Fabric Information Flow Calendar

MEng Project Report

Elizabeth Perk
ejp82

December 15, 2017

1 Introduction

The intention of this project was to port an information flow calendar application written in the JIF language to the distributed information flow language, Fabric. For this reason, the capabilities and specifications of the JIF calendar application will be discussed to know what exactly needs to be written in a Fabric version of the calendar. Afterwards, an array of topics regarding the calendar will be discussed, ranging from prior failed attempts to the numerous ideas regarding how the Fabric calendar architecture would be modified to fit better in a distributed system. These prior thoughts will not constitute the majority of the discussion, since discoveries of other related applications have caused the most recent version of the calendar to begin to take shape. The location of useful packages for finishing the Fabric application will be documented. After this, remarks will be made regarding everything that was accomplished as well as all that which remains unfinished. Then, various pitfalls that were encountered this semester will be described in detail—with the hope that future newcomers to the Fabric language will not get stuck in the same places. Finally, various contributions that were made throughout the semester and additional suggestions regarding Fabric and the associated code base will be mentioned.

2 JIF Calendar

To port the calendar successfully, it is necessary to know the capabilities provided by the JIF distribution to ensure that everything is transferred over to the new distribution. It is also essential to know the capabilities and limitations of each language, namely the differences between JIF and Fabric, to determine any functional limitations, if any. It may become necessary to explore possible workarounds if some of the functionality does not easily port to Fabric.

Intended Functionality—as prescribed by Chong, Vikram, and Myers—for the SIF calendar[1]:

1. Authenticated users may create, edit, and view events.
2. Events have a time, title, list of attendees, and description.
3. Events are controlled by expressive security policies, and application users are able to customize these policies.
4. Users can only edit an event if the user acts for the creator of the event.
5. Only the event creator or attendees may view details of an event (title, attendees, and description).
6. An event may specify an additional list of users who may view the time of the event.
7. If a user may view the time of an event, but not the details, then the user will see "Busy" as the event description.

Definition: A user's calendar is the set of all events for which the user is either the creator, attendee, or viewer.

Application Capabilities:

1. Update session state with date to display.
2. Update session state with which individual user's calendar to display.
3. Fresh ID for new event.
4. Update and retrieve info from the database.
5. Go to View/Edit Event page.
6. Error editing event.
7. Changing attendees or viewers of an event.
8. Delegation to CalRoot (the calendar's root principal).

9. Capability to change the color of the calendar with hot-keys to depict which users have what confidentiality or integrity rights.

The JIF calendar is not distributed, and therefore does not have to conceptualize the interaction between different nodes and the resulting information flow implications this can cause. Beyond that, this implementation relies on a MySQL database to perform authentication and preserve consistency. A **db_defn** text file is provided to automatically generate the necessary tables, and the users and their passwords are hard-coded into insert statements in this file. Furthermore, the class **CalendarDB.jif** is an empty class which primarily returns null or 0 everywhere. Whereas the rest of the classes get compiled down from JIF to Java, a **CalendarDB.java** class is already provided that supplies much of the desired functionality.

2.1 Running the JIF Calendar

What we found in the actual execution of the JIF calendar was that there were some issues getting it to work. The build file for automating the insertion of dependencies into the necessary Tomcat directories was not working correctly. After struggling with Tomcat, I switched to Jetty, which meant that I could not use the script any longer. However, I put the built JIF application into various Jetty directories until I managed to get the servlet to display stacktraces.

What worked was to put the necessary dependencies in the subdirectories **classes/** and **lib/** under **$jetty/webapps/calendar/WebINF**. From there I was able to determine which other ***.jar** files from the JIF libraries were dependencies to the application. Unfortunately, at this point there was a struggle trying to figure out how to get Jetty to use the shared directory for the common JIF dependencies that would be needed across different JIF web applications. However, after some effort, no solution was found regarding this issue.

After this point, by attempting to find necessary dependencies via the stacktraces, working from error to error, I got stuck on an error that I could not resolve. Finally, Jed revealed to me that the issue involved methods that had been deleted out of the current repositories of JIF more than four years ago. After he pushed the master changes, I attempted to recompile JIF. However, after the pull, I started to have issues with the new **build.xml** file, which I had to modify in several places before compilation worked. I eventually decided to comment out lines 473 and 547 of the **build.xml** because I assumed that I did not need the unix runtime, and this turned out to be the fix I needed. Finally, I was able to run the JIF-SIF calendar application!

