SSMMP/1.1 communication APIs: the code and its detailed explanation

Stanisław Ambroszkiewicz, Institute of Computer Science, University of Siedlce, Poland,
stanislaw.ambroszkiewicz@uws.edu.pl

Formal specification of SSMMP is at https://arxiv.org/abs/2305.16329 , and also in the companion SSMMPv1.1_specification.pdf

Contents:

- Java API for service (opening and closing TCP connections, terminating service instances)
- Java class Agent (execution and killing service instances; message forwarding)
- Java API for Manager (execution requests, reconfiguration, recovery from failures, and termination of CNApp)

1. Java API for service

// all needed parameters (taken from messages of SSMMP) for the constructor of the API

```
/*
service_name: A
service_instance_id: i
service_instance_address: NA_i
socket_configuration: [configuration of sockets]
plug_configuration: [configuration of plugs]
*/
```

/*
Note that the above parameters are also necessary for execution of an instance of service A by an Agent. Manager assigns a unique identifier, say i, (a positive integer) to a new instance of A to be executed. Manager also determines port numbers to all sockets of the instance. It is called socket configuration, and is an object of class String in the form of a sequence of pairs (separated by `" & "`):

```
                    socket_name, port_number
```

Configuration of plugs is an object of class String in the form of a sequence of triples (separated by `" & "`):

```
                    plug_name, socket_name, service_name
```

Each of such configurations corresponds to a connection, say (A, (P,S), B), of the abstract graph of CNApp.
These `plug_name` (P) and `socket_name` (S) are in the code of service A. They are assigned `service_name` (B) by Manager to instance i of service A according to the CNApp abstract graph. In other words, each configuration of a plug means that this abstract plug P is assigned a service name B, where the corresponding abstract socket S is constructed as a concrete server socket. This abstract pug and this abstract socket make up together a communication protocol between this instance i of service A, and an instance of the service B (`service_name`).

