH="joana/dist/*:examples/eop2-demo//dist/*:jython-standalone-2.7.2.jar:jsc
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
```

Build the demo application:

```
cd examples/eop2-demo/
ant
cd ../..
```

IMPORTANT In file `capo/Java/examples/eop2-demo/src/com/peratonlabs/closure/eop2/video/m` ensure that webroot is initialized to `capo/Java/examples/eop2-demo/resources`

Run the conflict analyzer from `capo/Java`:

```
java -cp $CLASSPATH org.python.util.jython zincOutput.jy -m
 ↪  './examples/eop2-demo/src/com/peratonlabs/closure/eop2/video/manager/VideoManager.jav
 ↪  -c './examples/eop2-demo/dist/TESTPROGRAM.jar' -e
 ↪  'com.peratonlabs.closure.eop2.video.manager.VideoManager' -b
 ↪  'com.peratonlabs.closure.eop2.'
```

The resulting cut.json will be produced in the directory the above command is invoked from.

3 Detailed Usage and Reference Manual

3.1 Annotations

The CLOSURE toolchain relies on source-level annotations to specify the cross-domain constraints. Developers annotate programs using CLOSURE Language Extensions (CLE) to specify cross-domain security constraints. Each CLE annotation definition associates a *CLE label* (a symbol) with a *CLE JSON* which provides a detailed specification for cross-domain data sharing and function invocation constraints. These source-level annotations determine the following:

1. The assignments of classes to enclaves
2. The confidentiality of data between enclaves
3. Which functions can be called cross-domain
4. Guard rules that transform data as it crosses domains

These annotations are only applied to a subset of the program and then passed to a separate tool called the _conflict *analyzer* that can infer the CLE labels of the rest of the program elements.

The rest of this section will introduce examples of Java program features with CLE annotations.

3.1.1 Defining CLE Annotations

First, we define a custom java annotation shown below called *Cledef*. A *Cledef* annotation will be applied to each application-specific annotation that we will define and apply to fields, constructors, and methods. The Cledef annotation enables us to associate a CLE JSON with the application-specific annotation.

```
@Target(ElementType.ANNOTATION_TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Cledef
{
    String clejson() default "";
    boolean isFile() default false;
}
```

3.1.2 Field Annotations

In the example below, we show how we can apply a CLE annotation to a field in a Java program. First, we define our annotation in a java source file.

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Cledef(clejson = "{" +
                  "  \"level\":\"green\"" +
                  "}")
public @interface Green {}
```

Together the `Green` custom annotation and the `Cledef` applied to it are equivalent to the `#pragma cle def <LABEL> <JSON>` that we use with the C toolchain [1].

The above annotation can be applied to any field (static or non-static) in a class. The retention policy ensures that the annotation is accessible throughout compilation and at runtime.

The `clejson` specifies a `"level"` field set to `"green"`. The level is similar to a security level, but there is no requirement for ordering among the levels. A single level may correspond to many enclaves, but in most cases, they will be in a bijection with the enclaves. The level names can be any string.

Enclaves are isolated compartments with computation and memory resources. Each enclave operates at a specified level. Enclaves at the same level may be connected by a network, but enclaves at different levels must be connected through a cross-domain guard (also known as SDH or TA-1 hardware within the GAPS program).

The following is an example application of the Green annotation to variable test.

```
@Green
int test;
```

The following is an example field annotation that allows a cross-domain flow (CDF)

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Cledef(clejson = "{" +
                  "  \"level\":\"green\"," +
                  "  \"cdf\":[" +
                  "    {" +
                  "      \"remotelevel\":\"purple\"," +
                  "      \"direction\":\"egress\"," +
                  "      \"guarddirective\":{" +
                  "        \"operation\":\"allow\"" +
                  "      }" +
```

```
                    "    }" +
                    "  ]" +
                    "}")
public @interface GreenShareable {}
```

In the above example, the `"remotelevel"` field specifies that the program element the label is applied to can be shared with an enclave so long as its level is `"purple"`. The `"guarddirective": { "operation": "allow"}}` defines how data gets transformed as it goes across enclaves. In this case, `{ "operation": "allow" }` simply allows the data to pass uninhibited. The `"direction"` field is currently not used and is ignored by the CLOSURE toolchain (may be removed in a future release).

The `cdf` is an array, and data can be released into more than one enclave. Each object within the `cdf` array is called a `cdf`.

3.1.3 Method and Constructor Annotations

The following shows an example of a constructor annotation.

```
@Target(ElementType.CONSTRUCTOR)
@Retention(RetentionPolicy.RUNTIME)
@Cledef(clejson = "{" +
                  "  \"level\":\"purple\"," +
                  "  \"cdf\":[" +
                  "    {" +
                  "      \"remotelevel\":\"green\"," +
                  "      \"direction\":\"bidirectional\"," +
                  "      \"guarddirective\":{" +
                  "        \"operation\":\"allow\"" +
                  "      }," +
                  "      \"argtaints\":[]," +
                  "      \"rettaints\":[\"TAG_RESPONSE_EXTRA\"]," +
                  "      \"codtaints\":[\"Purple\"]" +
                  "    }," +
                  "    {" +
                  "      \"remotelevel\":\"purple\"," +
                  "      \"direction\":\"bidirectional\"," +
                  "      \"guarddirective\":{" +
                  "        \"operation\":\"allow\"" +
                  "      }," +
                  "      \"argtaints\":[]," +
                  "      \"rettaints\":[\"TAG_RESPONSE_EXTRA\"]," +
                  "      \"codtaints\":[\"Purple\"]" +
                  "    }" +
```

```
                    "  ]" +
                    "}")
public @interface PurpleGreenConstructable {}
```

Similarly, the following example shows a method annotation. The only difference between a constructor and method annotation in Java is the Target.

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Cledef(clejson = "{" +
                    "  \"level\":\"orange\"," +
                    "  \"cdf\":[" +
                    "    {" +
                    "      \"remotelevel\":\"purple\"," +
                    "      \"direction\":\"bidirectional\"," +
                    "      \"guarddirective\":{" +
                    "        \"operation\":\"allow\"" +
                    "      }," +
                    "      \"argtaints\":[]," +
                    "      \"rettaints\":[\"TAG_RESPONSE_GETVALUE\"]," +
                    "      \"codtaints\":[\"Orange\"]" +
                    "    }," +
                    "    {" +
                    "      \"remotelevel\":\"orange\"," +
                    "      \"direction\":\"bidirectional\"," +
                    "      \"guarddirective\":{" +
                    "        \"operation\":\"allow\"" +
                    "      }," +
                    "      \"argtaints\":[]," +
                    "      \"rettaints\":[\"TAG_RESPONSE_GETVALUE\"]," +
                    "      \"codtaints\":[\"Orange\"]" +
                    "    }" +
                    "  ]" +
                    "}")
public @interface OrangePurpleCallable {}
```

The following is an example showing how a method or constructor annotation can be applied.

```
@OrangePurpleCallable
void foo() {

}
```

In a method or constructor annotation, the `cdf` field specifies remote levels permitted to call the annotated function. Method and constructor annotations are also different from data annotations as they contain
`taints` fields.

A taint refers to a label or an assigned label by the conflict analyzer for a given data element. There are three different taint types to describe the inputs, body, and outputs of a function: `argtaints`, `codtaints` and `rettaints` respectively. Each portion of the method or constructor may only touch data tainted with the label(s) specified by the annotation:

- `rettaints` constrains which labels the return value of a function may be assigned
- `argtaints` constrains the assigned labels for each argument. This field is a 2D-array, mapping each argument of the method or constructor to a list of assignable labels.
- `codtaints` includes any other additional labels that may appear in the body

Method and constructor annotations can coerce between labels of the same level (that is, they can allow the annotated method to touch data at a level with different sharing constraints), so it is expected that these methods and constructors are to be audited by the developer. The developer must add a `cdf` where the `remotelevel` is the same as the `level` specified in the annotation, to perform coercion. The other `cdf` blocks signify that the method can be called from the `remotelevel` specified.

3.1.4 TAGs

In the constructor and method annotations, there is `TAG_RESPONSE_GETVALUE` and `TAG_RESPONSE_EXTRA` label. These are special labels that do not require users to define them. The definitions for these `TAG_` labels are generated automatically by the toolchain; for every cross-domain call, there are two `TAG_` labels generated for receiving and transmitting data, called `TAG_REQUEST_` and `TAG_RESPONSE_`. Each generated tag label has a suffix which is the name of the method or constructor it is being applied to in capital letters. The label indicates that associated data is the result of incoming or outgoing data specific to the RPC logic. This supports verification of data types involved in the cross-domain cut and that only intended data crosses the associated RPC.

3.2 Phase 2 CLOSURE conflict analyzer based on MiniZinc constraint solver

The role of the conflict analyzer is to evaluate a user annotated program and decide if the annotated program respects the allowable information flows specified in the annotations. As input, the conflict analyzer requires the user annotated Java source code. Based on this, if it is properly annotated and a partition is possible it will produce an assignment

for each class to an enclave (cut.json). If the conflict analyzer detects an inconsistency in the given annotations and program, it reports this and the user can run MinZinc on the model to produce diagnostics identifying problematic constraints.

The conflict analyzer uses a constraint solver called MiniZinc [6] to perform program analysis and determine a correct-by-construction partition that satisfies the constraints specified by the developer using CLE annotations. MiniZinc provides a high-level language abstraction to express constraint-solving problems in an intuitive manner. MiniZinc compiles a MiniZinc language specification of a problem for lower-level solvers such as Gecode. We use an Integer Logic Program (ILP) formulation with MiniZinc. MiniZinc also includes a tool that computes the minimum unsatisfiable subset (MUS) of constraints if a problem instance is unsatisfiable. The output of this tool can be used to provide diagnostic feedback to the user to help refactor the program.

Downstream tools in the CLOSURE toolchain will use the output of the solver to physically partition the code, and after further analysis (for example, to determine whether each parameter is an input, output, or both, and the size of the parameter), the downstream tools will autogenerate code for the marshaling and serialization of input and output/return data for the cross-domain call, as well as code for invocation and handling of cross-domain remote-procedure calls that wrap the function invocations in the cross-domain cut.

3.2.1 Introduction to the Conflict Analyzer

3.2.2 Usage

The usage of the conflict analyzer is described in the previous section.

3.2.3 Modeling data and control flows

Java CLOSURE uses JOANA [3] to construct a system dependency graph (SDG) which is composed of a call graph, control flow graph, and data flow graph among other things allowing us to soundly track taints throughout an application. Joana is written in java and builds on Wala which analyzes an IR form of java bytecode.

We convert the node and edge types from the SDG to the format presented in our C documentation [1]. The transformation is shown in detail in the appendix.

3.2.4 Data required by MiniZinc

The Java conflict analyzer uses three kinds of information in its model. It uses information about data and control flows from the SDG. It also collects information from the annotations using a jython script that uses java utilities to extract the CLE annotations

from a given jar file. Lastly, we make use of the reflection feature in Java to relate fields and methods to their associated classes as well as track any modifiers that may be present on those fields and methods.

These three pieces of data along with the constraints described in the next section are given to MiniZinc. MiniZinc will then either produce at least one enclave assignment per class or report no such assignment exists given the program and user annotations.

3.2.5 Design Decisions

The Java conflict analyzer permits unannotated classes to reside in multiple enclaves. This choice was made to make our approach more practical. This design choice allows developers to annotate child classes with different level taints without necessarily requiring the parent class to be annotated.

Exceptions are currently modeled as returns and the rettaints component of an annotation also applies to exceptions.

3.2.6 Detailed MiniZinc constraint model

In this section, we present an informal statement of constraints to be enforced by our conflict analyzer. We then present the main constraints coded in MiniZinc used by our model to achieve these constraints. More information about MiniZinc including its usage and syntax can be found here.

In the model below, the `nodeEnclave` decision variable stores the enclave assignment for each node, the `taint` decision variable stores the label assignment for each node, and the `xdedge` decision variable stores whether a given edge is in the enclave cut (i.e., the source and destination nodes of the edge are in different enclaves. Several other auxiliary decision variables are used in the constraint model to express the constraints or for efficient compilation.

