Final Documentation

Editor Scalability

G20 - Robins

Brett Mason - Manager - mason15@illinois.edu
Maurice Lam - Customer - lam25@illinois.edu
Michael Robinson - Developer - robins26@illinois.edu
Thomas Zhang - Developer - zhang156@illinois.edu
Andrew Kryczka - Developer - kryczka2@illinois.edu
Joe Wrenn - Scribe - wrenn2@illinois.edu

Table of Contents

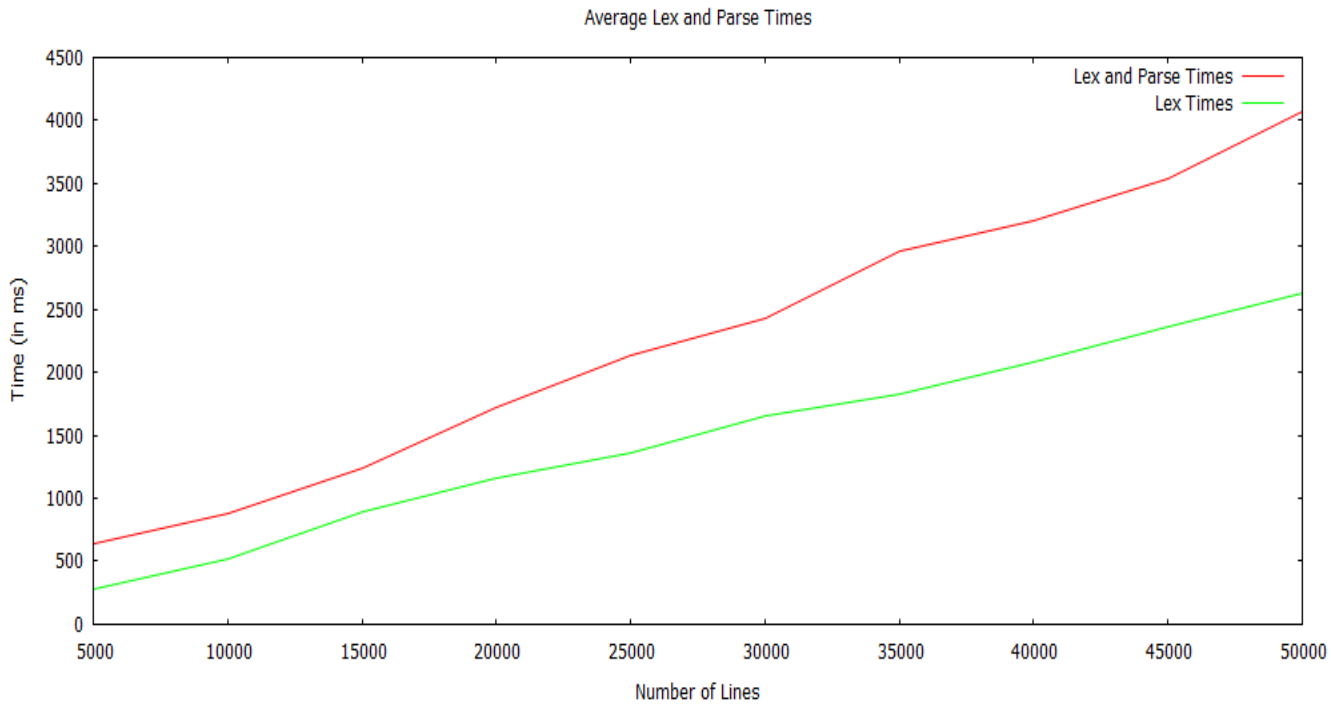
Description of Project

Project Overview

The Photran editor has a problem when files being edited become large. As file size increases the editor becomes less responsive, often freezing for more than a second while the user types. For our project, we were tasked with writing benchmarks to measure the performance of the editor on large files, making changes to Photran to improve its performance, and then using our benchmarks to show how our changes influenced Photran's performance on large files.

Our goal was to find inefficiencies in the code and give the user options to improve the performance (especially the responsiveness) of the editor. We first attempted to improve performance by adding an option to disable syntax highlighting on files larger than a certain threshold. However, no noticeable improvement resulted from this change. From our debugging and testing, we noticed that the UI thread was being blocked for multiple seconds upon changing a file, as the UI thread was performing a parse of the entire file. So, we decided to implement a concurrent model builder to parse in a background (i.e., non-UI) thread, as opposed to the default model builder which parses in the UI thread, interrupting the user's input. Our benchmarks confirmed a significant performance improvement from this change. Also, to confirm the correctness we implemented several new automated tests.