Manager also determines a network address (denoted NA_i) of the node where the instance i of A is to be executed by the Agent residing on that node. % It is also the network address of the Agent.
*/

```
public class ServiceComAPI {
```

```
static final String thisServiceName; // == "A"
static final String serviceInstanceId; // == "i"
static final String serviceInstanceNetworkAddress;
// == "NA_i"
static final String socketConfiguration;
// == [configuration of sockets];
static final String plugConfiguration;
// == [configuration of plugs];
// The state of ServiceComAPI consists of the values of the above objects, and
        // communications sessions collected in a vector defined below

    public static void main(String[] args) {

//   --- Let's introduce the basic data structures of ServiceComAPI ---

// create a vector from socketConfiguration;
// == [configuration of sockets];

// an example of a single configuration of a server socket
/*
String[] serverSocketConfig =
{
"socket_name: S",
"port_number: 12564"  // determined by Manager
}
*/
 // all elements of the sequence [configuration of sockets]
        // are added to vector serverSocketConfigVec.add(serverSocketConfig);

Vector<String[]> serverSocketConfigVec = new
Vector<String[]>();

String[] serverSocketConfigArray =
socketConfiguration.split(" & ");

for( String config : serverSocketConfigArray) {

String[] configInLines = config.split(", ");
String[] serverSocketConfig =
{
"socket_name: " + configInLines[0],
"port_number: " + configInLines[1],
};
serverSocketConfigVec.add(serverSocketConfig);
};


// create a vector from plugConfiguration; // == [configuration of plugs];
// an example of a single plug configuration
/*
String[] plugConfig =
{
```

```
"plug_name: P",
"socket_name: S",
"service_name: B"
};
*/

// all elements of the sequence [configuration of plugs]
// are added to the vector plugConfigVec.add(plugConfig);

Vector<String[]> plugConfigVec = new Vector<String[]>();

String[] plugConfigArray = plugConfiguration.split(" & ");

for( String config : plugConfigArray) {

String[] configInLines = config.split(", ");
String[] plugConfig =
{
"plug_name: " + configInLines[0],
"socket_name: " + configInLines[1],
"service_name: " + configInLines[2]
};
plugConfigVec.add(plugConfig);
};


// --- Let's introduce also two important objects:

String pendingOffOnReguestMessageId = null;
// The value of this object is set to "n" taken from "message_id: n"
        // of the Manager's request message of
        // (type: graceful_shutdown_request) to shutdown the service instance

String serviceInstanceStatus = null;
// the possible values are as follows:
// "on" means that the instance i of A is running,
// "pendingOff" means that the instance i of A is to be closed according to business logic,
        // and termination_info message of that closing is to be sent to Agent.
// "pendingOffConfirmed" means that the termination_info message has been already
        // sent, and System.exit() can be performed.
// "pendingOffOnRequest" means that the instance i of A is to be terminated on
        // Manager'request, and response to that request is to be sent to Agent.
// "pendingOffOnRequestConfirmed" means that the
        // graceful_shutdown_response message has been already sent,
        // and System.exit() can be performed.


// ---
// let's introduce the key data structure for communication session initially constructed immediately
        // after construction of an object of class Socket
// the sessions are stored in the following vector
```

```
Vector<String[]> sessionVec = new Vector<String[]>();

//  vector of current open communication sessions; each of the sessions is a String[] with fixed
        // number of elements indexed by 0,1,2, ... ,13,14,15,16

// The following object is an example of session of type plug for a communication session of
        // connection (A, (P,S), B) from the point of view of instance i of service A.
// The session is created, in the code of A, after constructing
        // Socket P = new Socket(NA_j, k),
        // where NA_j  is the network address of an instance of service B ,
        // and k is the port number of socket S of B
// an example of session of type plug
/*
String[] session =
{
"session_type: plug",
"session_status: on",
        // meaning that the session is already established;
        // or: off, pendingRequest, pendingResponse, pendingOn,
                // onToCofirm, pendingClose, broken
"session_id: m",  // m is from the line "source_plug_port: m"  below
"session_request_id: n",
        // n is from the line "message_id: n" of the message (type: session_request)
                // requesting this session establishing.
"request_to_close: false",
        // or true, then, the session must be closed (on a request of Manager)
                // by closing the plug on port m
        // below is the complete list of the parameters of a communication session
"source_service_name: A",
"source_service_instance_network_address: NA_i",
"source_service_instance_id: i",
"source_plug_name: P",
"source_plug_port: m",
"dest_service_name: B",
"dest_service_instance_network_address: NA_j",
"dest_service_instance_id: j",
"dest_socket_name: S",
"dest_socket_port: k",
"dest_socket_new_port: l",
"session_close_request_message_id: o"
}
*/
// each string is of the form "parameter: value".
// The parameters are fixed whereas values are subject of change.
// Some values may be unspecified, especially in  "dest_service_instance_id: j"

// Add the above object to  sessionVec
// sessionVec.add(session);

// The object below is an example of a communication session of connection (C, (P1,S1), A)
        // from the point of view of an instance i of service A.
// The session is of type socket created after constructing  ServerSocket S1,
        // and, then socket = S1.accept() in the code of service A.
// an example of such session
/*
String[] session =
{
"session_type: socket",
"session_status: on", // or: off, pendingClose, broken
"session_id: k", // k is from the line "source_plug_port: k" below
"session_request_id: ",
"request_to_close: false",
        // or true, then, the session must be closed (on a request of Manager)
                // by closing the socket on the port l from
                // the line "dest_socket_new_port: l",
"source_service_name: C",
"source_service_instance_network_address: NA_j",
"source_service_instance_id: j",
"source_plug_name: P1",
"source_plug_port: k",
"dest_service_name: A",
"dest_service_instance_network_address: NA_i",
"dest_service_instance_id: i",
"dest_socket_name: S1",
"dest_socket_port: m",
"dest_socket_new_port: l",
"session_close_request_message_id: ";
}
*/
// some values of the parameters may be unspecified
// add the above object to sessionVec
// sessionVec.add(session);


// --- the basic functionality of ServiceComAPI is as follows ---

Socket serviceToAgentSocket = new Socket (localhost, 55556);
// in ssmmp, the port number 55556 is fixed for communication from service instance to Agent
// Agent is on the same host (localhost)

// input stream inFromAgent, and output stream outToAgent

PrintWriter outToAgent = new
PrintWriter(serviceToAgentSocket.getOutputStream(), true);

BufferedReader inFromAgent = new BufferedReader(new
InputStreamReader(serviceToAgentSocket.getInputStream()));


// ----------------------------
// the main while loop is for:
// (1) sending messages (initiated by this instance i of A) to Agent, and sending responses to Agent,
// (2) receiving messages from Agent, and processing them.

int messageIdCounter = 1000;
```

```
public String getNewMessageId(){
String mId = String.valueOf(messageIdCounter);
messageIdCounter++;
return mId;
};

String messageId = null;
String messageToSend = null;
String[] sessionToSet = new String[16];
String[] sessionToRemove = new String[16];
int index = 0;
boolean check = true;

while(true) {
// (1) sending messages (this instance i of service A) to Agent via  outToAgent:
//   (1.1) termination of instance i of A (refer to Section 2.3 Service instance termination):
//     (1.1.1) resulting from business logic,
//     (1.1.2) resulting from request of Manager.
//   (1.2) request for session establishing.
//   (1.3) session_ack.
//   (1.4) confirmation of session closing resulting from business logic.
//   (1.5) confirmation (response) of session closing on request.

// ----- let us start to code
// (1.1) termination of instance i of A.
// Check the value of object serviceInstanceStatus  before  System.exit(0);

// ---
// (1.1.1) resulting from business logic,
if (serviceInstanceStatus.equals("pendingOff")){

// send termination_info (message) to Manager via Agent.
// the format of this message is as follows
/*
type: service_instance_termination_info
message_id: n
sub_type: service_instance_to_agent
service_name: A
service_instance_id: i
source_service_instance_network_address: NA_i
*/

messageId = getNewMessageId();

//  "  ~  "  is line separator in the message
messageToSend = "type:  service_instance_termination_info ~
message_id: " + messageId  + " ~ sub_type:
service_instance_to_agent ~
service_name: " + thisServiceName + " ~ service_instance_id: "
+ serviceInstanceId + " ~
```

```
source_service_instance_network_address: " +
serviceInstanceNetworkAddress;

outToAgent.print(messageToSend);

// display the message
// System.out.println(messageToSend);

serviceInstanceStatus = "pendingOffConfirmed";

}

// ---
// (1.1.2) resulting from request of Manager,
if
(serviceInstanceStatus.equals("pendingOffOnRequestToConfirm"))
{

// send message to Manager via Agent
//  "  ~  "  is line separator in the message
messageToSend = "type: graceful_shutdown_response ~
message_id: " + pendingOffOnReguestMessageId + " ~ sub_type:
service_instance_to_agent ~ status: 200";

outToAgent.print(messageToSend);

// display the message
// System.out.println(messageToSend);

serviceInstanceStatus = "pendingOffOnRequestConfirmed";

};


// ------
// (1.2) request for session establishing. Refer to 2.1 Session establishing for connection (A, (P,S), B)
// find session in vector sessionVec such that "session_status: pendingRequest"

index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if(sessionVec.get(index)[1].equals("session_status:
pendingRequest")){
check = false;
sessionToSet = sessionVec.get(index);
     };
index++;
}; // end of while loop

if (find.equals(false)) {
```

```
// modify sessionToSet
sessionToSet[1] = "session_status: pendingResponse";

index--;
// modify  sessionVec
sessionVec.set(index, sessionToSet);

// create the session_request message on the basis of this session, and send the message
        // to the Agent
// format of the session_request  message
/*
type: session_request
message_id: n
sub_type: service_to_agent
source_service_name: A
source_service_instance_id: i
source_service_instance_network_address: NA_i
source_plug_name: P
dest_service_name: B
dest_socket_name: S
*/

//note that   sessionToSet[3]  is the string of the form  "session_request_id: n"
        //where n is from the line "message_id: n"
        // of the message  type: session_request
        // this very message requesting this session establishing.

String[] line = sessionToSet[3].split(" ");
messageId = line[1];

//  "  ~  "  is line separator in the message
messageToSend = "type: session_request ~ message_id: " +
messageId + " ~ sub_type: service_to_agent ~ " +
sessionToSet[5] + " ~ " + sessionToSet[7] + " ~ " +
sessionToSet[6] + " ~ " + sessionToSet[8] +" ~ " +
sessionToSet[10] + " ~ " + sessionToSet[13];

outToAgent.print(messageToSend);

// display the message
// System.out.println(messageToSend);

};  // end of  if


// -----
// (1.3) session_ack.
//find session  in vector sessionVec such that "session_status: onToCofirm"

// resetting variables for the next while loop
sessionToSet = null;
