Nicole Maines, Richard Harker, Shayne Hayes, Son Huynh

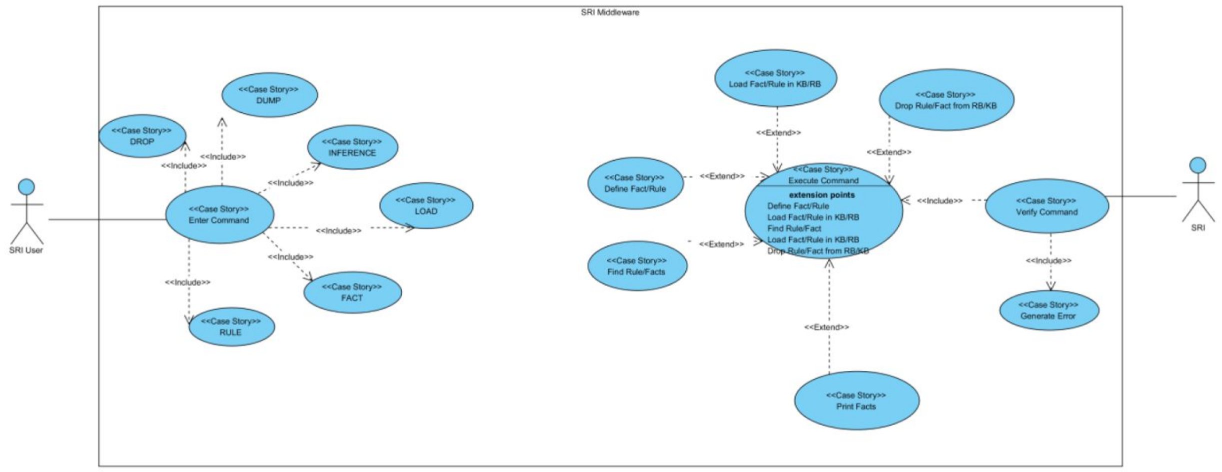
Dr. Karim Sobh

CMPS 109

17 March, 2017

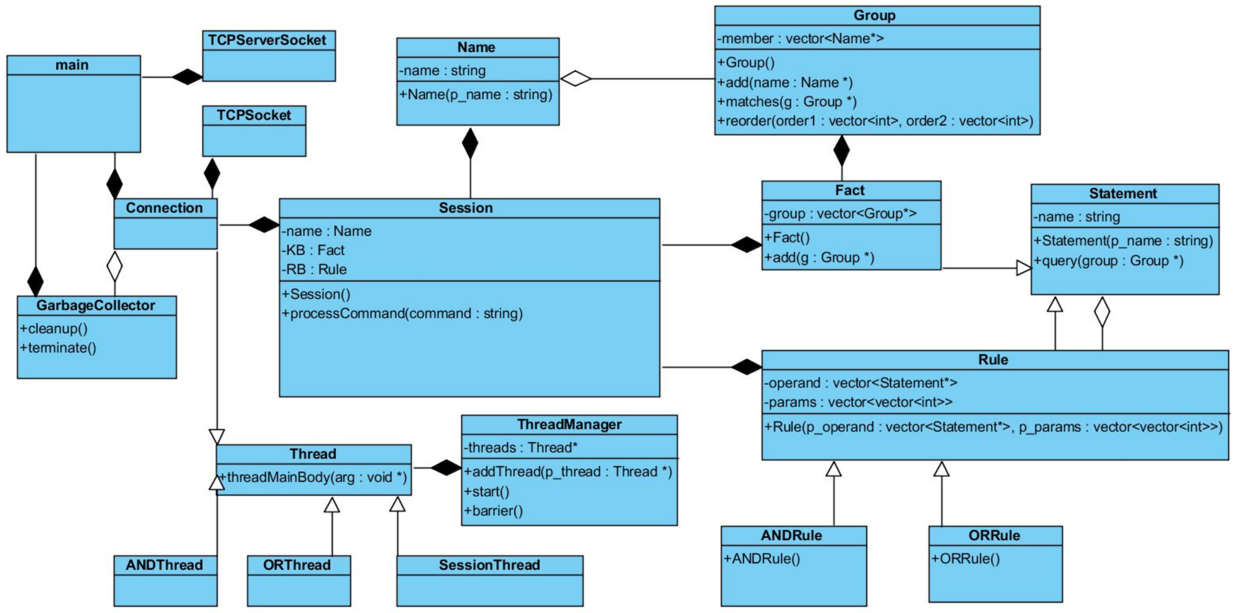
SRI UML Design Document (v4)

Use Case Diagram:


SRI Middleware
SRI User
SRI
<<Case Story>> DUMP
<<Case Story>> DROP
<<Case Story>> INFERENCE
<<Case Story>> LOAD
<<Case Story>> Enter Command
<<Case Story>> FACT
<<Case Story>> RULE
<<Include>>
<<Case Story>> Load Fact/Rule in KB/RB
<<Case Story>> Drop Rule/Fact from RB/KB
<<Case Story>> Define Fact/Rule
<<Case Story>> Find Rule/Facts
<<Case Story>> Execute Command
extension points
Define Fact/Rule
Load Fact/Rule in KB/RB
Find Rule/Fact
Load Fact/Rule in KB/RB
Drop Rule/Fact from RB/KB
<<Extend>>
<<Case Story>> Verify Command
<<Case Story>> Generate Error
<<Case Story>> Print Facts

Text Description: This diagram illustrates the user's view of using the SRI program. Specifically, it showcases the various commands available to the user and what functions SRI must fulfill.

Class Diagram:



Text Description: This diagram illustrates the class structure of our SRI implementation. Our implementation is separated into 17 classes: Session, Name, Group, Statement, Fact, Rule, ORRule, ANDRule, Thread, SessionThread, ORThread, ANDThread, ThreadManager, Connection, GarbageCollector, TCPSocket, and TCPServerSocket.

Session is the moderator class of the program. It contains all of the Statements and Names required for the program to function correctly. Its main two functions are to translate string commands to method invocations and to manage memory.

Name is really just a string. Not much else to see here.

Group is a collection of pointers to Names and is the data structure that is at the heart of our program. It contains many of the methods that make inferences possible, such as match(), reorder(), and addParams().

Statement is the father class of Fact and Rule and the grandfather class of ORRule and ANDRule. It stores any elements that facts and rules have in common, such as a name and a query() function.

Fact is a collection of pointers to Groups. It contains a method for adding new Groups and a method for querying its Groups.

Rule is the father class of ORRule and ANDRule and contains the functionality shared between the two. Most importantly, it stores a vector of pointers to Statements that represent a rule's predicates and a 2D vector of ints that represent parameter substitution rules.

Both ORRule and ANDRule are subclasses of Rule that contain different implementations of the query() function.

Thread encapsulates all functionality needed to create and manage pthreads.

ThreadManager stores a number of threads and runs them all in parallel.

SessionThread is used to multithread pointer fecthing when creating a new rule.

ORThread contains some of the OR operator logic and is used to parallelize OR queries (each predicate is queried in its own thread).

ANDThread is used to pipeline AND queries. Each group returned by a predicate is queried in its own thread for the next predicate. By the end, each "branch" of the tree is in its own thread.

Connection is used to create a new thread for each client that connects to the SRI Server. Each Connection object contains one SRI Session -- thus each client has their own Knowledge Base and Rule Base.

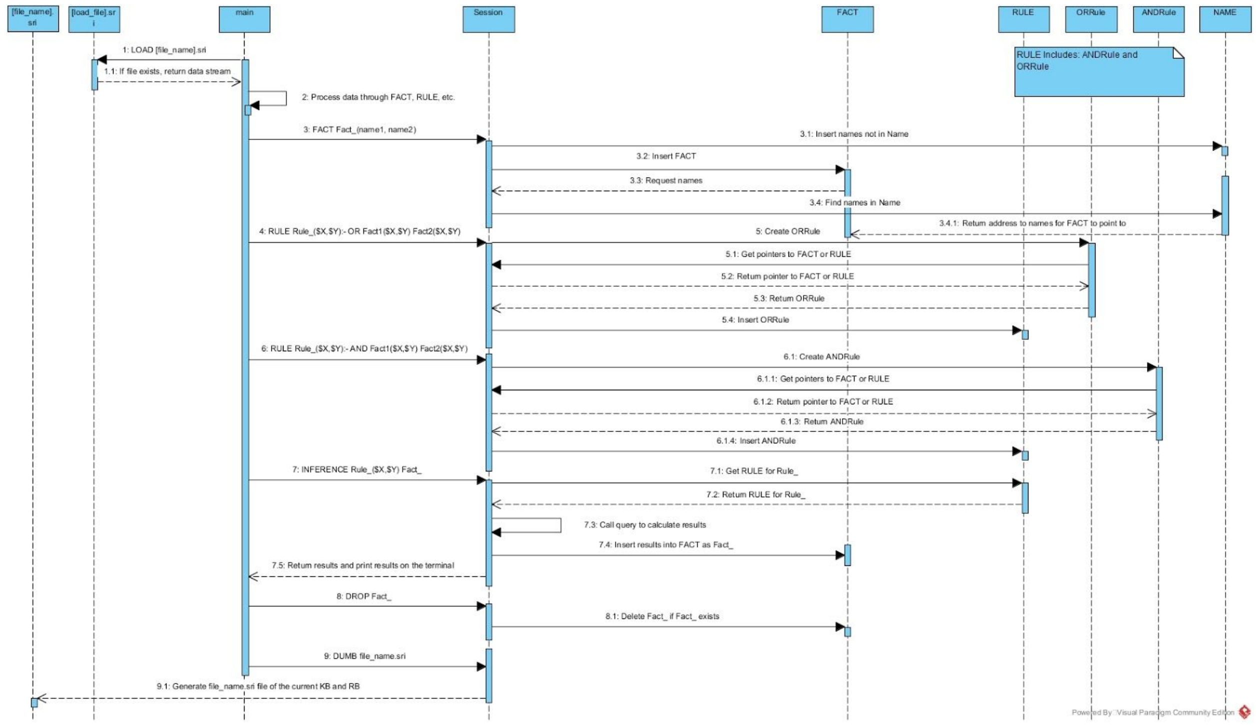
GarbageCollector cleans up Connections that are no longer running.

TCPSocket encapsulates all functionality needed to create and manage sockets.

TCPServerSocket is used to initialize new sockets representing various clients connecting to a single server.

Note: common.cpp simply contains some helper functions such as compressVector() and isDuplicate().

<u>Sequence Diagram:</u>


[file_name].sri
[load_file].sri
main
Session
FACT
RULE
ORRule
ANDRule
NAME
RULE Includes: ANDRule and ORRule
1: LOAD [file_name].sri
1.1: If file exists, return data stream
2: Process data through FACT, RULE, etc.
3: FACT Fact_(name1, name2)
3.1: Insert names not in Name
3.2: Insert FACT
3.3: Request names
3.4: Find names in Name
3.4.1: Return address to names for FACT to point to
4: RULE Rule_($X,$Y):- OR Fact1($X,$Y) Fact2($X,$Y)
5: Create ORRule
5.1: Get pointers to FACT or RULE
5.2: Return pointer to FACT or RULE
5.3: Return ORRule
5.4: Insert ORRule
6: RULE Rule_($X,$Y):- AND Fact1($X,$Y) Fact2($X,$Y)
6.1: Create ANDRule
6.1.1: Get pointers to FACT or RULE
6.1.2: Return pointer to FACT or RULE
6.1.3: Return ANDRule
6.1.4: Insert ANDRule
7: INFERENCE Rule_($X,$Y) Fact_
7.1: Get RULE for Rule_
7.2: Return RULE for Rule_
7.3: Call query to calculate results
7.4: Insert results into FACT as Fact_
7.5: Return results and print results on the terminal
8: DROP Fact_
8.1: Delete Fact_ if Fact_ exists
9: DUMB file_name.sri
9.1: Generate file_name.sri file of the current KB and RB
Powered By Visual Paradigm Community Edition

Text Description: This diagram illustrates the sequence of events when a user inputs each of the 7 commands: LOAD, FACT, RULE, INFERENCE, DROP, DUMP, and EXIT.

LOAD will open a file and execute each command inside it.

FACT will first determine if the fact exists using Fact's getName() method and will create the fact and add it to the Knowledge Base if it does not exist. Then it will create a new Group and loop through the entered names. For each name, the program will determine if the name exists, create a new Name and add it to a name vector if it does not exist or fetch the Name if it does exist, and add a pointer to the name to the newly created Group. Then it will simply add this Group to the Fact.

RULE will first loop through the KB and RB to fetch pointers to its predicates, and then it will use the parameters to create a 2D vector of substitution rules (ints that define what parameters are substituted where). Then the program will create a new ORRule or ANDRule, passing the predicates and parameter substitution rules to the Rule constructor.

INFERENCE will create a new Group and loop through the arguments of the command. For each name/parameter, it will fetch a pointer to the name/parameter from the name vector and add it to the Group. The program will then call the query() function of each fact and rule with the specified name, storing all results in a vector of Groups (and omitting duplicates). If the user specifies a name, the program will save this vector of results as a fact using the FACT command. Otherwise, it will print the results to the terminal and delete the results (query() always returns new groups).

DROP will loop through the KB and RB, deleting any facts or rules with the specified name (and deleting predicates that point to these facts/rules).

DUMP will loop through the KB and RB, generating a .sri file to store all facts and rules.

EXIT will delete the Session, which will in turn free all memory.

Assumptions Made:

- The predicates within a rule may be facts or other rules.
- The size of the groups returned from inferences are equal to the size of the list of parameters passed.
- Facts and rules may contain an unlimited number of parameters.

  Ex. Fact1($X,$Y) and Fact2($A,$B,$C,$D,$E) are both valid facts.
- There are no restrictions on the ordering of parameters.

  Ex. Rule1($X,$Y):- OR Rule2($X,$Y) Rule3($Y,$X,$Z) is a valid rule.
- Rules may contain an unlimited number of predicates.

  Ex. Rule1($X,$Y):- OR Rule2($X,$Y) Rule3($Y,$X) Rule4($X,$Y,$Z) is a valid rule.
- There is no limit on the number of facts that may be added to the Knowledge Base or rules that may be added to the Rule Base.
- Duplicate facts will not be added to the Knowledge Base and inferences will always eliminate duplicates before returning.
- Each predicate of an OR rule must contain all of the input parameters. Predicates missing one or more input parameters will be removed.

  Ex. In Rule1($X,$Y,$Z):- OR Rule2($Z) Rule3($X,$Y,$A) Rule4($B,$X,$Y,$Z), Rules 2 and 3 are invalid predicates and will be dropped.

  Note: These predicates will not even be written when the user inputs a DUMP command, since they are invalid.
- The predicates of an OR rule are independent of one another during an inference. The results of one predicate do not depend on the results of another.

Ex. In Rule1($X,$Y):- OR Rule2($X,$Y,$A) Rule3($A,$Y,$X), Rule 2 and Rule 3 are queried independently.

- The predicates of an AND rule are dependent on one another during an inference. Specifically, the result of each predicate depends on the result of the preceding predicates.

  Ex. In Rule1($X,$Y):- AND Rule2($A,$X) Rule3($Y,$A), Rule 3 cannot be inferenced until Rule2 is inferenced.

  Ex. Rule1($X,$Y,$Z):- AND Rule2($X,$A) Rule3($Y,$B) Rule4($A,$B,$Z) is a valid rule.
- Each parameter will be preceded by a dollar sign character.
- Names of facts and rules consist of the characters A-Z and/or a-z (though we also support numbers).
- Multiple facts and rules may share the same name. During an inference, every fact and rule will be queried and the results concatenated (omitting repeats).
- In addition to filtering names, the user may also filter parameters during inferences.

  Ex. Inference Fact1($X,$X) will only return pairs of the same names.

  (Note however that Fact1($X,$Y) should also return pairs of the same names in addition to different names!)
- OR is multithreaded by querying each of its predicates in parallel.
- AND is multithreaded by querying each predicate’s returned groups in parallel (pipelining).

- Each thread is given its own separate output vector to avoid situations where two threads push results simultaneously.
- Multiple clients may connect to a single server.
- Each client has their own unique Knowledge Base and Rule Base on the server.
- The server should run as a daemon, and should not close unless a connection error occurs.