Project ImpressiveACDC

Team: *Seekers*

Team Member:

- Haomo Tang
- Yiqing Xu
- Dong Yan
- Tianyu Zhao

Requirement

This visualization is designed as an improvement over the original dependency view of ACDC to further process the recovery result and incorporate more information in order to arrive at a meaningful representation of the system that has architectural implications visually accessible to human eyes.

The following features were specified in the proposal

1. Display component sized proportional to its complexity
2. Leave out insignificant component when there are too many components
3. Color dependency arrow according to strength of dependency
4. Visually group together closely related components
5. Click to on component to see its dependencies
6. Zoom in to illustrate code object within a component
7. Display connection between code objects

Feature #2 was ruled out for the correctness of the visualization. Feature #3 was changed with force simulation in the design. Feature #7 was overthrown for the purpose of abstraction. All the rest of the features was successfully implemented, together with some additional features that are introduced bellow.

General Design

ImpressiveACDC incorporates the results of ACDC, RELAX, and UCC, and displays them in three views in the form of web pages. The system has two parts: the preprocessing module and the visualization module.

The system is designed in a batch sequential style where first the preprocessing module process the results of RELAX, ACDC, and UCC, outputs to temporary files and then the visualization module takes the temporary files as input for display.

The preprocessing modules are made up of python scripts that applies simple analysis on the recovery results, such as counting dependencies amount clusters, counting classes within a cluster and deciding cluster labels. These analysis results together with the original recovery results are converted to JSON format and adapted so that they are ready to be read for visualization.

The visualization module is made up of three different views of the system that addresses three concerns.

Cluster dependency view
This view displays the dependency among components recovered by ACDC in a force-directed graph, with the goal of clearly depicting the configuration of components in the system by spatially grouping the components into communities. Dependency strength is calculated as the number of class dependencies between components, which is simulated by the forces between components so that components with stronger dependencies are closer together and connected components can be naturally separated. Size of each cluster is proportional to the number of compilation units within the cluster so that the clusters with more classes appear more significant in the image.

The idea behind this view is to sketch the system’s main characteristics and abstract away some of the details to give the viewer a clear impression of the system. Smaller, less connected clusters are negligible in size and located in the corner in this view while larger, well connected clusters may form a large community in the middle of the image. This way the most important messages will stand out and leave an impression.

There is a natural physical comparison for this view. Imagine the overall system to be recovered as a universe made up of galaxies. Each galaxy, that is, community of clusters, is made up of several star system grouped together through gravitational attraction. Each star system, or cluster, is made up of stars that are code objects in this simile. In the visualization, users can actually drag the star systems to feel the physical forces pulling in each directions.

This explains the logic behind this view: the classes are laid out as a result of gravitational forces simulation. It also justify using class dependency count as dependency strength between clusters: the gravitational forces between star systems are only the result of gravitational force between each pairs of stars.

Class bubble chart
This view displays the classes or compilation units inside a cluster with a bubble chart. The size of each bubble is proportional to the SLOC of the corresponding class obtained from UCC and the color of that bubble is consistent with the classification result of RELAX. The dependencies among classes are intentionally omitted for abstraction.

This view is intended to indicate the functionality of a component by revealing some of the details of that cluster. Sizing each class by its SLOC highlights the principal classes within each cluster where most code is devoted to. This way only the names of classes large enough is able to stand out. Each class is colored with its RELAX label so that the composition of concern of this cluster is made clear to the audience.