The solver will attempt to assign a node annotation label to all nodes except a user annotated function. Only user annotated functions may have a function annotation. Functions lacking a function annotation cannot be invoked cross-domain and can only have exactly one taint across all invocations. This ensures that the arguments, return and function body only touches the same taint.

3.2.7 General Constraints

- Instance and class fields can be annotated by the user with node annotations.
- Instance and class methods can be annotated by the user with method annotations.
- Constructors can be annotated by the user with constructor annotations.

- Only node annotations can be assigned by the solver to unannotated fields, methods or constructors.
- Method or constructor annotations cannot be assigned by the solver (these can only be assigned by the user).
- Each class containing one or more annotated elements (constructor, method, or field) must be assigned to exactly one enclave.
- Each class containing no annotated elements must be assigned to at least one enclave and at most every enclave.
- Across all accesses/invocations of an unannotated element, it may touch at most one label at each level.
- All elements (constructor, method, or field) of a class instance must be assigned the same enclave as the instance itself. This entails separate constraints for constructors, instance methods, instance fields, static methods and static fields.
- Contained nodes and parameters are assigned the same enclave(s) as their containing methods.

- Annotations can not be assigned to a valid enclave and they must be assigned to `nullEnclave`.
- Each (node,level) pair is assigned at most one valid label with that level.
- Only method entry nodes can be assigned a method annotation label.
- Only constructor entry nodes can be assigned a constructor annotation label.

```
constraint :: "NodeLevelAtTaintLevel"
forall (n in NonAnnotation)
( forall(l in nonNullEnclave)
( nodeLevel[n,l]==hasLabelLevel[taint[n,l]]));

constraint :: "NodeLevelAtEnclaveLevel"
forall (n in NonAnnotation)
( forall(l in nonNullEnclave)
( nodeLevel[n,l]==hasEnclaveLevel[nodeEnclave[n,l]]));

constraint :: "CannotBeNullEnclave"
forall (n in NonAnnotation)
( forall(l in nonNullEnclave)
( nullEnclave != nodeEnclave[n,l]));

constraint :: "FnAnnotationForFnOnly"
forall (n in NonAnnotation)
( forall(l in nonNullEnclave)
(isFunctionAnnotation[taint[n,l]] -> isFunctionEntry(n)));

constraint :: "FnAnnotationByUserOnly"
forall (n in FunctionEntry)
```

```
( forall(l in nonNullEnclave)
(isFunctionAnnotation[taint[n,l]] == userAnnotatedFunction[n]));

constraint :: "NodesHaveClassEnclave"
forall (n in PDGNodeIdx)
(forall (e in nonNullEnclave)
((classEnclave[hasClass[n],e]) == true -> nodeEnclave[n,e] == e));

constraint :: "UnannotatedFunContentTaintMatch"
forall (n in PDGNodeIdx)
(forall (l in nonNullEnclave)
(userAnnotatedFunction[hasFunction[n]]==false ->
 ↪  taint[n,l]==ftaint[n,l]));

constraint :: "ForceAnnotFuncToAnnoLvl"
forall (n in FunctionEntry)
(forall (l in nonNullEnclave)
(userAnnotatedFunction[hasFunction[n]] ->
 ↪  hasLabelLevel[taint[n,l]]==hasLabelLevel[ftaint[n,l]]));

constraint :: "AnnotatedFunContentCoercible"
forall (n in PDGNodeIdx where  isFunctionEntry(n)==false)
(forall (l in nonNullEnclave)
(userAnnotatedFunction[hasFunction[n]] -> isInArctaint(ftaint[n,l],
 ↪  taint[n,l], hasLabelLevel[taint[n,l]])));
```

3.2.8 2.2 Constraints on the Cross-Domain Control Flow

The control flow can never leave an enclave unless it is done through an approved cross-domain call, as expressed in the following constraints.

1) The only control edges allowed in the cross-domain cut are either call invocations or returns.
2) For any call invocation edge in the cut, the method annotation of the method entry being called must have a CDF that allows (with or without redaction) the level of the label assigned to the call site (caller).

```
constraint :: "EdgeSourceEnclave"
forall (e in PDGEdgeIdx)
(forall (l in nonNullEnclave)
(esEnclave[e,l]==nodeEnclave[hasSource[e],l]));

constraint :: "EdgeDestEnclave"
```

```
forall (e in PDGEdgeIdx)
(forall (l in nonNullEnclave)
(edEnclave[e,l]==nodeEnclave[hasDest[e],l]));

constraint :: "EdgeInEnclaveCut"
forall (e in ControlDep_CallInv)
(
  if (isClassAnnotated[hasClass[hasDest[e]]] == true)
  then
  (
    forall (l in nonNullEnclave)
      (xdedge[e,l]==(esEnclave[e,l]!=edEnclave[e,l]))
  )
  else
  (
    forall (l in nonNullEnclave)
    (xdedge[e,l]==false)
  )
  endif
);

constraint :: "OnlyCallsParamsAndRetsInCut"
forall (e in ControlDep_NonCall)
(forall (l in nonNullEnclave)
(xdedge[e,l]==false));

constraint :: "SourceFunctionAnnotation"
forall (e in ControlDep_CallInv union ControlDep_CallRet)
(forall (l in nonNullEnclave)
(esFunTaint[e,l] ==
(if sourceAnnotFun(e)
then taint[hasFunction[hasSource[e]],l]
else nullCleLabel endif)));

constraint :: "DestFunctionAnnotation"
forall (e in ControlDep_CallInv union ControlDep_CallRet)
(forall (l in nonNullEnclave)(edFunTaint[e,l] ==
(if destAnnotFun(e)
then taint[hasFunction[hasDest[e]],l]
else nullCleLabel endif)));

constraint :: "SourceCdfForDestLevel"
forall (e in ControlDep_CallInv union ControlDep_CallRet)
(forall (l in nonNullEnclave)(esFunCdf[e,l] ==
```

```
(if sourceAnnotFun(e)
then cdfForRemoteLevel[esFunTaint[e,l],
 ↪  hasLabelLevel[taint[hasDest[e],l]]]
else nullCdf endif)));

constraint :: "DestCdfForSourceLevel"
forall (e in ControlDep_CallInv union ControlDep_CallRet)
(forall (l in nonNullEnclave) (edFunCdf[e,l] ==
(if destAnnotFun(e) then
cdfForRemoteLevel[edFunTaint[e,l], hasLabelLevel[taint[hasSource[e],l]]]
 ↪
else nullCdf endif)));

constraint :: "XDCallBlest"
forall (e in ControlDep_CallInv)
(forall (l in nonNullEnclave) ( ( xdedge[e,l]) ->
 ↪  userAnnotatedFunction[hasDest[e]]));

constraint :: "XDCallAllowed"
forall (e in ControlDep_CallInv)
(forall (l in nonNullEnclave) (
xdedge[e,l] -> allowOrRedact(cdfForRemoteLevel[edTaint[e,l],
 ↪  hasLabelLevel[esTaint[e,l]]])));
```

3.2.9 2.3 Constraints on the Cross-Domain Data Flow

Data can only leave an enclave through parameters or return of valid cross-domain call invocations, as expressed in the following three constraints.

1) Any data dependency edge that is not a parameter or data return cannot be in the cross-domain cut.

2) For any data return edge in the cut, the taint of the source node (the returned value in the callee) must have a CDF that allows the data to be shared with the level of the taint of the destination node (the return site in the caller).

3) For any parameter passing edge in the cut, the taint of the source node must have a CDF that allows the data to be shared with the level of the taint of the destination node. This applies to the input parameters going from caller to callee and output parameters going from callee back to the caller.

Note: For cross-domain calls, the callee is assigned to a fixed enclave level. The caller may be unannotated and the label to be considered (e.g. for argument passing checks) would correspond to the label applicable at the level of the caller (instance).

```
constraint :: "XDCDataReturnAllowed"
forall (e in DataDepEdge_Ret)
(forall (l in nonNullEnclave)  (
xdedge[e,l] -> allowOrRedact(cdfForRemoteLevel[esFunTaint[e,l],
 ↪  hasLabelLevel[edTaint[e,l]]])));

constraint :: "XDCParmAllowed"
forall (e in Parameter)
(forall (l in nonNullEnclave)
(xdedge[e,l] -> allowOrRedact(cdfForRemoteLevel[esFunTaint[e,l],
 ↪  hasLabelLevel[edTaint[e,l]]])));
```

3.2.10 2.4 Constraints on Taint Coercion Within Each Enclave

For each level, each node in an unannotated method or constructor must have the same taint as the containing unannotated method or constructor itself.

For each level, for each parameter or data dependency (including returns) edges with at least one endpoint in an unannotated method or constructor, both endpoints must have the same taint.

Unannotated methods can be assigned to multiple enclaves as long as they touch only one taint within that enclave. Annotated methods, on the other hand, can only be assigned to a single enclave/level.

Each node in an annotated method or constructor must have a taint that is allowed by the argument taints (argtaints), code taints (codtaints), or the return taints (rettaints) of the corresponding method/constructor annotation.

For each parameter-in or parameter-out edge connected to an argument of an annotated method or constructor, the taint of the remote (caller side) endpoint must be allowed by the argument taints (argtaints) for that argument in the annotation applied to the method or constructor.

For each data return edge of an annotated method or constructor, the taint of the remote (caller side) endpoint must be allowed by the return taints (rettaints) of the annotation applied to the method or constructor.

For each data dependency edge (that is not a return or parameter edge) of an annotated method or constructor, the taint of both endpoints must be allowed by at least one of the following: argument taints (argtaints), code taints (codtaints), or return taints (rettaints) of the annotation applied to the method or constructor.

```
constraint :: "ArgumentTaintCoerced"
 forall (e in Parameter_In union Parameter_Out)
 (forall (l in nonNullEnclave)
```

```
  (if sourceAnnotFun(e) /\ xdedge[e,l] /\
 ↪  isParam_ActualOut(hasSource[e]) /\ (hasParamIdx[hasSource[e]]>0)
   then hasArgtaints[esFunCdf[e,l], hasParamIdx[hasSource[e]],
 ↪  taint[hasDest[e],l]]
   else true
   endif));

constraint :: "ReturnTaintCoerced"
 forall (e in DataDepEdge_Ret)
 (forall (l in nonNullEnclave)
 ((if sourceAnnotFun(e) /\ xdedge[e,l]
 then hasRettaints[esFunCdf[e,l], taint[hasDest[e],l]]
 else true endif)));

constraint :: "DataTaintCoercedData"
 forall (e in DataEdgeNoRet)
(forall (l in nonNullEnclave)
  (if ( sourceAnnotFun(e))
   then (isInArctaint(esFunTaint[e,l], taint[hasDest[e],l],
 ↪  hasLabelLevel[taint[hasDest[e],l]]) /\
                      isInArctaint(esFunTaint[e,l],
 ↪  taint[hasSource[e],l], hasLabelLevel[taint[hasSource[e],l]]))
   else true
   endif));
```

3.2.11 2.5 Class Constraints

- For each level, all elements of a class that contains no annotated elements must have the same taint.
- All taints on a static field must be at the same level. Unfortunately, this means that a class with a static field can only be assigned to a single enclave. This can be relaxed for final static variables because they will not change.
- The taint(s) of the object reference (this) must be allowed by the code taints (codtaints) of annotated methods. (If the object reference can take multiple labels, then unannotated methods are not possible within that class.)
- The taints of all elements of a class that contains an annotated element must have the same level, and the class is assigned to that enclave/level.

```
constraint :: "ClassEnclaveNonNull"
forall (cl in ClassNames)
  (exists (e in nonNullEnclave)
    (true == classEnclave[cl,e])
```

```
  );

constraint :: "BindClassLevelToEnclave"
forall (cl in ClassNames)
  (forall (e in nonNullEnclave)
    (classTaintedAtLevel[cl, hasEnclaveLevel[e]] == classEnclave[cl,e])
  );

constraint :: "BindClassTaintedAtLevel"
forall (n in PDGNodeIdx)
(
  (forall (e in nonNullEnclave)
    (classTaintedAtLevel[hasClass[n], hasLabelLevel[taint[n,e]]] ==
 ↪  true)
  )
);

constraint :: "BindAnnoClassToEnclaveLevel"
forall (cl in ClassNames)
(
(isClassAnnotated[cl])-> exactlyone(l in
 ↪  nonNullEnclave)(classEnclave[cl,l]));

constraint :: "CheckFieldTaint"
forall (method in FunctionEntry)
(
  forall(field in methodsFieldAccess[method])
  (
    forall (l in nonNullEnclave)
    (
      % Label level for fields of annotated classes is set by annotation
      % This constraint should be reviewed
       (not isClassAnnotated[hasClass[method]])->
       hasLabelLevel[fieldTaint[field,l]] ==
 ↪  hasLabelLevel[ctaint[hasClass[method],l]] /\
      (
      if userAnnotatedFunction[method]==false
      then
        (fieldTaint[field,l] == ftaint[method,l])
      else
      % This may be an issue for demo
        (if field in ClassFields_Static
        then
```

```
        % static field can become multiply tainted at this point since
 ↪  inside annotated function
          false
          %true (demo requires this to be true since it uses several
 ↪  static fields)
        else
          true
        endif)
      endif
      )
    )
  )
);
```