The following graph depicts the amount of time taken for the lexing and parsing, which were performed in the UI thread before our changes.



Architecture and Design

Approach

Our approach is to extend the existing `FortranModelBuilder` with a new module `ConcurrentFortranModelBuilder`, which overrides the `parse()` function with a new implementation. The purpose of having two model builders is to allow the user to choose between parsing the Photran editor code sequentially in the main thread or concurrently in a separate thread.

In `ConcurrentFortranModelBuilder`, we initialize an `ExecutorService` object that uses a single worker thread which operates off an unbounded queue of parsing tasks. We override the `parse()` function by canceling any previous unstarted tasks, interrupting the worker thread that is executing any unfinished tasks, and submitting a new parsing task to the `ExecutorService` object. This means that the unbounded queue will have no more than one task queued up at any given time.

The worker thread within `ExecutorService` will call the `parse()` function in `FortranModelBuilder` to begin the parsing task. During execution, this `parse()` function will return if the thread has been interrupted by checking the thread's status once before the actual parsing operation, and once before the loop replacement operation. These spots are chosen because parsing and loop replacement are by far the longest operations in `parse()`, and we do not want to execute these long operations if the thread has already been interrupted.

Photran Projects Modified

`org.eclipse.photran.cdtinterface`
`org.eclipse.photran.cdtinterface.vpg`
`org.eclipse.photran.core`
`org.eclipse.photran.core.vpg.tests`
`org.eclipse.photran.ui`

Photran Projects Added

`org.eclipse.photran.editortests`

Classes Added

`org.eclipse.photran.internal.core.model.ConcurrentFortranModelBuilder`
`org.eclipse.photran.internal.tests.AbstractEditorTest`
`org.eclipse.photran.internal.tests.ParameterizedResponsivenessTest`
`org.eclipse.photran.internal.tests.ParseTest`
`org.eclipse.photran.internal.tests.EditorTest`
`org.eclipse.photran.internal.tests.benchmarks.LexParseBenchmarks`

Classes Modified

`org.eclipse.photran.internal.core.model.FortranModelBuilder`
`org.eclipse.photran.internal.ui.editor.FortranEditor`
`org.eclipse.photran.internal.ui.editor.EditorPreferencePage`
`org.eclipse.photran.internal.ui.editor.FortranPreferences`

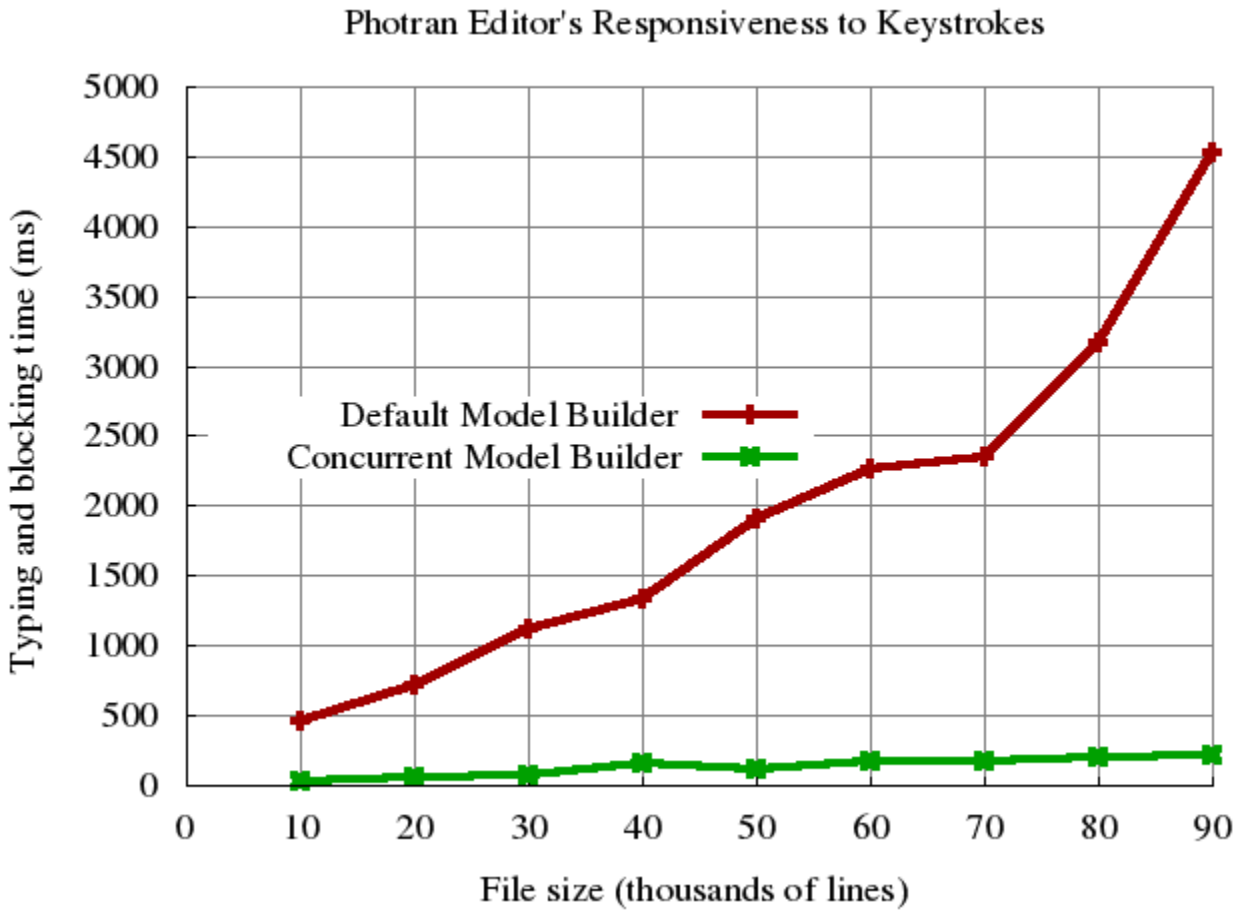
`org.eclipse.photran.cdtinterface.vpg/plugin.xml`

