Quarter Project Phase 4 Documentation

Nicole Kyriakopoulos nkyriako@ucsc.edu
Darwin Li dli39@ucsc.edu
Matthew Loyola mjloyola@ucsc.edu
Jeremy Hohman jhohman@ucsc.edu

Phase 3:

As an update to the previous phase, we have a fully functional OR that should produce correct results. In addition to producing correct results, it implements parallelization. In each recursive depth of inference, it will create two threads to search the two operand(which could be either facts or rules for each). This creates simultaneous recursive branches that will keep going until it bottoms out when there are only facts to inference. Then when the two threads are done searching the threads will join and the logical operation(OR/AND) will be performed. For the implementation of multithreading, we used the c++11 standard library thread. Also as a fix to phase 2, inference now takes in flags to allow it to print the result or to store the results in the knowledge base. As of now AND is not correctly implement and will not always produce correct results. However due the nature of our recursive algorithm, the AND is pipelined. We will fix by next phase.

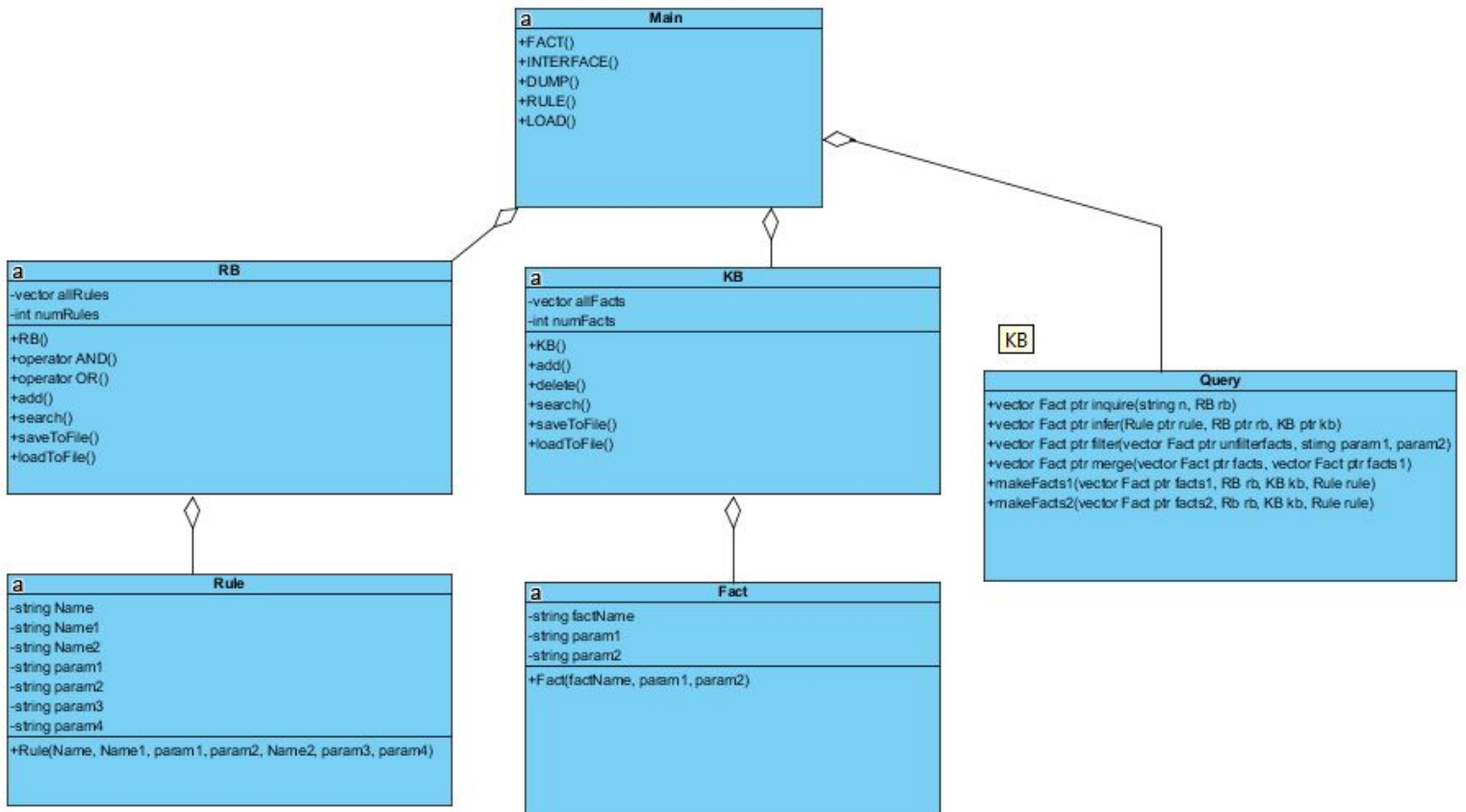For additional info, see inline comments.