3.2.11.1 Solution Objective In this model, we require the solver to provide a satisfying assignment that minimizes the total number of call invocations that are in the cross-domain cut. Other objectives could be used instead.

```
var int: objective = sum(e in ControlDep_CallInv, l in nonNullEnclave
 ↪  where xdedge[e,l])(1);
solve minimize objective;
```

Once the CAPO partitioning conflict analyzer has analyzed the CLE-annotated application code and determined that all remaining conflicts are resolvable by RPC-wrapping to result in a security-compliant cross-domain partitioned code, the conflict analyzer will produce a topology file (JSON) containing the assignment of every class to an enclave/level. An abbreviated sample topology JSON is provided below. A full-length version can be found in the appendix

```
{
    "enclaves": [
        {
            "level": "green",
            "assignedClasses": [

                ↪  "com.peratonlabs.closure.eop2.level.normal.VideoEndpointNormal",

                ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",

                ↪  "com.peratonlabs.closure.eop2.level.normal.VideoServerNormal"
            ],
            "name": "green_E"
        },
        {
```

```
            "level": "purple",
            "assignedClasses": [
                "com.peratonlabs.closure.eop2.transcoder.Transcoder",
                ↪ "com.peratonlabs.closure.eop2.video.manager.VideoManager"
            ],
            "name": "purple_E"
        },
        {
            "level": "orange",
            "assignedClasses": [
                ↪ "com.peratonlabs.closure.eop2.level.high.VideoEndpointHigh",
                ↪ "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
                ↪ "com.peratonlabs.closure.eop2.level.high.VideoServerHigh"
            ],
            "name": "orange_E"
        }
    ],
    "entry": {
        "mainClass":
         ↪ "com.peratonlabs.closure.eop2.video.manager.VideoManager",
        "enclave": "purple_E",
        "filepath":
         ↪ "./examples/eop2-demo/src/com/peratonlabs/closure/eop2/video/manager/VideoMa
    },
    "cuts": [
        {
            "callee": {
                "level": "orange",
                "type":
                 ↪ "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh"
            },
            "allowedCallers": [
                {
                    "level": "purple",
                    "type":
                     ↪ "com.peratonlabs.closure.eop2.video.manager.VideoManager"
                }
            ],
            "methodSignature": {
                "parameterTypes": [
```

```
                "int",
                "java.lang.String"
            ],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
            "name": "start",
            "returnType": "void"
        }
    },
    {
        "callee": {
            "level": "green",
            "type":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal"
        },
        "allowedCallers": [
            {
                "level": "purple",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager"
            }
        ],
        "methodSignature": {
            "parameterTypes": [
                "int",
                "java.lang.String"
            ],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
            "name": "start",
            "returnType": "void"
        }
    },
    {
        "callee": {
            "level": "orange",
            "type":
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh"
        },
        "allowedCallers": [
            {
                "level": "purple",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager"
```

```
            }
        ],
        "methodSignature": {
            "parameterTypes": [],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
            "name": "getRequest",
            "returnType":
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestHigh"
        }
    },
    {
        "callee": {
            "level": "green",
            "type":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal"
        },
        "allowedCallers": [
            {
                "level": "purple",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager"
            }
        ],
        "methodSignature": {
            "parameterTypes": [],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
            "name": "getRequest",
            "returnType":
             ↪  "com.peratonlabs.closure.eop2.video.requester.Request"
        }
    },
    {
        "callee": {
            "level": "orange",
            "type":
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh"
        },
        "allowedCallers": [
            {
                "level": "purple",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.transcoder.Transcoder"
```

```
                }
            ],
            "methodSignature": {
                "parameterTypes": [
                    "java.lang.String",
                    "byte[]"
                ],
                "fqcn":
                 ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
                "name": "send",
                "returnType": "void"
            }
        },
        {
            "callee": {
                "level": "green",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal"
            },
            "allowedCallers": [
                {
                    "level": "purple",
                    "type":
                     ↪  "com.peratonlabs.closure.eop2.transcoder.Transcoder"
                }
            ],
            "methodSignature": {
                "parameterTypes": [
                    "java.lang.String",
                    "byte[]"
                ],
                "fqcn":
                 ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
                "name": "send",
                "returnType": "void"
            }
        }
    ]
}
```

3.2.12 Remarks and Limitations

- A limitation of the current model is that it supports at most one enclave per level

- Class annotations are currently not used by CLE, but this can change in the future
- Class static fields are handled imprecisely
- No mechanism exists to apply user-defined function annotations to a lambda function
- The size of programs that can be analyzed is limited by the capabilities of JOANA (and in turn IBM WALA)
- Children of classes with annotated elements cannot be annotated for placement in an enclave with a different level, so it is best to keep annotations close to the leaf of the inheritance hierarchy

3.3 Auto Generation of Aspects for Partition Enforcement and Cross-Domain Communications

Given the annotated application and the topology, the Java code generation tool, CodeGenJava, does the following:

- creates a directory for each partition and copies the original app code into it; directory
- generates AspectJ definitions for each partition; aspect-example
- generates cross-domain tags and HAL configurations; xdconf and hal-purple.cfg
- generates remote procedure call handlers; handler
- generates an ant build script for each enclave build.xml and
- compiles and aspect weaves the resulting code.

Aspect-oriented programming (AOP) [7] is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does so by adding behavior to existing code ("advice") without modifying the code itself, instead separately specifying which code is modified (a "pointcut" specification). For example, an aspect can add behavior to log all function calls when the function name begins with `set`.

Aspect-oriented programming, illustrated in the diagram below from Cerny's dissertation [8], has the benefits of clean modularization of crosscutting concerns. For CLOSURE, this also means that the annotated source code need not be physically divided or modified, as the aspects are woven in by the aspect compiler [9] and weaver when generating the partitioned executable.

CLOSURE's approach to AOP is the following. A developer annotates Java application code using CLE and performs cross-domain analysis at the Java/Dalvik bytecode level. AspectJ code is auto-generated by CLOSURE, so the programmer need not learn AOP concepts. The CVI build process takes care of the invocation of the AspectJ compiler.

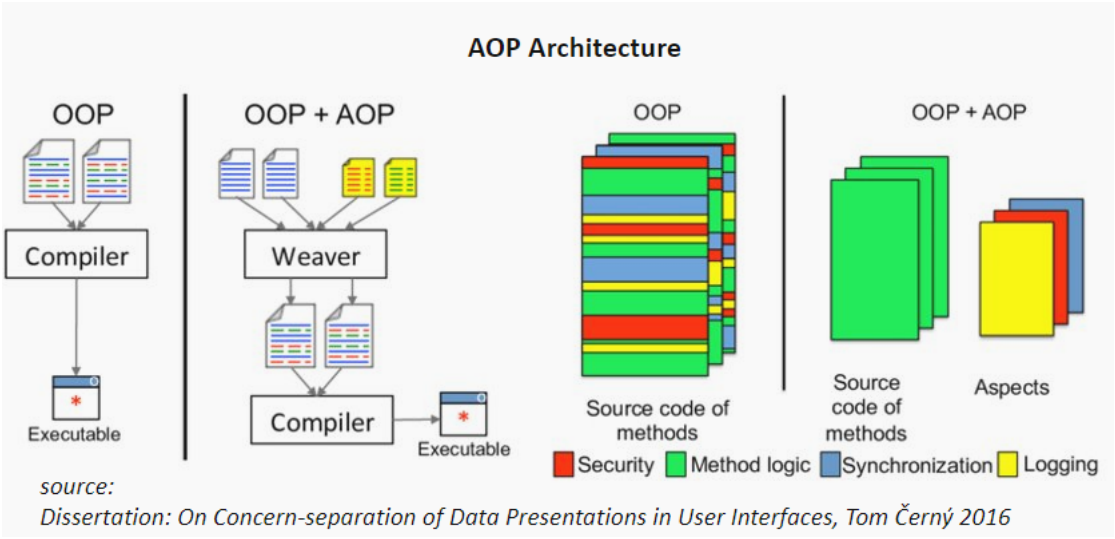


Figure 3: Aspect-Oriented Programming Concept

Based on the 'cuts' in the topology JSON file, CodeGenJava generates the necessary Aspect definitions to intercept cross-domain calls and forward them to the remote enclave via the HAL layer. The following diagram depicts the process of object instantiation and method invocation.

For cross-domain object instantiation, the generated AspectJ pointcut corresponding to the constructor intercepts the invocation, generates a shadow object and assigns an object id (oid). The constructor invocation and oid are then serialized to the remote enclave through RPC over HAL. The remote handler deserializes the constructor call, unmarshalls the arguments, instantiates the object by calling the constructor, and stores a local instance along with a map between the oid and the instance.

Cross-domain method invocations are handled similarly. For invoking an instance method, the Aspect defined on the caller side looks up the oid corresponding to the object, serializes the method request along with and oid and sends to the remote enclave through RPC over HAL.

The remote handler deserializes the call, looks up the object corresponding to the oid and invokes the specified method. The return value is serialized and passed back along the reverse path. Upon receiving the return value, the generated aspect code deserializes the response, unmarshalls the arguments, and provides it to the caller method.

To build CodeGenJava from the source, do the following:

```
$ cd CodeGenJava
$ ant
```

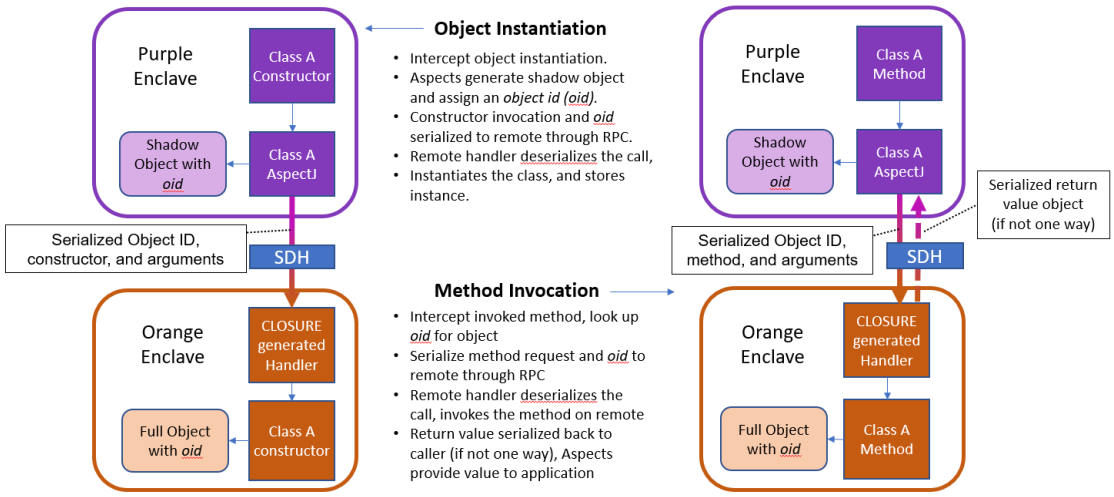


Figure 4: Constructor and Method Invocation

If successful, this will create a directory named code-gen containing a jar file, `code-gen.jar`, and a subdirectory named `resources`. For convenience in deployment, a zip file, `code-gen.zip`, which contains the same contents as the `code-gen` directory is also generated. See the AspectJ programming guide [10] for more details on aspect syntax used by the generated code.

