COM 1205 Project Description

Winter 2003

TraversalJ

Prepared by Karl Lieberherr and John Sung

Modified by Pengcheng Wu

Table of Contents

Introduction

A major software developer, KL Enterprises (KLE), wants to hire COM 1205, Inc. to create a prototype language system that will implement traversals in Java. In order to differentiate themselves from the DemeterJ product, they want to use AspectJ to implement the traversals. AspectJ is open source software from http://www.eclipse.org/aspectj.

KLE is aware that COM 1205, Inc. has minimal knowledge of AspectJ and the project is structured so that no AspectJ programming knowledge is needed. Some AspectJ reading knowledge is helpful and will be covered in class.

From a high-level view, COM 1205, Inc. is hired to perform a translation from one language A to a language B, using some predefined software. Such programming tasks are common, e.g., translate XML to HTML, Java to .class files, stylized English to stylized French, etc.

In order to investigate this further KLE has hired TA Consulting to do some prototyping and preliminary research to find out about the implementation issues and provide recommendations.

This document contains a description of the software tools that are available from the prototyping work and the recommendations from TA Consulting. If you have any questions about this document or the project itself, please send e-mail to com1205-grader@ccs.neu.edu. This will reach, KLE and the teaching assistant.

Environment Setup

You should use a different software setup than what you have used in your homework. To be concrete, you should have:

- Java 1.4
- DemeterJ
- DJ runtime library version 0.8.6 (you used 0.8.5 for homework)
- AspectJ compiler and runtime library.

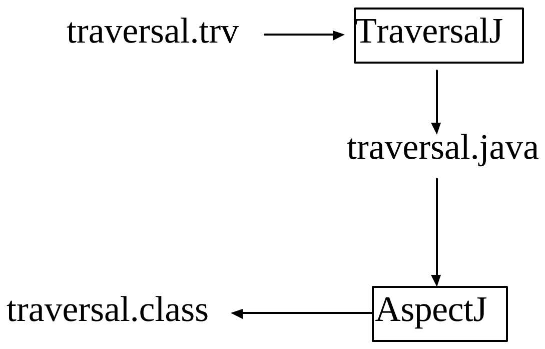
To setup an environment like that, your CCS's `.software` file should be like:

```
PATH=.:/usr/j2se/bin:/proj/demsys/demjava/bin

CLASSPATH=.:/proj/demsys/demjava/cur/demeterj.jar:
/proj/demsys/demjava/0.8.6/rt.jar:

/proj/demsys/demjava/packages/aspectj1.0.6/lib/aspectjrt.jar
```

For the `CLASSPATH` environment variable, you should put all of the paths in one single line, instead of separated lines. We encourage you to implement the project on CCS's solaris, where you can get the best support. If you have to work on Windows or Linux platform, besides the DemeterJ/DJ setup, you should also refer to http://www.eclipse.org/aspectj to figure out how to setup the AspectJ environment.

Software System Overview

The TraversalJ software should work as shown in the figure below. It should read in files that contain traversal specifications. Then, it should generate the implementation of the traversal. The implementation is in AspectJ. It allows the traversal code that is spread around many different classes to be in one place.



The main functionality of the TravarsalJ project is listed below:

?? Obtain Class Graph

?? Parse traversal files *.trv

?? Process the traversal with the obtained Class Graph

?? Using the Traversal Graph, output the implementation of the traversal in AspectJ

?? Compile the traversal with the original code

What's provided (both of the two files are available from the course directory in subdirectory project):

?? **CreateClassGraph.java** - using AspectJ and DJ, it intercepts the call to function main and creates a Class Graph object. Then, passes it to COM 1205 Inc. middleware using the ClassGraphFactor/Listener interface.

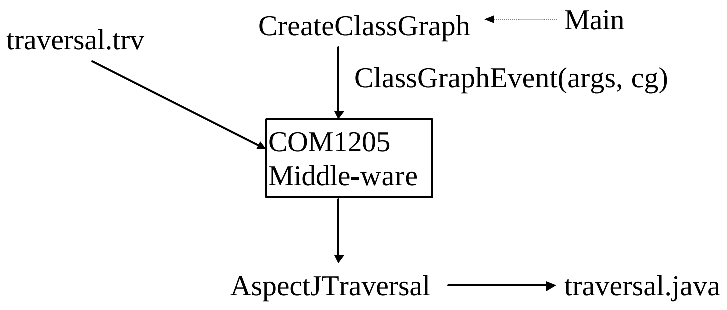
?? **AspectJTraversal.java** – provided by DAJ project (http://daj.sourceforge.net) originated by John Sung and being developed by Doug Orleans. It generates the traversal code in AspectJ given a Traversal or ClassGraph and a Strategy. Also, takes in traversal name for naming various components of the traversal. Class Traversal is a class newly added to DJ 0.8.6 and you can view it a synonym to class TraversalGraph.

