Katerina Chinnappan, Jakrarin Simrakut, Traik Zeid, Nathan Monahelis

CMPS 109 - Final Project Phase 2 report

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

Void SRI::dump() // It will access the current facts and rules defined
//in the runtime KB and RB and then save them to the SRI file given.

Void SRI::Inference(string query) //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.

Void SRI::Drop() // This will invoke a KB/RB method depending on what
//is being dropped. class KB : public Operator

Void SRI::dumpRF(ostream &os, KnowledgeBase *facts, RuleBase *brules)
```

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.

```
bool KnowledgeBase::findFactAssociation(Fact * fact)

void KnowledgeBase::AddFact(Fact * fact) //add a fact to the
//KnowledgeBase Dictionary

void KnowledgeBase::dropFact(string param)//drop the fact from the
//FactDictionary

KnowledgeBase::~KnowledgeBase() //destructor
```

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(Rule * rule)

void RuleBase::dropRule(string param)
```

Class Parse:

The Parse 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.

```
Bool getType(line) // determine fact/rule true=fact false=rule

String getFactAssoc(line) // returns fact Assoc

String  getRuleAssoc(line)//returns rule Assoc

Vector getFactParam(line)//returns a vector of string parameters

Vector getRuleParam(line)// returns a vector of string parameters

String getGateLine(line) // returns AND or OR of line
```

Class Component:

Our Component.h consists of three classes; Component, Fact and Rule.
Three constructors (default, with parameters and a copy constructor)
Bool find which takes in a vector of strings to determine if the component exists.
Bool AND which will return both parameters
Bool OR which will return either one of the parameters.

```
Component::Component()-Our default constructor

Component::Component(string identity, vector<string> members)-Main
constructor for our Components. This will take in a string for its
identity along with a vector of 'members' that represent the arguments
for a component.

Component::Component(const Component & comp)-Copy constructor

Component(Component && p);

bool find(vector<string> members); //Determines if this Component
exists.

bool Component::AND(bool first, bool second)-Method for AND operator
```

```
bool Component::OR(bool first, bool second)-Method for OR operator

Component::~Component-Deconstructor
```

Class Fact : Component:

Fact

Fact class inherits from Component

It contains a string called association which is the association **i.e Father, Mother, etc**

A vector of strings which represent the members of the fact, **i.e Mary, Romeo, Paul, etc**

A potential Boolean factExsists method which will take in a vector and check the database if the Fact exists, but we might not need it.

```
Fact(string identity, vector<string> members); //Main
//constructor for a Fact.
Fact(const Fact & f); //Copy constructor.
Fact(Fact && f); //Move constructor.

string association; //The association of the fact i.e Father,
//Mother, etc...
vector<string> members; //The members of the fact i.e John,
//Paul, Jones, etc...
// bool factExists(vector<string> members); //Takes in a vector
that represents the members of a fact.
```

Class Rule : Component

Rule class also inherits from Component.

Similar to the Fact class data structure.

Contains a string called association which is the association of the rule **i.e Grandma, Grandpa, Parent**

A vector of Component* elements

A logical operator – a vector of ints.

A Boolean validate functions which determines if a rule is valid based on the elements its built from.