The usage of the program CodeGenJava is straightforward:

```
$ java -jar code-gen/code-gen.jar -h
GAPS/Closure Java Code Generator
  -h/--help                    this help
  -c/--cutJson <cut.json>      cut JSON file (test/cut.json)
  -d/--dstDir  <pathname>      destination directory of the generated
   ↪  code (/home/closure/gaps/xdcc)
  -f/--config  <config.json>   config JSON file
  -i/--codeDir <source code>   code directory relative to srcDir (.)
  -j/--jar     <jar name>      name of the application jar file
   ↪  (TESTPROGRAM)
  -p/--compile <true|false>    Compile the code after partition (true)
  -s/--srcDir  <app src dir>   application source code
   ↪  (/home/closure/gaps/capo/Java/examples/eop2-demo)
```

Without arguments, CodeGenJava uses the default arguments shown in the parentheses at the end of the options above. A JSON config file can also be provided. An excerpt of the config is given below. A full sample config.json is given in the appendix.

```
{
  "dstDir": "/home/closure/xdcc",
```

```
  "cut": "test/cut.json",
  "srcDir": "/home/closure/gaps/capo/Java/examples/eop2-demo",
  "codeDir": ".",
  "jar": "TESTPROGRAM",
  "compile": true,
}
```

3.4 Interfacing with HAL

With the CLOSURE C toolchain [1], partitioned application programs use the xdcomms API library to interface with the HAL daemon which exchanges data through the GAPS Devices.

In the Java toolchain, there is no separate xdcomms API library, and the interaction with the HAL daemon is handled by the code generated by the CodeGenJava tool described in the previous section. AspectJ-woven applications interact with the HAL through the HalZmq class halzmq. HalZmq provides methods for marshaling/unmarshalling, serialization/de-serialization, and read/write to HAL, abstracting the details from the applications and the cross-domain software developer. The autogenerated code invokes methods provided by the HalZmq class.

The following flow diagram shows the steps taken from the method invocation intercepted by Aspect pointcut in one enclave to the point in another enclave where the actual method is invoked.

3.5 Example applications

We include the EoP2 application, a small example application, which is used to illustrate the capabilities of the CLOSURE toolchain for Java.

The application, when partitioned, creates a video-processing enclave that processes video from a webcam or IP camera, transcodes it, and sends it to two web servers, from which browsers can receive the video. Various filters can be applied to these frames using OpenCV to modify the images. The two web pages operate on different enclaves and permit different filters to alter the images sent to the web clients.

The quality of the frame received by the server depends on the level it is operating at. In the example, the video server running at level orange can receive higher fidelity frames. On the other hand, the video server running at level green can only receive lower fidelity frames (e.g. lower resolution, greyscale, etc.). The diagram below illustrates the intra-enclave and inter-enclave flows in the example application.

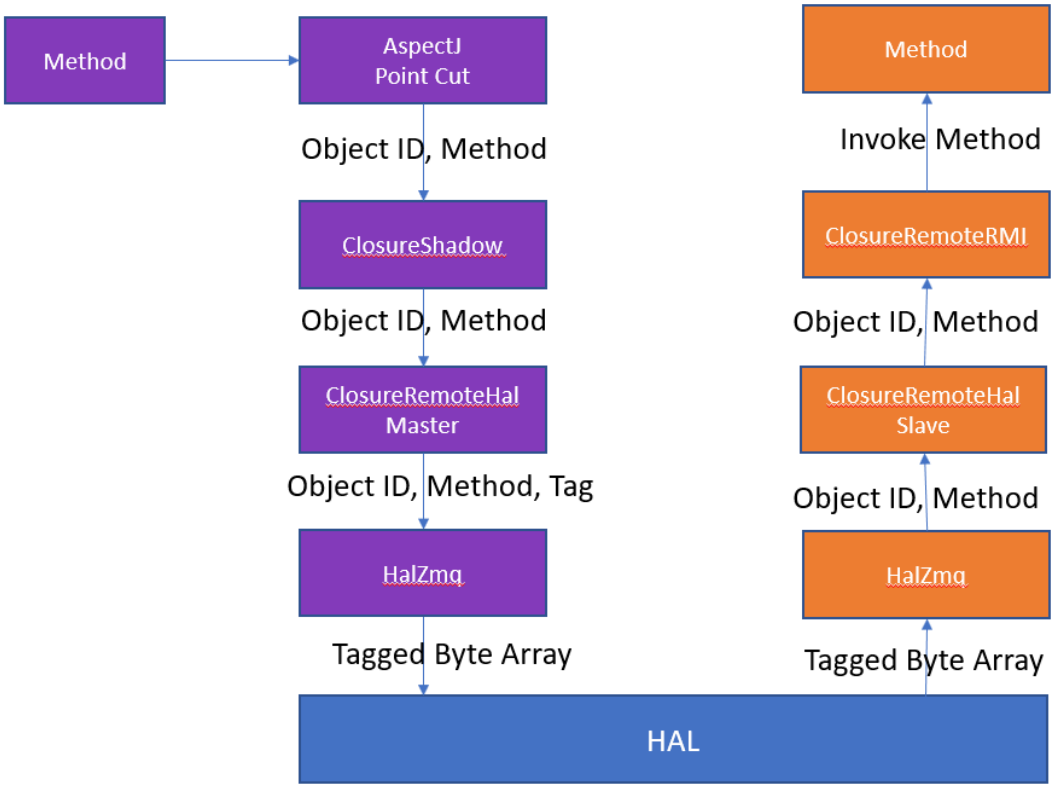


Figure 5: Java Cross-Domain Call Flow

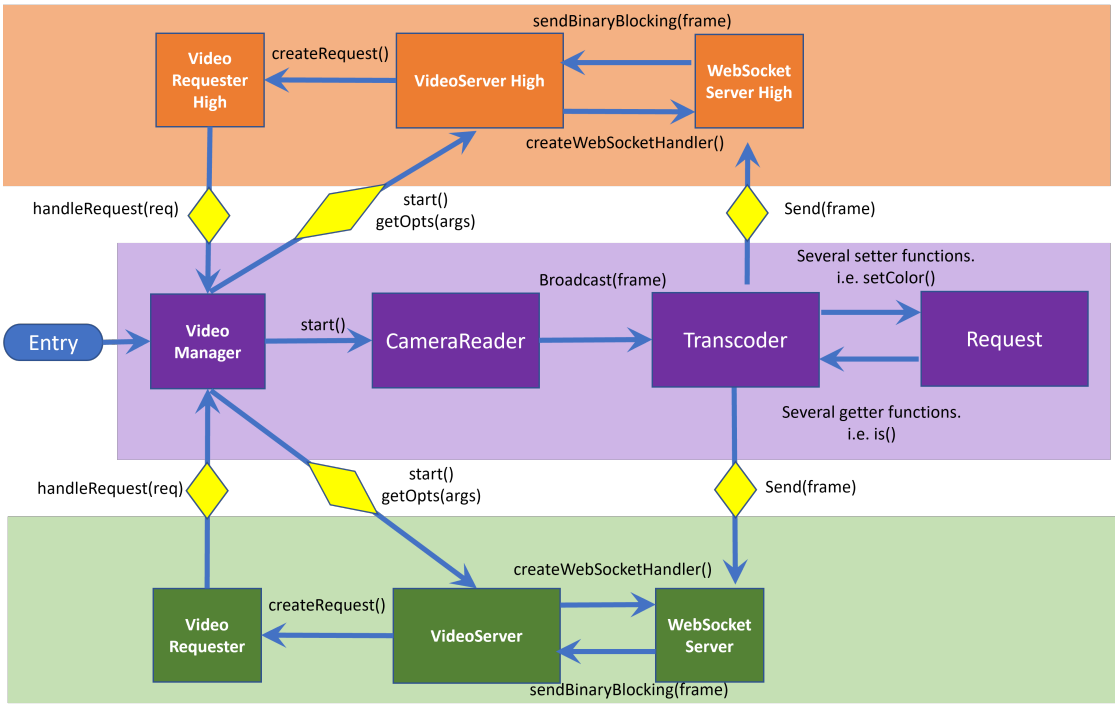


Figure 6: The diagram above shows the cross-domain flows for the EoP2 application

4 Limitations and Future Work

4.1 Limitations and language coverage

CLOSURE currently supports a subset of the Java language version 8. Notable current limitations are a lack of support for multi-threading applications and annotating lambda functions. Additionally, some underlying toolchains used have limited support for large programs. Lastly, we currently do not support Android applications. These language limitations are currently being addressed and we plan on supporting them in future releases. The CLOSURE Java toolchain has been demonstrated to support up to 3 enclaves, and can conceptually reason about an arbitrary number of enclaves.

4.2 Future Work

In future work, we will work on relaxing the known limitations. Also in the research pipeline are:

1. Support for Android applications
2. Support for analysis and partitioning of large applications
3. More complete coverage of the Java language

5 Appendices

5.1 CLE JSON example and schema

With CLE annotations for Java, we use the same CLE-JSON schema as with the C toolchain [1]. We describe an example CLE-JSON used with an annotation and the CLE-JSON schema below.

5.1.1 Example

Below is an example of `cle-json`. From the source code, the preprocessor produces a json with an array of label-json pairs, which are objects with two fields `"cle-label"` and `"cle-json"` for the label name and label definition/json respectively.

```
[
  {
    "cle-label": "PURPLE",
    "cle-json": {
      "level": "purple"
    }
  },
  {
    "cle-label": "ORANGE",
    "cle-json": {
      "level": "orange",
      "cdf": [
        {
          "remotelevel": "purple",
          "direction": "egress",
          "guarddirective": {
            "operation": "allow"
          }
        }
      ]
    }
  }
]
```

5.1.2 Schema

The preprocessor validates cle-json produced from the source code using jsonschema. The schema for cle-json is shown in detail below:

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "com.perspectalabs.gaps-closure.cle",
    "$comment": "JSON schema for GAPS-Closure CLE json definition",

    "oneOf":[
        {
            "description": "List of CLE entries",
            "type": "array",
            "default": [],
            "items": { "$ref": "#/definitions/cleLabel" }
        },
        {
          "$ref": "#/definitions/rootNode"
        }
    ],

    "definitions": {
        "guarddirectiveOperationTypes": {
            "$comment": "the guarddirective type enum",
            "description": "[ENUM] Guard Directive",
            "enum": [
                "allow",
                "block",
                "redact"
            ]
        },
        "directionTypes": {
            "$comment": "the direction type enum",
            "description": "[ENUM] traffic direction",
            "type": "string",
            "enum": [
                "egress",
                "ingress",
                "bidirectional"
            ]
        },

        "guarddirectiveTypes":{
            "description": "Guard Directive parameters",
            "type": "object",
            "properties": {
                "operation":{
                    "$ref": "#/definitions/guarddirectiveOperationTypes"
```

```
        },
        "oneway": {
            "description": "Communication only in one
             ↪  direction",
            "type": "boolean",
            "default": false

        },
        "gapstag": {
            "description": "Gaps tag to link remote CLE data
             ↪  [mux,sec,type]",
            "type": "array",
            "maxLength": 3,
            "minLength": 3,
            "items":[
                {
                    "type": "number",
                    "minimum": 0,
                    "description": "mux value"
                },
                {
                    "type": "number",
                    "minimum": 0,
                    "description": "sec value"
                },
                {
                    "type": "number",
                    "minimum": 0,
                    "description": "type value"
                }
            ]
        }
    }
},

"argtaintsTypes":{
    "description": "argument taints",
    "type": "array",
    "default": [],
    "uniqueItems": false,
    "items": {
        "description": "Taint levels of each argument",
        "type": "array",
        "default": [],
```

```
            "items":{
                "type": "string",
                "description": "CLE Definition Name"
            }
        }
    },