In running the application, I found that a lot of the capabilities outlined in its paper were lacking or not implemented to the degree that I had expected. Besides many UI issues where text fields emptied themselves the moment you changed one field in the edit event page, there was no time indicated in the calendar. As a result, the user only knew the day and not the time of an event (though this might suffice for a demo). However, beyond that, there were issues with getting the information flow coloring to work. First, the JIF-SIF calendar could not locate the **preamble.js** file, which I eventually located in the **src** directory of one SIF repository. However, finding it did not help initially because I struggled to get it loaded into Jetty in a way that the SIF servlet would recognize it. Finally, I changed the path in the **Servlet.java** file to be within the **webapp**'s directory, and after rebuilding SIF, Jetty finally loaded the preamble.

Next, there were issues with the accompanying JavaScript file. I could not get my Mac to recognize the keypress events in any of the three web browsers that I tested. In the developer console, I did not get any output for the code being called in the body tag element:

```
<body onkeypress="actionDown(event);" onkeyup="actionUp(event);" onload="initialize();">
```

After using a few HTML test files, I realized that the above method was having issues, so I decided to use JavaScript in the preamble instead:

```
document.addEventListener("DOMContentLoaded", function(event) {
    document.addEventListener("keydown", actionDown, false);
    document.addEventListener("keyup", actionUp, false);
```

```
});
```

After this, I could see that output was occurring within the console when I pressed keys, but I was not getting the expected behavior. For all three browsers, none were changing colors correctly based on the JavaScript in the **preamble.js** file, which was supposed to register a combination of key presses, namely the **alt key** combined with another key code. Presumably, the old JavaScript was fine a few years ago. However, it seemed that the newer browsers were registering different codes for one key. The following modified code mostly worked:

```
function actionDown(e)
{
    if(!e) { e = window.event; }

    var key = e.keyCode ? e.keyCode : e.which ? e.which : e.charCode;
    console.log( "Code: " + e.code );
    console.log( "KeyDown: " + key );

    if(!flag && e.altKey)
    { // AltKey was true
        if (key == 81 || key == 113) { // Q: simple coloring
            console.log( "Q key was pressed" );
            flag = true;
            colorScheme = 113;
        }
        else if (key == 87 || key == 119) { // W: HSV coloring
            console.log( "W key was pressed" );
            flag = true;
            colorScheme = 119;
        }
        else if (key == 73 || key == 105) { // I: Toggle between B&W background
            console.log( "I key was pressed" );
            isBkBlack = !isBkBlack;
        }
        else if (key == 79 || key == 111) { // O: Toggle between integrity & confidentiality
            console.log( "O key was pressed" );
            isInteg = !isInteg;
        }
        else if (key == 82 || key == 114) { // R: Saturation based color
            console.log( "R key was pressed" );
            flag = true;
            colorScheme = 114;
        }
        else if(key == 76 || key == 108) { //  L: Display Legend
            console.log( "L key was pressed" );
            displayLegend();
        }
                if(flag) {
                        conf(getColorFunc());
                }
        }
        console.log("\n");
}
```