Lessons Learned

There are a few things that we wish we would have know beforehand with this project. For example, syntax highlighting is not a major penalty on the performance of the editor. We did not discover this fact until we had already figured out a way to turn syntax highlighting off and added a preference option to toggle it. Another important fact is that the Photran code is very similar to the code for the CDT plugin. It was very helpful to look at the CDT code to find test examples and ideas to help implement our changes. A link to the CDT repository can be found in the Appendix.

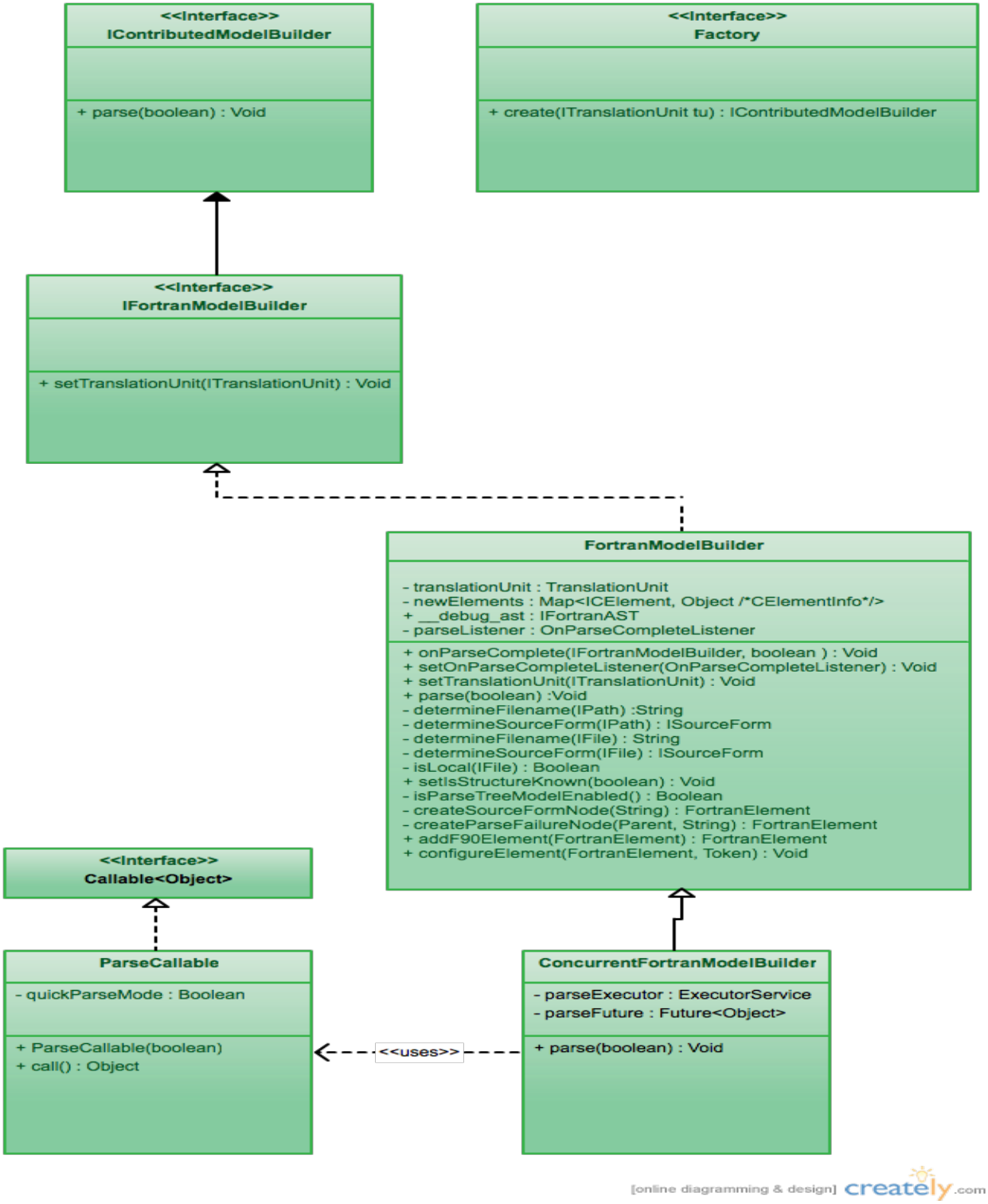
Results

With our addition of the `ConcurrentFortranModelBuilder`, we saw marked improvements in the performance of the editor on large files, as shown in the graph below.



One caveat with our approach is that the outline view will update with partial information or no information until the user clicks outside of the editor. Some of the refactoring options do not work, specifically the refactorings that edit the AST.

UML Diagram


<<Interface>>
IContributedModelBuilder
+ parse(boolean) : Void
<<Interface>>
Factory
+ create(ITranslationUnit tu) : IContributedModelBuilder
<<Interface>>
IFortranModelBuilder
+ setTranslationUnit(ITranslationUnit) : Void
FortranModelBuilder
- translationUnit : TranslationUnit
- newElements : Map<ICElement, Object /*CElementInfo*/>
+ __debug_ast : IFortranAST
- parseListener : OnParseCompleteListener
+ onParseComplete(IFortranModelBuilder, boolean) : Void
+ setOnParseCompleteListener(OnParseCompleteListener) : Void
+ setTranslationUnit(ITranslationUnit) : Void
+ parse(boolean) :Void
- determineFilename(IPath) :String
- determineSourceForm(IPath) : ISourceForm
- determineFilename(IFile) : String
- determineSourceForm(IFile) : ISourceForm
- isLocal(IFile) : Boolean
+ setIsStructureKnown(boolean) : Void
- isParseTreeModelEnabled() : Boolean
- createSourceFormNode(String) : FortranElement