    "cdfType": {
        "description": "Cross Domain Flow",
        "type": "object",
        "properties": {
            "remotelevel":{
                "description": "The remote side's Enclave",
                "type": "string"
            },
            "direction":{
                "$ref": "#/definitions/directionTypes"
            },
            "guarddirective":{
                "$comment": "active version guarddirective",
                "$ref": "#/definitions/guarddirectiveTypes"
            },
            "guardhint":{
                "$comment": "deprecated version of guarddirective",
                "$ref": "#/definitions/guarddirectiveTypes"
            },
            "argtaints":{
                "$ref": "#/definitions/argtaintsTypes"
            },
            "codtaints":{
                "description": "Taint level",
                "type": "array",
                "default": [],
                "items":{
                    "type": "string",
                    "description": "CLE Definition Name"
                }
            },
            "rettaints":{
                "description": "Return level",
                "type": "array",
                "default": [],
                "items":{
                    "type": "string",
```

```
                "description": "CLE Definition Name"
            }
        },
        "idempotent":{
            "description": "Idempotent Function",
            "type": "boolean",
            "default": true
        },
        "num_tries":{
            "description": "Num tries",
            "type": "number",
            "default": 5
        },
        "timeout":{
            "description": "Timeout",
            "type": "number",
            "default": 1000
        },
        "pure":{
            "description": "Pure Function",
            "type": "boolean",
            "default": false
        }
    },
    "dependencies": {
        "argtaints": {
            "required": ["argtaints", "codtaints", "rettaints"]
        },
        "codtaints": {
            "required": ["argtaints", "codtaints", "rettaints"]
        },
        "rettaints": {
            "required": ["argtaints", "codtaints", "rettaints"]
        }
    },
    "oneOf":[
        {
            "required": ["remotelevel", "direction",
             ↪  "guarddirective"]
        },
        {
            "required": ["remotelevel", "direction",
             ↪  "guardhint"]
        }
```

```
    ]
},

"cleLabel":{
    "type": "object",
    "required": ["cle-label", "cle-json"],
    "description": "CLE Lable (in full clemap.json)",
    "additionalProperties": false,

    "properties": {
        "cle-label": {
            "description": "Name of the CLE label",
            "type": "string"
        },
        "cle-json":{
            "$ref": "#/definitions/rootNode"
        }
    }
},

"rootNode":{
    "type": "object",
    "required": ["level"],
    "description": "CLE Definition",
    "additionalProperties": false,
    "properties": {
        "$schema":{
            "description": "The cle-schema reference (for
             ↪  standalone json files)",
            "type": "string"
        },
        "$comment":{
            "description": "Optional comment entry",
            "type": "string"
        },
        "level":{
            "description": "The enclave level",
            "type":"string"
        },
        "cdf": {
            "description": "List of cross domain flows",
            "type": "array",
            "uniqueItems": true,
            "default": [],
```

```
                    "items": { "$ref": "#/definitions/cdfType" }
                }
            }
        }
    }
}
```

5.2 System Dependency Graph (SDG)

The following introduces the node and edge types used by JOANA in the resulting SDG.

5.2.1 Node Types

The following lists the node types used in the SDG.

```
node_kind returns [SDGNode.Kind kind]
  : 'NORM' { kind = SDGNode.Kind.NORMAL; }
  | 'PRED' { kind = SDGNode.Kind.PREDICATE; }
  | 'EXPR' { kind = SDGNode.Kind.EXPRESSION; }
  | 'ENTR' { kind = SDGNode.Kind.ENTRY; }
  | 'CALL' { kind = SDGNode.Kind.CALL; }
  | 'ACTI' { kind = SDGNode.Kind.ACTUAL_IN; }
  | 'ACTO' { kind = SDGNode.Kind.ACTUAL_OUT; }
  | 'FRMI' { kind = SDGNode.Kind.FORMAL_IN; }
  | 'FRMO' { kind = SDGNode.Kind.FORMAL_OUT; }
  | 'EXIT' { kind = SDGNode.Kind.EXIT; }
  | 'SYNC' { kind = SDGNode.Kind.SYNCHRONIZATION; }
  | 'FOLD' { kind = SDGNode.Kind.FOLDED; }
  ;
```

In our MiniZinc model, we map each SDG node to a more convenient type using the following map below.

```
  nodeConversion = {
    "NORM" : "Inst_Other",
    "PRED" : "Inst_Br",
    "EXPR" : "Inst_Other",
    "SYNC" : "Inst_Other",
    "FOLD" : "Inst_Other",
    "CALL" : "Inst_FunCall",
    "ENTR" : "FunctionEntry",
    "EXIT" : "Inst_Ret",
```

```
    "ACTI" : "Param_ActualIn",
    "ACTO" : "Param_ActualOut",
    "FRMI" : "Param_FormalIn",
    "FRMO" : "Param_FormalOut",
}
```

5.2.2 Node Operations

The following lists the node operations used by the SDG.

```
node_oper returns [SDGNode.Operation op]
  : 'empty'          { op = SDGNode.Operation.EMPTY; }
  | 'intconst'       { op = SDGNode.Operation.INT_CONST; }
  | 'floatconst'     { op = SDGNode.Operation.FLOAT_CONST; }
  | 'charconst'      { op = SDGNode.Operation.CHAR_CONST; }
  | 'stringconst'    { op = SDGNode.Operation.STRING_CONST; }
  | 'functionconst'  { op = SDGNode.Operation.FUNCTION_CONST; }
  | 'shortcut'       { op = SDGNode.Operation.SHORTCUT; }
  | 'question'       { op = SDGNode.Operation.QUESTION; }
  | 'binary'         { op = SDGNode.Operation.BINARY; }
  | 'unary'          { op = SDGNode.Operation.UNARY; }
  | 'derefer'        { op = SDGNode.Operation.DEREFER; }
  | 'refer'          { op = SDGNode.Operation.REFER; }
  | 'array'          { op = SDGNode.Operation.ARRAY; }
  | 'select'         { op = SDGNode.Operation.SELECT; }
  | 'reference'      { op = SDGNode.Operation.REFERENCE; }
  | 'declaration'    { op = SDGNode.Operation.DECLARATION; }
  | 'modify'         { op = SDGNode.Operation.MODIFY; }
  | 'modassign'      { op = SDGNode.Operation.MODASSIGN; }
  | 'assign'         { op = SDGNode.Operation.ASSIGN; }
  | 'IF'             { op = SDGNode.Operation.IF; }
  | 'loop'           { op = SDGNode.Operation.LOOP; }
  | 'jump'           { op = SDGNode.Operation.JUMP; }
  | 'compound'       { op = SDGNode.Operation.COMPOUND; }
  | 'call'           { op = SDGNode.Operation.CALL; }
  | 'entry'          { op = SDGNode.Operation.ENTRY; }
  | 'exit'           { op = SDGNode.Operation.EXIT; }
  | 'form-in'        { op = SDGNode.Operation.FORMAL_IN; }
  | 'form-ellip'     { op = SDGNode.Operation.FORMAL_ELLIP; }
  | 'form-out'       { op = SDGNode.Operation.FORMAL_OUT; }
  | 'act-in'         { op = SDGNode.Operation.ACTUAL_IN; }
  | 'act-out'        { op = SDGNode.Operation.ACTUAL_OUT; }
```

```
  | 'monitor'        { op = SDGNode.Operation.MONITOR; }
  ;
```

5.2.3 Edge Types

The following lists the edge operations used by the SDG.

```
private edge_kind returns [SDGEdge.Kind kind]
// data dependencies
  : 'DD' { kind = SDGEdge.Kind.DATA_DEP; }           // data
 ↪  dependencies between local variables
  | 'DH' { kind = SDGEdge.Kind.DATA_HEAP; }          // data
 ↪  dependencies between field accesses
  | 'DA' { kind = SDGEdge.Kind.DATA_ALIAS; }         // data
 ↪  dependencies between aliasing fields accesses
// control dependencies
  | 'CD' { kind = SDGEdge.Kind.CONTROL_DEP_COND; }   // control
 ↪  dependencies between statements
  | 'CE' { kind = SDGEdge.Kind.CONTROL_DEP_EXPR; }   // control
 ↪  dependencies between nodes that correspond to the same statement
  | 'UN' { kind = SDGEdge.Kind.CONTROL_DEP_UNCOND; } // unconditional
 ↪  control dependencies
// control flow
  | 'CF' { kind = SDGEdge.Kind.CONTROL_FLOW; }       // control flow
 ↪  between statements
  | 'NF' { kind = SDGEdge.Kind.NO_FLOW; }            // control flow
 ↪  that is actually not possible (dead code)
  | 'RF' { kind = SDGEdge.Kind.RETURN; }             // control flow
 ↪  from method exit to call site
// method call related
  | 'CC' { kind = SDGEdge.Kind.CONTROL_DEP_CALL; }
  | 'CL' { kind = SDGEdge.Kind.CALL; }
  | 'PI' { kind = SDGEdge.Kind.PARAMETER_IN; }
  | 'PO' { kind = SDGEdge.Kind.PARAMETER_OUT; }
// summary edges
  | 'SU' { kind = SDGEdge.Kind.SUMMARY; }
  | 'SH' { kind = SDGEdge.Kind.SUMMARY_NO_ALIAS; }
  | 'SF' { kind = SDGEdge.Kind.SUMMARY_DATA; }
// method interface structure
  | 'PS' { kind = SDGEdge.Kind.PARAMETER_STRUCTURE; }
  | 'PE' { kind = SDGEdge.Kind.PARAMETER_EQUIVALENCE; }
// thread/concurrency related edges
  | 'FORK' { kind = SDGEdge.Kind.FORK; }
```

```
  | 'FORK_IN' { kind = SDGEdge.Kind.FORK_IN; }
  | 'FORK_OUT' { kind = SDGEdge.Kind.FORK_OUT; }
  | 'JOIN' { kind = SDGEdge.Kind.JOIN; }
  | 'ID' { kind = SDGEdge.Kind.INTERFERENCE; }
  | 'IW' { kind = SDGEdge.Kind.INTERFERENCE_WRITE; }
  | 'SD' { kind = SDGEdge.Kind.SYNCHRONIZATION; }
// general helper edges
  | 'HE' { kind = SDGEdge.Kind.HELP; }
  | 'FD' { kind = SDGEdge.Kind.FOLDED; }
  | 'FI' { kind = SDGEdge.Kind.FOLD_INCLUDE; }
// deprecated edges
  | 'RY' { kind = SDGEdge.Kind.READY_DEP; }
  | 'JF' { kind = SDGEdge.Kind.JUMP_FLOW; }
  | 'SP' { kind = SDGEdge.Kind.SUMMARY; }
  | 'VD' { kind = SDGEdge.Kind.DATA_DEP_EXPR_VALUE; }
  | 'RD' { kind = SDGEdge.Kind.DATA_DEP_EXPR_REFERENCE; }
  | 'JD' { kind = SDGEdge.Kind.JUMP_DEP; }
```

In our MiniZinc model, we map each SDG edge to a more convenient type using the following map below.

