User Manual for Data Dependency Capturer v0.1

Before getting started…

Data Dependency Capturer v0.1 is part of the whole program we built to do the experiments for our paper "**Do Data Dependencies in Source Code complement Call Dependencies for Understanding Requirements Traceability?**" accepted by the conference ICSM 2012. We didn't intend to build a stand-alone tool to capture the method data dependencies described in our paper. So the program we put here is just a prototype instead of a perfect tool. We will update this prototype and its user manual in the future.

This prototype is built upon JVMTI, which needs to include some JDK Directories in the project. Since different developers are using all kinds of JDKs (e.g. Open JDK) and different IDEs for C/C++, we think that pure source code without any IDE project structure is better than a precompiled dll file. Users can choose the best JDK and IDE for themselves and build a dll project with the source code we offered.

Because JVMTI has a unique callback mechanism for developers, users of this tool who has no experiences about JVMTI need some prior knowledge about it before jump into our code. We also use SQLite to record the messages dumped from JVM.

Limitations of this prototype

This prototype only captures the Method-Using-Data Records shown in Fig.1 during runtime of the target system because how to generate method data dependencies based on those records and use them can be ad hoc. Users can easily generate their own version of method data dependencies based on the algorithm we talked about in our paper (section IV.B).

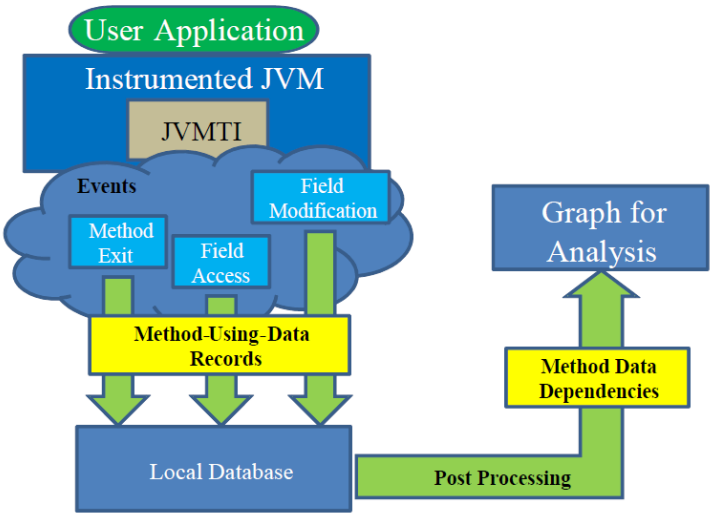

Fig.1 The approach of capturing method data dependencies

JVMTI has a special event for exception handling in Java, in the current version of our prototype, we ignored exception handling if it happened during runtime.

How to build a dll project for this prototype

First, build a dll project in your own IDE, put all the files in the “source code” and “sqlite library” folders into this project. Substitute the main program file (not the dllmain.cpp) in your dll project with the file “data_dependecy_capturer.cpp”. You may need to rename this file with the same name of your dll project in order to be consistent.

Then you need to include JDK directories, like Fig.2:

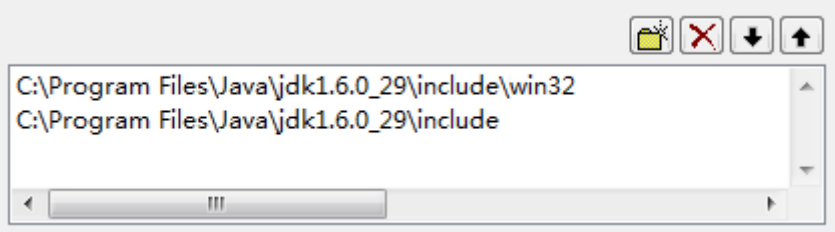

Fig2. Include JDK directories

Configure your own directories for JDK. You can see in Fig.2 that we are using the standard **jdk.1.6.0_29**. Some bug reports show that not all JDKs can support the JVMTI specification (Edition 6), please make sure that the JDK you are using can support this.