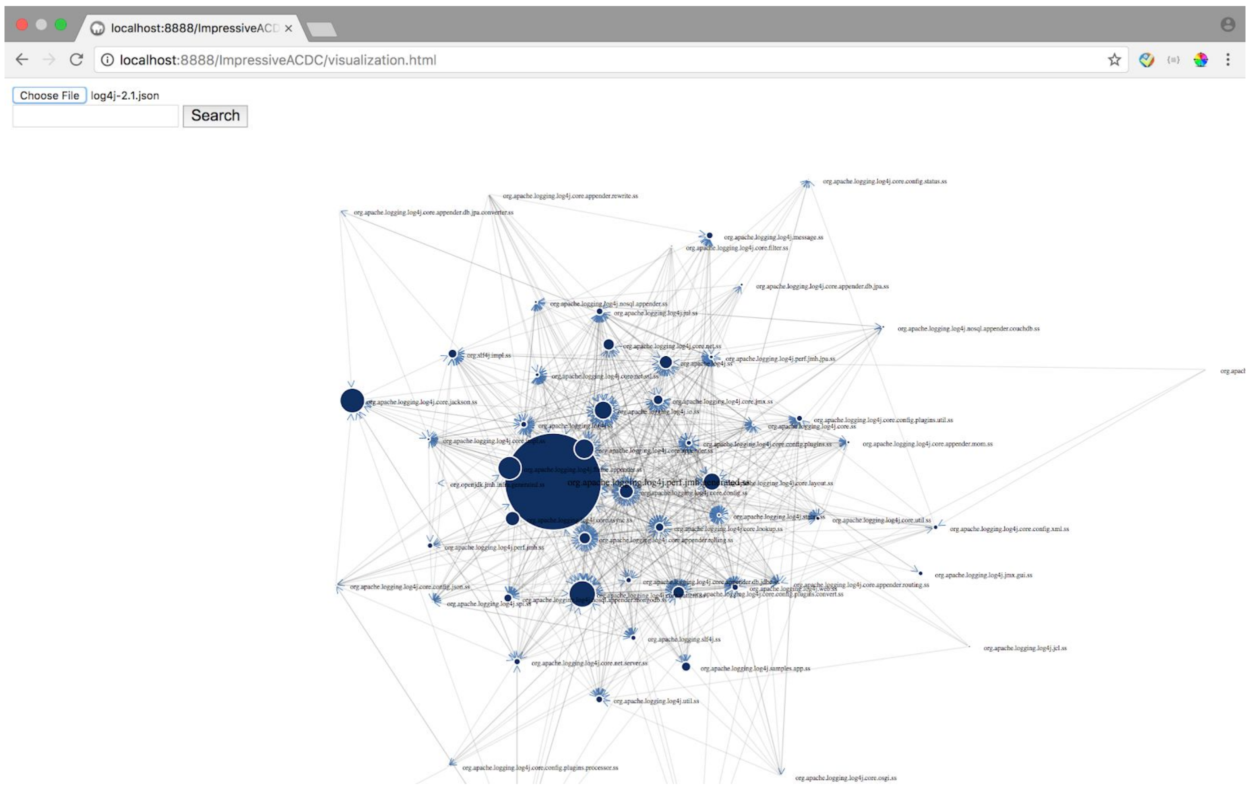
RELAX radar chart
This view arranges the components in a radar chart according to its composition of classes under each RELAX label. Each class is located according to its coordinate on its RELAX label counts and the color of the cluster is a majority vote of all the classes.

This view is intended as a relatively consistent view across different versions of the same system since we assume the RELAX coordinate of a cluster should not change much across different versions. Clusters of different versions of the same system should have roughly the same RELAX coordinate and thus be located around the same spot on the image. This makes it is useful for observing changes between different versions. The class count of the cluster and the dependencies are presented in this view for observations.