3 Fabric Calendar

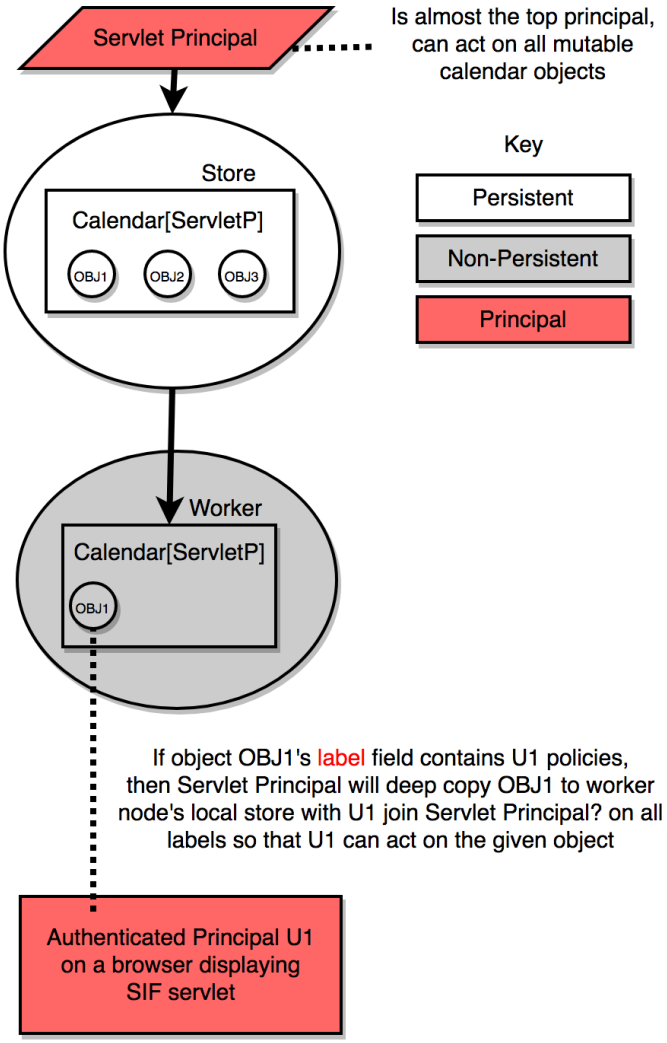
3.1 Previous Attempts

There have been numerous prior attempts to port the calendar from JIF to Fabric, all to no avail. I found these attempts in many different directories, such as some branches of Fabric and SIF. However, I found these examples to be a bit distracting since they were designed more as a direct port of the database version, rather than utilizing Fabric's capabilities for persistence and data storage that are integrated into the language. Instead of focusing too much on these, I decided to look at working Fabric examples and try to better understand how to utilize what was provided by the language.

3.2 Initial Design Ideas

I had many ideas regarding how to design the architecture for the calendar within Fabric, though most were scratched as I started to understand the language better. Initially, I thought I needed to write my own data structures to contain a user's events for their calendar, such as a HashMap. However, I learned later on that the TreeMap implemented in Fabric already did this well enough for me. One of the best resources for understanding Fabric was not the manual, but a paper titled "Fabric: A Platform for Secure Distributed Computation and Storage"[2]. I highly recommend that future users of Fabric read it.

I also initially worried about how one would copy and downgrade the various fields within events from one label to another, specifically when an event owner adds another user to the attendee's list. Because of this, I had some inspiration from the **FriendMap** example, thinking that I could in some sense "deep copy" event objects.



3.3 Ideas From other Packages

I think the best packages for inspiration are the following:

Package	Location	Useful Because
Travel	fabric/examples/travel	• Working Fabric application utilizing SIF • Has a login package similar to the userserv package used in the JIF calendar
AuthWiki	apl.cs.cornell.edu/git/authwiki	• Multiple users (user lists) • Working examples of servlet actions within atomic calls
Auth	fabric/tests/playground/auth	• Code which can extend the login package to include user related capabilities (this allows users to be created, verifying a username has not already registered)

I only discovered **AuthWiki** two days ago when Dr. Myers remembered it. Even though I could not get it to build and run due to information flow issues, even after finding the paths for missing dependencies, it has proven to be useful for rewriting some portions of the servlet actions for which the porting had been stuck. Also, because of this example's dependencies, the **Auth** package was discovered. If this package works, I think that it provides the most interesting addition to the calendar with the ability to create users dynamically instead of the hard-coding done in my and the prior implementation of the calendar.

3.4 Current Design Idea

Apart from moving away from the database to using Fabric storage nodes, after inspecting other working Fabric examples, it became apparent that a large amount of the SIF related code should be directly portable from the JIF version to Fabric version—assuming that amendments are made to how the information can flow in the newly distributed setting.

