Indus
Venkatesh Prasad Ranganath, Kansas State University
<rvprasad@cis.ksu.edu>

Table of Contents

Indus - The project ...................................................................................................... 1
Background ....................................................................................................... 1
What about the name? .......................................................................................... 1
Motivation ......................................................................................................... 1
Indus - The Core ......................................................................................................... 2
Packages ........................................................................................................... 2
Peccadillos ................................................................................................................ 4
Closing Note .............................................................................................................. 4

Indus - The project

Indus is a project aimed at developing and implementing program analyses and transformations that can be used to customize and adapt Java programs. It contains a bunch of implementations of various program analyses and transformations as subprojects which were developed to cater to client projects in SAnToS group. However, Indus and it’s subprojects are structured to facilitate their use by other projects as well.

Background

Indus was not conceived as stated above but rather by snowballing. As each developer of the project executed a subproject for a publication or as part of graduate work, each subproject was maintained and managed by the developers separately. These subprojects were related from the perspective of their application and purpose but could not be included into the client projects (Bandera) as they were generic and not tailored to Bandera. Hence, these subprojects (and the subprojects conceived for future) with a similar flavour and common purposes were bundled as Indus.

What about the name?

As for the name of the project, Indus is a river in the Indian subcontinent and it is the cradle of Indus valley civilization, one of oldest civilization in the world. Just to give the project a flavour of the country from where I come, I picked Indus as the name. Indus is not an acronym, it’s just a name.

Motivation

As mentioned above, Indus provides a collection of library classes that can be used in applications. Each of the libraries are provided by a subproject of Indus. For example, a bunch of static analyses are implemented in a subproject called "Staticanalyses" while the Java program slicer is implemented in a subproject called "JavaSlicer". Hence, each subproject deals with one cohesive theme and just that. If at any time the required functionality seems to be generic and not tightly coupled to the silent subproject, then that functionality will be provided by a new subproject.
Each subproject may require some piece of information from an analysis and there may be many flavours of the analysis. To enable the user to easily experiment with various implementations of the analysis, we split the analysis into its interface and implementation. Hence, the user can adapt any implementation of an analysis to conform to the given interface, plug in the adaptation into the system of modules, and it should be ready to use! As one may have observed the interfaces are not dependent on the implementation and they do not belong to the subproject which provides the implementation. Hence, the interfaces need to be housed in a different subproject.

It has been our past experience that related projects usually use common code snippets. To ease maintainability, it is best to avoid duplication and group these snippets into a subproject so that any new subproject requiring similar functionality can use off-the-shelf code. Recently, we also realized this sort of reusability could be exercised to cut down test setup time while testing with large-size systems.

To address the above issues we created a subproject of Indus by the same name, "Indus", which is discussed in detail in the next section.

**Indus - The Core**

From here on, we shall use Indus to refer to the subproject unless explicitly mentioned. The purpose of Indus is given below.

- To define a set of interfaces that are general enough to extract the information pertaining to analyses independent of their implementations.
- To provide a set of interfaces and abstract implementations of tiny frameworks that are used in more than one subproject of Indus (the project).
- To provide a set of interfaces and abstract implementations to expose analyses and transformations as tools via well defined interfaces to enable the analyses and transformations to be easily used in other toolkits and IDEs.

All implementations in Indus dealing with AST is based on Jimple, a IR for Java in the Soot toolkit available from Sable group at McGill University.

**Packages**

The following is a list of packages in Indus subproject along with their description. All the packages are rooted in a package called `edu.ksu.cis.indus`.

- **common**
  
  This package contains classes which provide functionality similar to helper functions. They are either classes of a particular library which are tailored for use in any subproject in Indus or classes which contain methods that are contain common operations of a sort used repetitively in the subprojects. This package will be populated as the subprojects grow.

- **common.datastructures**
  
  This package contains data structures commonly used in the subprojects such as `Pair`, `Triple`, `WorkBag`, etc. It may be the case that the client projects may extend these. However, the basic functionality is provide the classes in this package.

- **common.graph**
  
  This package contains classes which represent directed graphs with the most common graph operations such as SCC discovery, topological sort, etc. Oddly it also includes the implementation of
basic block graph and a manager of basic blocks as basic blocks graphs are just graphs and they are common to program analysis and transformations. For those curious about why another implementation of graph instead of using the graph support in Soot, the answer is that we found the graph support in Soot lacking in areas.

common.soot This package contains classes that provide functionality that are specific to Soot such as discovering the class that introduces the method into the class hierarchy branch and such.

interfaces This package contains the interfaces via which various implementation can provide the same information, say for example the call graph info in Soot can be used in the subprojects of Indus if they can provide an adapter of the form of ICallGraphInfo. These interface will also provide auxiliary classes or interface which are used in the parent interface as well. In case of ICallGraphInfo, it contains a class which captures the call information.

processing This package contains a small framework to visit AST nodes in a batched fashion. It is typical in transformations to perform a bunch of analyses and these analyses will walk the AST. Since, many of these analyses are independent of each other, their walk can be batched, hence, the framework. It is also the case that during such walk one may want to avoid visiting certain parts and this is possible via using the filters in this class or extending the filters.

tools This package contains a set of interfaces and abstract implementations via which various implementations can be exposed as cohesive tools to be absorbed in other toolkits and IDEs. Please see the JavaSlicer subproject for an instance of how to use the classes in this package.

transformations.common This package contains classes which are common to transform programs. Cloning of ASTs is one such operation that is provided by classes in this package.

edu.ksu.cis.indus.xmlizer This package contains classes that functionality required to xmlize jimple in a canonical way such that the generate data is comparable. This is used in our regression testing framework extensively and also in most of our driver/example classes.

The users/developers are encouraged to cast their solutions in a form which can provide the above interfaces defined in the above package to improve reusability across Indus projects.

There are no concrete classes in this subproject, hence, there are no example based exposition to the classes in this subproject. Please refer to the java docs or the doxygen docs for more information about how to used a particular class or interface.

As a matter of configurability, we use a interface-id-to-implementation mapping. Each interface can provide (or should provide) an id which the implementation cannot override. Hence, a client requiring an implementation can just lookup for a particular id in a map during configuration phase and use the provided implementation. We could have used the interface type-based resolution, but it is more simpler this way.
Peccadillos

As Indus is based on Soot, the analyses and transformations work on the runtime artifacts of Soot constructed using a particular configuration of Soot. The options to reach enable this particular Soot configuration is provided via `Util.getSootOptions()` method. It is strongly recommended to initialize Soot with the options obtained by this method before loading classes when using Indus or it's subprojects.

Closing Note

The XMLizing classes used by this project and it's parent and sibling projects use the xmlizing framework to drive the slicer. So, we urge you to peruse the source code of these classes before asking questions on the forum or the mailing list. We will be glad to answers any question you may have regarding the usage, but it probably would be faster if the user mocked an existing working piece of code while starting to use a new tool.

The reader is encouraged to use the modules as is or to extend them as required. In the due process, the users are urged to submit bug reports of any bugs uncovered with suitable information about the triggering input and configuration.

The interface of the modules are not fixed as the development team has not foreseen all possible applications and tweaks to the slicer. Hence, the users are encouraged to raise change requests to the development team along with any feature requests they may have. However, please note that the development team may not be able to implement all requested features in which case they will assist by providing any information or alterations to enable the requested features.

Please refer to Indus [http://indus.projects.cis.ksu.edu] for more documentation, distribution, mailing list, forums, and links to other subprojects.

We hope you have a pleasant experience using our product.