Tomcat Architecture

Our goal for this project is to recover the architecture of apache's tomcat. Tomcat is an open source implementation of Java Servlet, Server, Expression and WebSocket. To recover the architecture, we used a combination of Eclipse and Arch Studio. The source code for tomcat contains many components, each serving a specific role in this server application.

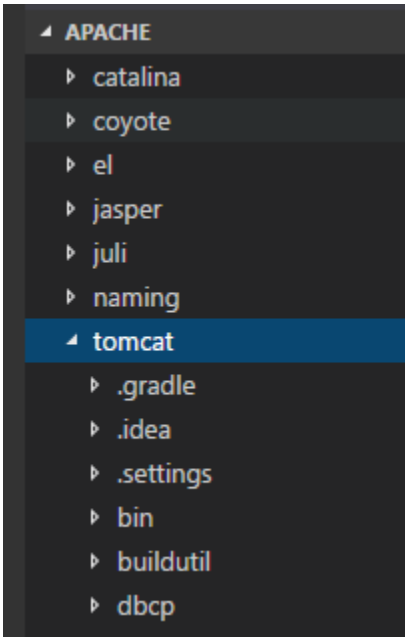


Figure 1

Building the Source Code

Our first task was to build the server application by downloading the source code from the apache website. Once we downloaded the code we then built it using apache ant (version 1.10.1). Ant is a tool for automating software build processes. For a successful build, we need the right version of the JDK and JRE, we used versions 1.8.0_144 and 1.8.0_151.

The build process is outlined at the following url:

https://tomcat.apache.org/tomcat-8.5-doc/BUILDING.txt

We followed these instructions and were able to successfully build the source code and run the server.

Figure 1 on the left shows the basic folder structure of the source-code. This is actually several folders deep, but it is what we use when we are compiling, and this is the folder structure we will be building the architecture from.

Figures 2 and 3 which are shown below display the build being successful.

```
cobertura-report:

test:
    [concat] Testsuites with skipped tests:
    [concat] TEST-org.apache.catalina.connector.TestRequest.NIO.txt
    [concat] TEST-org.apache.catalina.connector.TestRequest.NIO2.txt
    [concat] TEST-org.apache.catalina.core.TestDefaultInstanceManager.NIO.txt
    [concat] TEST-org.apache.catalina.core.TestDefaultInstanceManager.NIO2.txt
    [concat] TEST-org.apache.catalina.loader.TestWebappClassLoaderThreadLocalMemoryLeak.NIO.txt
    [concat] TEST-org.apache.catalina.loader.TestWebappClassLoaderThreadLocalMemoryLeak.NIO2.txt
    [concat] TEST-org.apache.catalina.nonblocking.TestNonBlockingAPI.NIO.txt
    [concat] TEST-org.apache.catalina.nonblocking.TestNonBlockingAPI.NIO2.txt
    [concat] TEST-org.apache.coyote.http11.upgrade.TestUpgradeInternalHandler.NIO.txt
    [concat] TEST-org.apache.el.parser.TestELParser.NIO.txt
    [concat] TEST-org.apache.el.parser.TestELParser.NIO2.txt
    [concat] TEST-org.apache.tomcat.jni.TestFile.NIO.txt
    [concat] TEST-org.apache.tomcat.jni.TestFile.NIO2.txt
    [concat] TEST-org.apache.tomcat.jni.TestSocketServer.NIO.txt
    [concat] TEST-org.apache.tomcat.jni.TestSocketServer.NIO2.txt
    [concat] TEST-org.apache.tomcat.jni.TestSocketServerAnyLocalAddress.NIO.txt
    [concat] TEST-org.apache.tomcat.jni.TestSocketServerAnyLocalAddress.NIO2.txt
    [concat] TEST-org.apache.tomcat.util.net.TestSsl.NIO.txt
    [concat] TEST-org.apache.tomcat.util.net.openssl.TestOpenSSLConf.NIO.txt
    [concat] TEST-org.apache.tomcat.util.net.openssl.TestOpenSSLConf.NIO2.txt
    [concat] TEST-org.apache.tomcat.websocket.TestConnectionLimit.NIO.txt
    [concat] TEST-org.apache.tomcat.websocket.TestConnectionLimit.NIO2.txt
    [concat] TEST-org.apache.tomcat.websocket.TestWsWebSocketContainerWithProxy.NIO.txt
    [concat] TEST-org.apache.tomcat.websocket.TestWsWebSocketContainerWithProxy.NIO2.txt
    [concat] TEST-org.apache.tomcat.websocket.pojo.TestEncodingDecoding.NIO.txt
    [concat] TEST-org.apache.tomcat.websocket.pojo.TestEncodingDecoding.NIO2.txt
    [concat] TEST-org.apache.tomcat.websocket.server.TestAsyncMessages.NIO.txt
    [concat] TEST-org.apache.tomcat.websocket.server.TestAsyncMessages.NIO2.txt
    [concat] TEST-org.apache.tomcat.websocket.server.TestWsRemoteEndpointImplServer.NIO.txt
    [concat] TEST-org.apache.tomcat.websocket.server.TestWsRemoteEndpointImplServer.NIO2.txt
    [concat] Testsuites with failed tests:

BUILD SUCCESSFUL
Total time: 49 minutes 20 seconds

C:\Program Files\apache-tomcat-9.0.1-src>ant
Buildfile: C:\Program Files\apache-tomcat-9.0.1-src\build.xml

download-compile:
```