COM 1205 Inc.'s Middleware:

- ?? **Traversal Specification Parser** - Write a class dictionary in DemeterJ to parse the traversal language specified.
- ?? **Command Line Processing** - Process command line such that it processes the files listed and an option to specify the destination directory of the generated code.
- ?? **Interfacing with CreateClassGraph** - Write the ClassGraphListener/Factory to obtain the command line arguments and the Class Graph of the program.
- ?? **Processing Strategy declarations** - Process the strategy declarations such that they can be used within the traversal declaration.
- ?? **Processing Traversal declarations** - Process the traversal declaration using the strategy declarations.
- ?? **Outputting Generated Files** - Write the generated traversal code out to file as specified in this document.
- ?? **Testing TraversalJ** - Implement a traversal used in developing TraversalJ, using the TraversalJ's traversal specification and run to see the traversal works.

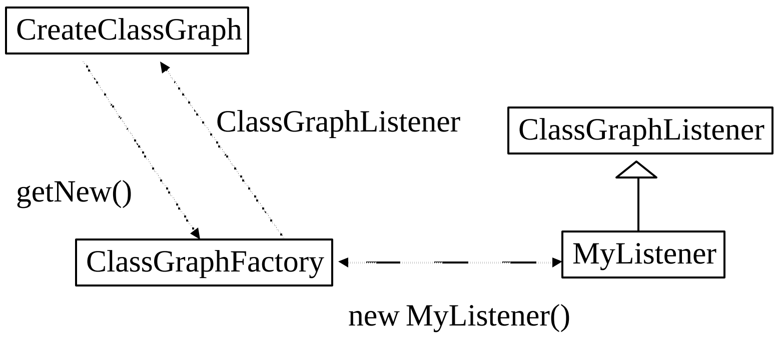
Provided Files

The provided files are basically the software layer above and below the COM1205 Inc.'s middleware. They are very thin, as they only need to do very small amount of low level work. The top layer is the CreateClassGraph layer that obtains the Class Graph and the command line arguments. The Bottom layer is the AspectJTraversal layer that generates the actual Traversal in AspectJ.

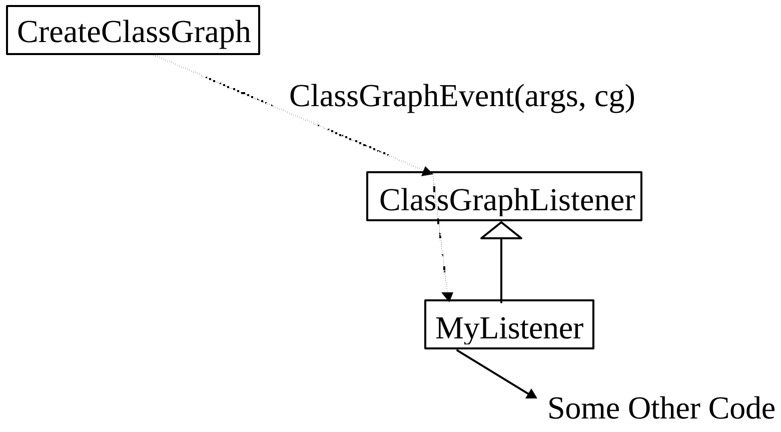


The CreateClassGraph class uses AspectJ to intercept the call to main method. Then, it uses DJ to obtain the Class Graph of the current running program. It also obtains the command-line arguments. It expects two classes, ClassGraphListenerFactory and ClassGraphListener to be defined. ClassGraphListenerFactory allows CreateClassGraph to obtain an instance of type ClassGraphListener. And ClassGraphListener should implement the abstract method ClassGraphEvent(String [] args, ClassGraph cg).

The figure below shows how CreateClassGraph obtains an instance of ClassGraphListener class.



The figure below shows how COM1205 Inc. middleware obtains the command-line arguments and the Class Graph.



COM 1205 Inc. should implement using DemeterJ:

?? The relationship between ClassGraphListener and MyListener (this can actually be any class name that you'd like, i.e. MyClassGraphListener, COM1205Listener, etc.)

?? The static method getNew() in ClassGraphListenerFactory.

?? The abstract method ClassGraphEvent(String [], ClassGraph) in class ClassGraphListener.