The current implementation follows the following format:

Calendar Package

- **CalendarServlet** - Main entrypoint for the web application
- **Calendar** - An abstract calendar principal, reflecting the broker in travel application
- **Config** - Configuration for the calendar
- **Create** - A FabIL script to initialize the calendar's state
- **Event** - The event class
- **FrontPageAction** - Front page of the web application, should require the login for authentication
- **User** - Object containing the user's information
- **Session sub package**
  - **CalendarSessionState** - Holds the state for the calendar of an authenticated session
  - **ChangeDisplayDateAction** - An authenticated action to change the display date
  - **CreateEvent** - Small wrapper action to set things up before invoking the CreateEditEvent action. In particular, this action creates a new Event, and gives that to the createEditEvent action
  - **CreateEditEvent** - A reusable Action for either creating, editing, or viewing events
  - **EditEventAction** - An authenticated action to edit an Event in the calendar
  - **FinishEditEventReceiver** - Receives notification that a calendar Event edit is complete
  - **FinishEditingEvent** - An event that occurs when an edit on a calendar Event is complete
  - **SelectDisplayUser** - An authenticated action to change the display user

  - **ShowCalendar** - An authenticated action to show the calendar
  - **ShowCalendarSessAction** An authenticated action to produce all the SIF elements available in the different views of the calendar

- **Utility sub package**
  - **DateUtil** - A utility to convert a string to a date, a date to a string, and to check that a string is a date
  - **Date** - A necessary utility for the calendar that Fabric did not port from JIF
  - **Declassifier** - Methods for downgrading various calendar objects
  - **UserComparator** - A comparator to check if two objects represent the same UserPrincipal
  - **Util** - Intended to become a deep-copy utility for Event objects if needed, and may require downgrading or endorsing to pass rights from one principal to another. A question regarding this involves: *how would you revoke rights to an event if you delete a user from the list of attendees?*

Principals Package

- **ServletPrincipal** - A principal representing the calendar root principal, and may act on all servlet-related mutable objects. Many of the classes either get instantiated with this label or are instantiated with this label's authority.
- **UserPrincipal** - A principal who may only act on objects for which a specific UserPrincipal has privilege.

Modified Login Package (Starred items come from **Auth** and cause the **Login** to be more similar to the **Userserv**)

- **AbstractAction** - The most basic action, with methods to create SIF nodes that contain tables with either a **banner** or a **sifbody** element. It also contains a method to return a request's default value if it is available.
- **AuthenticatedAction** - A class extending abstract action, which create actions that are protected by requiring a successful login.
- **AuthSessState** - A class which extends SIF's **SessionState**, associating the session state to an authenticated user.
- **CreateUserAction*** - An action which allows for the creation of a user.
- **HTMLAuthServlet** - A class which extends SIF's **HTMLServlet** class, such that the servlet is related to an authenticated user.
- **LoginAction** - An action for a user to log into the servlet.
- **LoginClosure** - Allow the session to act for an authenticated user.
- **LogoffAction** - An action for a user to log off the servlet.
- **SubmitLoginAction*** - A more comprehensive action for a user to log into the servlet.
- **SubmitNewUserAction*** - An action to submit the user to the servlet.

For the current design idea, I considered modeling the application architecture in a way similar to a combination of the **AuthWiki** and **Travel** examples. The state is initialized by the **Create.fil** class. Then, the main entry point to the application is through **CalendarServlet**, which initializes two possible actions, the **FrontPageAction** and the **LogoffAction**. It also should initialize the actual Calendar object with the ServletPrincipal (**Calendar.fab** and **ServletPrincipal.fab**). The **Calendar** needs to hold **CalendarSessionState**, so that it has the current user and the list of **Events** pertaining to that user's calendar. After this, the events should have the lists of Attendees and Viewers, and only allow these principals to view the fields within that **Event** object for which they have privilege. Then, I was hoping to tie all of this together with the actions that were found in the JIF calendar implementation, but I didn't manage this step. I also did not complete the instantiations of SIF elements yet.

3.5 Prototype: The Implementation Thus Far and What's Left

Create.fil: Like the **JIF-SIF Calendar**, **Travel**, and **AuthWiki**, the users are initially hardcoded. However, instead of a text file defining database tables and inserts, we use a FabIL script that calls the necessary methods to do this. As per the **AuthWiki** example, I created this hardcoding outside of the actual Fabric code, so that this is at least done by an "init-state" operation rather than leaving passwords in the actual application as the **Travel** example does. Then, like **AuthWiki**, I add the users to a list.