Specification

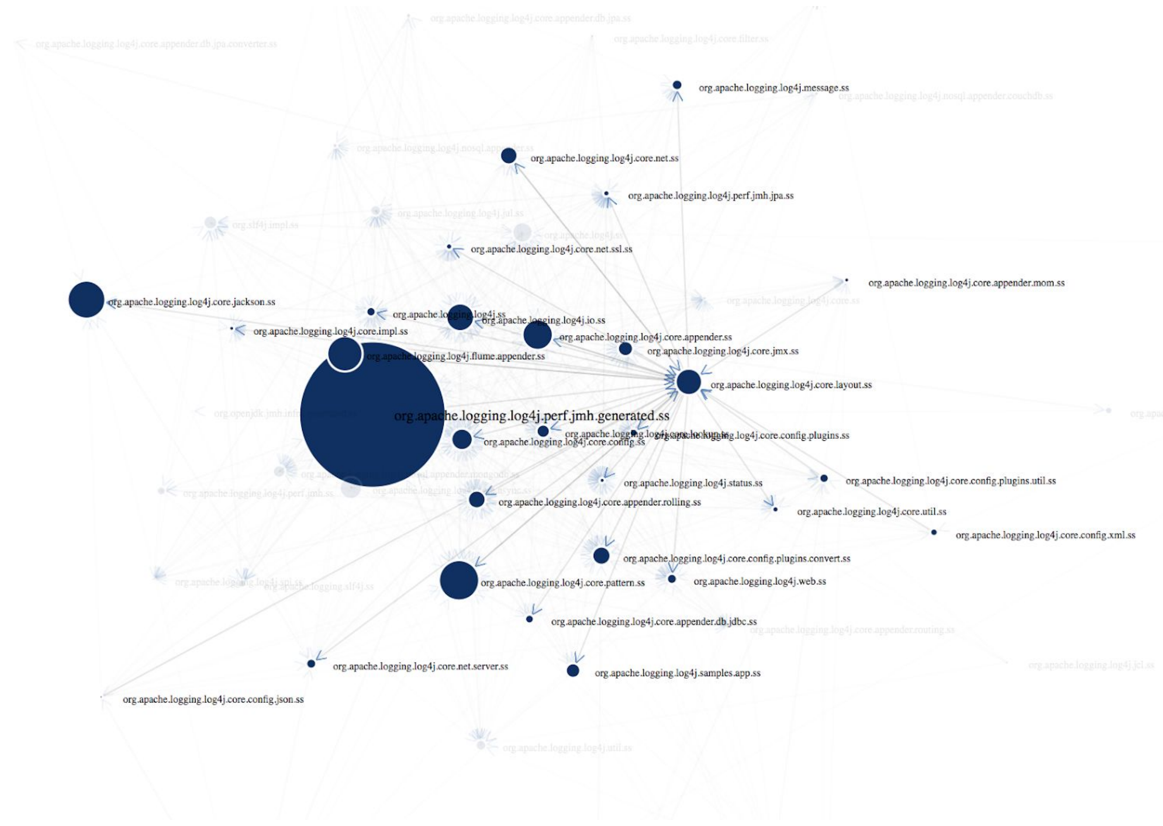
The design of our ImpressiveACDC visualization is clearly followed in its implementation. We allow user to select a json file which includes nodes and links information as input and generate related cluster dependency view as well as consistent view. User is also able to inspect a cluster to see more detailed information by selecting one node.

The cluster dependency view
The cluster dependency view can be found in visualization.html. To see the dependency view, user need to select a input file by clicking “choose file” button. The json file should be located in input folder and be generated by *main.py* in required format. After selecting input file, the dependency view will be generated automatically as shown in Figure a.
Figure a.


localhost:8888/ImpressiveACDC
localhost:8888/ImpressiveACDC/visualization.html
Choose File
log4j-2.1.json
Search
org.apache.logging.log4j.core.appender.rewrite.ss
org.apache.logging.log4j.core.config.status.ss
org.apache.logging.log4j.core.appender.db.jpa.converter.ss
org.apache.logging.log4j.message.ss
org.apache.logging.log4j.core.filter.ss
org.apache.logging.log4j.core.appender.db.jpa.ss
org.apache.logging.log4j.nosql.appender.coachdb.ss
org.apache.logging.log4j.nosql.appender.ss
org.apache.logging.log4j.jmx.ss
org.slf4j.impl.ss
org.apache.logging.log4j.core.net.ss
org.apache.logging.log4j.perf.jmh.jpa.ss
org.apache.logging.log4j.core.jackson.ss
org.apache.logging.log4j.core.jmx.ss
org.apache.logging.log4j.core.config.plugins.util.ss
org.apache.logging.log4j.core.appender.mom.ss
org.apache.logging.log4j.perf.jmh.generated.ss
org.apache.logging.log4j.core.layout.ss
org.apache.logging.log4j.core.util.ss
org.apache.logging.log4j.core.config.xml.ss
org.apache.logging.log4j.core.lookup.ss
org.apache.logging.log4j.perf.jmh.ss
org.apache.logging.log4j.jmx.gui.ss
org.apache.logging.log4j.core.config.json.ss
org.apache.logging.log4j.spi.ss
org.apache.logging.log4j.core.appender.routing.ss
org.apache.logging.log4j.core.config.plugins.convert.ss
org.apache.logging.log4j.jcl.ss
org.apache.logging.slf4j.ss
org.apache.logging.log4j.core.net.server.ss
org.apache.logging.log4j.samples.app.ss
org.apache.logging.log4j.util.ss
org.apache.logging.log4j.core.config.plugins.processor.ss
org.apache.logging.log4j.core.osgi.ss