?? The implementation of the abstract method ClassGraphEvent(..) in MyListener that will start the processing of the traversal files. In a sense, this is the "main" method for COM1205 Inc.'s middleware.

Compiling with the provided code:

?? The CreateClassGraph.java and AspectJTraversal.java are available from the class directory.

?? Create a subdirectory under your main project directory and copy the files to there.

?? Add the directory in your .prj file's JAVADIRS option.

?? Change the COMPILER to ajc and no compiler options.

The AspectJTraversal Methods

There are several methods and data members that you should be aware of in AspectJTraversal.

?? **AspectJTraversal(String tn, Traversal tg)** – A constructor. Given a traversal name String and the Traversal object, generate the traversal methods implemented in AspectJ. You can access the generated code in the form of a String by calling toString() method on this AspectJTraversal object.

?? **AspectJTraversal(ClassGraph cg, String tn, String st) -** Same as above, but it takes in ClassGraph, traversal name string and Strategy string. It generates the Traversal from the ClassGraph object and the Strategy string and calls the above constructor.

?? **public String getAspectName() -** Returns the aspect name of the generated code.

?? **public String toString() -** Returns the generated AspectJ code as a String.

?? **static boolean debug -** the data member that is conditioned on for debug printing. If set to true, it outputs information as the traversal code is generated. Defaulted to false.

?? **static boolean addPrintingAdvice -** The data member variable that determines whether extra code is generated to output useful information when the traversal methods are called. Defaults to false.

Traversal Specification

A traversal specification file must have the definition of an **aspect**, which contains a list of traversal-related declarations. The following is an example of a traversal specification: file **Example.trv**:

```
aspect Example {
  declare strategy AtoB: “from A to B”;
  declare strategy AtoBSkipTail: intersect(AtoB, skipTail);
  declare strategy AtoBviaTelephone: intersect(AtoB,”from *
via Telephone to *”);
  declare traversal: trav1: AtoBSkipTail;
  declare traversal: trav2: intersect(AtoBviaTelephone,
“from A to C”);
   declare strategy: skipTail:
      “from * bypassing -> *,tail,* to *”;
}
```

Any declaration can be either of the following: a traversal strategy declaration, or a traversal declaration with or without visitors (in this project, we only focus on traversal declarations without visitors).

The traversal strategy declaration can have two different forms: **Simple form** or **Composed form**. The `AtoB` and `skipTail` in the above example are simple form strategies, just like those strategies we have used in a regular DJ program. The `AtoBSkipTail` and `AtoBviaTelephone` in the above example are composed form strategies, in which we can produce more complex strategies from simpler ones, which can either be a strategy variable referring to another strategy declaration (not necessarily a simple form strategy declaration), or a traversal strategy string, e.g., `from * via Telephone to *`.

Notice that if we use a strategy variable to compose a composed strategy, that variable must match a strategy declaration in the same aspect, which can be declared either before or after the current declaration. The semantics of a composed form strategy declaration is to construct a new traversal strategy by applying the second argument as a constraint to the first argument. It is very similar to what we have used in DJ program to construct a class graph slice, e.g., `ClassGraph cg2=new ClassGraph(cg1,”from A to C”);`

The `trav1` and `trav2` in the above example are traversal declarations. They can have two forms too. Traversal `trav1` is defined on a predefined strategy declaration, while traversal `trav2` is defined on a newly defined anonymous strategy. Traversal declarations are the traversals which we want to be generated into AspectJ traversal code, and the strategy declarations are only for supporting the definition of traversal declarations.

A file can contain only more than one aspect definition. And you can also have more than one .trv files. But be aware of that the **traversal name** is some sort of global variable, so you must make sure there is no two traversals having the same name, either in the same aspect or in different aspects.

What you need to implement is given such an aspect definition containing a list of strategy/traversal declarations, try to figure out the relationship/constraint between them. Then generate the traversal code in AspectJ using the ClassGraph object created by CreateClassGraph.java and the interfaces provided by AspectJTraversal.java

Testing TraversalJ

During your development of TraversalJ you should create some traversal specification file to test your project. After you feel that you have everything completed to your satisfaction, you should test the TraversalJ project by generating a traversal in AspectJ from a traversal defined in TraversalJ in the TraversalJ specification. Then, generate the file traversal, recompile and run TraversalJ again. But you should set AspectJTraversal.addPrintingAdvice to true. This will generate extra code that will print the traversal methods.

?? Set AspectJTraversal.addPrintingAdvice to true somewhere in your code.

?? Compile TraversalJ.