Phase 2:

Changes: AND is not functional as of now. KB and RB hold fact and rule pointers instead now. KB holds a vector of facts and RB holds a map of the rules, to make searching less time consuming. AND code is partially implemented, but is not tested. If one tries to input an AND query, it returns the OR Results instead. The parameters for facts and rules are in a vector now, so now rules and facts can have more than 2 parameters per relation. Most functions in main pass a KB and RB parameter to function correctly. Furthermore, there is a Query class, a static class that encapsulates all the methods necessary to complete a query. First the inquire method takes in the inference.the infer method makes an inference based on the given rule. Then, Filter handles the instances where a query is asking for a specific name in a position. Merge merges returned vectors (inferred and from KB directly) together and passes a vector of the vector of answers to the query. As of now, INFERENCE only prints and does not store, however, we are have unfinished implementation on this part.

Makefile Commands:

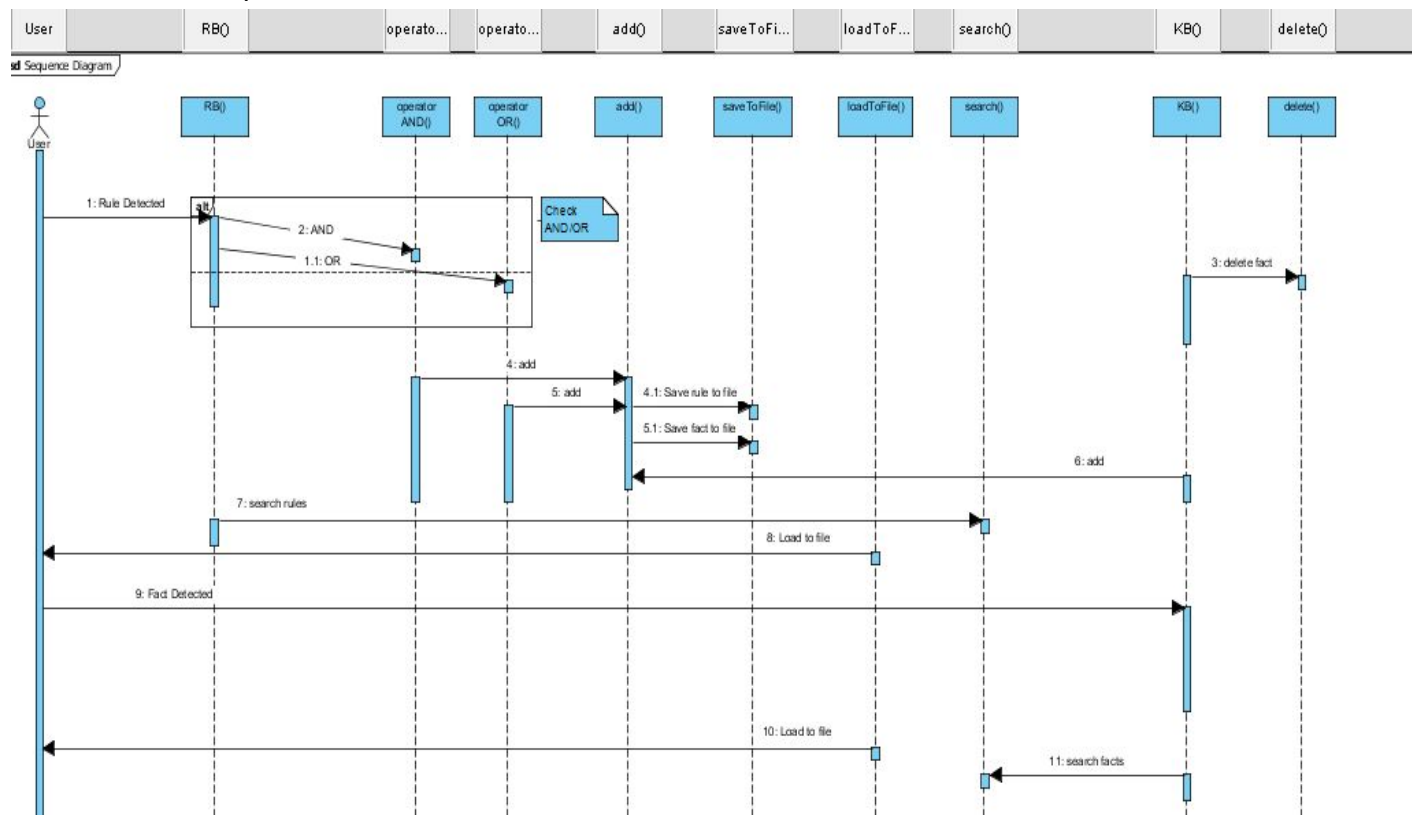make clean (cleans object files and executables), make (makes all), make program1 (runs main)
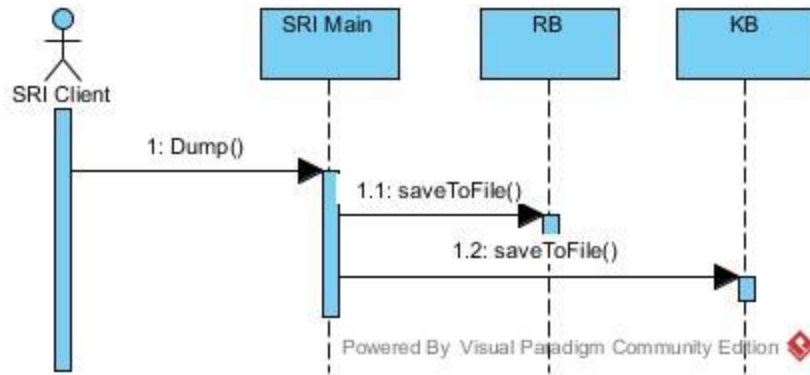For additional info, see inline comments.