```
edgeConversion = {
    "CD" : "ControlDep_Other",
    "CE" : "ControlDep_Other",
    "UN" : "ControlDep_Other",
    "CF" : "ControlDep_Other",
    "NF" : "ControlDep_Other",
    "RF" : "ControlDep_CallRet",
    "CC" : "ControlDep_CallInv",
    "CL" : "ControlDep_CallInv",
    "SD" : "ControlDep_Other",
    "JOIN" : "ControlDep_Other",
    "FORK" : "ControlDep_Other",
    "DD" : "DataDepEdge_Other",
    "DH" : "DataDepEdge_Other",
    "DA" : "DataDepEdge_Alias",
    "SU" : "DataDepEdge_Other",
    "SH" : "DataDepEdge_Other",
    "SF" : "DataDepEdge_Other",
    "FD" : "DataDepEdge_Other",
    "FI" : "DataDepEdge_Other",
    "PI" : "Parameter_In",
    "PO" : "Parameter_Out",
    "PS" : "Parameter_Field",
```

```
    "PE" : "DataDepEdge_Alias",
    "FORK_IN" : "DataDepEdge_Other",
    "FORK_OUT" : "DataDepEdge_Other",
    "ID" : "DataDepEdge_Other",
    "IW" : "DataDepEdge_Other",
}
```

5.3 The cross-domain cut specification: cut.json

The `cut.json` file is a description of level and enclave assignments produced by the conflict analyzer and is used as input for the java code generator. It also contains information about the callee and caller that will be in the cut.

The `cut.json` contains:

1. the set of enclaves and levels relevant to the program
2. an assignment from each class to a level and an enclave
3. Callee and caller info for cross-domain calls

The `cut.json` generated for eop2 is as follows:

```
{
    "codeDir": "examples",
    "enclaves": [
        {
            "level": "green",
            "assignedClasses": [
                "com.peratonlabs.closure.eop2.camera.CameraReader",
                "com.peratonlabs.closure.eop2.camera.CameraType",
                "com.peratonlabs.closure.eop2.level.VideoRequester",
                "com.peratonlabs.closure.eop2.level.VideoServer",
                 ↪  "com.peratonlabs.closure.eop2.level.high.VideoEndpointHigh",
                 ↪  "com.peratonlabs.closure.eop2.level.normal.VideoEndpointNormal",
                 ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
                 ↪  "com.peratonlabs.closure.eop2.level.normal.VideoServerNormal",
                "com.peratonlabs.closure.eop2.video.manager.Config",
                 ↪  "com.peratonlabs.closure.eop2.video.manager.GetHttpSessionConfigurat
                 ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager",
```

```
            "com.peratonlabs.closure.eop2.video.requester.Request",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestDecoder",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestDecoderHigh",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestEncoder",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestEncoderHigh",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestHigh"
        ],
        "name": "green_E"
    },
    {
        "level": "purple",
        "assignedClasses": [
            "com.peratonlabs.closure.eop2.camera.CameraReader",
            "com.peratonlabs.closure.eop2.camera.CameraType",
            "com.peratonlabs.closure.eop2.level.VideoRequester",
            "com.peratonlabs.closure.eop2.level.VideoServer",
            ↪ "com.peratonlabs.closure.eop2.level.high.VideoEndpointHigh",
            ↪ "com.peratonlabs.closure.eop2.level.normal.VideoEndpointNormal",
            "com.peratonlabs.closure.eop2.transcoder.Transcoder",
            "com.peratonlabs.closure.eop2.video.manager.Config",
            ↪ "com.peratonlabs.closure.eop2.video.manager.GetHttpSessionConfigurato
            ↪ "com.peratonlabs.closure.eop2.video.manager.VideoManager",
            "com.peratonlabs.closure.eop2.video.requester.Request",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestDecoder",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestDecoderHigh",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestEncoder",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestEncoderHigh",
            ↪ "com.peratonlabs.closure.eop2.video.requester.RequestHigh"
        ],
        "name": "purple_E"
```

```
    },
    {
        "level": "orange",
        "assignedClasses": [
            "com.peratonlabs.closure.eop2.camera.CameraReader",
            "com.peratonlabs.closure.eop2.camera.CameraType",
            "com.peratonlabs.closure.eop2.level.VideoRequester",
            "com.peratonlabs.closure.eop2.level.VideoServer",
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoEndpointHigh",
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoServerHigh",
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoEndpointNormal",
            "com.peratonlabs.closure.eop2.video.manager.Config",
             ↪  "com.peratonlabs.closure.eop2.video.manager.GetHttpSessionConfigurato
             ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager",
            "com.peratonlabs.closure.eop2.video.requester.Request",
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoder",
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoderHigh",
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoder",
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoderHigh",
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestHigh"
        ],
        "name": "orange_E"
    }
],
"rootDir": "/home/rbrotzman/gaps/build/src/capo/Java",
"assingments": [
    {
        "className":
         ↪  "com.peratonlabs.closure.eop2.camera.CameraReader",
        "enclave": "green_E"
    },
    {
```

```
    "className":
     ↪  "com.peratonlabs.closure.eop2.camera.CameraType",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.VideoRequester",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.VideoServer",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.high.VideoEndpointHigh",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.normal.VideoEndpointNormal",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.normal.VideoServerNormal",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.Config",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.GetHttpSessionConfigurator",
    "enclave": "green_E"
},
```

```
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.Request",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoder",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoderHigh",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoder",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoderHigh",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestHigh",
    "enclave": "green_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.camera.CameraReader",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.camera.CameraType",
    "enclave": "purple_E"
```

```
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.VideoRequester",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.VideoServer",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.high.VideoEndpointHigh",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.normal.VideoEndpointNormal",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.transcoder.Transcoder",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.Config",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.GetHttpSessionConfigurator",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.Request",
```

```
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoder",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoderHigh",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoder",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoderHigh",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.RequestHigh",
    "enclave": "purple_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.camera.CameraReader",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.camera.CameraType",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.VideoRequester",
    "enclave": "orange_E"
},
{
```

```
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.VideoServer",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.high.VideoEndpointHigh",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.high.VideoServerHigh",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.level.normal.VideoEndpointNormal",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.Config",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.GetHttpSessionConfigurator",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager",
    "enclave": "orange_E"
},
{
    "className":
     ↪  "com.peratonlabs.closure.eop2.video.requester.Request",
    "enclave": "orange_E"
},
```

```
        {
            "className":
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoder",
            "enclave": "orange_E"
        },
        {
            "className":
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestDecoderHigh",
            "enclave": "orange_E"
        },
        {
            "className":
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoder",
            "enclave": "orange_E"
        },
        {
            "className":
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestEncoderHigh",
            "enclave": "orange_E"
        },
        {
            "className":
             ↪  "com.peratonlabs.closure.eop2.video.requester.RequestHigh",
            "enclave": "orange_E"
        }
    ],
    "entry": {
        "mainClass":
         ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager",
        "enclave": "purple_E",
        "filepath":
         ↪  "./examples/eop2-demo/src/com/peratonlabs/closure/eop2/video/manager/VideoMa
    },
    "jar": "TESTPROGRAM.jar",
    "cuts": [
        {
            "callee": {
                "level": "orange",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh"
            },
            "allowedCallers": [
                {
                    "level": "purple",
```

```
                "type":
                 ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager"
            }
        ],
        "methodSignature": {
            "parameterTypes": [
                "int",
                "java.lang.String"
            ],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
            "name": "start",
            "returnType": "void"
        }
    },
    {
        "callee": {
            "level": "green",
            "type":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal"
        },
        "allowedCallers": [
            {
                "level": "purple",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager"
            }
        ],
        "methodSignature": {
            "parameterTypes": [
                "int",
                "java.lang.String"
            ],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
            "name": "start",
            "returnType": "void"
        }
    },
    {
        "callee": {
            "level": "orange",
            "type":
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh"
```

```
    },
    "allowedCallers": [
        {
            "level": "purple",
            "type":
             ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager"
        }
    ],
    "methodSignature": {
        "parameterTypes": [],
        "fqcn":
         ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
        "name": "getRequest",
        "returnType":
         ↪  "com.peratonlabs.closure.eop2.video.requester.RequestHigh"
    }
},
{
    "callee": {
        "level": "green",
        "type":
         ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal"
    },
    "allowedCallers": [
        {
            "level": "purple",
            "type":
             ↪  "com.peratonlabs.closure.eop2.video.manager.VideoManager"
        }
    ],
    "methodSignature": {
        "parameterTypes": [],
        "fqcn":
         ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
        "name": "getRequest",
        "returnType":
         ↪  "com.peratonlabs.closure.eop2.video.requester.Request"
    }
},
{
    "callee": {
        "level": "orange",
        "type":
         ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh"
```

```
        },
        "allowedCallers": [
            {
                "level": "purple",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.transcoder.Transcoder"
            }
        ],
        "methodSignature": {
            "parameterTypes": [
                "java.lang.String",
                "byte[]"
            ],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh",
            "name": "send",
            "returnType": "void"
        }
    },
    {
        "callee": {
            "level": "green",
            "type":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal"
        },
        "allowedCallers": [
            {
                "level": "purple",
                "type":
                 ↪  "com.peratonlabs.closure.eop2.transcoder.Transcoder"
            }
        ],
        "methodSignature": {
            "parameterTypes": [
                "java.lang.String",
                "byte[]"
            ],
            "fqcn":
             ↪  "com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal",
            "name": "send",
            "returnType": "void"
        }
    }
]
```

```
} []
}
```

5.4 Constraint Model in MiniZinc

The following contains type declarations for the MiniZinc model used within the conflict analyzer. These type declarations, along with a model instance generated in python are inputted to MiniZinc with the constraints to either produce a satisfiable assignment or some minimally unsatisfiable set of constraints.

5.4.1 Type declarations

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SDG Nodes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

int: Inst_FunCall_start;
int: Inst_FunCall_end;
int: Inst_Ret_start;
int: Inst_Ret_end;
int: Inst_Br_start;
int: Inst_Br_end;
int: Inst_Other_start;
int: Inst_Other_end;
int: Inst_start;
int: Inst_end;

int: FunctionEntry_start;
int: FunctionEntry_end;

% Need to check that there isn't an issue with non-contiguious arg
↪  indicies
int: Param_FormalIn_start;
int: Param_FormalIn_end;
int: Param_FormalOut_start;
int: Param_FormalOut_end;
int: Param_ActualIn_start;
int: Param_ActualIn_end;
int: Param_ActualOut_start;
int: Param_ActualOut_end;
int: Param_start;
int: Param_end;
```

```
int: PDGNode_start;
int: PDGNode_end;

set of int: Inst = Inst_start .. Inst_end;
set of int: FunCall = Inst_FunCall_start .. Inst_FunCall_end;

set of int: FunctionEntry = FunctionEntry_start .. FunctionEntry_end;

set of int: Param_FormalIn = Param_FormalIn_start .. Param_FormalIn_end;
set of int: Param_FormalOut = Param_FormalOut_start ..
 ↪  Param_FormalOut_end;
set of int: Param_ActualIn = Param_ActualIn_start .. Param_ActualIn_end;
set of int: Param_ActualOut = Param_ActualOut_start ..
 ↪  Param_ActualOut_end;
set of int: Param = Param_start .. Param_end;

set of int: PDGNodeIdx  = PDGNode_start .. PDGNode_end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SDG Edges
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

int: ControlDep_CallInv_start;
int: ControlDep_CallInv_end;
int: ControlDep_CallRet_start;
int: ControlDep_CallRet_end;
int: ControlDep_Other_start;
int: ControlDep_Other_end;
int: ControlDep_start;
int: ControlDep_end;

int: DataDepEdge_Ret_start;
int: DataDepEdge_Ret_end;
int: DataDepEdge_Alias_start;
int: DataDepEdge_Alias_end;

int: DataDepEdge_Other_start;
int: DataDepEdge_Other_end;
int: DataDepEdge_start;
int: DataDepEdge_end;

int: Parameter_In_start;
int: Parameter_In_end;
```

```
int: Parameter_Out_start;
int: Parameter_Out_end;
int: Parameter_Field_start;
int: Parameter_Field_end;
int: Parameter_start;
int: Parameter_end;

int: PDGEdge_start;
int: PDGEdge_end;

set of int: ControlDep_CallInv = ControlDep_CallInv_start ..
 ↪  ControlDep_CallInv_end;
set of int: ControlDep_CallRet = ControlDep_CallRet_start ..
 ↪  ControlDep_CallRet_end;
set of int: ControlDep_Other = ControlDep_Other_start ..
 ↪  ControlDep_Other_end;
set of int: ControlDep = ControlDep_start .. ControlDep_end;

set of int: DataDepEdge_Ret  = DataDepEdge_Ret_start ..
 ↪  DataDepEdge_Ret_end;
set of int: DataDepEdge_Alias = DataDepEdge_Alias_start ..
 ↪  DataDepEdge_Alias_end;
set of int: DataDepEdge_Other = DataDepEdge_Other_start ..
 ↪  DataDepEdge_Other_end;
set of int: DataDepEdge = DataDepEdge_start .. DataDepEdge_end;

set of int: Parameter_In = Parameter_In_start .. Parameter_In_end;
set of int: Parameter_Out = Parameter_Out_start .. Parameter_Out_end;
set of int: Parameter_Field = Parameter_Field_start ..
 ↪  Parameter_Field_end;
set of int: Parameter = Parameter_start .. Parameter_end;

set of int: PDGEdgeIdx = PDGEdge_start .. PDGEdge_end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Java OO Features
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

int: ClassNames_start;
int: ClassNames_end;
int: ExternalClass;

int: ClassFields_Instance_start;
int: ClassFields_Instance_end;
```

```
int: ClassFields_Static_start;
int: ClassFields_Static_end;
int: ClassMethods_Instance_start;
int: ClassMethods_Instance_end;
int: ClassMethods_Static_start;
int: ClassMethods_Static_end;

set of int: ClassNames = ClassNames_start .. ClassNames_end;
set of int: AllClassNames = ClassNames_start .. ClassNames_end union
 ↪  {ExternalClass};

set of int: ClassFields_Instance = ClassFields_Instance_start ..
 ↪  ClassFields_Instance_end;
set of int: ClassFields_Static = ClassFields_Static_start ..
 ↪  ClassFields_Static_end;
set of int: ClassFields = ClassFields_Instance_start ..
 ↪  ClassFields_Static_end;

set of int: ClassMethods_Instance = ClassMethods_Instance_start ..
 ↪  ClassMethods_Instance_end;
set of int: ClassMethods_Static = ClassMethods_Static_start ..
 ↪  ClassMethods_Static_end;
set of int: ClassMethods = ClassMethods_Instance_start ..
 ↪  ClassMethods_Static_end;

set of int: ClassElements = ClassFields_Instance_start ..
 ↪  ClassMethods_Static_end;

array[ClassFields]    of ClassNames:  fieldOfClass;
array[ClassMethods]   of ClassNames:  methodOfClass;
array[ClassNames]     of AllClassNames:  immediateParent;
array[ClassNames]     of set of AllClassNames:  allParents;
array[ClassNames]     of set of AllClassNames:  implementsInterface;
array[FunctionEntry]  of set of ClassFields:  methodsFieldAccess;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Containing Class for PDG Nodes, Containing Function for PDG Nodes,
 ↪  Endpoints for PDG Edges, Indices of Fucntion Formal Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

array[PDGNodeIdx]     of ClassNames:  hasClass;
array[PDGNodeIdx]     of FunctionEntry:  hasFunction;
array[PDGEdgeIdx]     of int:  hasSource;
array[PDGEdgeIdx]     of int:  hasDest;
```

```
array[Param]           of int:  hasParamIdx;
array[FunctionEntry]  of bool: userAnnotatedFunction;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convenience Aggregations of PDG Nodes and Edges
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set of int: NonAnnotation       = Inst union FunctionEntry union Param;
set of int: ControlDep_Call     = ControlDep_CallInv union
 ↪  ControlDep_CallRet;
set of int: ControlDep_NonCall  = ControlDep_Other;
set of int: DataEdgeNoRet       = DataDepEdge_Other union
 ↪  DataDepEdge_Alias;
set of int: DataEdgeNoRetParam  = DataEdgeNoRet union Parameter_Field;
set of int: DataEdgeParam       = DataDepEdge union Parameter;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Security Levels and Enclaves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

enum Level;
set of Level: nonNullLevel = { x | x in Level where x!=nullLevel };
enum Enclave;
set of Enclave: nonNullEnclave = { x | x in Enclave where x!=nullEnclave
 ↪  };

array[Enclave] of Level: hasEnclaveLevel;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CLE Input Model
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

enum cleLabel;
enum cdf;
enum GuardOperation = {nullGuardOperation, allow, deny, redact};
enum Direction      = {nullDirection, bidirectional, egress, ingress};

int: MaxFuncParms;  % Max number of function parameters in the program
 ↪  (C<128, C++<256)
set of int: parmIdx = 1..MaxFuncParms;

array[cleLabel]                     of Level:          hasLabelLevel;
array[ClassNames]                   of bool:
 ↪  isClassAnnotated;
```

