Thesis Proposal: Implementation and Evaluation of a Pluggable Type Checker for Loci 2.0

Amanj Mahmud Sherwany and Tobias Wrigstad

Uppsala University

1 Background

The Loci system proposes a simple, statically checkable set of annotations that lets programmers express thread-locality. It was designed with Java in mind, but should apply to Java-like languages with only a modicum of changes. The original system extended Java with two annotations: `(@Thread)`, to denote potentially thread-local objects and `(@Shared)` to denote shared objects.

Encouraged by e.g., Doug Lea (the concurrency lead for Java) we are intending to make Loci an official JSR for Java 8. This requires availability of a solid platform and IDE-independent implementation of the system to promote industrial uptake, and experimentation with the Loci annotations by programmers involved in the Java Community Process.

Furthermore, subsequent extension to Loci are only formalised but not implemented, and the current prototype implementation is tied to Eclipse, buggy, and does not incorporate annotation information for the Java standard libraries. A re-implementation of Loci incorporating this information would allow us to redo and extend the evaluations of the original Loci paper and hopefully get much improved results.

2 Task Description

Reimplementing Loci requires a deep understanding of the system. Understanding its formal description (including the Loci 2.0 extensions) and how it extends to full Java will be the first milestone, which will be achieved through careful study of the paper and the prototype implementation. We expect this to take about a full month and the outcome to be an extensive set of test cases in the form of small Java programs

that each test a certain feature of Loci and is expected to pass or fail with some message.

These test cases will be instrumental for verifying the correctness of the tool.

The second part of the project is implementing the tool using the JSR 308 checker framework for pluggable type systems. The use of this technology allows us to improve on Loci by allowing annotations on types (previously we were limited to declarations), and since it is included in Java 7, the implementation will work "automatically" in the upcoming Java release. Getting the implementation right, both in terms of correctness and future extensibility is important.

The third part of the project will be taking a number of representative programs and annotating them and seeing to what extent we can capture any thread-locality in them. We will report on the number of annotations needed for these systems, and any difficulties arising where the Loci system could not capture the system's thread-locality straightforwardly or at all. This is an extension of the evaluation done in the original Loci paper, but with some crucial extra information concerning thread-locality in the standard Java classes. This study will be the main part of the master thesis report.

Last, the project also entails the development of a manual for the tool, and a decent web page for the entire project, hosted at IT.

In terms of master thesis requirements, the project will involve both solid engineering of the tool, type system work (most notably the improvements to Loci made possible by the flexible annotations in JSR 308), and a practical evaluation where selection of representative programs is key. The project therefore combines engineering and scientific methods and should lead to a solid report on both the tool and its applicability to real-world Java code. We will be required to work with both industry tools and considerations, as well as research results from the pluggable types field and the Loci paper.

3 Procedure

A key aspect of the thesis work is the program selection. We plan to select programs from the DaCapo benchmark suite and annotate them, including the programs that we have already annotated for the previous

version of the system. The DaCapo programs have a lot of concurrency and are also standard measurements in the Java community.

Examples of literature relevant for the project:

1. Loci: Simple Thread-Locality for Java [Wrigstad et al. 2009]
2. Pluggable Type Systems [Bracha 2004]
3. The Checker Framework: Custom pluggable types for Java [Ernst 2008–]
4. JavaCop: Declarative Pluggable Types for Java [Markstrum et al. 2010]

And additional papers in the relevant fields. Presumably the report could be published at e.g., FTfJP or similar workshop.

4 Delimitations

Depending on the time available in the analysis phase, we might cut down the number of, or size of programs in the corpus. This will make the result of the analysis slightly more sketchy.

To be clear, the implementation of the Loci tool is not a prototype, but meant to be "industry quality."

5 Time Plan

The work is divided into four main parts:

1. Understanding Loci **(1 month)**
   (a) Go through the Loci type system in detail
   (b) Go through the Loci extensions
   (c) Look at capabilities for uniqueness in Scala
   (d) Investigate back-porting the Scala capabilities into Loci
   (e) Play around with the existing Eclipse tool for Loci 1.0
   (f) Rewrite the type system rules into a set of Java test cases

2. Learn the JSR 308 checkers framework **(1 month)**
   (a) Go through the checkers framework and JSR 308 documentation *(2 weeks)*
   (b) Write a small checker for pure functions in Java using the checker framework *(1 week)*
   (c) Implement Loci using the checkers framework *(1 week)*
3. Evaluation of the Loci 2.0 system using the type checker on a small number of medium-size programs **(2 months)**
   (a) Corpus selection and analysis *(1 month)*
   (b) Corpus annotation *(2 weeks)*
   (c) Bug fixing, etc. *(2 weeks)*
4. Writing up the report and presenting **(2 months)**

The work starts the first week of November, 2010 and is expected to be finished on April 30th, 2011.

5.1 Meetings with Reviewer

We plan five meetings with the reviewer. The goal of these is two folds: to give the reviewer confidence in the progress of the thesis work, and also introduce the reviewer to Loci, the key technologies and analysis results incrementally during the project. Tentatively, a meeting should be held on the last of each month, except the last month.

5.2 Meetings with Supervisor

Weekly meetings on Wednesdays.