- createParseFailureNode(Parent, String) : FortranElement
+ addF90Element(FortranElement) : FortranElement
+ configureElement(FortranElement, Token) : Void
<<Interface>>
Callable<Object>
ParseCallable
- quickParseMode : Boolean
+ ParseCallable(boolean)
+ call() : Object
<<uses>>
ConcurrentFortranModelBuilder
- parseExecutor : ExecutorService
- parseFuture : Future<Object>
+ parse(boolean) : Void
[online diagramming & design] creately.com

Future Plans

What will happen to the project?

We have no plans to continue development on this project. We have intentions to upload our changes to GitHub for individuals or groups who are interested in using or building upon our ideas. Some members of our group expressed interest in continuing developing but as a group we will cease involvement in the Photran project.

Group Member Reflections

The following are personal reflections from each of our group members.

Brett Mason

I was assigned to the Editor Scalability project with my two group members I worked with on the previous mps and three new group members. I enjoyed the opportunity to work as a group using the XP process on a real life project. In this project we were assigned a goal but the path towards that goal was up to us to decide. This fueled our creativity and allowed our group hands on experience most of us never experienced. However, this also created new problems, such as group dilemmas on how to reach our goal and using the new XP process. With my background in business and my lower level of understanding of computer science, I decided to become the manager of the group. Overall the position was not intellectually challenging but demanding. I had to ensure we stay on task with group meeting, ensuring we met the deadlines, and efficiently used the XP process. In conclusion, I enjoyed working on a real life project and using my strengths and building on my weaknesses as we progressed to accomplishing our goal.