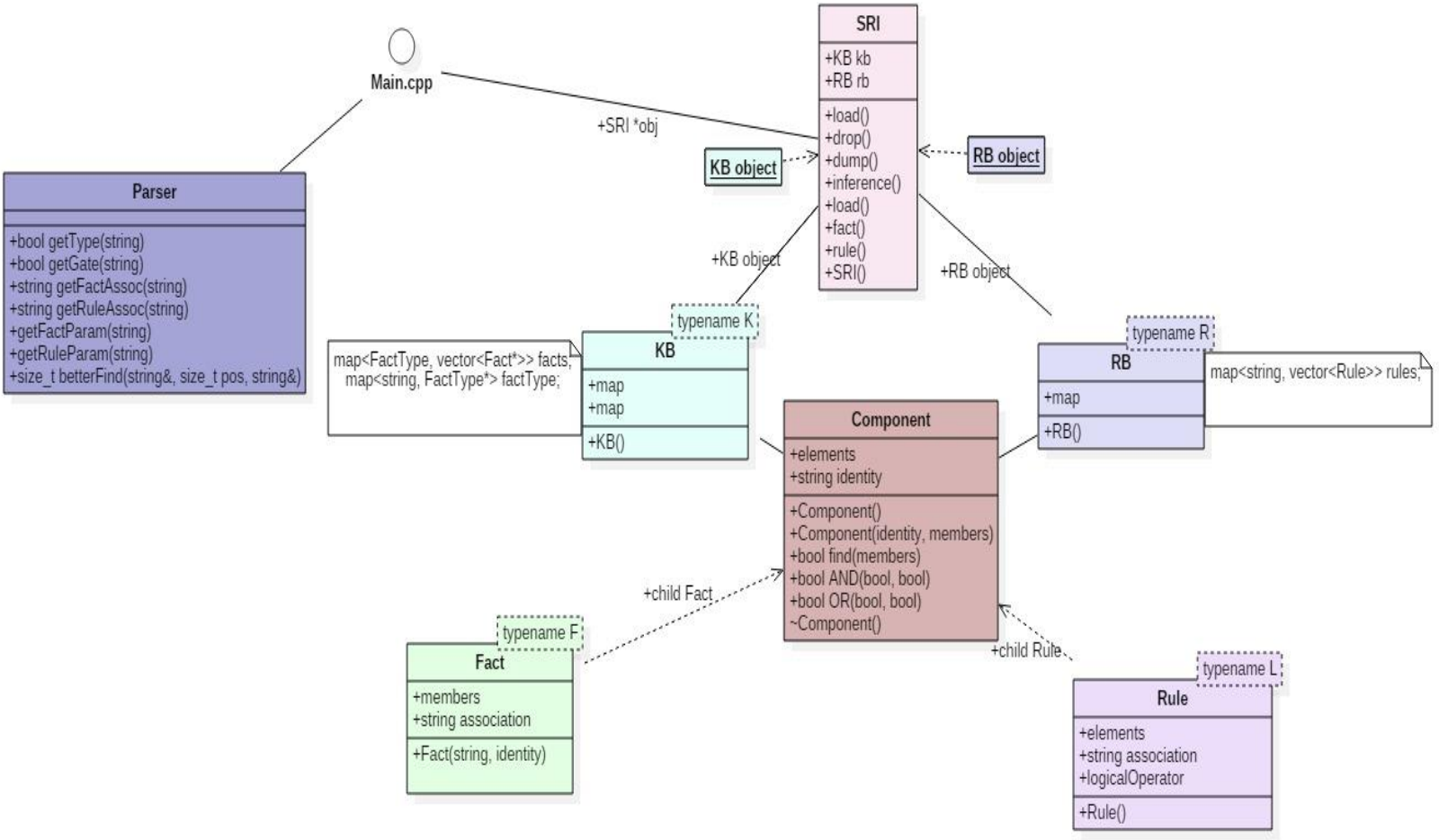
```
Rule();
Rule(string identity, vector<Component*> elements, vector<bool>
operators); //constructor that can accept rules with multiple
//components in a vector of type Component.
Rule(const Rule & r); //copy constructor.
```

```
Rule(Rule && r); //Move constructor.

string association; //The association of the rule i.e
//Grandfather, Grandmother, Parent, etc...
vector<Component*> elements;
vector<int> functionOperator; //These point to AND/OR, similar
//to boolean ops.

bool validate(vector<string> members); //Determines if a rule is
//valid based on the elements it is built from.
```

