Intraprocedural Dataflow Analysis for Software Product Lines: A Prototype

Társis Tolêdo[2] twt@cin.ufpe.br
Márcio Ribeiro[2,3] mmr3@cin.ufpe.br
Claus Brabrand[1,2] brabrand@itu.dk
Paulo Borba[2] phmb@cin.ufpe.br

January 23, 2012

[1] IT University of Copenhagen, Denmark
[2] Federal University of Pernambuco, Brazil
[3] Federal University of Alagoas, Brazil

Preface

This document is a companion for the prototype implementation of the Intraprocedural Dataflow Analysis for Software Product Line concept. We give an overview of the architecture and key concepts. Basic knowledge of the SOOT framework is required for understanding the implementation details described. More details at the AOSD'12 paper: Intraprocedural Dataflow Analysis for Software Product Lines [1].

1 Overview

Intraprocedural Feature-Sensitive Dataflow Analyses allows one to analyze a method annotated with preprocessor directives without the need to explicitly generate all variants of that method [1].

In the specific implementation described here, CIDE [2], an Eclipse IDE [7] plug-in, is used as the preprocessor technology and SOOT, a Java transformation and analysis framework, is extended to perform feature-sensitive analyses.

Figure 1 shows an architectural view of this prototype. In the *Feature-sensitive dataflow analyses* component[1], analyses are special because they take the feature annotations – preprocessor directives like `#ifdef`, or colors in case of CIDE – into consideration when computing the least fixed-point. The

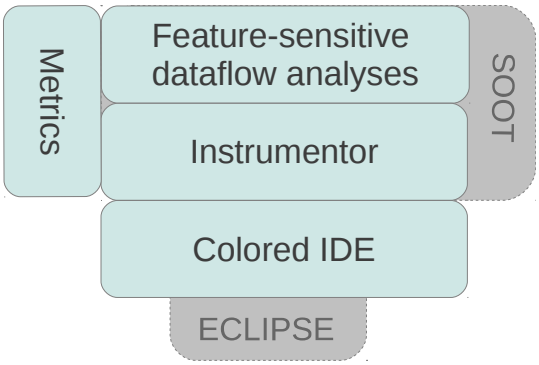

Figure 1: Architecture

intermediate *Instrumentor* component translates the feature annotations into an representation that the analyses can use. We call this the *instrumentation process*.

SOOT provides mechanisms for attaching and reading information to/from the nodes of the control-flow graph and the infrastructure for extending the analysis' framework. More details about this in Section 2.

In the case of this implementation, the CIDE plug-in provides the feature annotation for the Eclipse IDE. This is the reason why this prototype is also an Eclipse plug-in.

The measurements made to prospect performance and space data are scattered among these two components. The *Metrics* component is responsible for receiving and storing information as the other collects them. It then dumps the information into a sheet for a more flexible statistical analysis.

[1] No strict meaning for the term *component* is intend

2 Implementation details

2.1 The instrumentation process

We implemented the *Instrumentor* component as a SOOT Transformer by the class `FeatureModelInstrumentorTransformer`. It's job is to attach to every `Unit` a `Tag` (class `FeatureTag`) containing information about which features the `Units` belongs to. With CIDE, this means the colors that surround the code that generated the `Unit`.

In addition, the *Instrumentor* component also attaches a `Tag` (class `ConfigTag`) the `Body` of every method. This tags contains the total set of configurations that the method can assume.

These `Tags` allows the *lifting* of the analysis as described below in Section 2.2.

2.2 Analyses

We describe three different ways to achieve feature-sensitivity in the dataflow analysis [1]. All of them require that this instrumentation process takes places before the analysis can execute. We achieve this by assigning an instance of the `FeatureModelInstrumentorTransformer` class to one of SOOT's transformation phases. Refer to the paper for theoretical details.

2.2.1 The Simultaneous and Lazy forms

Both the simultaneous (A3) and the lazy (A4) form of analysis require an extended implementation of lattice elements, known as `FlowSets` in SOOT. You can find this extended implementation at class `MapLiftedFlowSet`. `MapLiftedFlowSets` maps configurations to `FlowSets` in case of A3, or set of configurations to `FlowSets` in case of A4. We use the `ConfigTag` class to make such mappings.