CalendarServlet.fab This is started but not completed.

Config.fab This probably maps the necessary items.

User.fab This is seems finished.

User.fab This should be done.

Date.fab: Beyond that, the Calendar needed a **Date** class, since Fabric did not have this. Most of the other classes could not be written until I had this because of them using Date objects. As such, this class has been completed, but has not yet been tested to see if the flow policies are too restrictive for the desired use cases.

DateUtil.fab: This should be done.

UserComparator.fab: This is borrowed from **Auth** with modifications to improve readability.

Login Package This is essentially done because it only needed a few small modifications. Besides that, an overhaul was done to the syntax to remove deprecated symbols and formatted to improve readability.

Principals Package This should be done.

Session Package This has been started but not completed. None of the SIF objects have been ported yet, especially considering some issues of being stuck with information flow compiler errors. Looking at **AuthWiki** helped get this package started, as I discovered the idea of creating a new label and checking if that should be allowed to flow to various stores that might learn something from accesses of objects on the stores.

Event.fab: This should be done.

The rest of the Event Package I am unclear as to why the JIF Calendar had extra **FinishEditEvent** classes, so right now some of these might be placeholders, or possibly should be moved to the session package.

bin/* This should be done. It contains shell scripts necessary for initializing the state, starting the nodes for the stores, workers, and web applications. Additional scripts were added for generating certificates which are automatically trusted and imported into the respective keystores prescribed by the calls to **make-keystore** within methods in **init-keys** differentiated by name. The only thing that is unclear at this point is what these specific nodes should be called, and if the application should require separate key stores per user added to the calendar.

build.xml This should be done. The only change the user should do is edit the **build.properties** in the **etc** directory to point to their path to **$FABRIC**

prototype, src-bck, src-bck2 These are playgrounds where I was trying different things.

3.6 Possible Extensions for Fabric Calendar Application

- Allowing a way to accept or decline an event and a way to leave an event
- Improved UI behavior when compared to the JIF version
- A demo which scales the application from one node to many
- Ordering of events by time
- An expanded view for days
- Allow creators of events to transfer event ownership to other users

4 Road Blocks

4.1 Environment

Throughout the semester, numerous issues were encountered at all levels of installation of the environment needed to run the JIF and Fabric applications. This ranged from compatibility issues with newer versions of the Java JDK, often pertaining to differences in the **.jar** files provided by Oracle from their prior distributions. Changes in installation steps had to be made to get SIF to work. There were also issues getting Tomcat and Jetty to work, so that the JIF web applications could be run. Luckily, the Fabric web applications do not require Tomcat or Jetty, as this is integrated by the Fabric nodes running the application. In addition, some of the methods needed by many of the examples, including the calendar, were no longer in the current distributions of JIF—or Fabric. Many build files had broken paths due to reorganization of the code bases that had occurred over the years, and digging was required to find the necessary packages. All of this, especially the fixes that were found, is documented on my Github in various markdown files.

4.2 Documentation and Syntax

Throughout the semester, there have been numerous points of confusion that have blocked progress. One of the biggest time sinks involved the lack of awareness concerning the fact that much of the code is compiler generated. Thankfully, Tom straightened out this confusion, keeping me from spending another month wasting hours scrutinizing this confusing code. Additionally, Fabric, JIF, and FabIL each have syntax that is difficult to comprehend and is not well documented. There are confusing intricacies that I did not understand even after reading the Fabric manual several times, which lead me to think that a more concrete walkthrough would be useful for future Fabric newcomers, which is why I created a web page for that purpose. Finally, across the numerous classes and packages in the various repositories associated with the project, much of the code either uses deprecated, confusing syntax or it uses workarounds that defeat the purpose of the language itself just to allow the to code compile. It was hard to look at the numerous implementations and decide which were actually using Fabric in a way that was correct and "safe," especially because bad examples littered the examples folder, even in the official 3.0.0 release. Some of the examples were broken and did not run, and some of them had no documentation.