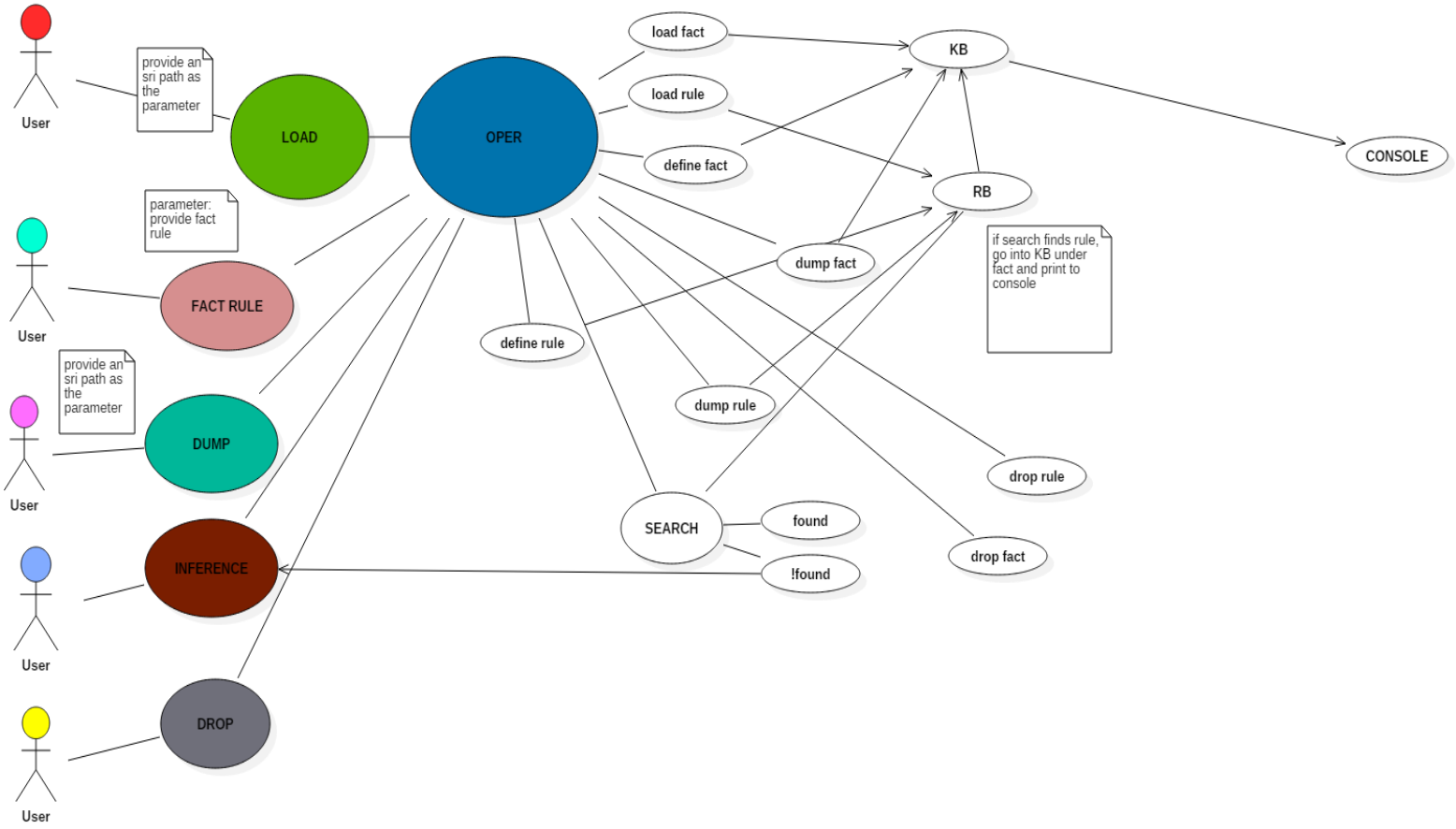
Main.cpp
This is the main function which has one SRI pointer object. Provides the user with a menu, takes in user input and performs the operations.

