SRI DESIGN DOCUMENT

OUTLINE OF RULE:

Rules are comprised up of components, of which can be either Facts or other Rules. A Rule is valid if its required components are valid.

Name	**Description**
Components	A vector of strings representative of either other rules or facts. The evaluate() function inside of Query will utilize this to check validity of those facts or rules.
Name	String name identifier
ops	A vector of integers (either 1 or 0) that represents AND or OR operators that will be used to evaluate the validity of the rule.
toString()	Returns string representation of the rule.
Rule(string s, vector<int> ops, vector<string> c)	Constructor (ops is a vector of ints that act as a switch for AND and OR operators to use)

OUTLINE OF RULE BASE:

Holds references to each Rule, along with methods to manage them. Is a part of "Working Memory."

Name	**Description**
rules	Map of rule pointers. Contains all rules in working memory.
Add(Rule * r)	Adds rule to the rule base. No two rules may have the same name.
Remove(Rule * r)	Remove rule from the rule base. Returns removed rule if successful.
RB()	Initializes an empty rule base
toString()	Returns string representation of all rules into a string so it can be printed
Find()	Takes in a string
Load()	Takes input from an SRI file and load all the rules into the rule base
Dump()	Saves all the facts into an SRI file so it can be used for input later

OUTLINE OF KNOWLEDGE BASE:

Holds maps of maps of maps… of strings representing Facts, along with methods to manage them. Is a part of “Working Memory.”

Name	**Description**
facts	A map of maps of maps of…. Of strings that represent combinations of actors (strings). Each combination is a “Fact.”
Add(vector<string>)	Adds fact to the knowledge base. Adds a string to the map if it doesn’t exist already exist, steps into the index and add string there if it does. Does not add if fact already exists (using evaluate function)
Remove(string)	Remove fact from the knowledge base. Uses map’s clear() and remove() as necessary.
KB()	Initializes an empty knowledge base
toString()	returns string representation of all facts into a string so it can be printed
evaluate(vector<string> actors)	Recursively traverses the KB to evaluate the validity of a fact by checking if it exists within the KB. Takes a vector of strings as input. If vector.size() == current map level (number of elements - 1) then the fact is valid.

OUTLINE OF PARSE:

Holds maps of maps of maps… of strings representing Facts, along with methods to manage them. Is a part of “Working Memory.”

Name	Description
RB* RuleBase	Pointer to the RuleBase
KB* KnowledgeBase	Pointer to KnowledgeBase
Query* QQ	Pointer to Query
int numRuns	The number of ‘)’’s in a line
Parse(KB* knowledgeBase, RB* ruleBase, Query* QQ)	Constructor for Parse
vector<vector<string> > Preds	Takes vectors of strings that if it encountered “Father($X,$Y)” would have vector of strings with [“Father”, “$X”, “$Y”]
vector<bool> Logic	Stores the ‘AND’ or ‘OR’ encountered
int searchLength(int start, int end)	Function that subtracts the two inputs
void ParsePred(string input, bool FactMode);	Gives Preds the vectors of strings with [“Father”, “$X”, “$Y”] and if in FactMode adds that to the KB.
void ParseRule(string input)	Controls the Parsing of a line when inserting a rule
Int numPreds(string input)	Calculates numRuns
Void ParseLine(string input)	Gets a line of input

Void ParseFile(string fileName)	Controls the Parsing of input from a file
Void DumpToFilestring fileName,string input)	Prints KB and RB to a file
Void ParseTerminalInput ()	Runs an infinite loop waiting for input that will break if a ‘q’ is entered as the only command
Void AddFact(vector<string>)	Tries to add a vector of strings to the KB
Void AddRule(int numFcns)	Adds a rule to RB