You also need to add sqlite3.lib as the Additional Dependencies for the Linker. After that you will be able to compile the dll file!

How to configure this prototype (in the source code)

If you want to monitor different Java programs, you have to change the package name in line 507 and line 590 in the data_dependecy_capturer.cpp file. For example, if you want to monitor GanttProject, the package name is “net.sourceforge.ganttproject”, you will need to add “Lnet/sourceforge/ganttproject/” to the IF clause in line 507 and 590.

If you want to change paths for database files, you can look at line 835, 850 and 865. The database name “FA” means “Field Access”. “FM” means “Field Modification”. “PP” means “Parameter Passing”. These databases keep the Method-Using-Data records which are dumped by the prototype.

How to start this prototype along with the JVM

If you successfully compiled your project, you will get a dll file. Start your target program with this JVM option: ”-agentpath:%the path of your dll file%”, an example is shown in Fig.3 (In Eclipse):

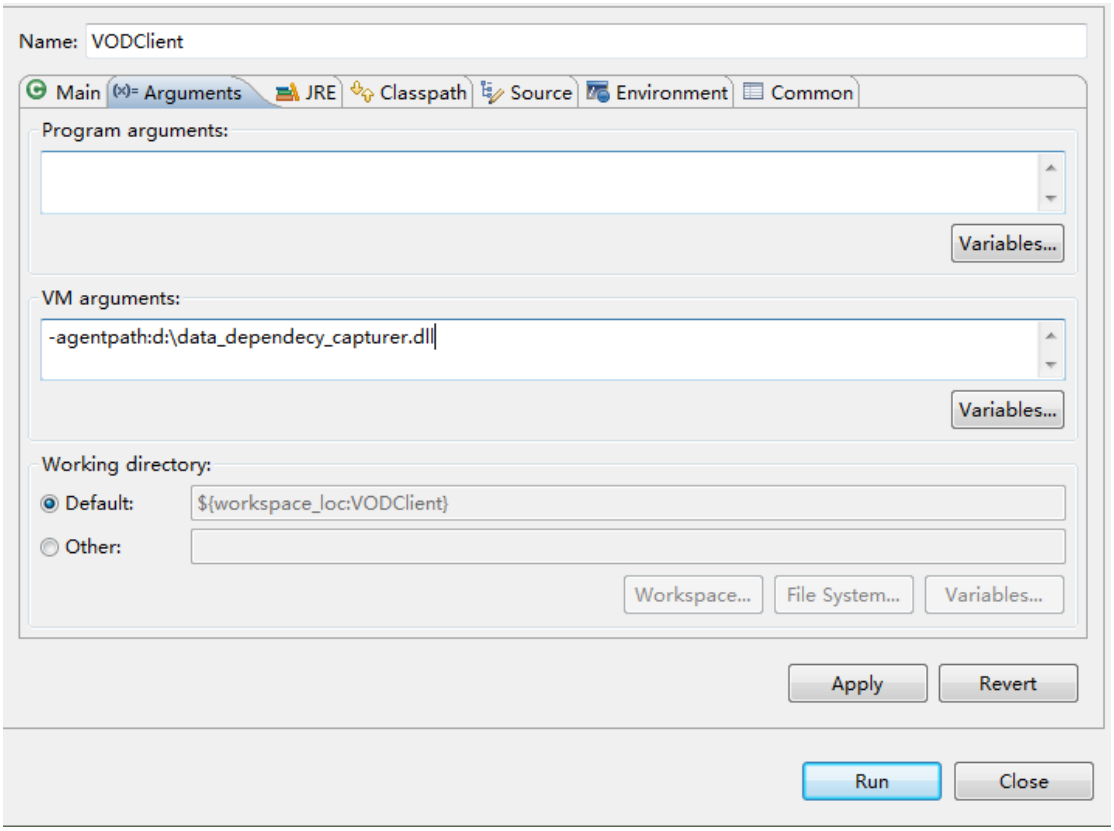

Fig.3 Start the JVM with the prototype