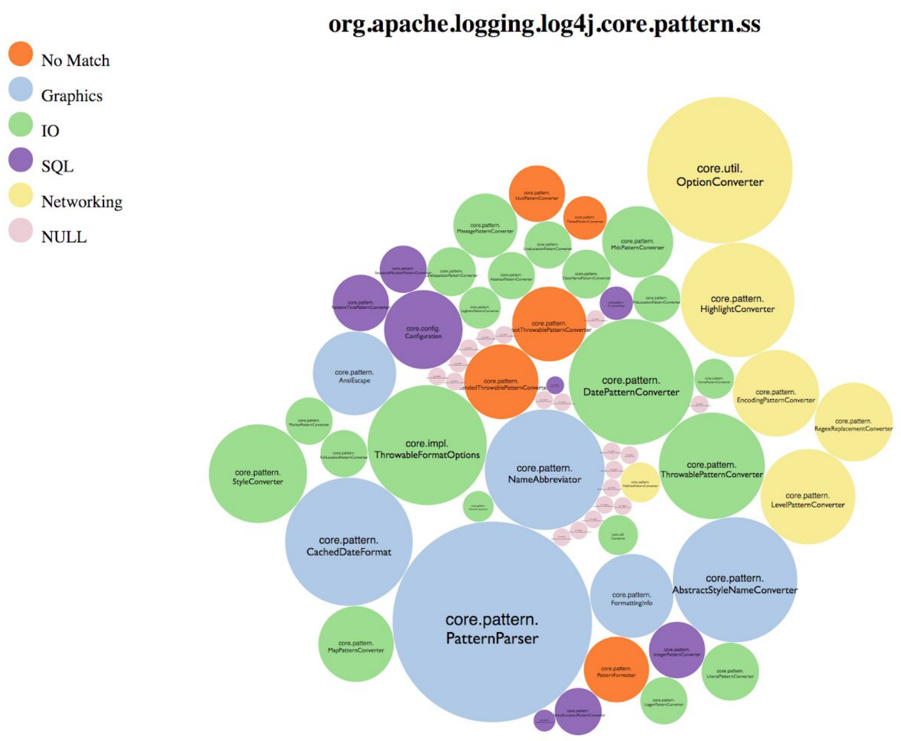
The cluster dependency view is implemented in a force-directed graph by D3.js. Each of the dark bule nodes represents a different cluster with the cluster name as label. The cluster with more compile units will be shown in larger node. The arrow between two nodes represents the relationship of these two clusters (If A depends on B, then the arrow points from A to B, like so: A -> B). Those nodes staying in the center share stronger dependencies, and those nodes away from the center are less dependent on other clusters or less depended by other clusters.
User are able to see more clear relationships about specific node by clicking on the it. Only related nodes and links will be highlight, and other part of the view will be grayed out as shown in Figure b.

Figure b


org.apache.logging.log4j.message.ss
org.apache.logging.log4j.core.net.ss
org.apache.logging.log4j.perf.jmh.jpa.ss
org.apache.logging.log4j.core.net.ssl.ss
org.apache.logging.log4j.core.appender.mom.ss
org.apache.logging.log4j.core.jackson.ss
org.apache.logging.log4j.core.impl.ss
org.apache.logging.log4j.ss
org.apache.logging.log4j.io.ss
org.apache.logging.log4j.core.appender.ss
org.apache.logging.log4j.flume.appender.ss
org.apache.logging.log4j.core.jmx.ss
org.apache.logging.log4j.core.layout.ss
org.apache.logging.log4j.perf.jmh.generated.ss
org.apache.logging.log4j.core.config.ss
org.apache.logging.log4j.core.config.plugins.ss
org.apache.logging.log4j.status.ss
org.apache.logging.log4j.core.config.plugins.util.ss
org.apache.logging.log4j.core.appender.rolling.ss
org.apache.logging.log4j.core.util.ss
org.apache.logging.log4j.core.config.xml.ss
org.apache.logging.log4j.core.config.plugins.convert.ss
org.apache.logging.log4j.core.pattern.ss
org.apache.logging.log4j.web.ss
org.apache.logging.log4j.core.appender.db.jdbc.ss
org.apache.logging.log4j.core.net.server.ss
org.apache.logging.log4j.samples.app.ss
org.apache.logging.log4j.core.config.json.ss