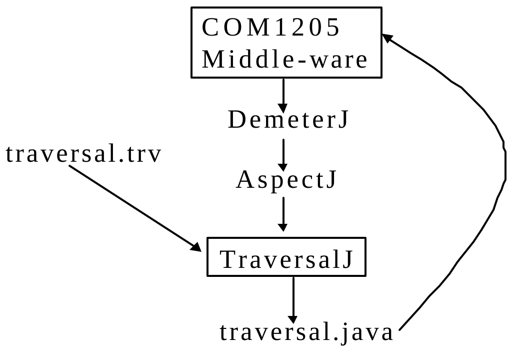
?? Implement a non-trivial AspectJTraversal used in TraversalJ in the TraversalJ's traversal specification language.

?? Run the TraversalJ on the file you wrote in the previous step and store the file into different directory.

?? Then, recompile with the newly generated file by setting the JAVADIRS to be the one specified previously.

?? Rerun the TraversalJ. You should see bunch of calls (method signature) in the output along with all of the other output. This verifies that it actually generated the correct traversal for the given traversal specification.

The figure below shows the process:



Suggested Project Phases

This section describes the project phases for implementing the TraversalJ traversal code generator. It is highly recommended that COM1205 Inc. follow these phase as it optimizes for maximum points in case of incomplete project by the project deadline.

The recommended phases are divided into 4 phases listed below, You are encouraged to break down the project into even smaller phases, practicing incremental software development using a spiral software development process.

?? Phase1 – Write Class Dictionary for the traversal language

?? Phase 2 – Implement –d directory option, with simple traversals

?? Phase 3 – Implement strategy definition and traversal definition

?? Phase 4 – Test code by generating a traversal used in TraversalJ and in other Java programs, e.g., a Basket example.

In Phase 1, you should generate the Class Dictionary for the traversal language.

In Phase 2, you should implement the ClassGraphListener/Factory interface and the -d option. You should be able to obtain the String array of command-line arguments and find the directory specified by the "-d" destination directory option. And, also be able to access the ClassGraph object. You should also implement the basic traversal code generation using AspectJTraversal layer.

In Phase 3, you should implement the strategy and traversal definitions. The program should be able to recognize other declared strategies and generate the correct traversal with a ClassGraph passed to it.

In Phase 4, you should select a non-trivial traversal that is within TraversalJ and implement it using TraversalJ's traversal language. Then, generate the traversal code, recompile and run. Make sure that you turn on the option to generate printing advice and call the traversal method somewhere in your .beh files. Also test your project with other Java programs to which you add traversals.

In this project breakdown, Phase 2 is the biggest step as you need to implement the ClassGraphListener/Factory interface and traversals that will use AspectJTraversal to generate the traversal code. You should attempt to finish this part with 1 week left 'til the deadline. Don't forget to save some time for the write-up as it may take longer than anticipated.

After each phase, you should save the state of the project to some other directory. This allows you to roll back to the last phase if you happened to took the wrong approach in solving the next phase. Not only that, if your program does not compile and you need to submit it, then you can submit the last save of the project. This is usually a good practice for big projects.

Debugging Strategies

This section outlines some debugging strategies for TraversalJ. Some of the strategies listed here can be useful in general.

Debug Printing

When inserting printing methods for observing what the program is doing, it best to make it conditional. The conditional printing should be predicated on a boolean variable that can be set by any other classes, i.e. public. This allows you to turn all of the debug printing on an off. In the C or C++ program, you can use macros to create these printing methods easily. But in Java, you need to just have an if statement for the println methods. Look at AspectJTraversal as an example.

Note: If you know some AspectJ, you could use a debugging aspect.

You should also utilize the PrintingVisitor and DisplayVisitor with the universal traversal to printout the object graph. This will help you to understand what's going on with the parsing of the input file.

Debugging Traversals

When a traversal does not work the way you intended, it can be daunting to attempt to find out what's going on. Here are couple of hints:

?? You may use the debug printing method.

?? You could also print out the traversal graph specified by the strategy using DJ.

?? You should also check the demeterj compilation output very carefully, it outputs lots of messages and some warnings or errors could be missed.

?? You can take a look at Java code in gen directory to see what's really happening.

?? You can attempt to use jdb or ajdb (AspectJ debugger) to poke around.

Miscellaneous Parts

This section describes all the miscellaneous parts of the TraversalJ program.

Command-Line Processing

The only command line option is the "-d" option which specifies the directory in which the output of the file should be written. All of the other strings in the string array should be treated as a traversal specification file, *.trv.