The analysis achieve feature-sensitivity in these forms by implementing the transfer functions to act on such extended lattice elements, which implies checking the *applicability* of transfer functions.

This is where the `FeatureTags` attached to the Units are required. The analyses' transfer functions iterate over all possible configurations and decide whether or not to apply the transfer function by checking a configuration against the feature information from the `FeatureTag`.

This is straightforward, as you can see on the implementation of the reaching definitions (`LiftedReachingDefinitions` class for A3 and `LazyLiftedReachingDefinitions` for A4) and definite assignments analyses (`LiftedUninitializedVariables` class for A3 and `LazyLiftedUninitializedVariables` for A4). The results of the analysis can be iterated as it is usually done with SOOT. The only twist is that the `MapLiftedFlowSet` contains analysis information about all possible configurations for one given point.

2.2.2 The Consecutive form

The consecutive (A2) form does not need an extended lattice, but also requires the applicability test and has the most simple implementation of the three forms. See classes `UnliftedUninitializedVaribles` and `UnliftedReachingDefinitions` for the A2 forms of the uninitialized variables and reaching definitions analysis respectively.

Keep in mind, however, that A2 analyses can only compute information about a single configuration at a time.

3 Running the experiment

To reproduce the experiment described in the paper, you need a number of requirements. Note that the experiment itself is an Eclipse plug-in.

Follow the instructions on CIDE [2], Antenna Preprocessor [3] and AJDT [6] websites and install their plug-ins.

Check out the source [4] for our eclipse plug-in. Then import it to your eclipse by clicking on File → Import then select Existing projects into workspace. Set the path to the directory of your check out and click Finish.

Now enable the preprocessor for `cide_ei_rmk` by right-clicking it and checking Antenna Preprocessor. Enable the METRICS feature right-clicking the `cide_ei_rmk` project and clicking on Properties and then Antenna Preprocessor. On the Defines field, simply write METRICS and click OK.

If everything built correctly, then you are ready to go. Right-click on the `cide_ei_rmk` project and click Run As → Eclipse Application. Make sure

you have allocated a generous amount of memory to your JVM, for the analysis process is very memory-intensive.

After the new Eclipse spawns, import the benchmarks you wish to run the analysis on. Four of them are available at the companion website [5]. Right-click on one of them and choose Run FSA (FSA means Feature-Sensitive Analysis). The analyses will execute and the components collect several metrics and dump into an Excel file (fs.xls) in your home directory.

4 Final notes

The code is heavily sprinkled with time measurements in order to collect the performance information seen in the paper, along with other features implemented with `#ifdef` and the Antenna Preprocessor [3]. The main functionality of this software does not require any of those features to be active, however, the only exception being the A4 form.

We implemented the A4 form as an feature alternative to the A3 form, and you can activate it by enabling the LAZY feature in the Antenna Preprocessor.

As this is an prototype implementation, we make no guarantees with respect to the API preservation in future versions. In fact, there are many changes scheduled for the source code, and some of them will inevitably break the API.

Feel free to contact any of the paper authors if you have questions, contributions or want to learn more.

References

[1] Claus Brabrand, Márcio Ribeiro, Társis Tolêdo and Paulo Borba, *Intraprocedural Dataflow Analysis for Software Product Lines*, In proceedings of AOSD'12, Potsdam, Germany.

[2] *Colored IDE (CIDE) plug-in*, http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/.

[3] *Antenna Preprocessor plug-in*, http://antenna.sourceforge.net/wtkpreprocess.php

[4] *Prototype repository*, https://www.assembla.com/code/cide_ei_rmk/subversion/nodes

[5] *Paper companion website*, http://twiki.cin.ufpe.br/twiki/bin/view/SPG/EmergentAndDFA

[6] *AJDT plug-in*, http://www.eclipse.org/ajdt/

[7] *Eclipse IDE*, http://www.eclipse.org/