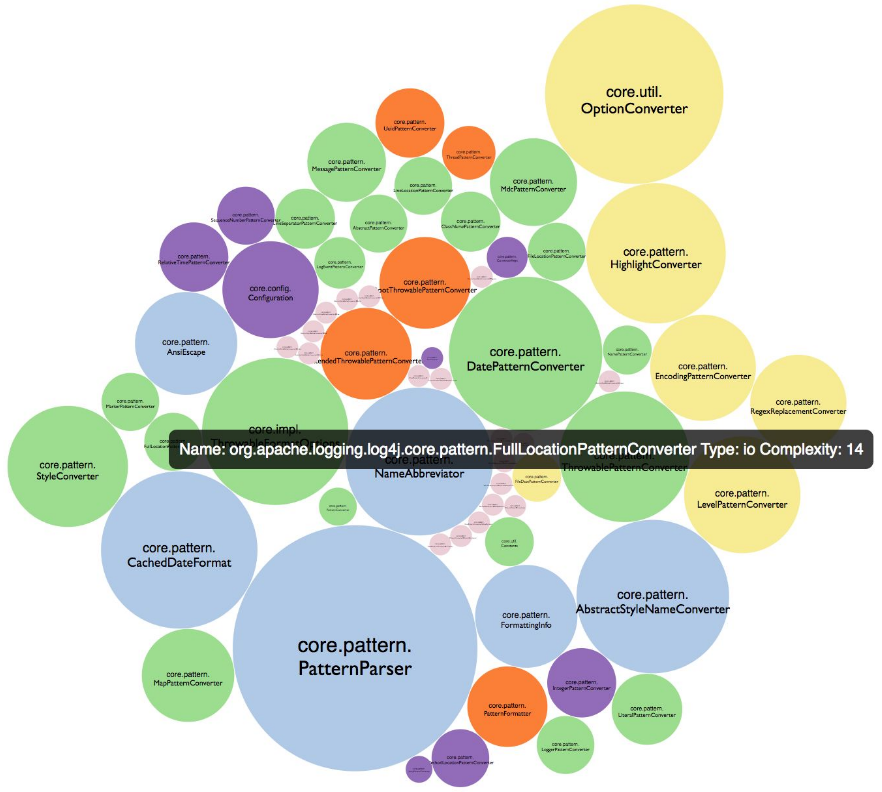
User can zoom in the view by double-click on the page.
User are also able to drag one of the nodes and move around to see it's effects on the whole system architecture.
To find specific node, user can type the cluster name in search field and select the node in autocomplete dropdown list. The selected node will be highlight and other part will be grayed out for five seconds.

Class bubble chart
The class bubble chart is implemented in *classVisualization.html*. User can go the specific cluster class view by clicking the name of the node in dependency view. To generate the class view successfully, user need to make sure generating corresponding "-class.json" file from preprocessing steps, and put the file in "input" folder as well.
After the page loading, detailed class view will be generated automatically as shown in Figure c.
Figure c.



The cluster class view is implemented in bubble chart graph by D3.js with the header showing the cluster name. There are six types of classes in total with different colors. The size of circle means complexity. More complex class will have larger size. User can have a general idea about the structure of this cluster by looking at the size and color of all nodes. All classes with NULL type (bubble in pink) have hardcoded complexity equals two to reduce the visual effect. User are able to hover over the circle to see clear information about selected class with type and complexity as shown in Figure d.