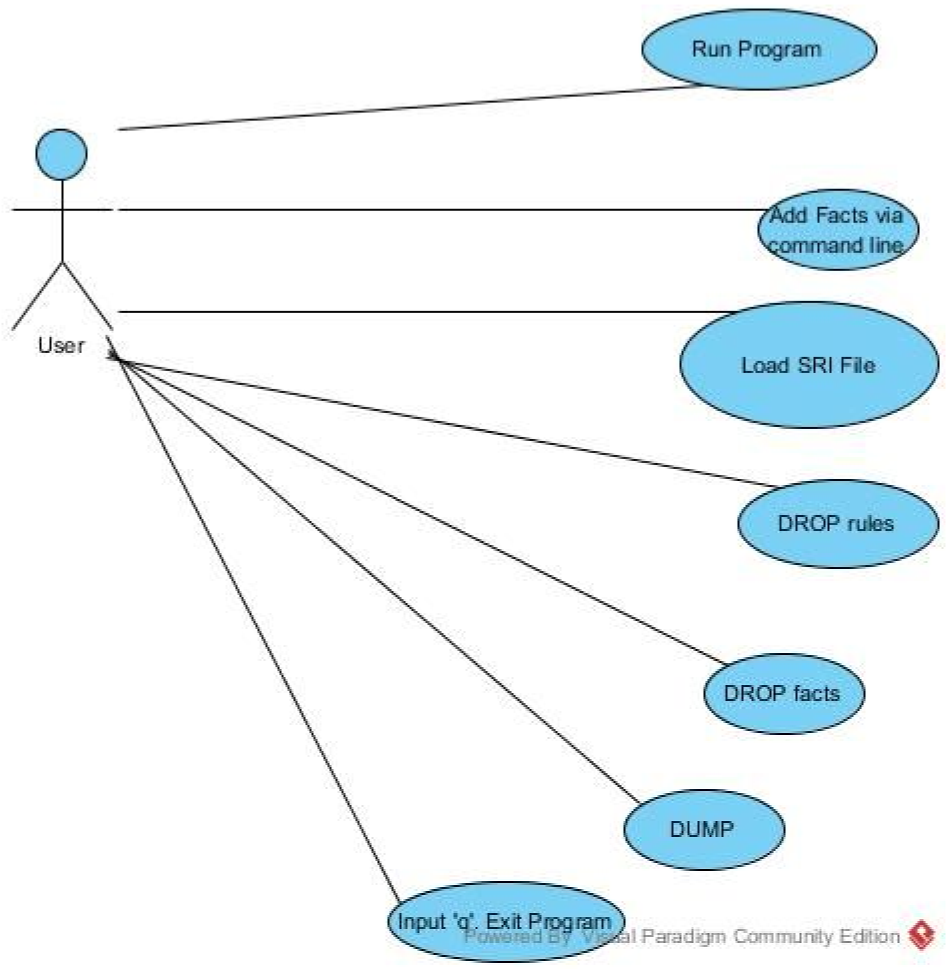
<u>OUTLINE OF QUERY</u>
The “Thinking” function that drives inferences. “Binds” actors together into combinations, which are then passed into each rule in the RB. The rules recursively evaluate their validity and populate the KB as necessary. Functions are put in place to not recur into duplicate combinations.

Name	**Description**
kb*	Pointer to a KB object
rb*	Pointer to a RB object
evaluateRule()	Takes in a single vector<string> with format (Relationship, component, component,...). If the rule exists as a fact in the KB, return true. If not, check components for validity. If the component is a fact, check if it exists within KB. Else if the component is a rule, check if it exists in the RB. If the rule exists within the RB, return true. Else, return false..
printResults()	Prints results of an inference run
inference()	“Binds” actors together into combinations, which are then passed into each rule in the RB. The rules recursively evaluate their validity and populate the KB as necessary.