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

Updated Class Diagram

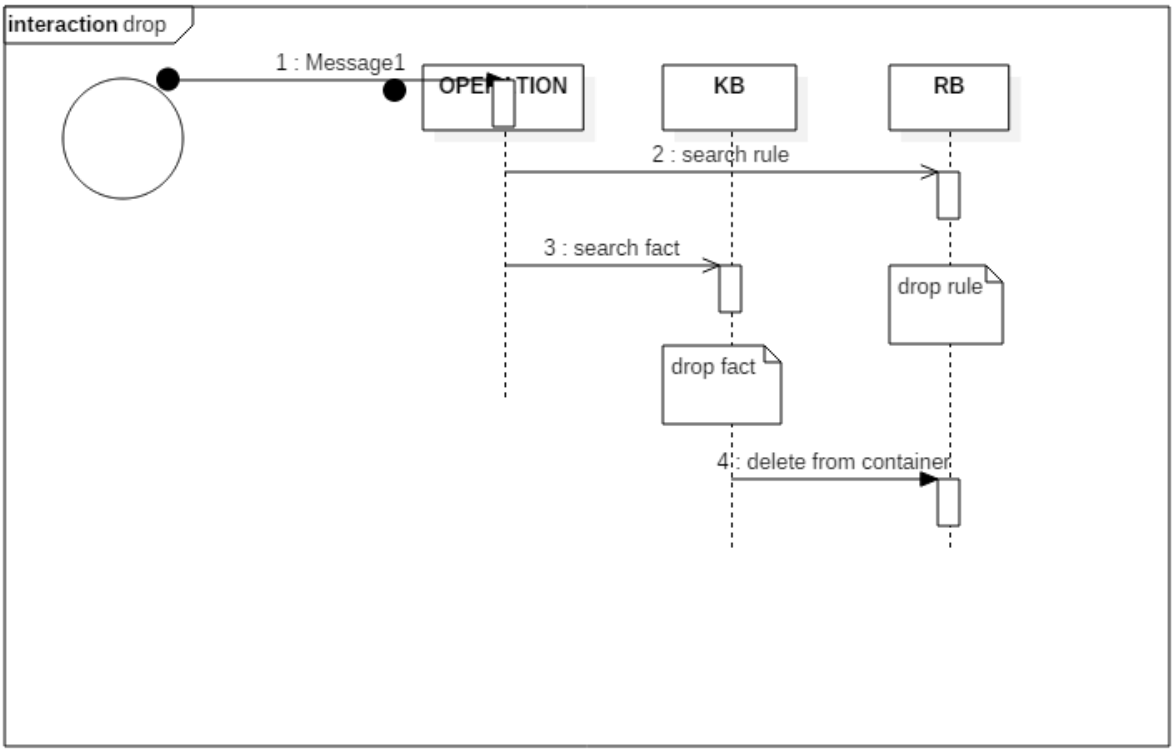