Class diagram:

Main
- +FACT()
- +INTERFACE()
- +DUMP()
- +RULE()
- +LOAD()

RB
- -vector allRules
- -int numRules

- +RB()
- +operator AND()
- +operator OR()
- +add()
- +search()
- +saveToFile()
- +loadToFile()

KB
- -vector allFacts
- -int numFacts

- +KB()
- +add()
- +delete()
- +search()
- +saveToFile()
- +loadToFile()

KB

Query
- +vector Fact ptr inquire(string n, RB rb)
- +vector Fact ptr infer(Rule ptr rule, RB ptr rb, KB ptr kb)
- +vector Fact ptr filter(vector Fact ptr unfilterfacts, sting param1, param2)
- +vector Fact ptr merge(vector Fact ptr facts, vector Fact ptr facts1)
- +makeFacts1(vector Fact ptr facts1, RB rb, KB kb, Rule rule)
- +makeFacts2(vector Fact ptr facts2, Rb rb, KB kb, Rule rule)

Rule
- -string Name
- -string Name1
- -string Name2
- -string param1
- -string param2
- -string param3
- -string param4

- +Rule(Name, Name1, param1, param2, Name2, param3, param4)

Fact
- -string factName
- -string param1
- -string param2

- +Fact(factName, param1, param2)

Sequence Diagrams

Load Sequence:

Rule and Fact Sequence:

User | RB() | operato... | operato... | add() | saveToFi... | loadToF... | search() | KB() | delete()

sd Sequence Diagram

User

RB()
operator AND()
operator OR()
add()
saveToFile()
loadToFile()
search()
KB()
delete()

