WARP Reliability Analysis Technical Specification

Authors: Jackson Grant, Jackie Mills, Matt Boenish, Andy Luo
Created On: 3/29/2023
Group Meeting Times: 3:30-4:30 on Mondays at the Library, 12:30-1:30 on Wednesdays in lab

1. Introduction:
    a. Overview: We need to complete the ReliabilityAnalysis and ReliabilityVisualization classes in the WARP code base so that they produce the *.ra output files. These files are a visualization of an end-to-end reliability analysis of a WARP input file.
        i. **Program Requirements**:
            1. ReliabilityVisualization must create a file visualization of a ReliabilityAnalysis for Warp to match the example *.ra files included with the program specification.
            2. ReliabilityAnalysis must perform a complete analysis of the given input flows to meet the reliability requirements specified by the WorkLoad.
2. Solution:
    a. Design:
        i. Creating *.ra files will interact with input files that contain information about the flows being used. These input files are what the reliability analysis is going to be based off of. The design and solution will not alter these files, but rather, will create an analysis of the reliability of the flow as called for by the Warp main program.
        ii. **ReliabilityVisualization**: The design will incorporate steps to create the necessary layout for the *.ra files: a header, parameters used, nodes in the flow, and the math in each step of computing the Reliability Analysis.
            1. The class will need to override the methods visualization(), createVisualizationData(), and createHeader() at minimum, as these are called by VisualizationImplementation in the process of building a visualization through the Warp main. The method createFile() may need to be overridden as well, but is currently implemented in VisualizationObject, the superclass. The method createFooter() may not need to be fully implemented, since the example *.ra output does not have a footer.
        iii. **ReliabilityAnalysis**: The design will create a table that computes the reliability of each node in each time slot used in the analysis. This class will include the math to commute that reliability, the code that will create the table when created with the constructor taking a Program as a parameter, and will access the necessary instruction parameters given by WarpDSL.
            1. The current computational functionality of this class and its interaction with WorkLoad needs to be revised to account for

numFaults values other than the default, due to an error in the HW5 code.

2. The class needs to complete its entire analysis after being created using the constructor taking a Program as a parameter, and store it as an attribute(s).
3. The reliability analysis will be accessible through the getter methods getReliabilities() and verifyReliabilities(). The former will be used within the reliability suite, and the latter is used by WarpSystem.

b. Test Framework:
   i. Unit testing will be used to verify the functionality of the methods developed as part of the solution, and will be included with the verification of the *.ra output file result as the complete test suite.
   ii. **Tests**:
      1. Test if the correct math is being used in each step within Reliability Analysis from src->sink node.
      2. Test if *.ra file has the correct header.
      3. Test if *.ra file has the correct parameters listed.
      4. Test if *.ra file has the correct data in the schedule
      5. Implement unit tests for every method in ReliabilityAnalysis
      6. Implement unit tests for every method in ReliabilityVisualization

3. Timeline:
   a. Sprint 1 (Due April 10):
      i. Fill out this document -Group
      ii. Put high-level plans into ReadMe -Jackie
      iii. Create a full sequence diagram for when Warp is run with the -ra option -Matt, Jackson
      iv. Create plans and assign work -Group
   b. Sprint 2 (Due April 21):
      i. Update ReadMe to reflect updated plans and document who did what -Jackie
      ii. Fully code ReliabilityVisualization
         1. Implement createHeader() -Jackson
         2. Implement createVisualizationData() -Andy
         3. Implement visualization() -Andy
      iii. JavaDoc ReliabilityVisualization -Group, Jackson Review
      iv. Update UML diagrams -Matt
      v. Make tests for ReliabilityVisualization - Jackie, Matt
         1. Unit tests for each method -Matt
         2. Test to verify output formatting -Matt
         3. Test to check math -Jackie
      vi. Plan Sprint 3 -Group
   c. Sprint 3 (Due May 5):
      i. Update ReadMe - Jackie

ii. Fully code ReliabilityAnalysis
    1. Bug fix (2.a.iii.1) -Jackson
    2. Run reliability analysis from constructor -Jackson, Andy
    3. Implement getReliabilities() -Andy
    4. Implement verifyReliabilities() -Andy

iii. JavaDoc ReliabilityAnalysis -Group, Jackson Review

iv. Update UML diagrams -Matt

v. Make tests for ReliabilityAnalysis -Jackie, Matt
    1. Unit tests for each method -Jackie, Matt
    2. Test to verify output data -Jackie, Matt

3/29 Meeting Notes:
PROGRAM SPECIFICATION: complete the ReliabilityAnalysis and ReliabilityVisualization classes such that they create the *.ra files available from ICON and evaluate the end-to-end reliability of WARP flows, as requested in the WarpTester main program. Do this even for the k-Fault model version of the code, using the -e and -m parameters if specified. Be sure to complete all methods declared in the class.