Maurice Lam

When I chose the project to work on, my criteria was to choose something I know about. Not that I need to know how to fix the problem, but I need to know about *the problem*. Since I am not a Fortran developer, I do not know about the idioms and the styles commonly used in Fortran. I could not have fixed the problem well if I don't know what that is. Editor scalability is different. I used code editors and IDEs very often and I really hate it when it becomes slow. In the process of trying to improve the scalability, I can really feel how it's like to develop in a huge project - we basically spent 2 weeks getting to know how Photran works here and there, coming up with "solutions" that didn't really help. It's frustrating at first, but it is a really important experience that I wish I had earlier.

Michael Robinson

I chose the Editor Scalability project because I am interested in concurrent programming and enjoy reverse engineering. The project also had the bonus of not requiring Fortran knowledge, which I preferred not to have to develop. This project certainly presenting challenges, especially in determining how we could test and quantify performance enhancements. Doing this manually was out of the question, so figuring out how to automate this process was interesting. Overall, the project has been rewarding intellectually and the group has been professional and reliable.

Thomas Zhang

I chose the Editor Scalability project along with Maurice (whom I worked with on the MPs) because we felt that this was a project that would allow us to learn about Photran architecture without being bogged down by Fortran semantics. The most challenging aspect about this project was that we had to first specifically trace down the reason for the editor lag on large documents, and that caused us to try several potential fixes before finally being able to pinpoint the cause of the problem. Once we had that, we were quickly able to successfully come up with a spike solution and iteratively improve upon it through a mixture of group brainstorming and pair programming. Overall, I believe that the project was very rewarding as it allowed me to both improve my skills in Java and learn about an existing large scale system all while using the XP development process.

Andrew Kryczka

I was excited to work on the Editor Scalability project because I have been wanting to improve my knowledge and skills in concurrent programming in Java. Additionally, like my other group members, I did not want to learn much about the Fortran language since I cannot see such knowledge being useful beyond CS427. One aspect of the project I found challenging was learning a sufficient amount about the existing Photran project to make our changes. In our other CS classes, we typically work on standalone projects and do not have to learn complex software systems, so this was a new type of challenge for me. To overcome this challenge, our group successfully used a few of the reverse engineering techniques taught in CS427, e.g., reading the source code, reading the documentation, writing test cases, and monitoring execution with the debugger. In terms of our group, I found our group enjoyable to work with, and I agree with Michael that everyone was professional and reliable. Overall, I am happy with the useful knowledge and skills in concurrent programming and the XP process that I've acquired during the project, and I'm also somewhat satisfied with our final product.

Joe Wrenn

The editor scalability project for Photran has been a really interesting experience. Photran is the largest amount of code I have ever worked with so it was an interesting experience to dive into at the beginning of reverse engineering. I started with the documentation and that gave me an idea of where I needed to place a few breakpoints and from there I used the debugger until I had a good understanding of how our section of code was working. This project was especially interesting to me because I use eclipse a lot and I now have a greater respect and understanding for how eclipse works. I think our group did really well with solving the problem and we were able to do it in a timely fashion due to everyone involved being responsible for their share of the workload and respectful to the other group members. Overall this has been a very positive experience.