Figure 2

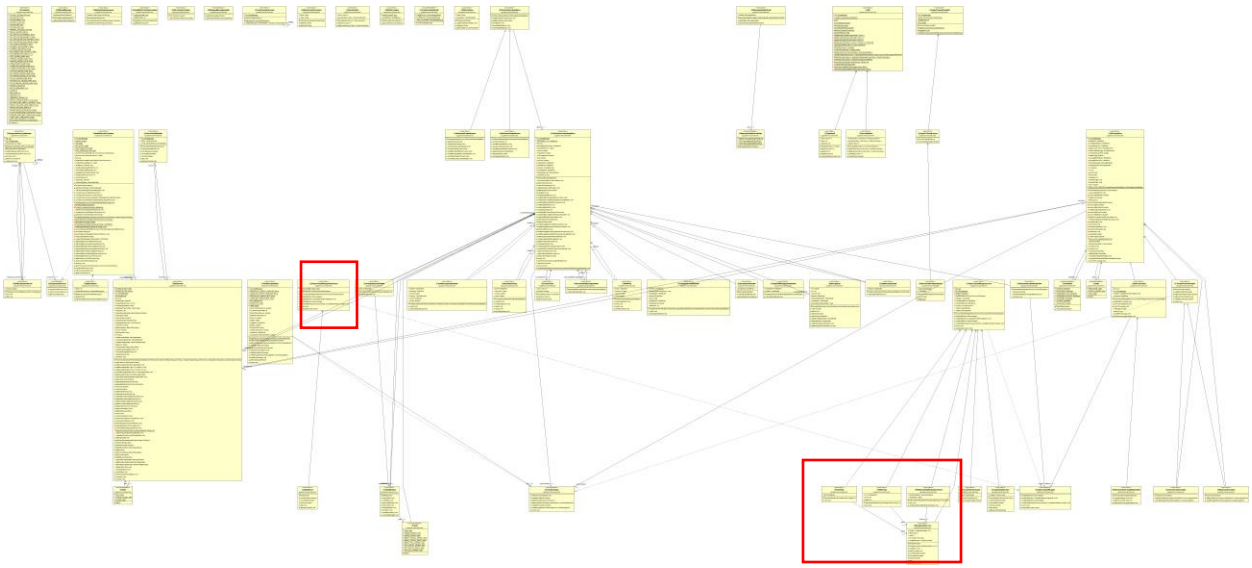
```
   [delete] Deleting directory C:\Program Files\apache-tomcat-9.0.1-src\output\build\temp
    [mkdir] Created dir: C:\Program Files\apache-tomcat-9.0.1-src\output\build\temp

compile-prepare:

download-validate:

validate:

compile:

build-manifests:

package:

build-docs:
     [xslt] Transforming into C:\Program Files\apache-tomcat-9.0.1-src\output\build\webapps\docs
     [xslt] Transforming into C:\Program Files\apache-tomcat-9.0.1-src\output\build\webapps\docs\appdev
     [xslt] Transforming into C:\Program Files\apache-tomcat-9.0.1-src\output\build\webapps\docs\funcspecs
     [xslt] Transforming into C:\Program Files\apache-tomcat-9.0.1-src\output\build\webapps\docs\config
     [xslt] Transforming into C:\Program Files\apache-tomcat-9.0.1-src\output\build\webapps\docs\architecture
     [xslt] Transforming into C:\Program Files\apache-tomcat-9.0.1-src\output\build\webapps\docs\tribes

build-tomcat-jdbc:
     [echo] Building Tomcat JDBC pool libraries

prepare:

download:

build-src:

build:

compile-webapp-examples:

deploy:

examples-sources:

BUILD SUCCESSFUL
Total time: 3 seconds

C:\Program Files\apache-tomcat-9.0.1-src>
```

