1. Project title: Visualization and Query-based Analysis of JPF Traces

- Aditya Kapre, Graduate Student, CSE Department, University at Buffalo

2. Project description:

The goal of this project is to facilitate a more visual and high-level understanding of JPF's textual log output. We will provide a visualization of individual execution states, the history of execution, as well as the execution tree of choices made by JPF. A more detailed account of the project description can be found at below JIVE website: http://www.cse.buffalo.edu/jive/JPF.pdf

Problem Definition:

For a concrete example, consider the Dining Philosopher's problem. When we run JPF on the Dining Philosopher's code, we get a textual trace which finally shows the deadlocked state. This textual trace can be presented in a more intuitive manner. It is desirable to see not only the individual states but also the history of execution that led to the deadlock. At the user's direction, the tree of execution choices made by JPF's scheduler should also be viewable in an intuitive manner.

Proposed Solution:

A promising approach would be to interface JPF with JIVE, a state-of-the-art visualization system that depicts the run-time states and execution trace of a Java program in a visual manner. We propose to provide three main visualizations of the JPF execution trace: an object diagram which shows a particular execution state; a sequence diagram which summarizes the history of interaction along one execution path; and a tree diagram that shows the scheduler choices in execution paths. Our proposed object and sequence diagrams are based upon similar notations provided by UML, but JIVE makes some extensions to the UML notation and also allows the user to filter and abstract the diagrams for better readability. Since JIVE also supports queries on the execution history, we will provide this capability so that the user can analyze when certain conditions occurred. These queries pertain to the values of variables, methods, statements, objects, exceptions, and threads.

3. What is the development methodology you propose to use for the project? Please include information on possible deliverables and the tentative schedule.

Development methodology

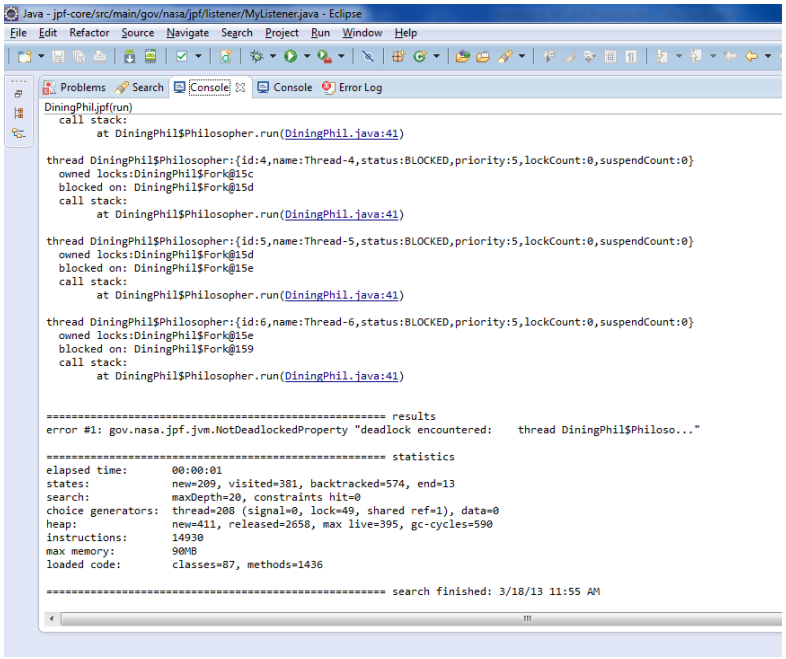
We are proposing 'Rapid application development' methodology for this project. We would break this project into small tasks precisely into four tasks and iteratively complete the development process. To summarize it all, following will be the tasks.

a. JPF-JIVE Eclipse Plug-in Creation: We will create an eclipse plugin in JIVE such that it can handle XML files of type "jpf://"

b. JPF Custom Listener Creation: We will write a custom listener that will implement SearchListener and VMListener. This listener will write the desired events to be visualized to a jpf log file.