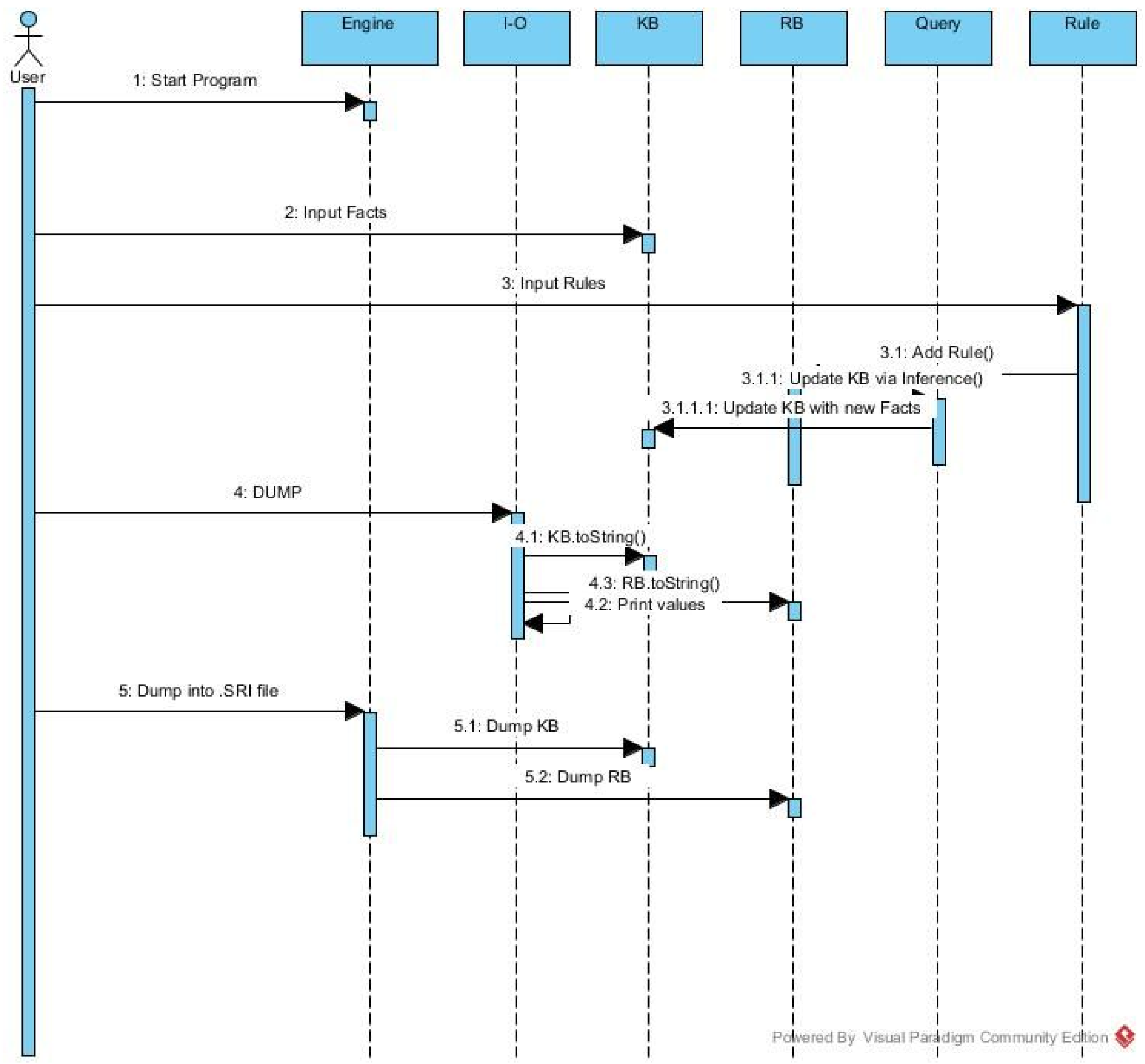
<u>ASSUMPTIONS:</u>
We are assuming the user is feeding the program proper input.
Rules are comprised of existing Facts and/or Rules already in working memory
A Fact is true if it can be found in the Knowledge Base
There can be no duplicate facts or rules

To compile:
Use: “make” without quotes in the root directory.