USE CASE DIAGRAM

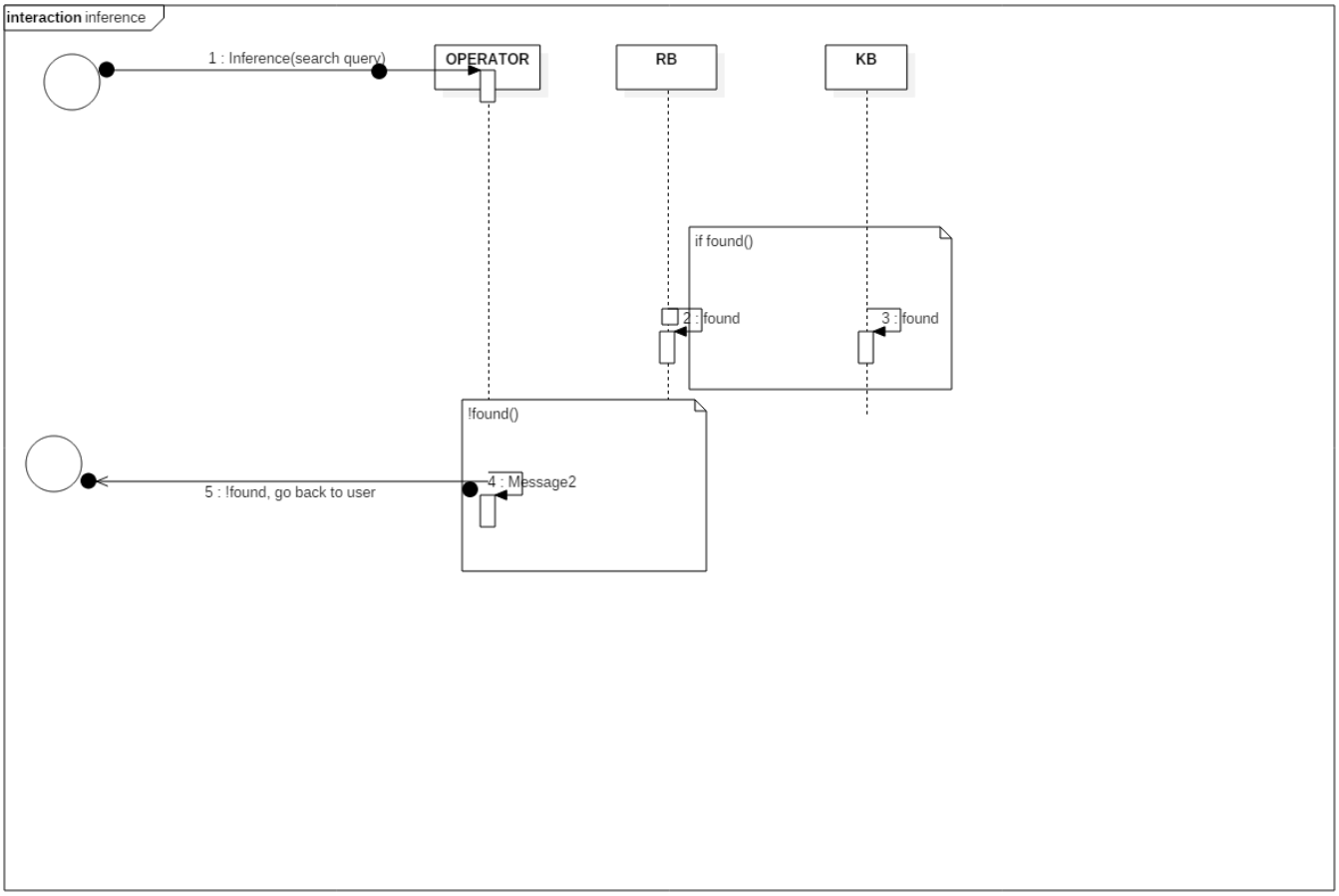


SEQUENCE DIAGRAM

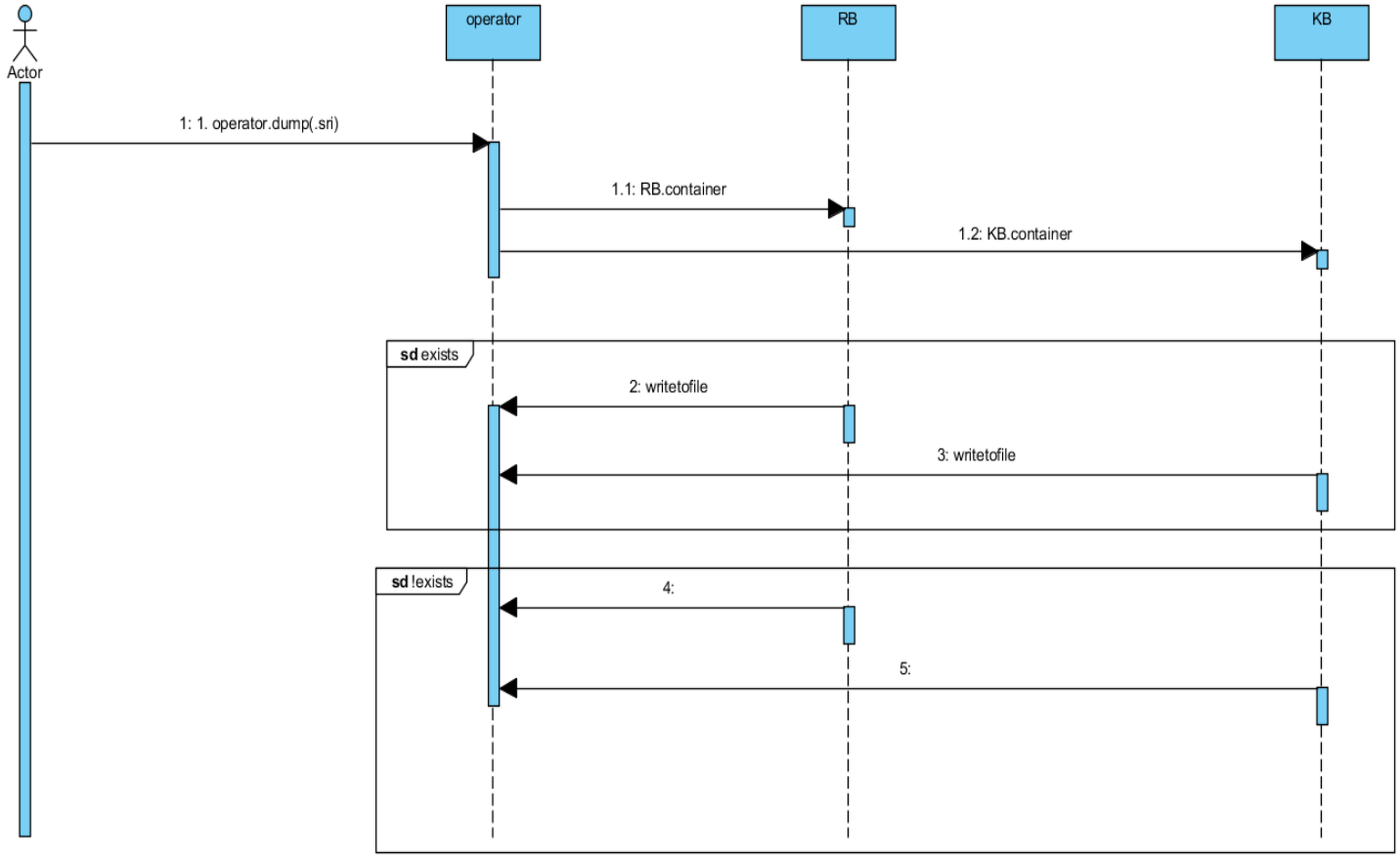
