Weekly Report

Team Dec 15-12: @PaniniJ

Week 5: *Feb. 16th - Feb. 22rd*

Advisor	Dr. Rajan	
Client	Dr. Rajan	
Team Members	Dalton Mills	*Webmaster*
	David Johnston	*Team Lead*
	Kristin Clemens	*Communication Lead*
	Trey Erenberger	*Key Concept Holder*

Weekly Summary

Work this week on the project was focused on two things. First, we drafted a first version of the Project Plan document and sent it to Dr. Rajan for review, which he approved. The Project Plan was submitted on Saturday evening. Next, we looked more closely at the prior work that is a part of the PaniniJ project. In particular, we have been reading the code in `org.paninij.runtime` and the intermediate Java code generated by `panc` (made available using the `-XD-printflat` flag) in order to understand how they interact.

Thursday evening, David observed a potential conflict between the way that we were expecting to generate capsule classes and the way that runtime code is generated. This issue was described it in an email, “SE 491: Template Class Name Conflicts with Auto-Generated Interface Name.”

On Friday, while discussing this issue and some other potential project risks with Dr. Rajan, it was agreed that @PaniniJ-`panc` interoperability should not be a top priority for our project. As such our goal is to build a solution which uses existing the runtime classes but with some modifications.

On Sunday, the group spent time analyzing the source code artifacts generated by `panc`, especially the parts which relate to duck futures. We performed a semi-systematic analysis of the code generated by procedures of various shapes. Our interpretation of the results has led us to believe that a simpler duck-future-generation process is possible. This alternative solution would preserve the one-object-per-procedure-invocation constraint, increase procedure invocation performance, and reduce the variety of duck future classes which need to be generated. We will be further discussing and investigating this potential solution this week.

Technical Progress

- Dalton did a lot of research on pluggable type checkers, more specifically, the Checker Framework. Type checking is an area of research done by the University of Washington where they developed the Checker Framework for Java 1.6 and 1.7. Much of what they accomplished was included in Java 1.8 (JSR 308 http://types.cs.washington.edu/jsr308/) Since type annotation processing is an immature area for Java, the documentation is underwhelming. Type annotation processing will require much more in depth research to see exactly how it can be used to check capsule integrity. Dalton also continued to work on the project website.

- David performed some refactoring of the existing capsule-generation code that should make it easier to create multiple source-code artifacts from a single template class. (This work can be found on the `[capsule-generation/multiple-artifacts]` branch.) He also drafted some ideas about what capsule initialization and wiring might look like from the template-programmer's perspective. (This work can be found on the `[examples/original-benchmarks]` branch.) The basic idea is to break up state initialization, capsule design, and capsule wiring into three separate functions which can be called by the runtime as appropriate. The intention is that the contents of these functions can be relatively declarative, short, and follow simple conventions. At some point in the future, we hope that these conventions can be enforced by the compiler.

- Trey looked into the changes necessary to allow for the code generated from our project to be compatible with the existing org.paninij runtime. This involved scrutinizing the java code created during the `panc` compile process, understanding the transformations that occurred, and thinking about how java 1.8 changes what we are able to do during this step.

- Kristin participated in analyzing the java code generated during the `panc` compile process and identified a number of areas that will require either a complete rework of the source or, at least, heavy refactoring.

Meetings

Weekly Administrative Meeting

Members Present: All
Additional Participants: N/A
Date & Location: Tuesday 17 of February; Molecular Biology 1414
Minutes:

- Start: 2:50
- Kristin report on Maven progress:
    - Going well, David ironing out some errors
    - On GitHub issues
- David report on `org.paninij.runtime` integration:
    - Issues with dependencies in the Panini runtime.
    - Attempting to pull in the project manually and comment out the lines/modules causing the problems.
    - Will report the results of this reduction to determine what needs to be done on the panini.runtime package.
    - Possible Maven wrapping of the pieces of `org.paninij.runtime` to allow for dependencies to be resolved in smaller chunks.
- Dalton going to add a maven module and work on type checking this week
    - Might spin off into a separate repo to get a proof of concept working. Type checking is becoming more of a 2nd semester objective, but we would still like proof that the idea will work.
- Trey going to work on refactoring tasks once Maven transition is stabilized.
    - Matching Dr. Rajan's specified conventions [interoperability]
- Group going to transition from trello to github issues and wiki to store information
    - Will allow more transparent progress and task tracking.
    - Will allow documentation to be aggregating in the final place.
- Module Diagram Task
    - Map out module connections
    - High level view of our project to demonstrate how the system is composed
    - May depend on understanding of the panini.runtime which could potentially delay this.
- Concluded 3:37

Extra Collaboration Meeting

Date & Location: Thursday 19 of February; Google Hangouts
Members Present: All
Length: 5 hours
Purpose: Draft Project Plan
Result: Project Plan completed, submitted to Dr. Rajan for review.

Bi-Weekly Advisor Meeting

Members Present: Trey, David, Dalton
Additional Participants: Dr. Rajan
Date & Location: Friday 20 of February; Atanasoff 101
Minutes:

- **Start**: 10:30am
- David explained the naming conflict issue he previously described in his email “SE 491: Template Class Name Conflicts with Auto-Generated Interface Name”.
- Dr. Rajan agreed that @PaniniJ-`panc` interoperability was not a top priority (i.e. not a functional requirement). There is no problem modifying and adapting existing runtime files to better fit our capsule-generation system.
- Dr. Rajan indicated that we should expect some difficulties associated with wiring capsules together.
- Dr. Rajan reiterated a few priorities for our project:
    - User-defined code that serves as input to our capsule-generation system should be not generate any Eclipse warnings.
    - Capsule systems generated by @PaniniJ must be performant. Performance should be comparable with capsule systems generated by `panc`. We will need to implement benchmarks to empirically compare our @PaniniJ with `panc`.
- **End**: 10:55am

Weekly Collaboration Meeting

Date & Location: Sunday 22 of February; Google Hangouts
Members Present: All
Additional Participants:
Minutes:

- **Start**: 1:20PM
- Testing of procedure shapes when compiled with `panc`
    - Examination of Panini example: HelloWorld.java
    - Built ProcShapes.java with various method shapes to analyze code generated by `panc`.
    - Ran through `panc` in several waves.
    - Examination of generated duck classes, their superclasses, and how they are used.
    - Building understanding of how duck classes are grouped when generated.
        - Ducks are currently grouped by return type and `capsuleName`.
        - Possible to change the way this works to make more sense.
            - Looking primarily at the generated "enum" that is used to switch the duck message to the method it needs to call.
            - Duck constructor assigns parameters to every argument 'hole' on every method call.
- Building future ducks.
- Future work:
    - Looking into making duck class generation more sane.
    - Adding interfaces with default keywords to handle code duplication.
    - Changing the way ducks are grouped to cover multiple capsules.
    - Using type parameters to simplify duck interfaces/classes.
    - Code generation helper methods.
    - Making generated code and methods that result in generated code more friendly.
    - Creating classes that handle various portions of source code.
- **End:** 6:20PM

Individual Hourly Contributions

Trey Erenberger	15 Hours
David Johnston	15 Hours
Kristin Clemens	10 Hours
Dalton Mills	17 Hours

Culumulative Time Contribution

Trey Erenberger	42.5 Hours
David Johnston	57.7 Hours
Kristin Clemens	33.5 Hours
Dalton Mills	42 Hours

Tentative Plans for Week 6

The focus of the coming week will be on *duck futures*. We need to plan out how the duck futures will be generated from a capsule's procedure signatures. We will be meeting Monday (Feb. 23nd) at 1:00pm with the specific purpose of identifying what tasks need to be done and who will tackle these tasks.