Appendix

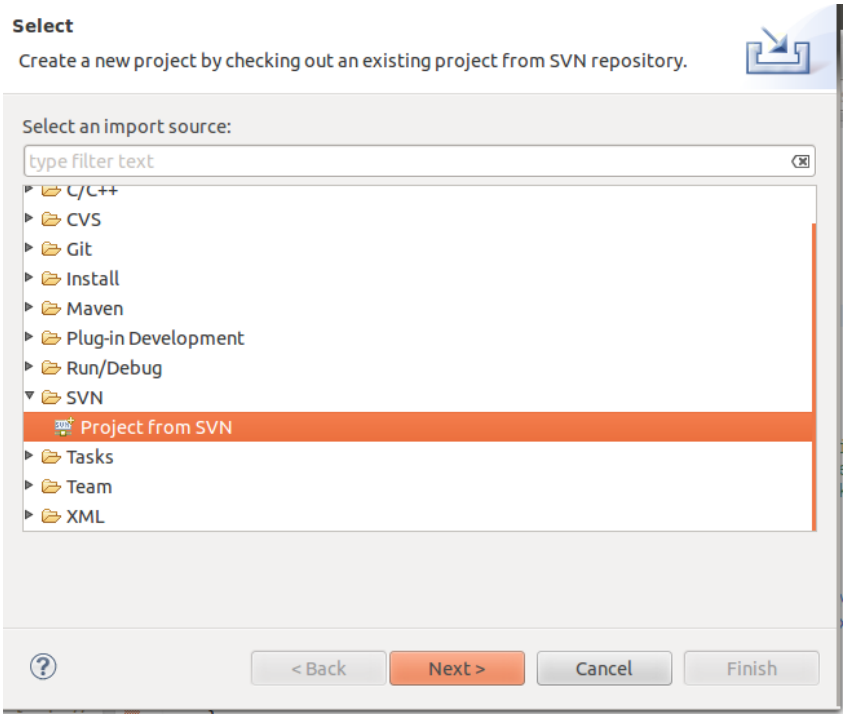
Installation

Group Repository Location: https://subversion.ews.illinois.edu/svn/fa12-cs427/_projects/G20/trunk
Photran Repository Location: https://subversion.ews.illinois.edu/svn/fa12-cs427/_shared/photran/trunk
Photran Website: http://www.eclipse.org/photran/
CDT Repository: https://github.com/eclipse/cdt

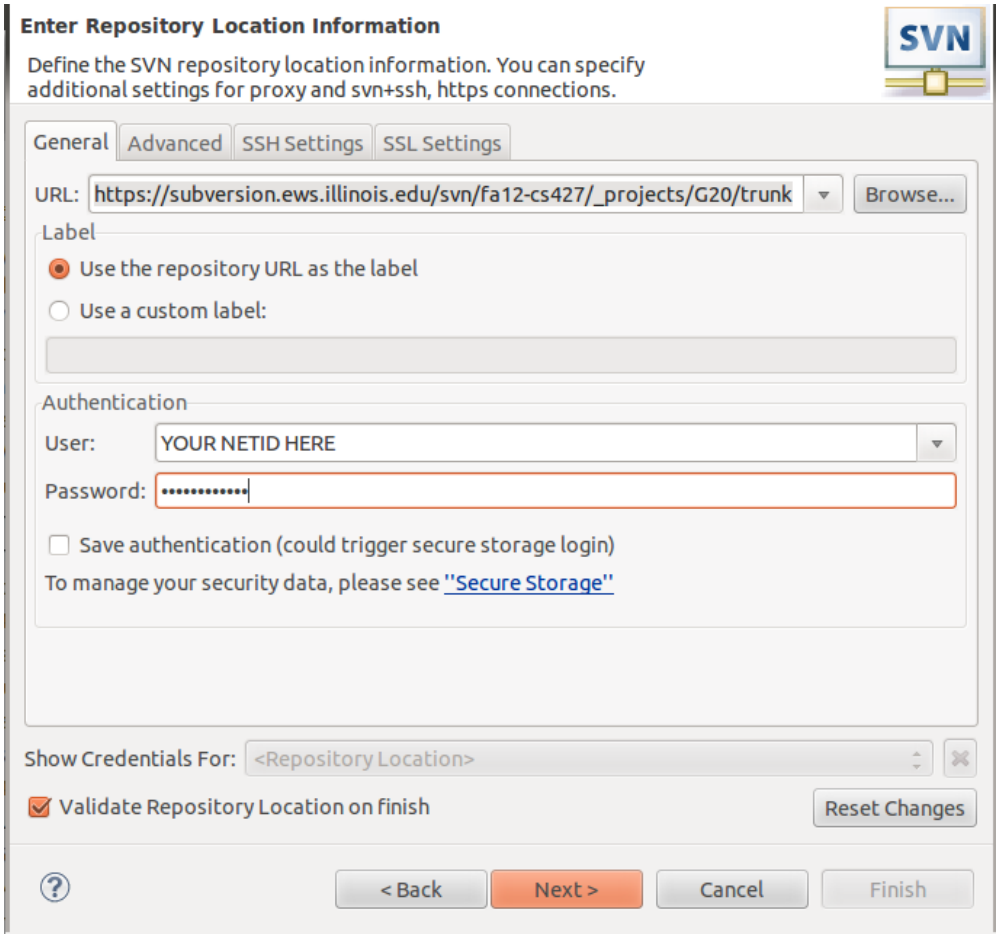
Follow the instructions at https://wiki.engr.illinois.edu/display/cs427fa12/MP4 to setup Eclipse for Photran with the base version. This is a prerequisite to checking out our project and must be completed in full. Note that you must have the SVN Team Provider plugin installed in Eclipse to be able to checkout any SVN project.