DROP



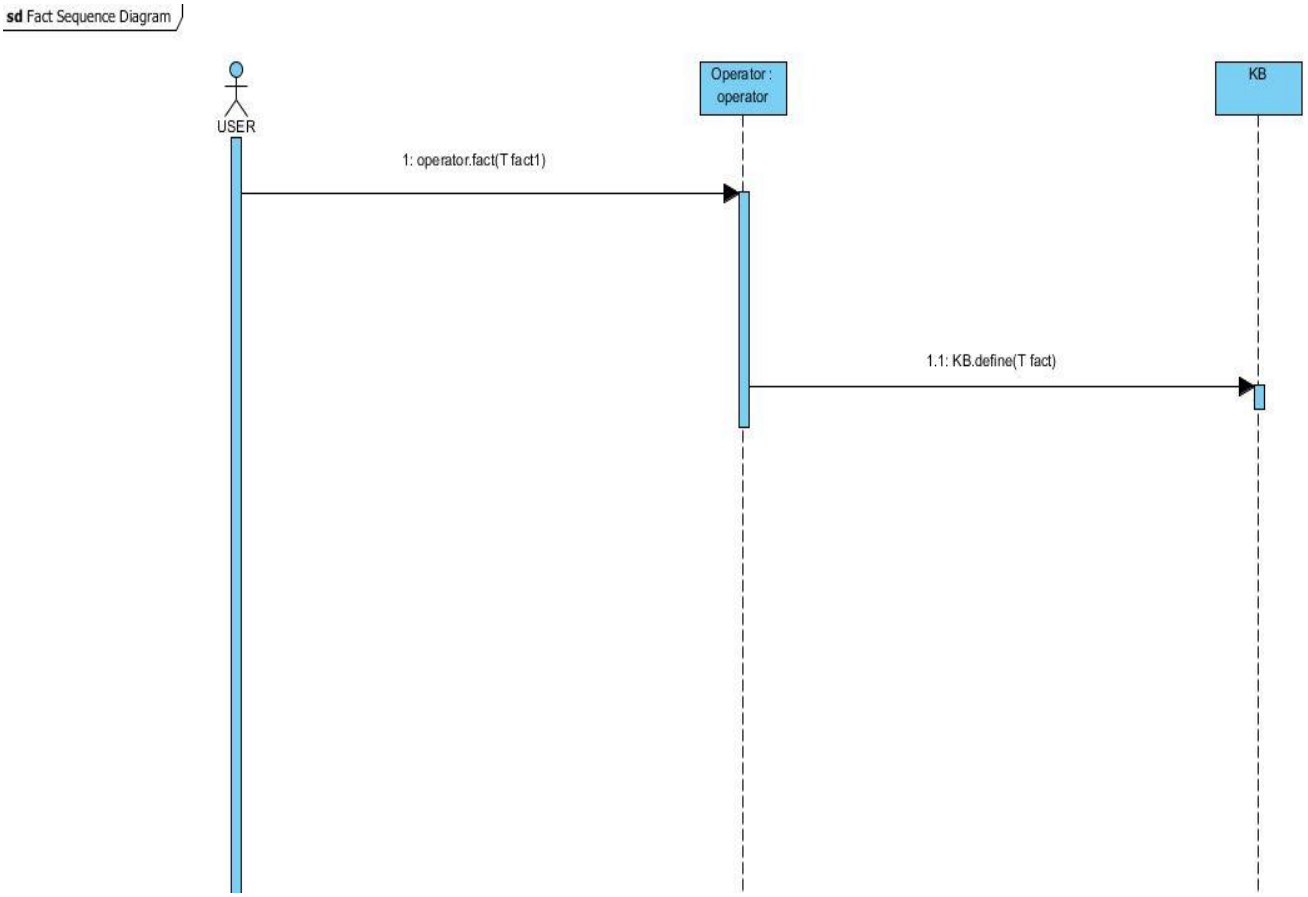
INFERENCE


interaction inference
1 : Inference(search query)
OPERATOR
RB
KB
if found()
2 : found
3 : found
!found()
4 : Message2
5 : !found, go back to user