- To retrieve the instructions in each time slot, which is required to know which nodes probabilities to update and when, use ArrayList<InstructionParameters> getInstructionParameters(String instruction) in the WarpDSL class to parse the instruction string contained in a ProgramSchedule table entry. The class InstructionParameters() (defined in WarpDSL) allows you to access the parameters.
- Let M represent the Minimum Packet Reception Rate on an edge in a flow. That is, M = minPacketReceptionRate.The end-to-end reliability for each flow, flow:src->sink, is computed iteratively as follows:
    - The flow:src node has an initial probability of 1.0 when it is released. All other initial probabilities are 0.0. (That is, the rest of the nodes in the flow have an initial probability value of 0.0.)
    - Each src->sink pair probability is computed as newSinkNodeState = (1-M)*prevSnkNodeState + M*prevSrcNodeState This value represents the probability that the message has been received by the node SinkNode. Thus, the NewSinkNodeState probability will increase each time a push or pull is executed with SinkNode as a listener.

Main Task: Let ReliabilityAnalysis and ReliabilityVisualization perform reliability analysis and create an output file visualization with the *.ra suffix

Entry Point: Warp.java

Input: Input files are flows
Output: *.ra file and a reliability analysis

Control Flow:

File → Warp → VisualizationFactory → VisualizationImplementation
—

Make a function that:

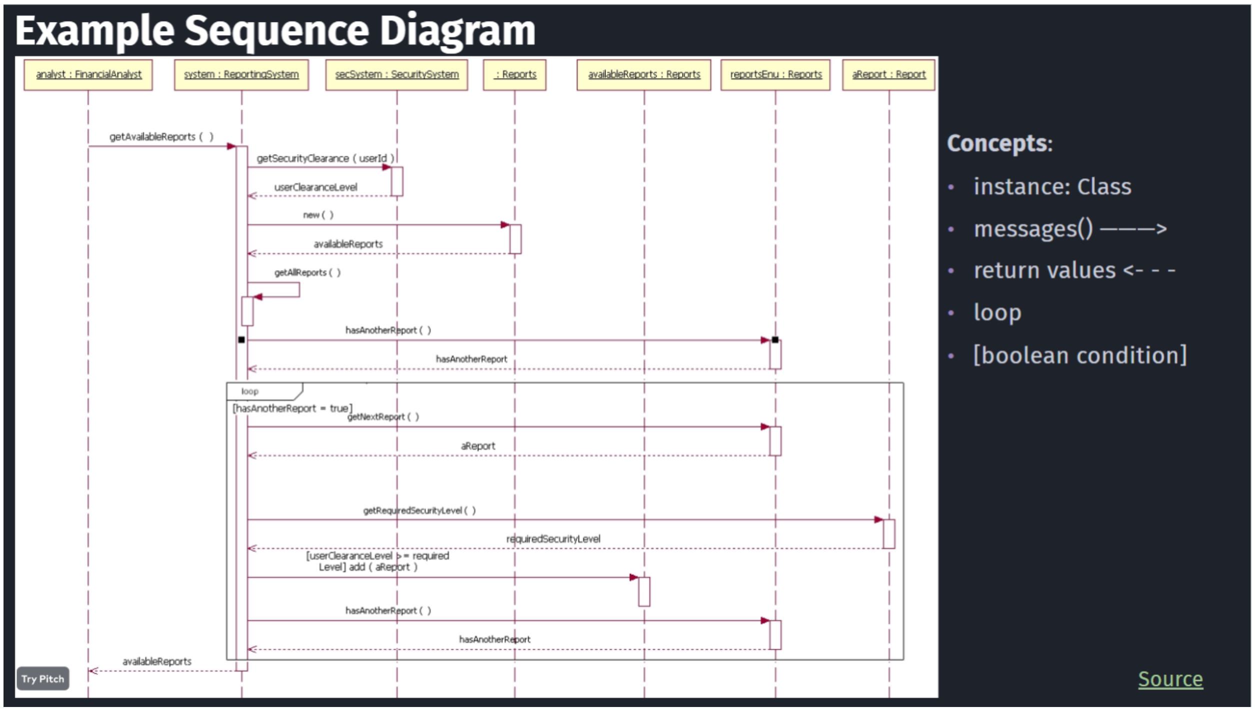
1) Creates a WarpDSL object and gets instructions
2) DO the .ra and perform an analysis
3) Put it into the constructor

4/3 Meeting Notes:

- Reliability Visualization will need the following methods, which are called by VisualizationImplementation:
    - visualization
    - fileVisualization
    - createFile

Sequence Diagram

4/10 Meeting Notes:

- 

Example Sequence Diagram

Concepts:

- instance: Class
- messages() ———>
- return values <- - -
- loop
- [boolean condition]

Source