Several examples of confusing syntax using the deprecated way of doing things are shown below:

Old Syntax	**New Syntax**	**Meaning**
{*!:}	{$\top \leftarrow$}	Top principal can write (has integrity privilege)
{*:}	{$\top \rightarrow$}	Top principal can read (has confidentiality privilege)
{p:;p!:;*lbl}	{$p \rightarrow$; $p \leftarrow$;*lbl}	Principal p confidentiality and integrity privilege joined with a pointer to the privileges given by label lbl

The ambiguous * made it extremely hard for me to follow what was happening in parts of the code, and the use of syntax such as {p:;p!:;*lbl} felt extremely illegible at times between the cluster of colons and semicolons. Updating all the repositories to remove this syntax, or at least throw library packages and examples that use this into a "fix me"

directory or the "testing/playground" would greatly help newcomers in comprehending the semantic meaning of the code.

As someone with limited experience with less-distributed and polished programming languages made for research purposes, or languages that cannot be easily searched online when one experiences difficulties in the process of writing said code, I was often stuck for hours at a time. These issues ranged from both confusion with the syntax to compiler errors that were actually issues with the compilers themselves. As the semester progressed, Tom Magrino became an invaluable resource and great help to me by guiding me away from traps into which I would frequently fall. With his help, I steered away from looking at compiler generated code, and did not spend as much time being stuck on bugs that weren't actually my fault as I had previously done. We discovered that Fabric does not have a good way of implementing final constant arrays, and found bugs in **fabc**, **jlc**, and even **Polyglot**. I was really happy to hear his thoughts regarding the direction that the APL code repositories should head—focusing on improving future development by implementing integration/unit tests, continuous integration, and improving comments, coding standards, and documentation.

5 Random Contributions

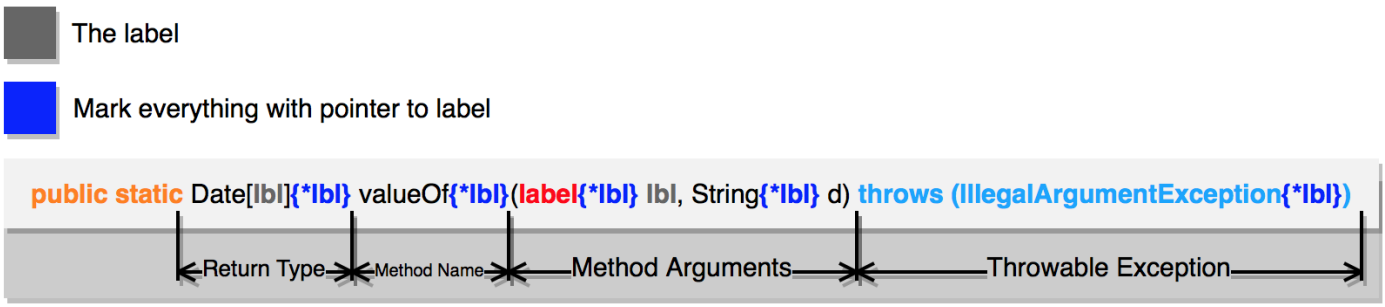
Due to the many issues encountered this semester regarding this project, different additional contributions were made to help others who might want to work with Fabric in the future. These include the following:

- Testing installation of Polyglot-JIF-Fabric stack, SIF, and Polyglot-JIF-Fabric IDE stack in different environments
  - Discovering changes that needed to be made in various build files
  - Discovering that Polyglot's call to Java's Toolprovider method no longer works with certain versions of Java
  - Discovering other compatibility issues with newer versions of the JDK:
    - May need **JCE.jar**
    - May need **tzdb.dat**, **local_policy.jar**, **us_export_policy.jar**
  - Writing manuals for the successful installations
- Discovering issues with the **preamble.js** file, regarding set up, key press recognition, and keybinding—and documenting a workaround.
- Helping to uncover and verify various compiler issues in **fabc**, **jlc**, and even **Polyglot**
- Creating bash scripts (which work on Linux) which generate signed certificates that are automatically trusted and imported into Fabric keystores. (This does not yet work on Mac due to **fabric/bin/import-cert** using **sed -i**)
- Improving documentation
  - Making a printable version of the label syntax table from the JIF manual, since the doxygen-generated web manual was not printing in a readable manner. This lead to discovering several errors in the JIF manual regarding label syntax.
  - Creating a web page intended to help newcomers to Fabric to have an easier time integrating into the environment and understanding the syntax and language capabilities through concrete examples.