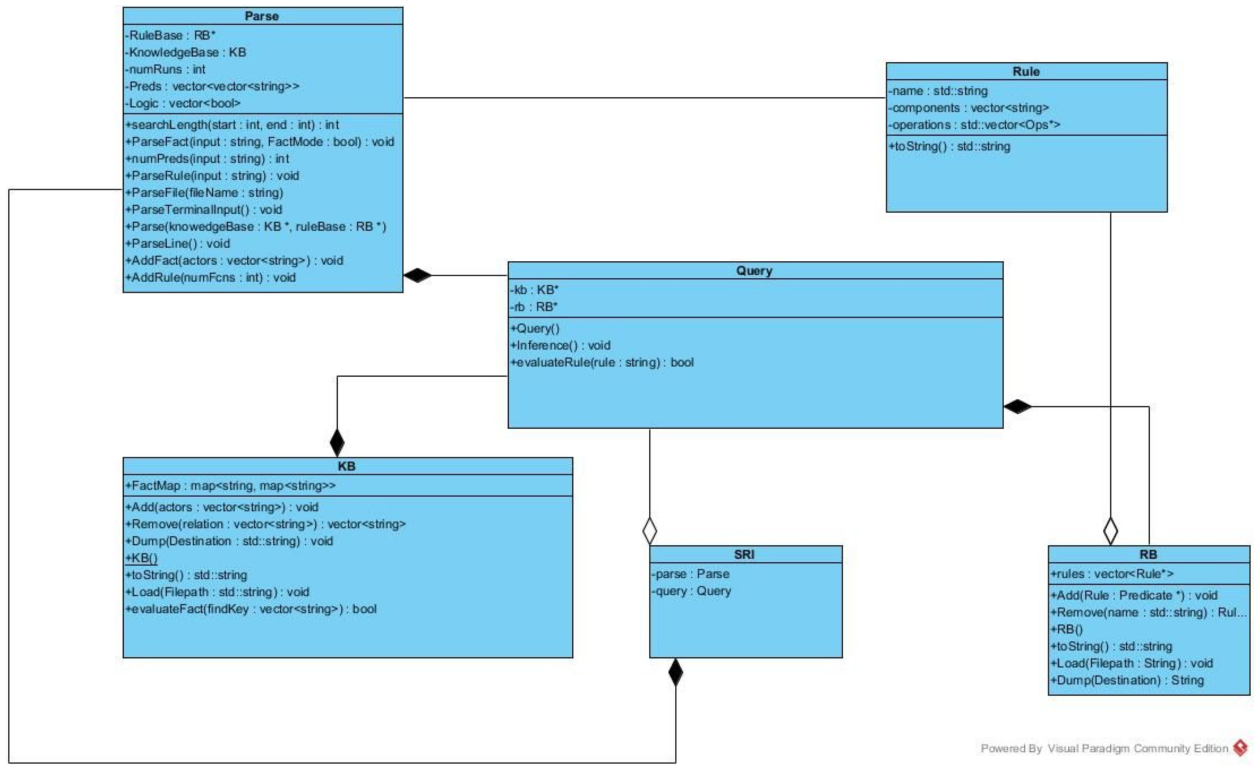
USE CASE DIAGRAM:



SEQUENCE DIAGRAM



CLASS DIAGRAM:


Parse
-RuleBase : RB*
-KnowledgeBase : KB
-numRuns : int
-Preds : vector<vector<string>>
-Logic : vector<bool>
+searchLength(start : int, end : int) : int
+ParseFact(input : string, FactMode : bool) : void
+numPreds(input : string) : int
+ParseRule(input : string) : void
+ParseFile(fileName : string)
+ParseTerminalInput() : void
+Parse(knowedgeBase : KB *, ruleBase : RB *)
+ParseLine() : void
+AddFact(actors : vector<string>) : void
+AddRule(numFcns : int) : void
Rule
-name : std::string
-components : vector<string>
-operations : std::vector<Ops*>
+toString() : std::string
Query
-kb : KB*
-rb : RB*
+Query()
+Inference() : void
+evaluateRule(rule : string) : bool
KB
+FactMap : map<string, map<string>>
+Add(actors : vector<string>) : void
+Remove(relation : vector<string>) : vector<string>
+Dump(Destination : std::string) : void
+KB()
+toString() : std::string
+Load(Filepath : std::string) : void
+evaluateFact(findKey : vector<string>) : bool
SRI
-parse : Parse
-query : Query
RB
+rules : vector<Rule*>
+Add(Rule : Predicate *) : void
+Remove(name : std::string) : Rul...
+RB()
+toString() : std::string
+Load(Filepath : String) : void
+Dump(Destination) : String
Powered By Visual Paradigm Community Edition