Figure 3

Creating UML Diagrams

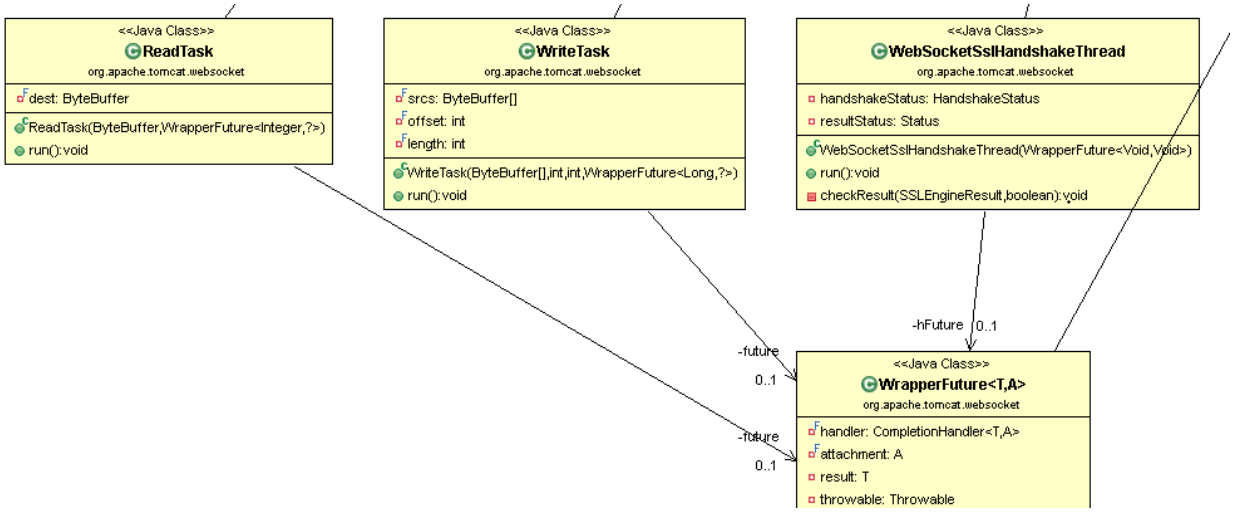
Our second task was to understand what task each component was responsible for, which include many sub-components. To gain a better understanding, we created a UML diagram of all of the classes in this project using ObjectAids in Eclipse. We recovered the UML diagram of each component. ObjectAid allowed us to create uml diagrams of each sub-component with relative ease as you can see below.

The WebSocket component

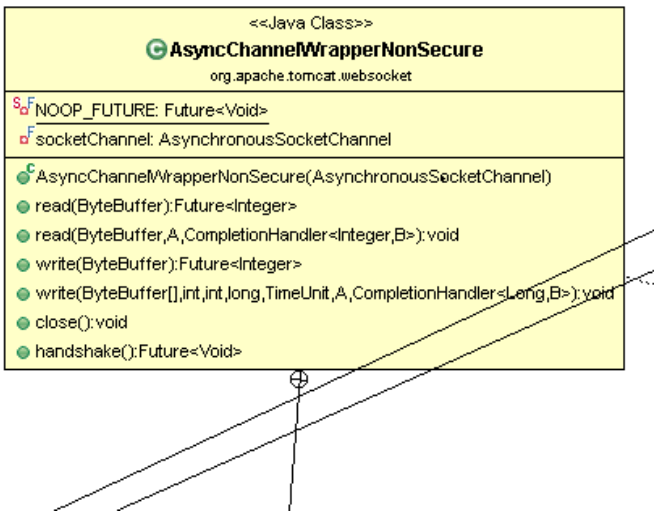


As you can see from the image, this is a very complicated piece of software. There are a lot of interconnections between the different classes. Generally, the lower the classes in this diagram are sub-

classes of the classes listed at the top, or are used in the classes listed near the top. Since the UML diagram is so detailed and zoomed out, it may be hard to read, but this is the bottom right close up of some of the classes, as marked with the red box.