Next, use Eclipse to import our project, overwriting portions of the base Photran package as follows.

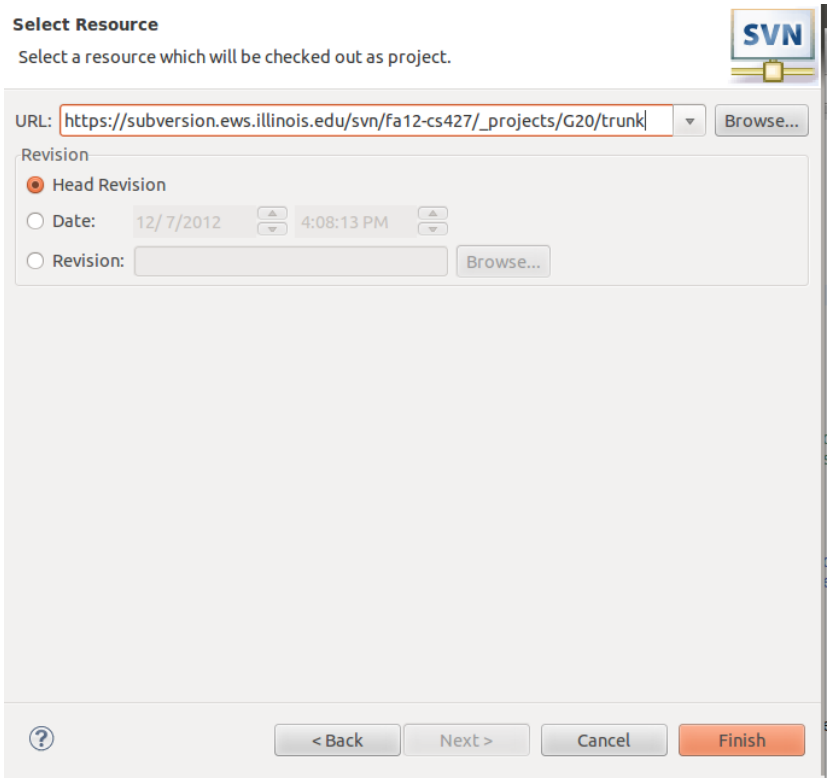
Open the import wizard by using File -> Import, and select “Project from SVN”



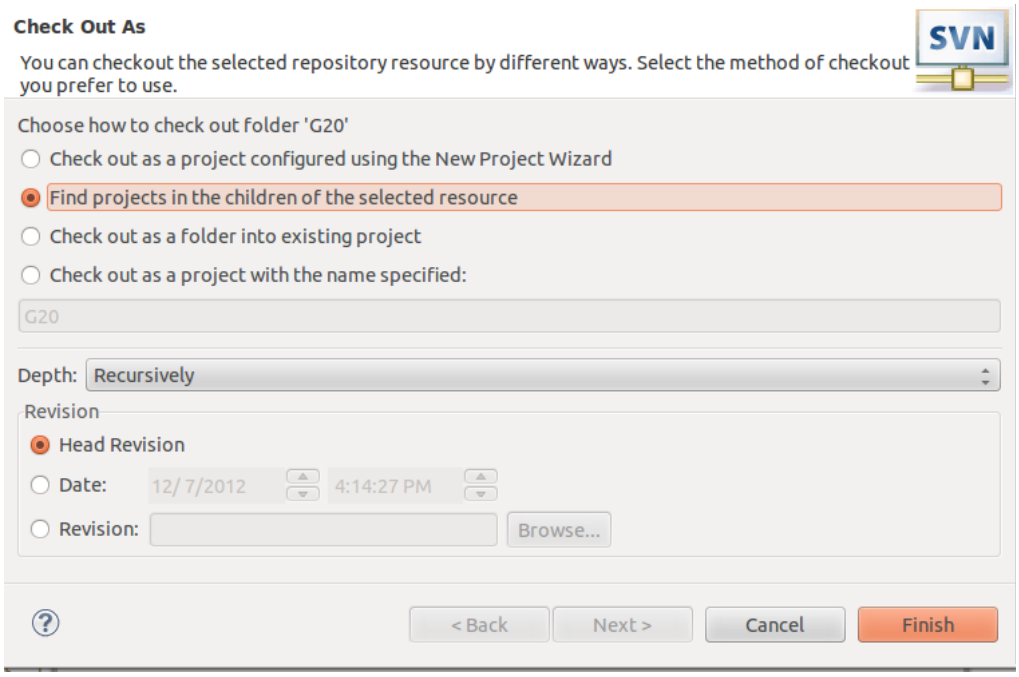
Then select “Create a new repository location” and click “Next”.
Enter the group repository location as displayed at the beginning of this section, also enter your NetID and password. Click “Next”. If it asks if you want to “normalize the URL by cutting its last segment off”, click “No”.