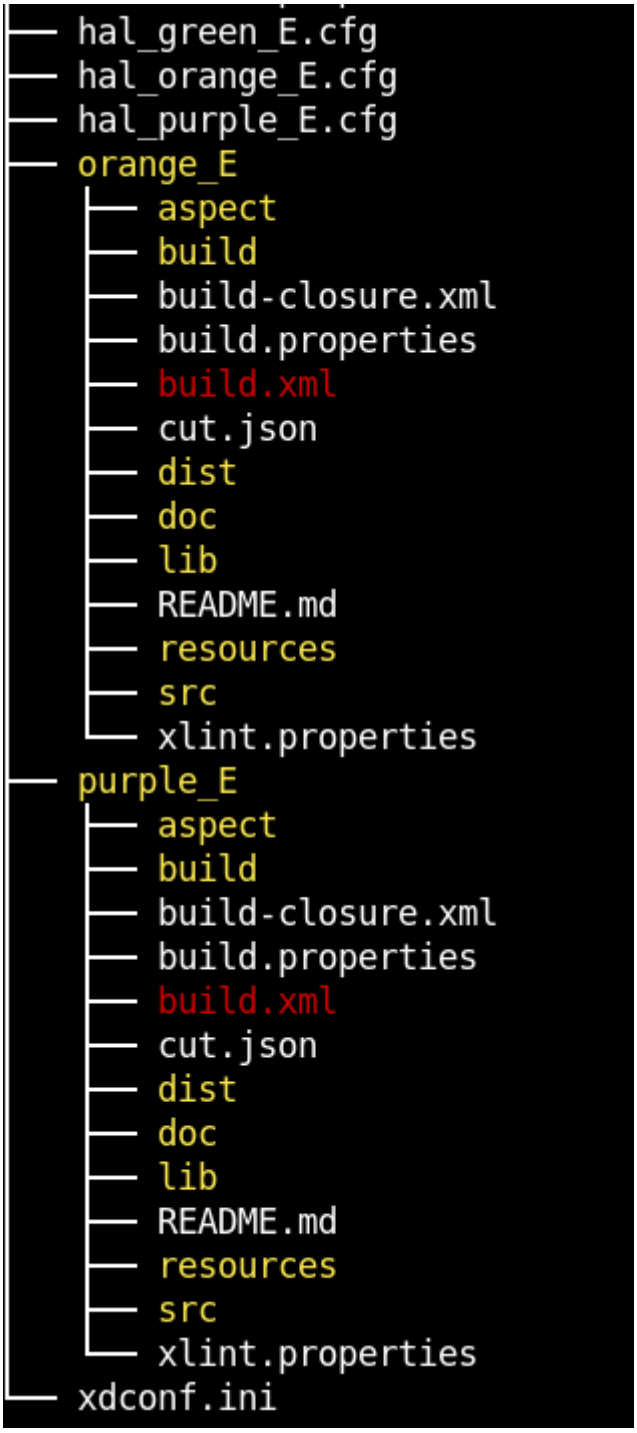
```
array[cleLabel]                          of bool:
 ↪  isFunctionAnnotation;

array[cdf]                               of cleLabel:       fromCleLabel;
array[cdf]                               of Level:          hasRemotelevel;
array[cdf]                               of GuardOperation:
 ↪  hasGuardOperation;
array[cdf]                               of Direction:      hasDirection;
array[cdf]                               of bool:           isOneway;
array[cleLabel, Level]                   of cdf:
 ↪  cdfForRemoteLevel;

set of cdf: functionCdf = { x | x in cdf where
 ↪  isFunctionAnnotation[fromCleLabel[x]]==true };

array[functionCdf, cleLabel]             of bool:           hasRettaints;
array[functionCdf, cleLabel]             of bool:           hasCodtaints;
array[functionCdf, parmIdx, cleLabel]    of bool:           hasArgtaints;
array[functionCdf, cleLabel]             of bool:           hasARCtaints;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debug flag and decision variables for the solver
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bool:                                                       debug;
debug = true;

array[ClassNames,nonNullEnclave]                       of var bool:
 ↪  classEnclave;
array[ClassNames,Level]                                of var bool:
 ↪  classTaintedAtLevel;

array[PDGNodeIdx,nonNullEnclave]                       of var Enclave:
 ↪  nodeEnclave;
array[PDGNodeIdx,nonNullEnclave]                       of var Level:
 ↪  nodeLevel;
array[PDGNodeIdx,nonNullEnclave]                       of var cleLabel:
 ↪  taint;
array[PDGNodeIdx,nonNullEnclave]                       of var cleLabel:
 ↪  ftaint;
array[ClassNames,nonNullEnclave]                       of var cleLabel:
 ↪  ctaint;
```

```
array[ClassFields,nonNullEnclave]                    of var cleLabel:
 ↪ fieldTaint;

array[PDGEdgeIdx,nonNullEnclave]                     of var Enclave:
 ↪ esEnclave;
array[PDGEdgeIdx,nonNullEnclave]                     of var Enclave:
 ↪ edEnclave;

array[PDGEdgeIdx,nonNullEnclave]                     of var cleLabel:
 ↪ esTaint;
array[PDGEdgeIdx,nonNullEnclave]                     of var cleLabel:
 ↪ edTaint;

array[PDGEdgeIdx,nonNullEnclave]                     of var cleLabel:
 ↪ esFunTaint;
array[PDGEdgeIdx,nonNullEnclave]                     of var cleLabel:
 ↪ edFunTaint;

array[PDGEdgeIdx,nonNullEnclave]                     of var cdf:
 ↪ esFunCdf;
array[PDGEdgeIdx,nonNullEnclave]                     of var cdf:
 ↪ edFunCdf;

array[PDGEdgeIdx,nonNullEnclave]                     of var bool:
 ↪ xdedge;
array[PDGEdgeIdx,nonNullEnclave]                     of var bool:
 ↪ tcedge;

array[PDGEdgeIdx,nonNullEnclave]                     of var bool:
 ↪ coerced;
```

5.5 AspectJ Code Generator Outputs

5.5.1 Output Directory Structure

Below is the sample output directory structure created by the Java toolchain for the demo application. The AspectJ definitions and other artifacts, along with the original application, for each enclave are placed in a separate directory. In addition, the HAL configuration files (`xdconf.ini` and `hal_*.cfg`) are put at the top level.



Inside each enclave, AspectJ-related files are placed under the aspect subdirectory. Below is a sample for the purple enclave.

```
purple_E/aspect/
├── ipc.txt
├── lib
│   ├── aspectjrt-1.9.9.1.jar
│   ├── aspectjrt.jar
│   ├── aspectjtools.jar
│   ├── aspectjweaver-1.9.9.1.jar
│   ├── aspectjweaver.jar
│   ├── codeGen.jar
│   ├── gson-2.8.0.jar
│   └── jzmq-3.1.0.jar
├── purple_E.aj
├── tags.txt
├── VideoRequesterHighClosureAspect.aj
└── VideoRequesterNormalClosureAspect.aj
```