AsyncChannelWrapperSecure is a main class that contains several sub-classes within it. The close up in this section of the images shows some of these sub-classes. As we can see, Read Task, WriteTask, and WebSocketSslHandshakeThread are all using WrapperFuture<T,A>. WrapperFuture is a sub-class of AsyncChannelWrapperSecure as well and it's used within the other sub-classes. The WrapperFuture class is responsible for handling handshake requests and failures. It is uually used at the beginning of the class setup and in the end or catch-block to report when a handshake succeeds or fails. This is obviously used for asynchronous communication, and is an important part of the web-socket component.



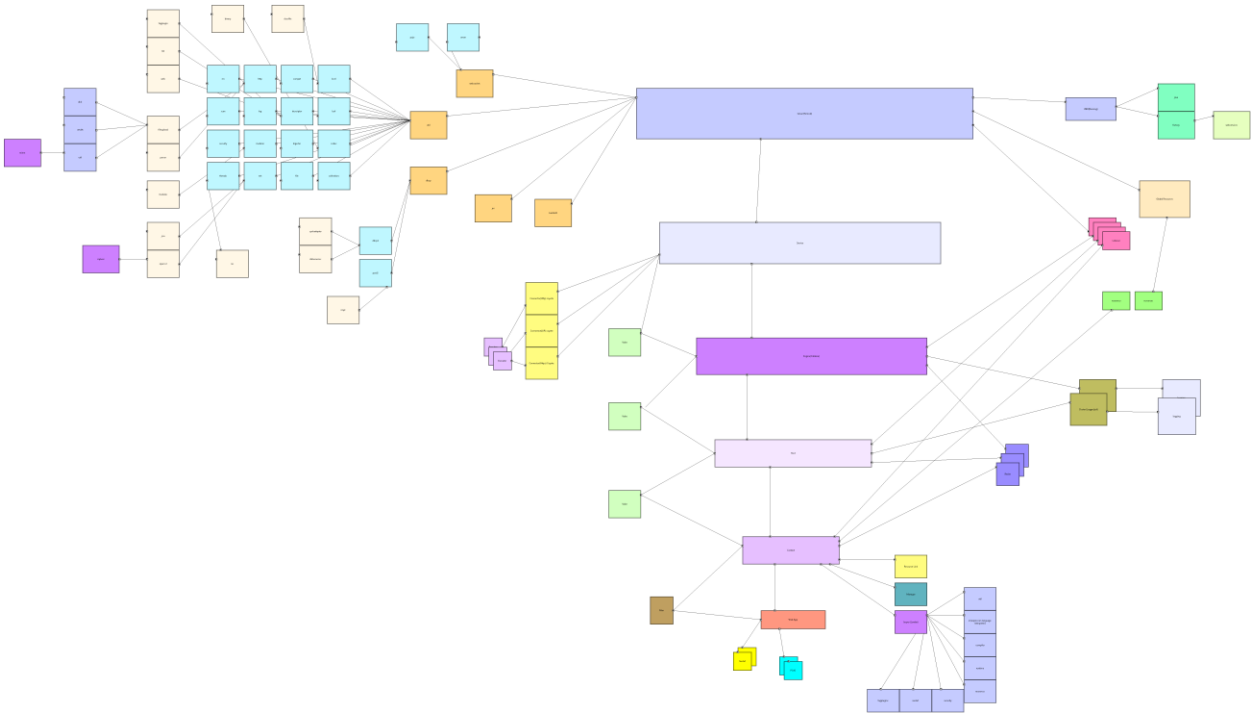
The *non-secure* AsyncChannelWrapper class is significantly less complicated because it does not require the SSL verification. It only contains a single child class that has basic methods for reading, writing, cancelling, etc. the handshake. There are many other parts to the WebSocket component, but this illustrates how complicated each component can be.

A similar layout is repeated for all the UML components that we have created. There are 43 UML diagrams just for the Tomcat Server code, one for each component, and each one contains a variety of classes like the ones we just discussed.

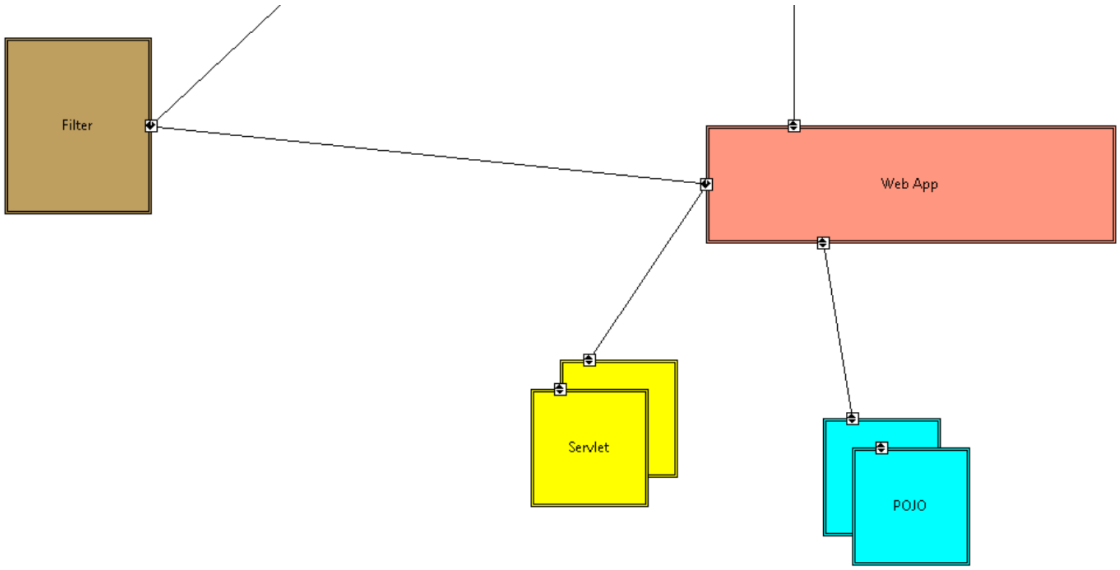
Once we ran through the uml diagrams and understood how each component interacted with each other, we decided to start working on the architecture of the software. This part was done using Arch Studio.

Architecture in Arch Studio

The architecture we recovered for apache tomcat was a hierarchal architecture pattern. This is a layered architecture with many sub-components.