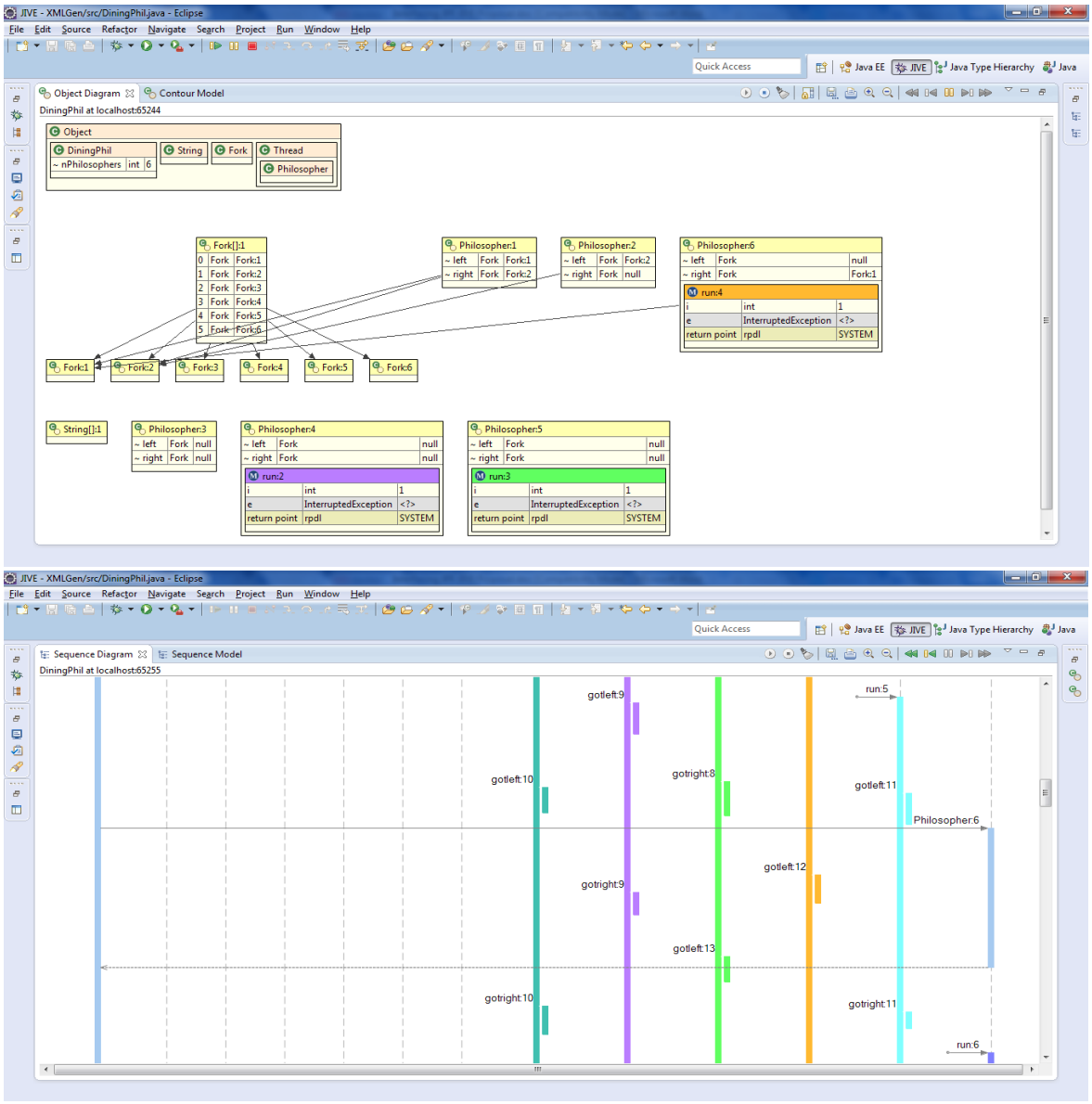
c. XML Generator Creation: This program will convert the JPF custom trace to XML events in the form read and visualized by JIVE.
d. JPF Importer Creation: We will write a class to interpret various events of interest that JPF emits. This call will also handle the visualization i.e Object and Sequence diagrams

We have carried out some initial experiments using JPF to understand its behavior. We can use the extension point listeners SearchListener and VMListener to listen the events of interest: we can create a custom Listener that will implement SearchListener as well as VMListener and log the events. Following is the output logged on console by running the JPF verification on the DiningPhilosopher.jpf file. Our aim is to provide a visualization of all the paths tried by JPF, and the path leading to deadlock.



JIVE's visualization is driven by a sequence of events that it gets from the JVM. These events pertain to such operations as field write, method call, object creation, thread start, exception thrown, etc. JIVE's visualizations can also be driven by input from an XML file of suitable format. Thus, by writing the events logged from our custom JPF Listener into the required XML format, we can interface JPF with JIVE.