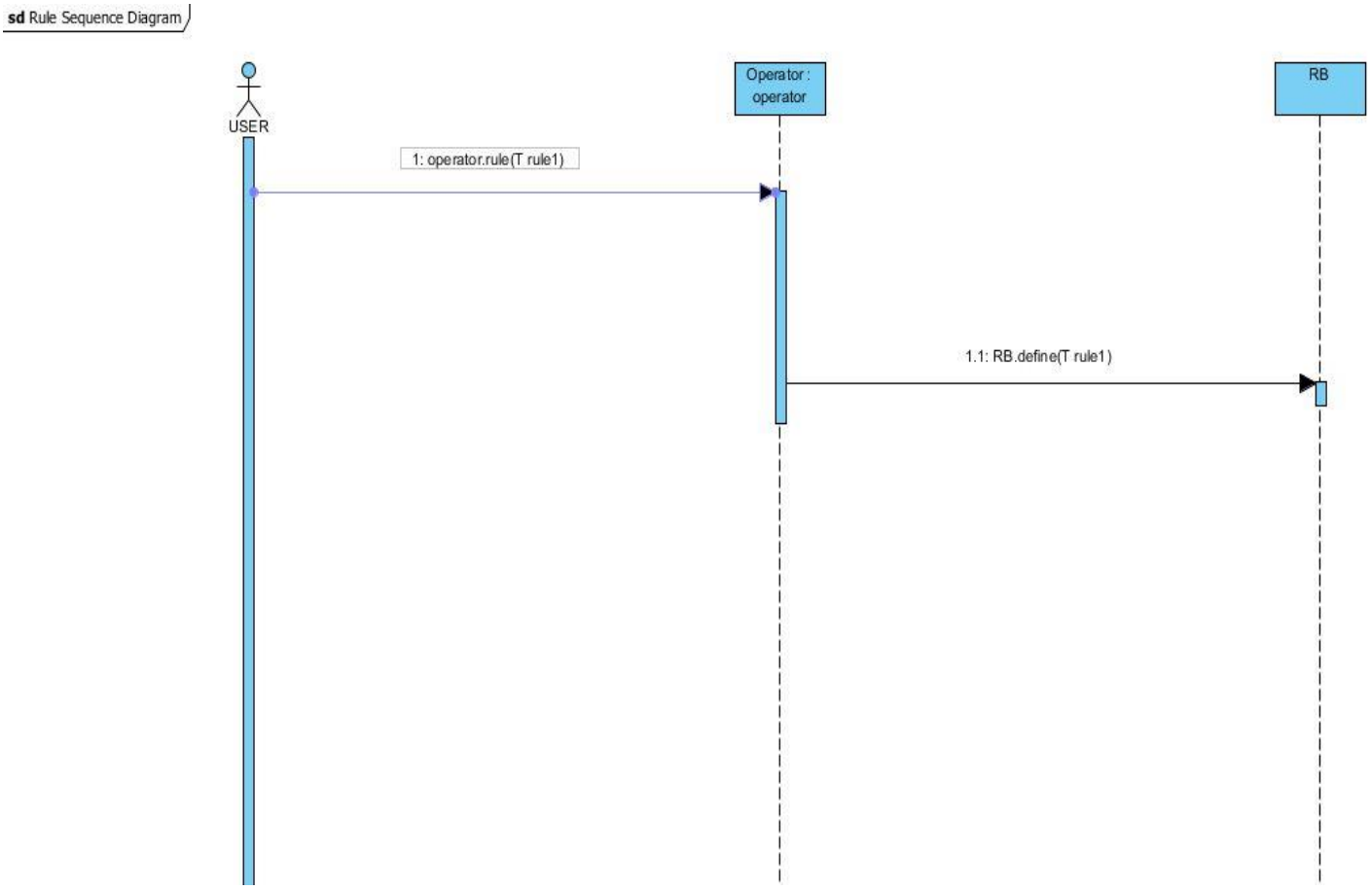
DUMP


Actor
operator
RB
KB
1: 1. operator.dump(.sri)
1.1: RB.container
1.2: KB.container
sd exists
2: writetofile
3: writetofile
sd !exists
4:
5:

DEFINE FACT


sd Fact Sequence Diagram
USER
Operator : operator
KB
1: operator.fact(T fact1)
1.1: KB.define(T fact)

DEFINE RULE


sd Rule Sequence Diagram
USER
Operator : operator
RB
1: operator.rule(T rule1)
1.1: RB.define(T rule1)

LOAD RULE/FACT