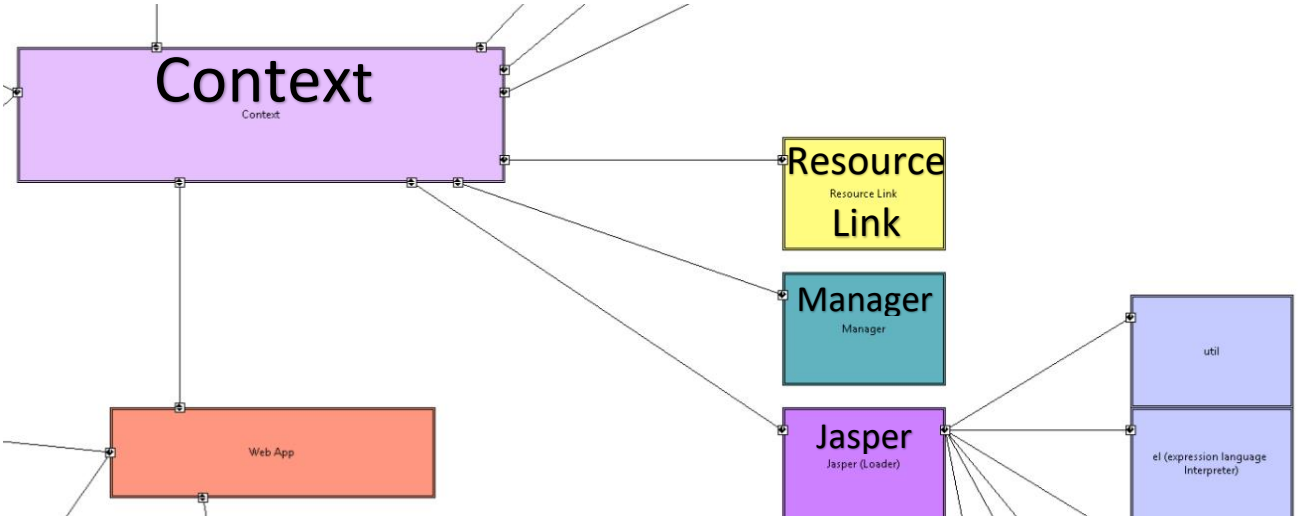


The labels may be hard to read, so I will talk about each component in more detail along with a screenshot from a close-up on the architecture.



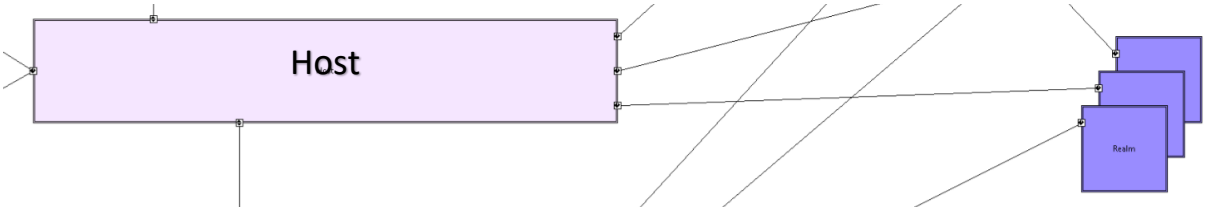
Web App

This component is responsible for the actual application that the server will be hosting. This uses a few components from the engine, the web app itself can be anything that is compatible, such as a java application running on the server.



Context

The context is basically something that servers as a parser which mainly uses Jasper to interpret expressions, compile, and manager resources.



Host

The host is responsible for the realm of the application. This is a virtual host, and supports things like auto-deployment and background processing delay. Auto-deployment includes the ability to live-deploy and application while tomcat is running. Realms serve as contexts for the host, such as database users, and roles.

Engine

The next item up from the host is the Engine component which is using Catalina, which handles things like logging and pumps data through to the service component.

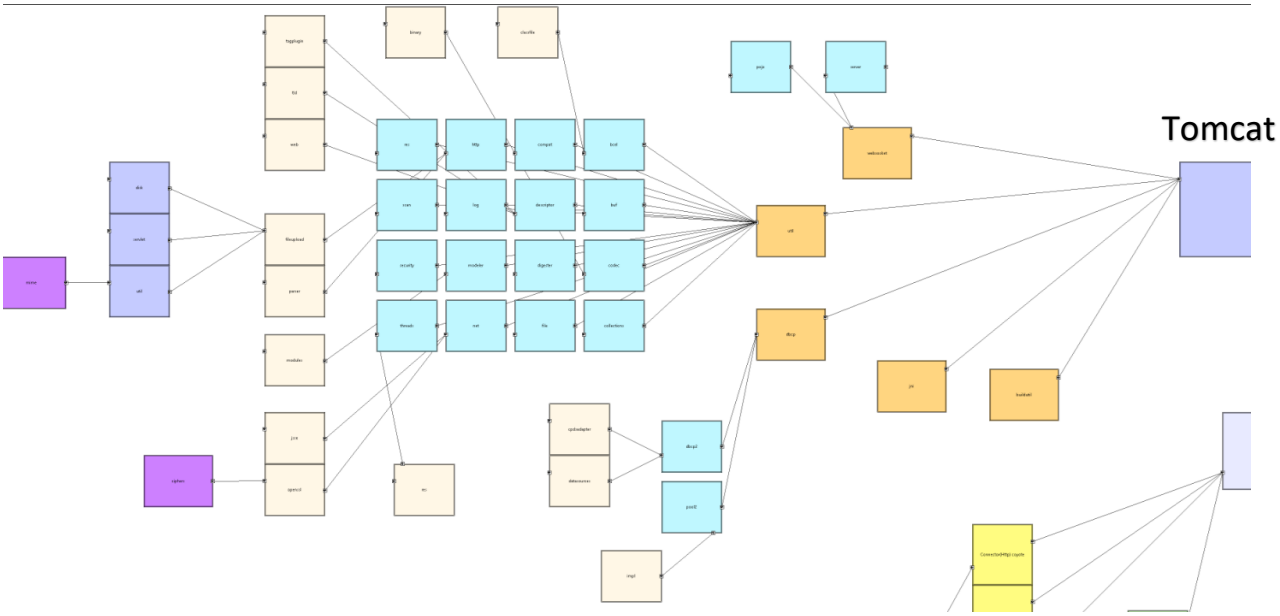
Service

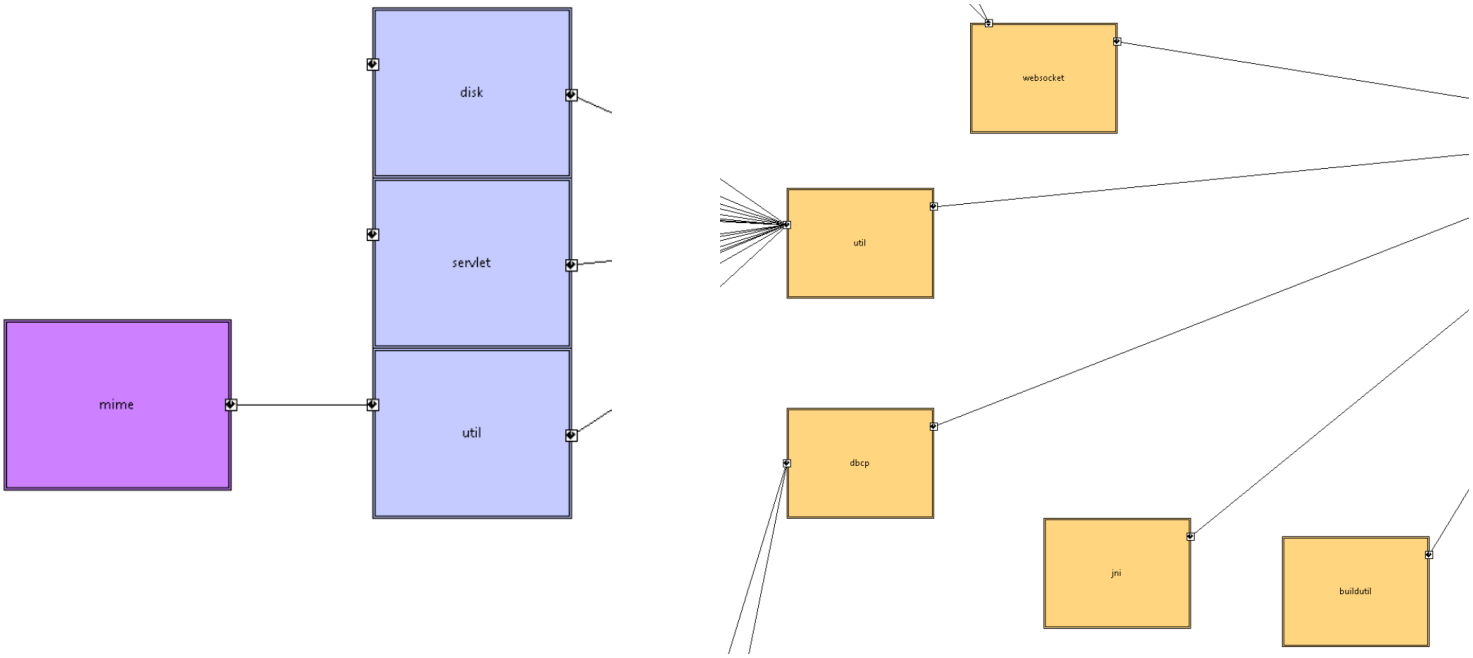
The service layer serves as an abstraction layer between tomcat and the Catalina engine.

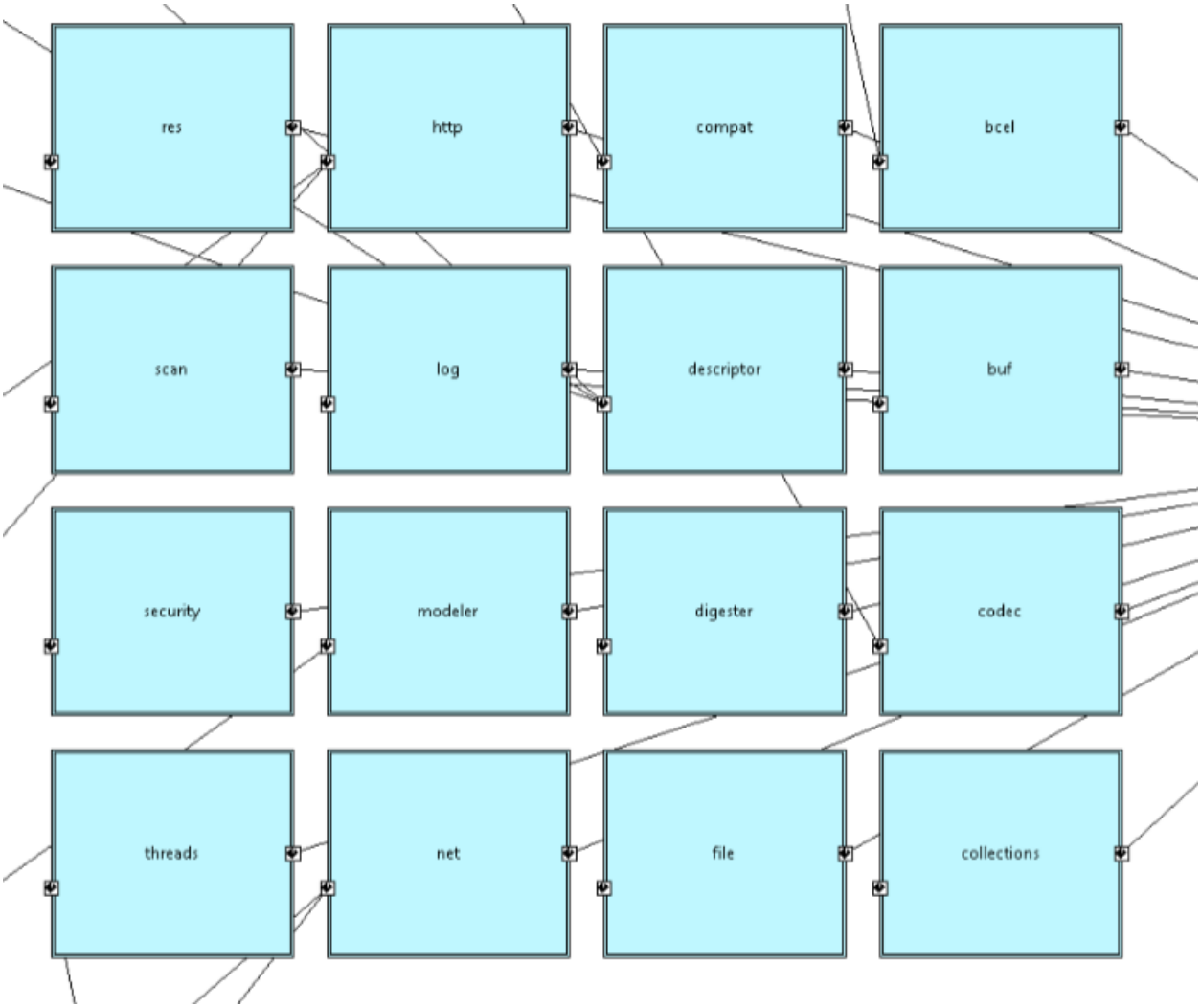
Server (Tomcat)

This is where the majority of the application and processing occurs. The websocket, utils etc are all here.

You can find all of the hi-resolution screenshots at the git repo we are keeping the project in:
https://github.com/DavidSaiyan/SoftArch


Tomcat


disk
servlet
util
mime
websocket
util
dbcp
jni
buildutil


res
http
compat
bcel
scan
log
descriptor
buf
security
modeler
digester
codec
threads
net
file
collections