Make sure that “Head Revision” is selected, and click “Finish”.

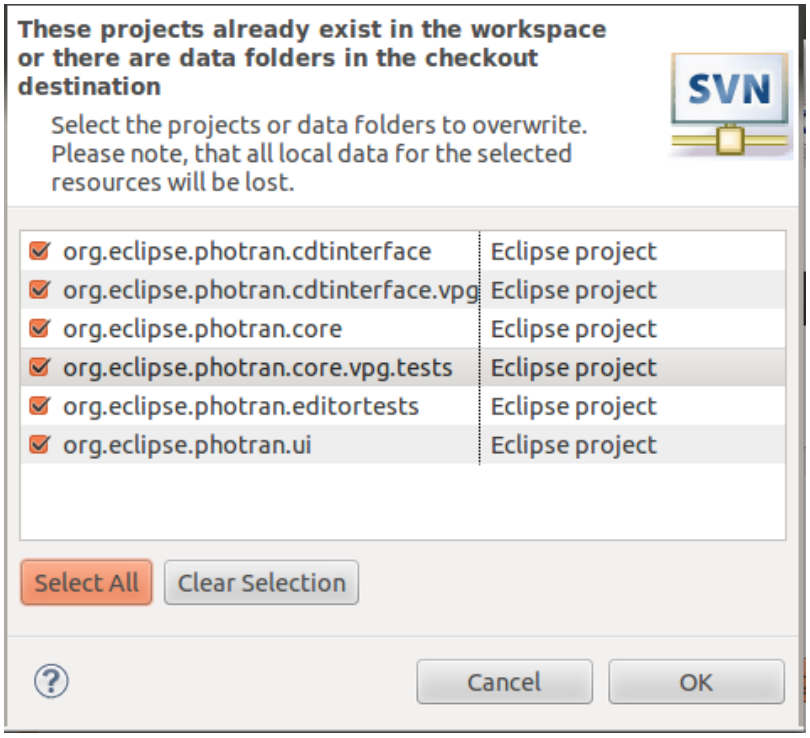


Select “Find projects in the children of the selected resource” and click “Finish”.



Be patient at it scans the repository. Then click “Finish” to “Check out as projects into workspace”.

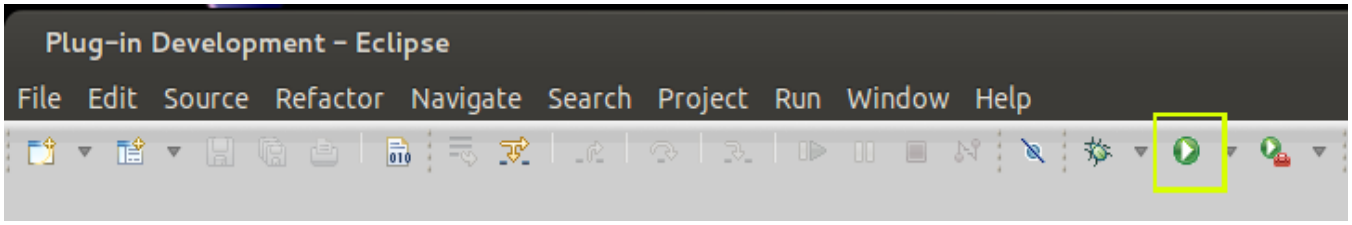
A dialog will appear asking if you wish to overwrite existing data in the workspace. Select all of the projects and click “Ok”.



Be patient as it checks out the project.

Running the Program

After following the steps above to install Photran with our changes, simple select any Photran project in your workspace and click the green “Run” button in the Eclipse toolbar (highlighted in the image below).



Running the Test Suite

After following the steps above to install Photran with our changes, you can run the test suite as follows.

Select the `org.eclipse.photran.core.vpg.tests` project in your workspace. Right click it and select “Run As” and select “JUnit Plug-in Test”. Do the same for the `org.eclipse.photran.editortests` project. Notice that the ParameterizedResponsivenessTest class runs benchmarks and outputs its results to graph/results.dat, which contains average running time for both model builders on files of various numbers of lines.