So, a sample arguments in .prj for TEST_ARGS could be:

```
-d trav MyTraversal1.trv MyTraversal2.trv

MyTraversal.trv -d gen/trav
```

Output File Name Format

The output file name should be **aspectname.java**, where **aspectname** is the name of the aspect you have defined in a .trv file. There's a method getAspectName() in AspectJTraversal to return the correct file name format without the .java extension.

.prj Options

JAVADIRS - List the directories for the provided files and generated traversal code for testing.

COMPILER - For this project, it should be changed to ajc, the AspectJ compiler

COMPILE_ARGS - This should be blank since no arguments are needed.

TEST_ARGS - This is to specify the actual command line to TraversalJ. It should be able to handle -d option and a list of traversal specification files.

TEST_INPUT - This doesn't matter, but until the command-line option is implemented it should specify a traversal specification file for phase 1.

Project Write Up

For the project documentation you should document the Class Graph as a UML class diagram, the traversals used in TraversalJ, which traversal you chose to implement and some analysis of TraversalJ. The Class Graph UML diagram should be in same format as the ones from Homework1. It should have all of the is-a and has-a relationships. This should include only the classes defined in your .cd file.

The traversal documentation should include the code and a high level description of what the traversal is doing, i.e. This traversal is used to visit all the commands. It uses the strategy from CommandList to {Command1, Command2}. It does blah blah blah for Command2 and blah blah blah for Command2.

The documentation for the testing phase of the project should include the place in which the traversal that you picked is located in .beh file and where you call the generated traversal method. It should also include step by step instructions on how to compile, run, and test your project. This should also be included in the README file that should be in your project directory at the time of your submission.

The analysis should document your analysis of the utility of TraversalJ in general. Discuss possible productivity gains and in which areas these productivity increases can be gained. There is no right answer for this section, but you should demonstrate that you have thought about this subject and give some logical rationale for your analysis. It should also include possible reasons for having the listener and factory pattern for the interface between ClassGraphListener/Factory and COM1205 Inc.'s middle-ware.

Make sure to include the names of the project members.

README

This file should include the names of the persons that have worked on this project and the step by step directions on how to compile, run, and test your project. Also, include the places in which the traversal that was selected to test and where the traversal method is called.

Submission

For submission, use com1205-submission@ccs.neu.edu. run demeterj clean and zip to package the project.

You can do this multiple times, but it's highly discouraged. We may mistake yours as late or grade your earlier submission.

Due Dates: Start now. No extensions are possible.

Project Submission: Tuesday, March 3, 2003, 11:59pm.

Write-Up: Friday, March 7, 2003, 11:59pm.

Non-CCS Project Environments

There are some software hoops that you'll need to go through in order to compile and run the Java program.

The basic steps in creating your own project environment are:

?? Obtaining DemeterJ and DJ (0.8.6)

?? Obtaining AspectJ from www.aspectj.org

?? Setting up path and classpath environment variables for DemeterJ and AspectJ

Linux Environment

If you have a Linux machine then you should be all set with the steps mentioned above. No other extra care is needed. If you happen to want to develop on a Windows machine then you need to read on and follow the directions listed in the next section.

Windows Environment

On Windows98, there's a limitation on how long the command line can be and if you have long classpath variable, then demeterj's compilation system will not work correctly when attempting to run. The limitation could be in Windows 2000 as well, but I don't have a machine to test it out on.

Also, AspectJ's compiler doesn't seem to like the command-line options that DemeterJ is passing to it. Thus, you need to actually execute these steps yourself. It would be best to write a batch file to compile, test, and recompile with generated traversal code.

Here are the steps involved:

First copy the provided files to a directory under the main project directory. For this example call the directory "interf".

```
mkdir interf
copy c:\download\*.java interf
```

Run demeterj to generate gen\classes directory. It'll might give some compilation error if you have any code that accesses AspectJTraversal.

```
demeterj
```

Then, run the AspectJ compiler on it from the gen\classes directory:

```
cd gen\classes
ajc ..\..\interf\*.java
```

Then, run demeterj to regenerate, run ajc to compile everytime you change your code.

```
demeterj
cd gen\classes
ajc ..\..\interf\*.java ..\.*.java
```

If you want to run the test:

```
set classpath = gen\classes;%classpath%
mkdir trav
java Main -d trav myTrav1.trv
```

If you want to compile with generated code:

```
cd gen\classes
ajc ..\..\interf\*.java ..\..\trav\*.java ..\*.java
```

This seems like it's daunting at first, but if you have two command prompt running with doskey installed then it's much easier. One is used for running ajc and the other is used for demeterj and running the test cases. This coupled with emacs for windows and emacs' shell buffer, you have a very good system for developing TraversalJ.