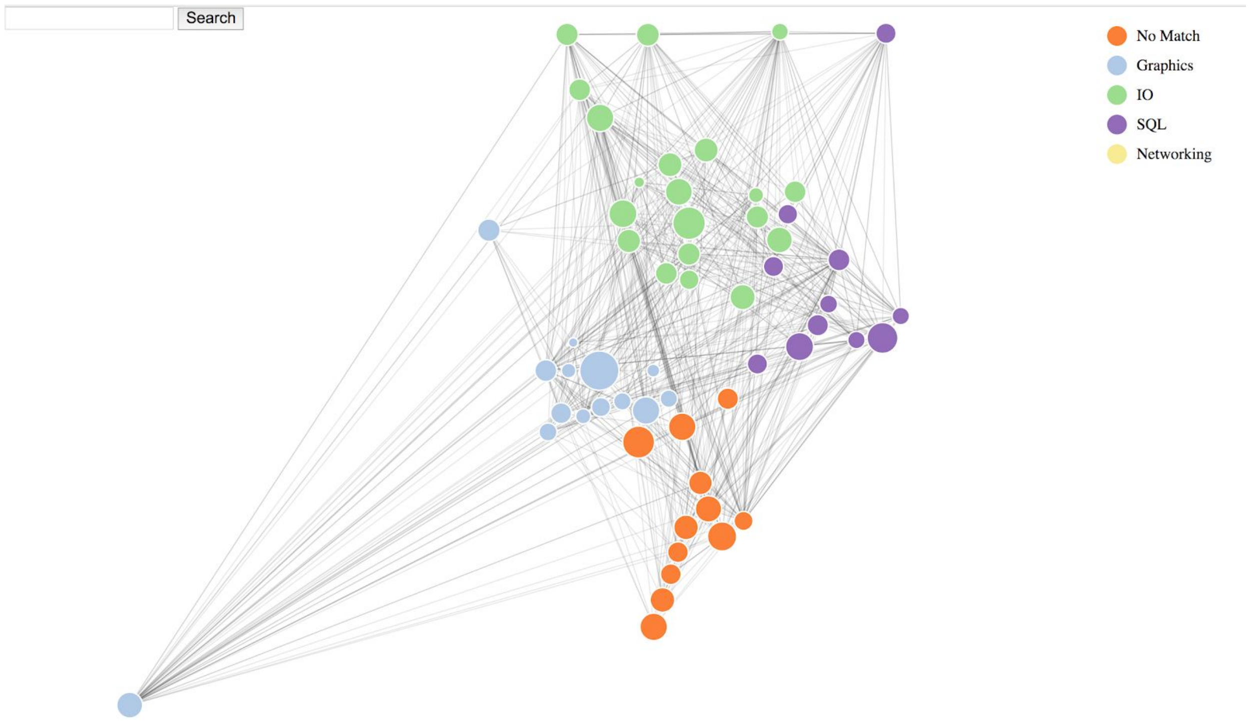
Figure d.


Name: org.apache.logging.log4j.core.pattern.FullLocationPatternConverter Type: io Complexity: 14
core.util. OptionConverter
core.pattern. HighlightConverter
core.pattern. DatePatternConverter
core.pattern. EncodingPatternConverter
core.pattern. RegexReplacementConverter
core.pattern. ThrowablePatternConverter
core.pattern. LevelPatternConverter
core.pattern. AbstractStyleNameConverter
core.pattern. FormattingInfo
core.pattern. PatternParser
core.pattern. NameAbbreviator
core.impl. ThrowableFormatOptions
core.pattern. AnsiEscape
core.config. Configuration
core.pattern. StyleConverter
core.pattern. CachedDateFormat
core.pattern. MapPatternConverter
core.pattern. MdcPatternConverter
core.pattern. MessagePatternConverter
core.pattern. UuidPatternConverter
core.pattern. PatternFormatter
core.pattern. IntegerPatternConverter
core.pattern. LiteralPatternConverter
core.pattern. LoggerPatternConverter

RELAX radar char

After selecting input json file in cluster dependency view, a button “go to consistent view” will show up. By clicking on the button, a new page with consistent view will show up as shown in Figure e. To generate the consistent view successfully, user need to make sure generating corresponding “-consistent.json” file from _____previous python step, and put the file in input folder as well.

Figure e



Each node in this view represents a different cluster, and the color shows its main type.

When user click on one node, all related nodes will labels will be highlight, and other part will be grayed out.

User are able to type the cluster name in search field and select the node in autocomplete dropdown list to find specific cluster. The selected cluster will be highlight with label and all related nodes will also be highlight.

User can zoom in the view by double-click on the page.

User are able to compare two different versions of a system by looking at this radar chart since all nodes positions are calculated from recovery method, and will stay in similar position if the system architecture is unchanged.