1: Rule Detected

alt

Check AND/OR

2: AND

1.1: OR

3: delete fact

4: add

5: add

4.1: Save rule to file

5.1: Save fact to file

6: add

7: search rules

8: Load to file

9: Fact Detected

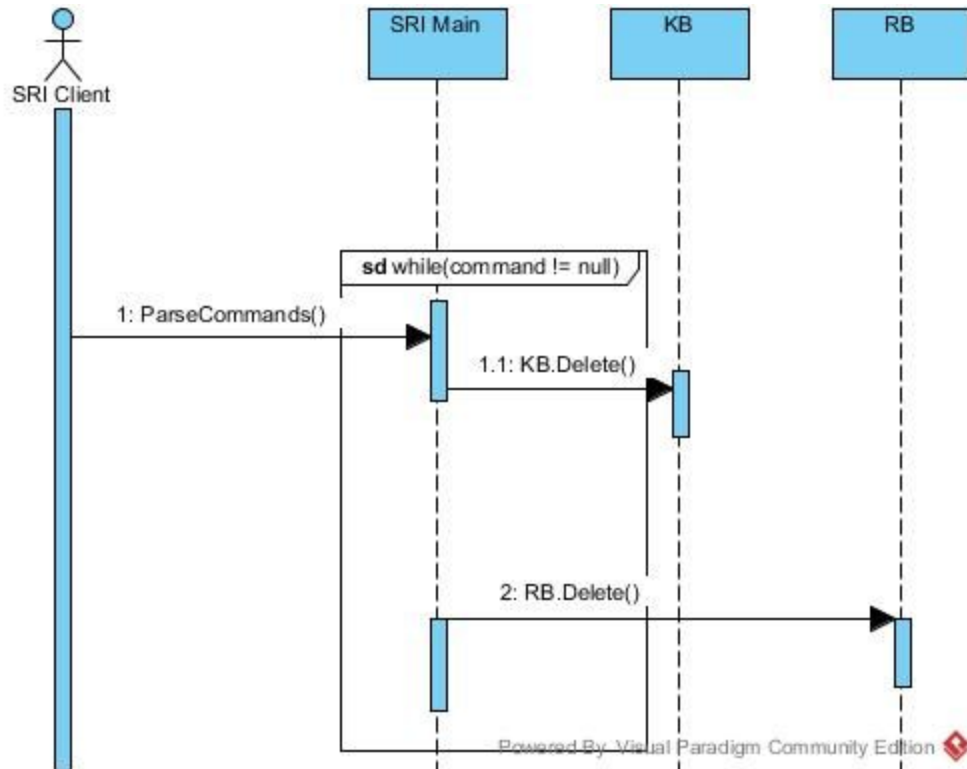10: Load to file

11: search facts

Notes: Rules are checked for AND/OR and added accordingly. Rules and Facts can be searched. They are saved to a file and can be loaded from some already filled file. Facts can be deleted. This diagram references the parsing that occurs in a different diagram because it was necessary to show the flow of events as the rules and facts are involved. Also for visual reasons, the method invocations were shown instead of the classes or objects - we will probably tweak this representation in a later version, or as the design changes.
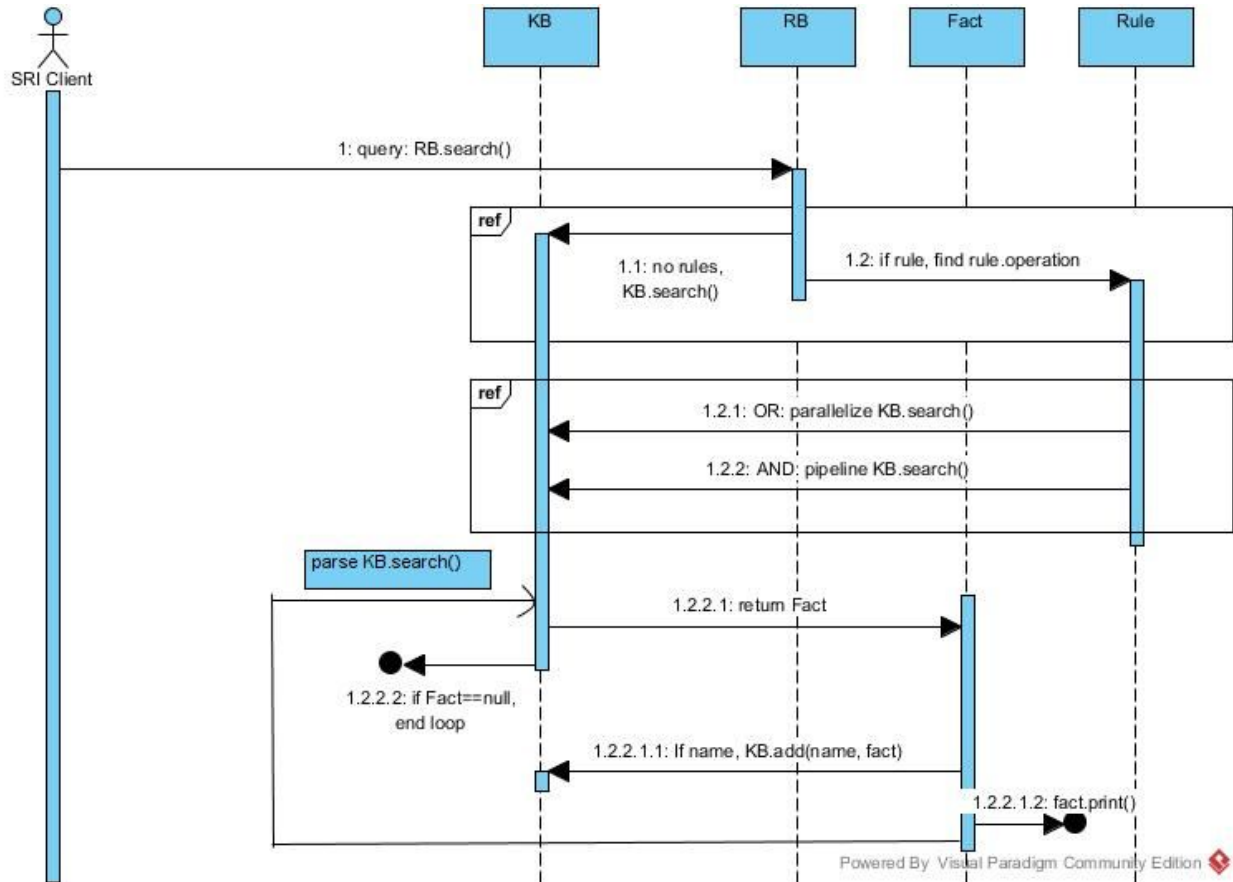
DUMP sequence:

DROP Sequence:

Inference Sequence:

Powered By Visual Paradigm Community Edition

Use Case Diagram: