Katerina Chinnappan, Jakrarin Simrakut, Traik Zeid, Nathan Monahelis

CMPS 109 - Final Project Phase 4 report

Basically, we have put in a lot of time and effort in this project, and getting feedback from phase3 and phase2, we have changed a lot for phase4. We added multithreading. We hope we can get as much as partial credit as we can so it can pay off the work.

Running in into some last minute decisions and bugs, we have put together a server and client modules for phase4 but unfortunately couldn't get to compile. We have used Karim's code for TCP Sockets, Garbage Collector and Connection. Basically our idea was to have the SRIMain class begin the SRI program within the constructor which took a TCPSocket pointer. The Connection class invokes this SRIMain object and gets the program running.

Please do notice that we added multithreading for phase3 and more functionality for phase2.

Class SRI:
SRI is our main class. It implements the commands, load, dump, drop(rules and facts), and inference. Inference is yet to be completed. The SRI class uses the KnowledgeBase and RuleBase pointer objects.

```
Void SRI::inputLine(stringStream &)

Void SRI::addFact(string)

Void SRI::addRule(string)

Void SRI::load(string) // It will then parse the SRI file line by line
//adding rules or facts into their respective databases.

Void SRI::dump(string command)

Void SRI::dumpRF(ostream &os,KnowledgeBase *kb, RuleBase *rb)

Void SRI::drop(string param)

Vector<map<string,string>> inferenceFact(string, vector<string> &);

Vector<map<string,string>> inferenceRule(string, vector<string> &);

map<string,vector<string>> findRule(string, int);

Void SRI::Inference(string) //This will print the results of the
//given query to the terminal. Inference
```

```
//will have an option to declare the results of the query under a fact
//with a given variable name.
```

Class KnowledgeBase:
The KnowledgeBase class contains public string map of Fact* vectors which will hold the facts. Public methods such as the findFactAssociation which returns true if the association of a certain fact is found or vice versa.

```
KnowledgeBase()

void KnowledgeBase::AddFact(vector<string> mems) //add a fact to the
//KnowledgeBase Dictionary

void KnowledgeBase::dropFact(vector<string> mems)//drop the fact from
the //FactDictionary

void removeAllFactsWithkey(string key);

int totalFacts();

vector<vector<string>> findFact(string assoc);

bool exists(string);

void dumpFact(ostream &os, string command)
```

The KnowlegeBase class uses a Fact pointer object.

Class RuleBase:
The RuleBase class contains a public string map of Rule* vectors. Just like in KnowledgeBase, we can add and drop a rule we specify. The RuleBase class contains a Rule pointer object.

```
RuleBase::RuleBase()

void RuleBase::AddRule(map<string, vector<string>> mems)

void RuleBase::dropRule(map<string, vector<string>> mems)

void RuleBase::removeAllRulesWithkey(string key)

int totalRules();

bool exists(string)

vector<map<string, vector<string>>> findRule(string);
```

```
RuleBase::~RuleBase()
```

Class Parser:

The Parser class will open up a specified .sri file and parse it. Basically, take it apart and separate the Facts from the Rules as well as process the logical operators.

```
Parser();

Bool getType(string) // determine fact/rule true=fact false=rule

Bool getGate(string)//returns the type of gate used in any given rule

String getFactAssoc(string) // returns fact Assoc

String  getRuleAssoc(string)//returns rule Assoc

String  getInferAssoc(string)//returns the association of an inference command

statement

Size_t betterFind (const string& haystack, size_t pos, const string&
needle, size_t n)

vector<string> addFact(string p_string);

map<string< vector<string>> addRule(string p_string);

Vector<string> parseRule(string p_string)//returns a vector of string
parameters

Vector<string> inference(string p_string)// returns a vector of string
parameters
```

~Parser();

Class Threads:

```
Vector<thread *> threadList //our list of threads to access

ThreadContainer()

void executeThreads()
```

```
Void insert(thead *)//insert a thread into our list

~Thread Container()
```

Class Thread:

```
pthread_attr_t pthread_attr; // pthread attribute data member

long cpu_count;     // Number of CPUs data member

pthread_t pthread;  // pthread_t identifier data member

char identifier[128]; // A printable thread identifier represented by
the time th thread created

bool started;   // A flag indicating if a thread is started or not

bool running;   // A flag indicating if a thread is running or not

pthread_mutex_t mutex;  // A mutex that controls the execution of the
thread

bool termination_request;   // A flag indicating that a termination
request is initiated

void *(*threadRoutine   ) (void *); // A pointer to the start routine
of the thread execution

void setRunning (bool _running);    // Sets the running flag data
member of the thread

static void  cleanup(void * target_thread); // A static method that
performs house keeping after the thread terminates

Thread(void *(*_threadRoutine) (void *) =NULL); // Constructor

bool isRunning ();      // Check if thread is running

pthread_t * getThreadHandler(); // Returns a pointer to the thread
identifier

void start (); // A jacket wrapper method that fork the thread
execution

virtual void * threadMainBody (void * arg) = 0; // A pure virtual
method whose implementation is the thread main function

static void * run (void * arg);     // A static method that is passed
to pthread_create and invokes threadMainBody from within
```

```
void waitForRunToFinish (); // Blocks until the running thread
finishes execution

char * getThreadIdentifier ();  // Return the thread identifier string

bool isAlive ();    // Checks if the thread start is initiated

virtual ~Thread();  // Virtual Thread Destructor
```

Class Connection:

Connection extends from thread class provided by Professor Karim.

```
TCPSocket * tcpSocket// TCP Socket for communication with client

Connection * next_connection// A way to build a linked list of connections for the garbage
collector to be able to track them

Connection(TCPSocket * p_tcpSocket)// Constructor: Set client connected TCP socket

Void * threadMainBody(void * arg)// Main thread body that serves the connection

Void setNextConnection()// Set the next pointer: used by the Garbage Collector

Connection * getNextConnection()// Get a pointer to the next connection

~Connection()// Destructor
```

Class TCPServerSocket:

```
Int sock;// Socket Handler

Struct sockaddr_in serverAddr // Server Socket Address Structure

Struct sockaddr_in serverAddr// Client Socket Address Structure

Char * address// Local address for the server socket to bind to

Int port;// Local port for the server socket to bind to

Int backlog// Maximum length of the queue fo pending connections.

TCPserverSocket(const char * _address; int _port, int
_backlog)//Constructor

Bool initializeSocket()// Initailize server socket
```

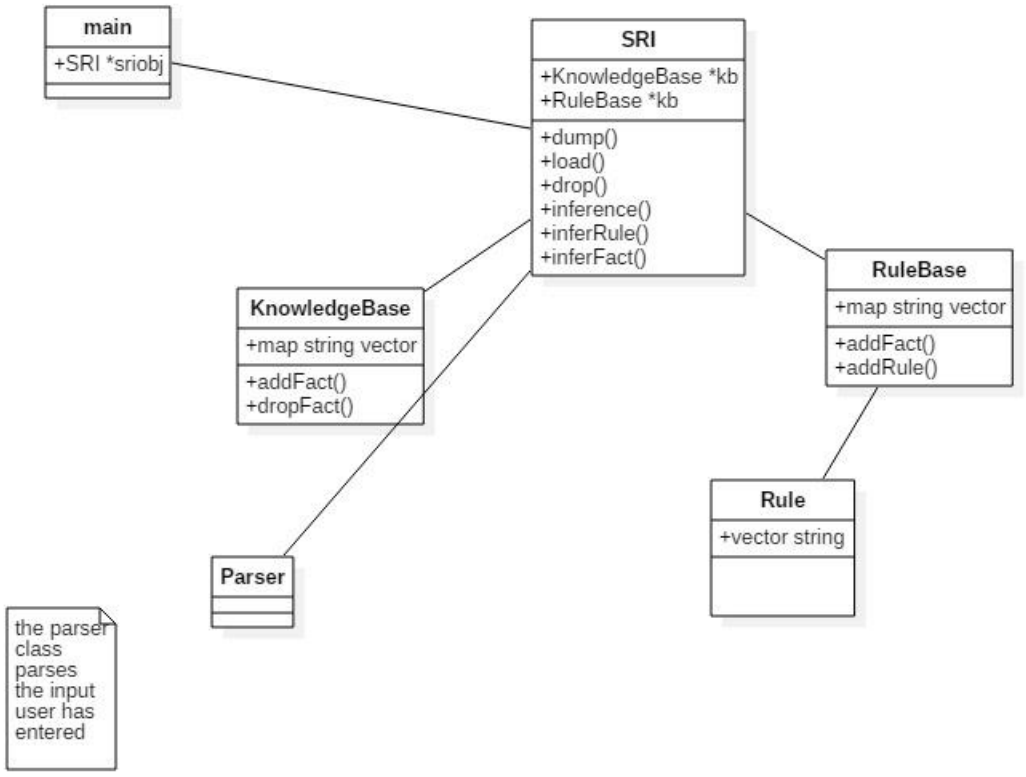
```
TCPSocket * getConnection (int timeoutSec=0, int timeoutMilli=0, int
readBufferSize=10*1024*1024,int writeBufferSize=10*1024*1024)// Wait for a
client connection and return a TCPSocket object that represents the client

~TCPServerSocket()// Destructor
```

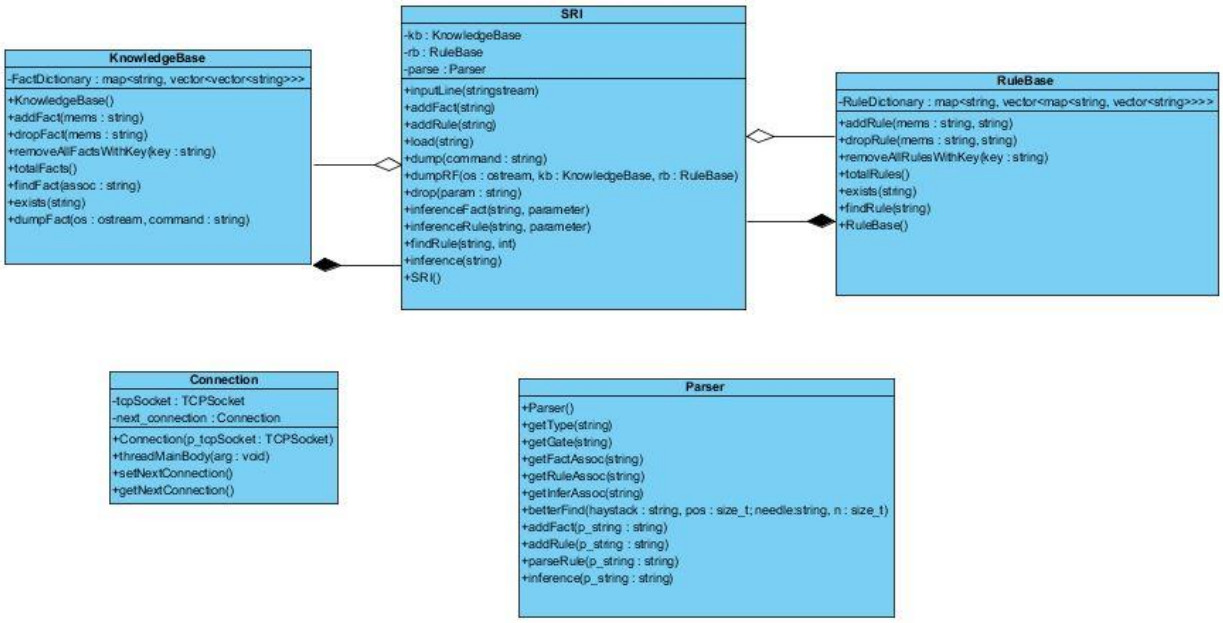
Class SRIMain
The wrapper class for phase4, just the constructor which gets the program running

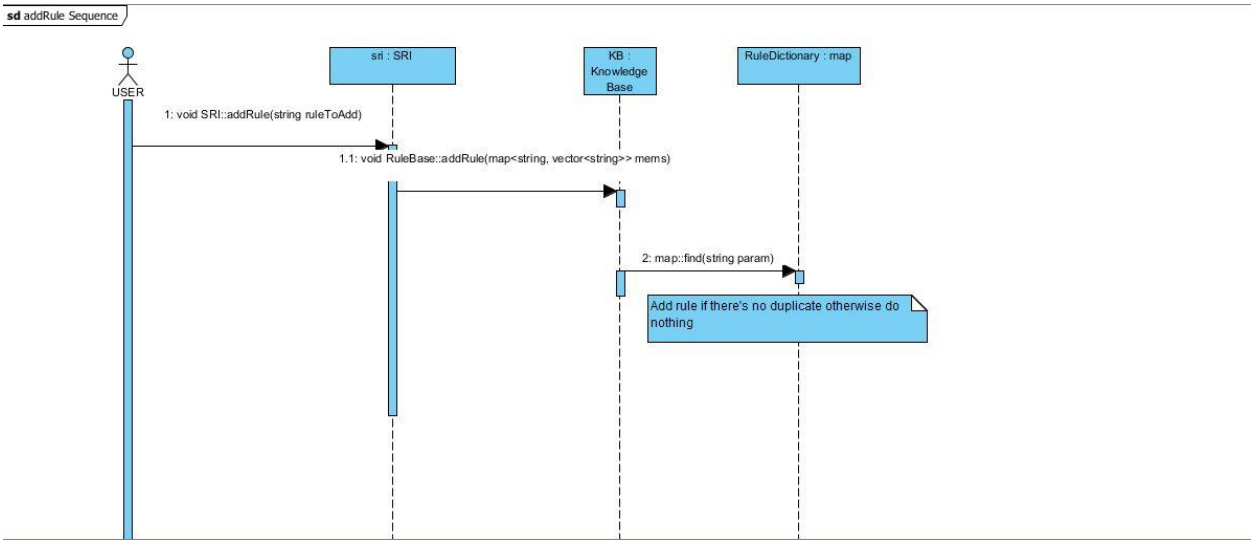
Main.cpp
This is the main function which has one SRI pointer object. Provides the user with a menu, takes in user input and performs the operations.

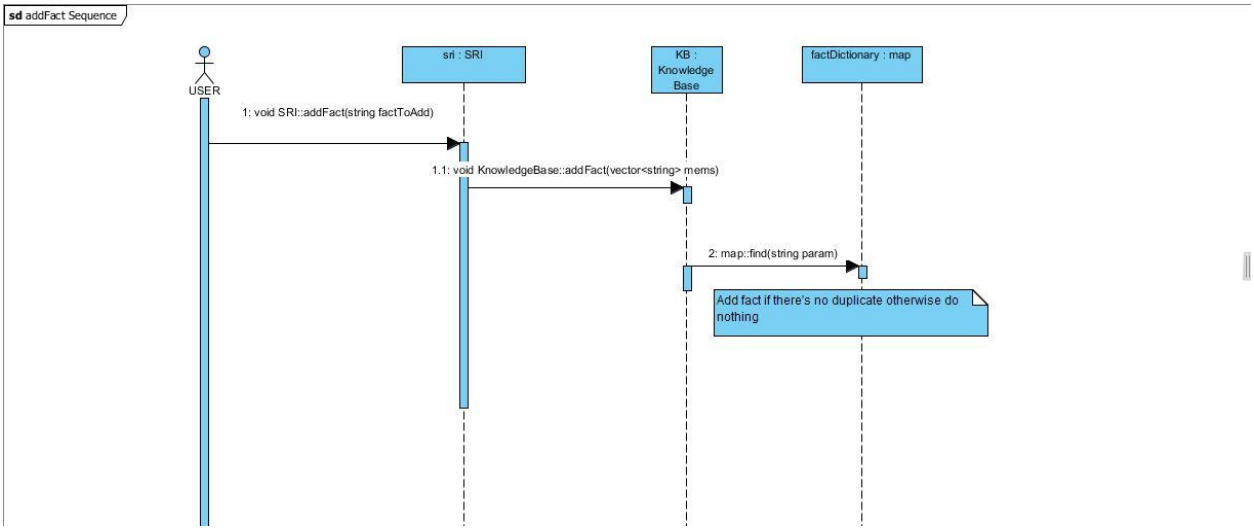
common.h
This is a header file which includes ALL C++11 headers, and std::functions to make life easier.

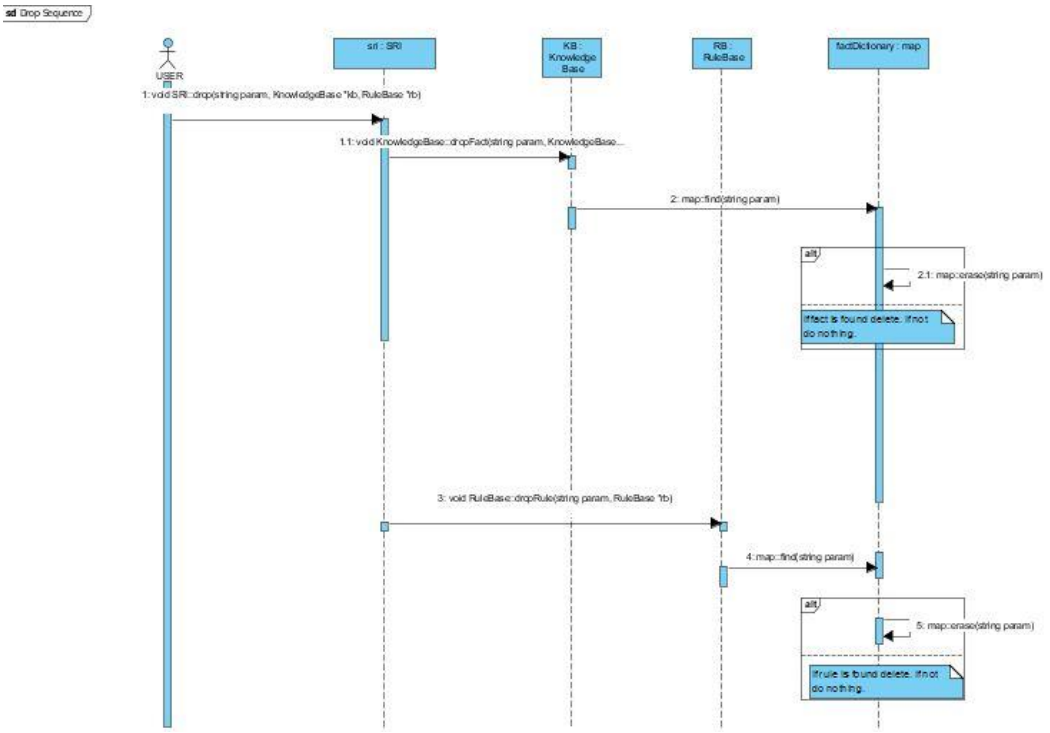
Makefile
Creates the .o for SRI

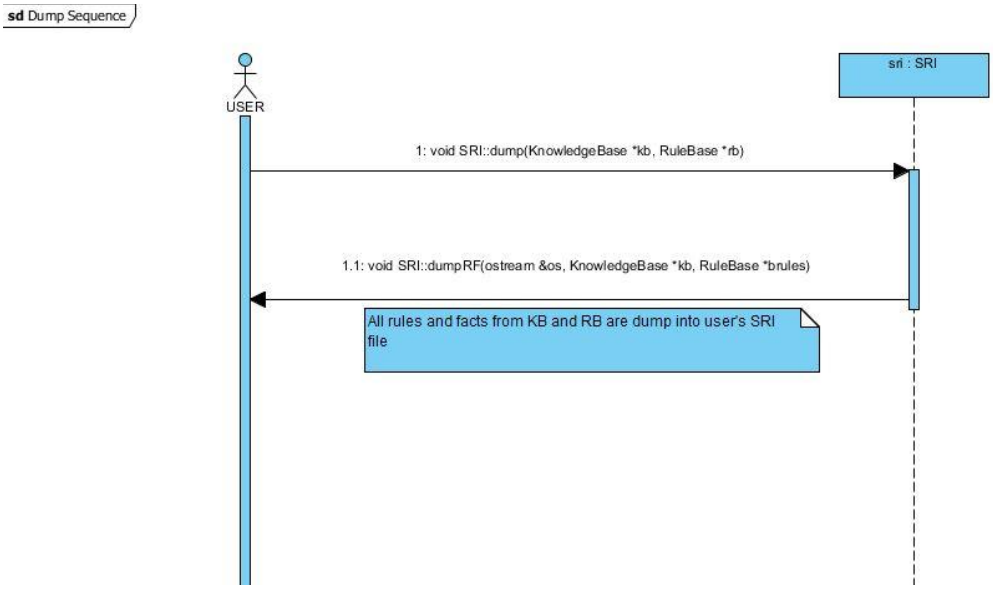



KnowledgeBase
-FactDictionary : map<string, vector<vector<string>>>
+KnowledgeBase()
+addFact(mems : string)
+dropFact(mems : string)
+removeAllFactsWithKey(key : string)
+totalFacts()
+findFact(assoc : string)
+exists(string)
+dumpFact(os : ostream, command : string)
SRI
-kb : KnowledgeBase
-rb : RuleBase
-parse : Parser
+inputLine(stringstream)
+addFact(string)
+addRule(string)
+load(string)
+dump(command : string)
+dumpRF(os : ostream, kb : KnowledgeBase, rb : RuleBase)
+drop(param : string)
+inferenceFact(string, parameter)
+inferenceRule(string, parameter)
+findRule(string, int)
+inference(string)
+SRI()
RuleBase
-RuleDictionary : map<string, vector<map<string, vector<string>>>>
+addRule(mems : string, string)
+dropRule(mems : string, string)
+removeAllRulesWithKey(key : string)
+totalRules()
+exists(string)
+findRule(string)
+RuleBase()
Connection
-tcpSocket : TCPSocket
-next_connection : Connection
+Connection(p_tcpSocket : TCPSocket)
+threadMainBody(arg : void)
+setNextConnection()
+getNextConnection()
Parser
+Parser()
+getType(string)
+getGate(string)
+getFactAssoc(string)
+getRuleAssoc(string)
+getInferAssoc(string)
+betterFind(haystack : string, pos : size_t; needle:string, n : size_t)
+addFact(p_string : string)
+addRule(p_string : string)
+parseRule(p_string : string)
+inference(p_string : string)


sd addRule Sequence
USER
sri : SRI
KB : Knowledge Base
RuleDictionary : map
1: void SRI::addRule(string ruleToAdd)
1.1: void RuleBase::addRule(map<string, vector<string>> mems)
2: map::find(string param)
Add rule if there's no duplicate otherwise do nothing


sd addFact Sequence
USER
sri : SRI
KB : Knowledge Base
factDictionary : map
1: void SRI::addFact(string factToAdd)
1.1: void KnowledgeBase::addFact(vector<string> mems)
2: map::find(string param)
Add fact if there's no duplicate otherwise do nothing


sd Drop Sequence
USER
sri : SRI
KB : Knowledge Base
RB : RuleBase
factDictionary : map
1: void SRI::drop(string param, KnowledgeBase *kb, RuleBase *rb)
1.1: void KnowledgeBase::dropFact(string param, KnowledgeBase...
2: map::find(string param)
alt
2.1: map::erase(string param)
If fact is found delete. If not do nothing..
3: void RuleBase::dropRule(string param, RuleBase *rb)
4: map::find(string param)
alt
5: map::erase(string param)
If rule is found delete. If not do nothing..


sd Dump Sequence
USER
sri : SRI
1: void SRI::dump(KnowledgeBase *kb, RuleBase *rb)
1.1: void SRI::dumpRF(ostream &os, KnowledgeBase *kb, RuleBase *brules)
All rules and facts from KB and RB are dump into user's SRI file