5.5.2 Sample AspectJ Class

The following is the AspectJ definition generated for the VideoRequesterHighClosure class, which is located in the orange enclave and accessed from the purple enclave in the partitioned demo application.

```
package com.peratonlabs.closure.aspectj;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;

import org.aspectj.lang.reflect.ConstructorSignature;
import org.aspectj.lang.reflect.MethodSignature;

import com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh;
import com.peratonlabs.closure.remote.ClosureRemoteRMI;
import com.peratonlabs.closure.remote.ClosureShadow;

public aspect VideoRequesterHighClosureAspect {
    // declare error : noInstance() : "Instantiation of the
     ↪  VideoRequesterHigh class is not allowed in this enclave";

    declare precedence: purple_E, *;
```

```
    public void invalid(String message) {
        throw new RuntimeException(message);
    }

    // constructor invocation
    VideoRequesterHigh around(Object myObject) :
↪   call(VideoRequesterHigh.new(..)) &&
↪   !within(VideoRequesterHighClosureAspect) && this(myObject) {
        ConstructorSignature signature = (ConstructorSignature)
↪   thisJoinPoint.getSignature();
        invalid("Not allowed to call the constructor: " + signature);

        return null;
    }

    // constructor execution: this also captures invocation via
     ↪  reflection
    Object around(Object myObject) :
     ↪  execution(VideoRequesterHigh.new(..)) && this(myObject) {
        ConstructorSignature signature = (ConstructorSignature)
↪   thisJoinPoint.getSignature();
        invalid("Not allowed to invoke this Constructor " + signature);
        return null;
    }

    // object finalization
    after(Object myObject) : execution(void
     ↪  VideoRequesterHigh.finalize()) && this(myObject) {
        ConstructorSignature signature = (ConstructorSignature)
↪   thisJoinPoint.getSignature();
        invalid("Not allowed to call finalize() " + signature);
    }

    /******* fields *******/
    // all class or instance field reads
    private pointcut fieldGet() : get(*
     ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.*) &&
     ↪  !within(com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh);
    declare error : fieldGet() : "direct read from VideoRequesterHigh's
↪   fields is not allowed in this enclave. Use a getter";

    // all instance field writes
```

```
    private pointcut fieldSet() : set(*
     ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.*) &&
     ↪  !within(com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh);
    declare error : fieldSet() : "direct write to VideoRequesterHigh's
↪   fields is not allowed in this enclave. Use a setter";

    // all field reads via reflection
    Object around(Field field, Object myObject):
        call(public Object Field.get(Object)) &&
        target(field) &&
        args(myObject) {

        Object result = null;
        if (field.getDeclaringClass() ==
         ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.class)
            result = proceed(field, myObject);
        else {
            invalid("Not allowed to read field via reflection:
             ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh."
             ↪  + field.getName());
        }

        return result;
    }

    // all field writes via reflection
    void around(Field field, VideoRequesterHigh myObject, Object
     ↪  newValue):
         call(public void Field.set(Object, Object)) &&
         target(field) &&
         args(myObject, newValue) {

        //Object result = null;
        if (field.getDeclaringClass() ==
         ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.class)
            proceed(field, myObject, newValue);
        else {
            invalid("Not allowed to write field via reflection:
             ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh."
             ↪  + field.getName());
        }
    }

    // static field reads via reflection
```

```
    Object around(Field field):
        call(public Object Field.get(Object)) &&
        target(field) {

        Object result = null;
        if (field.getDeclaringClass() ==
         ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.class)
            result = proceed(field);
        else {
            invalid("Not allowed to read static field via reflection:
             ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh."
             ↪  + field.getName());
        }

        return result;
    }

    // all static field writes via reflection
    void around(Field field, Object newValue):
         call(public void Field.set(Object, Object)) &&
         target(field) &&
         args(newValue) {

        if (field.getDeclaringClass() ==
         ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.class)
            proceed(field, newValue);
        else {
            invalid("Not allowed to write static field via reflection:
             ↪  com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh."
             ↪  + field.getName());
        }
    }

    /******* Method Invocations **********/

    // method invocation/execution
    private pointcut methodExec() : execution(*
     ↪  VideoRequesterHigh.*(..));

    // method invocation/execution: this also captures invocations via
     ↪  reflection
    Object around(Object myObject): methodExec() && target(myObject) {
        MethodSignature signature = (MethodSignature)
↪   thisJoinPoint.getSignature();
```

```
        invalid("Not allowed to invoke this method " + signature);

        return null;
    }

    // static method invocation/execution
    private pointcut staticMethod(): execution(static *
     ↪  VideoRequesterHigh.*());

    Object around(): staticMethod() {
        MethodSignature signature = (MethodSignature)
 ↪  thisJoinPoint.getSignature();

        invalid("Not allowed to invoke static method " + signature);

        return null;
    }
}
```

5.5.3 ZeroMQ URL (ipc.txt)

The ipc.txt file is loaded at application startup time to connect to the ZeroMQ for publication and subscriptions. The following is a sample for the purple enclave.

```
ipc:///tmp/tchalsubpurple_e
ipc:///tmp/tchalpubpurple_e
```

5.5.4 XDCC Tags (tags.txt)

The tags.txt file is loaded at application startup time to initialize mux/sec/type of cross-domain calls.

```
com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.start.int.java.lang.String 4 4
com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.start.int.java.lang.String_rsp
com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal.start.int.java.lang.String
com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal.start.int.java.lang.String
com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.getRequest 4 4 5
com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.getRequest_rsp 6 6 6
com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal.getRequest 3 3 7
com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal.getRequest_rsp 1 1 8
com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.send.java.lang.String.byte[] 4
```

```
com.peratonlabs.closure.eop2.level.high.VideoRequesterHigh.send.java.lang.String.byte[]_r
com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal.send.java.lang.String.byte
com.peratonlabs.closure.eop2.level.normal.VideoRequesterNormal.send.java.lang.String.byte
```

5.5.5 Ant Build for AspectJ Weaving

An ant build file, build-closure.xml, is generated to handle the AspectJ weaving task.

```
<project name="Build app and aspect lib then weave" default="weave">
  <property file="./build.properties"/>

  <taskdef classpath="${aspectj.home}/lib/aspectjtools.jar"

 ↪  resource="org/aspectj/tools/ant/taskdefs/aspectjTaskdefs.properties"/>

  <path id="project.class.path">
      <pathelement location="${aspectj.home}/lib/aspectjrt.jar"/>
      <pathelement location="${aspectj.home}/lib/codeGen.jar"/>
      <pathelement location="dist/TESTPROGRAM.jar"/>
      <fileset dir="./lib">
        <include name="**/*.jar" />
      </fileset>
  </path>

  <!-- build the CLOSURE aspectj library -->
  <target name="compile">
    <mkdir dir="dist" />
    <iajc
      source="1.5"
      classpathref="project.class.path"
      outjar="dist/closure-aspect.jar"
      xlintfile="xlint.properties">
      <sourceroots>
        <pathelement location="aspect" />
      </sourceroots>
    </iajc>
  </target>

  <target name="initialize" depends="compile">
    <mkdir dir="dist" />
    <copy todir="dist">
      <fileset dir="./dist">
        <include name="TESTPROGRAM.jar" />
```

```
      </fileset>
      <fileset dir="./dist">
        <include name="closure-aspect.jar" />
      </fileset>
    </copy>
  </target>

    <!-- weave the app and the CLOSURE aspectj library -->
  <target name="weave" depends="initialize">
    <mkdir dir="dist" />
    <iajc injars="dist/TESTPROGRAM.jar"
          aspectpath="dist/closure-aspect.jar"
          outjar="dist/weaved-TESTPROGRAM.jar"
          classpathref="project.class.path">
    </iajc>
    <delete file="dist/TESTPROGRAM.jar"/>
  </target>
</project>
```

5.5.6 Slave Handler

The slave handler is used to listen for cross-domain calls. It replaces the entry point of the original app via an AspectJ pointcut.

```
package com.peratonlabs.closure.aspectj;

import com.peratonlabs.closure.remote.ClosureRemoteHalSlave;

public aspect VideoManagerMainAspect {
    // static method invocation/execution
    private pointcut staticMain(): execution(public static void
     ↪  com.peratonlabs.closure.eop2.video.manager.VideoManager.main(String[]));

    Object around(): staticMain() {
        ClosureRemoteHalSlave.init();
        return null;
    }
}
```

5.5.7 Sample Config for CodeGenJava

Below is a sample configuration file for the CodeGenJava tool.

```
{
  "dstDir": "/home/closure/xdcc",
  "cut": "test/cut.json",

  "srcDir": "/home/closure/gaps/capo/Java/examples/eop2-demo",
  "codeDir": ".",
  "jar": "TESTPROGRAM",
  "compile": true,

  "halCfg":
   ↪  "/home/closure/gaps/hal/java-eop2-demo-hal/hal_autoconfig-multienclave.py",
  "deviceFile":
   ↪  "/home/closure/gaps/hal/java-eop2-demo-hal/devices_eop2_java_alllocal.json"
}
```

5.6 HAL Configuration Files

See the C documentation [1].

5.7 Dockerfile

5.7.1 Dockerfile for Source release

The following dockerfile is used to build a source release from scratch.

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND noninteractive
ENV HOME /home/closure
ENV TZ="America/New_York"
ENV JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
ENV PATH=PATH=$JAVA_HOME/bin:$HOME/MiniZincIDE-2.5.5-bundle-linux-x86_64/bin:$PATH
ENV CAPO=$HOME/gaps/capo/Java
ENV CLASSPATH=$CAPO/joana/dist/*:$CAPO/examples/eop2-demo/dist/*:$CAPO/jython-standalone-

RUN apt-get update && \
    apt-get install -y net-tools iproute2 netcat dnsutils curl iputils-ping iptables nmap

RUN apt-get -y install gcc mono-mcs
RUN apt-get -y install make

RUN apt-get -y install openjdk-8-jdk-headless
```

```
RUN apt-get -y install libzmq3-dev
RUN apt-get -y install libconfig-dev
RUN apt-get -y install ant
RUN apt-get -y install maven
RUN apt-get -y  install build-essential

RUN apt-get -y install python3
RUN apt install -y python3-pip
RUN pip3 install libconf

RUN apt-get -y install tmux
RUN apt-get -y install libzmq-java
RUN apt-get -y install openssh-client

RUN apt-get -y install git

######### Create and run as closure
RUN useradd -u 8877 -s /bin/bash closure
RUN mkdir -p /home/closure
RUN chown closure.closure /home/closure

USER closure

RUN mkdir -p $HOME/opencv-4.6.0/build/
RUN mkdir -p $HOME/gaps

WORKDIR $HOME/gaps
RUN git clone -b tcp https://github.com/gaps-closure/hal.git
RUN git clone https://github.com/gaps-closure/CodeGenJava.git
RUN git clone -b develop https://github.com/gaps-closure/capo.git
RUN git clone https://github.com/joana-team/joana.git $HOME/gaps/capo/Java/joana

WORKDIR $CAPO/joana
RUN git submodule init
RUN git submodule update

RUN mv setup_deps setup_deps.orig && \
    cp ../setup_deps .

WORKDIR $CAPO
RUN wget https://repo1.maven.org/maven2/org/python/jython-standalone/2.7.2/jython-standal
RUN wget https://sourceforge.net/projects/jscheme/files/jscheme/7.2/jscheme-7.2.jar
```

```
WORKDIR $HOME
RUN wget https://github.com/MiniZinc/MiniZincIDE/releases/download/2.5.5/MiniZincIDE-2.5
RUN tar xzf MiniZincIDE-2.5.5-bundle-linux-x86_64.tgz
RUN rm -f MiniZincIDE-2.5.5-bundle-linux-x86_64.tgz

RUN echo "bind X kill-session" > .tmux.conf
RUN echo "set-option -g status-keys emacs" >> .tmux.conf

WORKDIR $HOME/opencv-4.6.0/build
RUN wget https://github.com/gaps-closure/capo/releases/download/T0.2/opencv-4.6.0.tgz
RUN tar xzf opencv-4.6.0.tgz
RUN rm -f opencv-4.6.0.tgz

WORKDIR $HOME/gaps
RUN cp capo/Java/examples/eop2-demo/resources/scripts/* .
RUN cp capo/Java/release/README-src.md README.md

EXPOSE 8080 8081
```

References

[1] "CLOSURE toolchain user manual for CLanguage." GitHub, 2022. Available: https://github.com/gaps-closure/gaps-closure.github.io/tree/develop/docs/C

[2] "Visual studio code," *Visual Studio Code - Code editing. Redefined.* Microsoft, 2022. Available: https://code.visualstudio.com/

[3] J. Graf, M. Hecker, and M. Mohr, "Using JOANA for information flow control in java programs - a practical guide," in *Proceedings of the 6th working conference on programming languages (ATPS'13)*, Feb. 2013, pp. 123–138.

[4] M. J. Beckerle and S. M. Hanson, "Data format description language (DFDL) v1.0 specification," *Data Format Description Language (DFDL) v1.0 Specification*. The Apache Software Foundation, 2022. Available: https://daffodil.apache.org/docs/dfdl/

[5] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[6] P. J. Stuckey, K. Marriott, and G. Tack, "The minizinc handbook," *The MiniZinc Handbook - The MiniZinc Handbook 2.5.5*. 2022. Available: https://www.minizinc.org/doc-2.5.5/en/index.html

[7] G. Kiczales *et al.*, "Aspect-oriented programming," in *European Conference on Object-Oriented Programming*, Jun. 1997, vol. 1241, pp. 220–242. doi: 10.1007/BFb0053381.

[8] T. Černý, "On concern-separation of data presentations in user interfaces," PhD thesis, 2016.

[9] "The AspectJ development environment guide." Xerox, 2005. Available: https://www.eclipse.org/aspectj/doc/next/devguide/printable.html

[10] "The AspectJ programming guide." Xerox, 2003. Available: https://www.eclipse.org/aspectj/doc/next/progguide/index.html