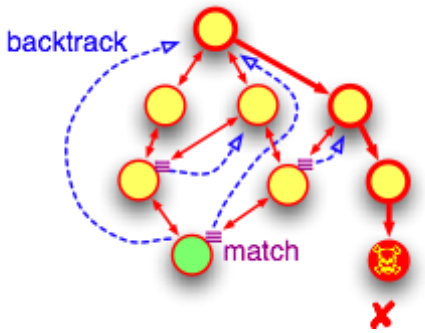
Following are screenshots of the kind of object and sequence diagram that would be produced by JIVE for a single path through the DiningPhilosopher.java code. These screenshots were generated by directly running the Java code directly through JIVE. The object diagram captures one execution state along one execution path. The aim of this project is to interface the JPF output with JIVE so that we can visualize all execution states along all execution paths generated by JPF.



3. Proposed Work:

1. We would like to create a custom listener that will, in turn, implement the SearchListener and VMListener interfaces. The custom listener will extract the data of interest, which we will be stored in flat file. We will create a plug-in to transform the flat file into an XML file of events that is supported by JIVE's XSD schema. We will create an extension to JIVE such that it is able to handle the files generated by JPF in JIVE format. We will implement a JPF importer that will read input XML, interpret it, and display events by mapping them to various diagrams using the standard JIVE code for diagram generation.
2. We will also develop a new visualization that displays how the paths are searched, as depicted by the following picture (taken from the JPF website). We can use the JPF classes

such as ThreadInfo, MethodInfo, ElementInfo and corresponding protected members such as lastThreadInfo, lastThreadInfo, lastElementInfo of JVM class to get the data relevant to the different methods of the custom listener.



Finally, we will deploy this plug-in in Eclipse.

The table shows the tentative schedule for deliverables:

Date	Deliverable
June 14	JPF-JIVE Eclipse Plug-in
July 5	JPF Custom Listener
July 19	XML Generator
August 9	JPF Importer
August 23	Unit Testing and Unit Test Issues resolution
September 6	System Testing and System Test Issues resolution
September 13	Documentation
September 13 to September 27	Buffer (approximately 15%)

4. What are the goals that you hope to accomplish in your project?

My goal is to facilitate the analysis of software, especially in the discovery of concurrency-related errors. JPF is a powerful tool and it is a very promising approach to this problem. However, its textual output is not easy to follow. My project will greatly assist in understanding complex programs and speed up the analysis and understanding of the cause of errors, compared with reading a textual log file. By interfacing JPF with JIVE, I hope to provide the following capabilities: (i) object diagram for visualizing an individual state; (ii) sequence diagram for an execution path; (iii) tree diagram for scheduler choices; and (iv) query interface for exploring conditions along an execution path.

5. Tell us about yourself (university, department, year). What are your qualifications, interests, and expectations? Have you worked on an open-source project in that past, whether through JPF, Google Summer of Code, or otherwise?

I am graduate student in Computer Science at The State University of New York at Buffalo. I will be completing my M.S. by December 2013. I am very interested in programming languages and databases, and am interested in building practical systems that embody advanced concepts such as software model checking.

I have extensive hands-on experience in object oriented programming through 2.5 years of work experience with Bank of America through Infosys Limited. I have worked on open source technologies such as Java, MySql, Tomcat and frameworks such as Struts 2.0, Grails, and ASP.net. I have implemented visualization of Real time Health monitoring plant at Bank of America along with other projects.

6. What is your availability this summer? Do you have other commitments?

If selected, I am planning to devote the summer of 2013 to this project.

7. Briefly describe any discussions you have had with JPF mentors about your project. (We encourage that you contact the mentors and discuss ideas with them before submitting an application)

My interest in JPF and JIVE started in Fall 2012 when I took a graduate seminar (CSE 705) under Professor Dr. Bharat Jayaraman. This seminar was devoted to model checking and other techniques for rigorous software design. I did my seminar presentation on symbolic model checking, and I have been discussing the interfacing of JPF with JIVE ever since the seminar. This semester (Spring 2013) I have carried out a number of preliminary experiments, which includes coding, in order to understand JPF's VM Listener and Search Listener as well as JIVE's external interfaces. Through these explorations I am already in a position to show proof of concept for the project.

Among the JPF experts, I have contacted Dr. Corina Pasareanu and Dr. Neha Rungta with questions about JPF. My discussions about the proposed project have been with Dr. Bharat Jayaraman, and I have been interacting with him for the past two semesters regularly. I am confident that I can execute this project over the summer and I believe that this interface will be very valuable to JPF users.