6 Suggestions

6.1 Fabric Coding Style Standards

Throughout the many repositories created by the APL group, there are some very outdated examples and packages, as mentioned previously. Starting a project with syntax and coding styles that are standardized would be very useful for readability and comprehension for newcomers who are not as familiar with the group's practices. Additionally, because of how Fabric labels things, long methods with lots of parameters and joins within the labels start to get really difficult to visually parse.



What I found helpful was making one line methods—or multi-lined methods with arbitrary newlines and tabbing—into something similar to how elements in HTML code are often organized. That way, I could cluster the different pieces of the method together by type, as exemplified by the following:

```
public void invokeImpl{*lbl}
(
    label{*lbl} lbl,
    Request[ServletPrincipal]{*lbl} req,
    HTMLAuthServlet[ServletPrincipal]{*lbl} servlet_,
    AuthSessState{*lbl} state_,
    principal{*lbl} currentUser
)
throws  (ServletException{*lbl; req.session<-})
where   caller(req.session),
        lbl <= {req.session->},
        req.session equiv state_.sessionPrincipal,
        req.session actsfor currentUser
{
...
}
```

6.2 Future MEng Students and Fabric

Considering my experience this semester with the Calendar application, being that the scope of unfamiliar concepts was quite vast and that I needed to traverse significant territory primarily alone, I feel that I had unknowingly bitten off more than I could chew for a three credit hour project. Without better documentation and commented code, I feel that it might be best for the next batch of MEng students who work with Fabric to work more closely with a PHD student or another partner. Especially after talking to peers about their projects, I wished that I had worked with another student. I feel that the ability to share ideas and collaborate would have greatly assisted me, since usually different individuals struggle with grasping different concepts and can help each other better understand the concepts that make more sense to them than to their partner. Besides that, discussion and explanation of concepts helps to further understanding. Later in the semester, I tried to use friends who had graduated as rubberduckies[3] for my babblings, and whether or not they had any idea of what I was saying, I found that the talking helped me to understand what I was doing more. I truly feel that collaboration and interaction would greatly improve productivity and the overall depth of Fabric concepts that could be investigated.

7 Conclusion

In conclusion, though I struggled this semester to produce a deliverable of which I could feel proud, I am grateful for the opportunity that I was given to learn about Fabric and all the peripheral concepts of the projects surrounding it. Though I often felt out of my depth in the APL group meetings, especially in the beginning of the semester, I started to follow

the conversations a lot better as the semester progressed. I really appreciated sitting among the PHD students, since I felt that I learned a lot from them. Being someone who really measures my own accomplishments with tangible results, I have had various levels of exasperation at myself for not having a working prototype for the calendar application, but I believe that I have learned a lot from the endeavor which I will take with me into the future. I feel that I learned a lot of really cool conceptual ideas regarding information flow, ways to deal with persistence, and interesting implementations of certain capabilities that the Fabric language wanted to achieve. Another big takeaway for me is the overall project organization, which really ingrained into me the importance of code bases maintaining good documentation, commented code, and unit tests. With these necessary components, newcomers can assimilate and become productive far earlier in the process. It is great that Tom Magrino thinks the same way and is pushing to update the APL repositories to better follow this. Thank you, Dr. Myers and Tom, for letting me have this experience and for your guidance this semester on this project.

References

[1] Chong, Vikram, and Myers. "SIF: Enforcing Confidentiality and Integrity in Web Applications."

[2] Liu, Jed, Michael D. George, K. Vikram, Xin Qi, Lucas Waye, and Andrew C. Myers. "Fabric: A Platform for Secure Distributed Computation and Storage," *Proceedings of the ACM 2009 Symposium on Operating Systems Principles and Implementation (SOSP 2009)*. October 11-14, 2009 (Big Sky, Montana). pp. 321-334.

[3] *Rubber Duck Debugging*, available at `https://en.wikipedia.org/wiki/Rubber_duck_debugging`.

[4] *APL Cornell Wiki*, available at `https://apl.cs.cornell.edu/wiki/Main_Page`.