// or set all elements of sessionToSet to empty string " "
```

```
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if(sessionVec.get(index)[1].equals("session_status:
onToCofirm")){
check = false;
sessionToSet = sessionVec.get(index);
      };
index++;
}; // end of while loop

if (check.equals(false)){
// modify sessionToSet
sessionToSet[1] = "session_status: on";
index--;
// modify sessionVec
sessionVec.set(index, sessionToSet);

// create the session_ack message on the basis of this session, and send the message
// to the Agent
//note that sessionToSet[3] is of the form "session_request_id: n"

String[] line = sessionToSet[3].split(" ");
messageId = line[1];

// format of this message
/*
type: session_ack
message_id: n
sub_type: service_to_agent
source_service_name: A
source_service_instance_id: i
source_plug_port: m
dest_socket_new_port: l
status: [status code]
*/

// " ~ " is line separator in the message
messageToSend = "type: session_ack ~ message_id: " + messageId
+ " ~ sub_type: service_to_agent ~ service_name: " +
thisServiceName + " ~ service_instance_id: " +
serviceInstanceId +  " ~ " + sessionToSet[9] + " ~ " +
sessionToSet[15] + " ~ status: 200";

outToAgent.print(messageToSend);

// display the message
// System.out.println(messageToSend)

}; // end of if
```

```
// ------
// (1.4) session closing resulting from business logic or from an exception

//find session  in vector sessionVec such that "session_status: off"
// or ("session_status: broken")
// and   "request_to_close: false"

// resetting variables for the next while loop
sessionToSet = null;
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if(
( (sessionVec.get(index)[1].equals("session_status: off")
||
(sessionVec.get(index)[1].equals("session_status: broken")
) )
&& (sessionVec.get(index)[4].equals("request_to_close:
false")) ) {
check = false;
sessionToRemove = sessionVec.get(index);
     }; // end of if
index++;
}; // end of while loop

if (check.equals(false)) {

// Remove the session from sessionVec.
index--;
sessionVec.remove(index);

messageId = getNewMessageId();

if(sessionToRemove[0].equals("session_type: plug")) {

// create (on the basis of this session) message source_service_session_close_info
/* format of this message
type: source_service_session_close_info
message_id: n
sub_type: source_service_to_agent
source_service_name: A
source_service_instance_id: i
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_service_name: B
dest_service_instance_id: j
dest_service_instance_network_address: NA_j
dest_socket_name: S
dest_socket_port: k
```

```
dest_socket_new_port: l
status: [status code]
*/

if(sessionToRemove[1].equals("session_status: off"){

//  "  ~  "   is line separator in the message
messageToSend = "type: source_service_session_close_info ~
message_id: " + messageId + " ~ sub_type:
source_service_to_agent ~ " + sessionToRemove[5] + " ~ " +
sessionToRemove[7] + " ~ " + sessionToRemove[6] + " ~ " +
sessionToRemove[8] + " ~ " + sessionToRemove[9] + " ~ " +
sessionToRemove[10] + " ~ " + sessionToRemove[12] + " ~ " +
sessionToRemove[11] + " ~ " + sessionToRemove[13] + " ~ " +
sessionToRemove[14] + " ~ " + sessionToRemove[15] + " ~ status:
111";

}else{  // note that only the status is to change

messageToSend = "type: source_service_session_close_info ~
message_id: " + messageId + " ~ sub_type:
source_service_to_agent ~ " + sessionToRemove[5] + " ~ " +
sessionToRemove[7] + " ~ " + sessionToRemove[6] + " ~ " +
sessionToRemove[8] + " ~ " + sessionToRemove[9] + " ~ " +
sessionToRemove[10] + " ~ " + sessionToRemove[12] + " ~ " +
sessionToRemove[11] + " ~ " + sessionToRemove[13] + " ~ " +
sessionToRemove[14] + " ~ " + sessionToRemove[15] + " ~
status: 122";
};  // end of else
      };  // end of if


if(sessionToRemove[0].equals("session_type: socket")) {

// create message dest_service_session_close_info
/*  format of this message
type: dest_service_session_close_info
message_id: o
sub_type: agent_to_Manager
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_service_name: B
dest_service_instance_network_address: NA_j
dest_service_instance_id: j
dest_socket_name: S
dest_socket_port: k
dest_socket_new_port: l
status: [status code]
*/

if(sessionToRemove[1].equals("session_status: off"){
```

```
//  "  ~  "  is line separator in the message
messageToSend = "type: dest_service_session_close_info ~
message_id: " + messageId + " ~ sub_type:
dest_service_to_agent ~ " + sessionToRemove[6] + " ~ " +
sessionToRemove[8] + " ~ " + sessionToRemove[9] + " ~ " +
sessionToRemove[10] + " ~ " + sessionToRemove[11] + " ~ " +
sessionToRemove[12] + " ~ " + sessionToRemove[13] + " ~ " +
sessionToRemove[14] + " ~ " + sessionToRemove[15] + " ~ status:
111";

}else{    // note that only the status is to change

messageToSend = "type: dest_service_session_close_info ~
message_id: " + messageId + " ~ sub_type:
dest_service_to_agent ~ " + sessionToRemove[6] + " ~ " +
sessionToRemove[8] + " ~ " + sessionToRemove[9] + " ~ " +
sessionToRemove[10] + " ~ " + sessionToRemove[11] + " ~ " +
sessionToRemove[12] + " ~ " + sessionToRemove[13] + " ~ " +
sessionToRemove[14] + " ~ " + sessionToRemove[15] + " ~ status:
122"
};   // end of else

     };  // end of  if

// Send the message to Agent
outToAgent.print(messageToSend);

// display the message
// System.out.println(messageToSend);

};  // end of  if


// -----
// (1.5) confirmation (response) of session closing on request from Manager

// find session  in vector sessionVec such that "session_status: off"
// and  "request_to_close: true"

// resetting variables for the next while loop
sessionToSet = null;
int index = 0;
boolean check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if((sessionVec.get(index)[1].equals("session_status: off")) &&
(sessionVec.get(index)[4].equals("request_to_close: true")) )
{
check = false ;
sessionToRemove = sessionVec.get(index);
```

```
     };
index++;
}; // end of while loop

if (check.equals(false)){

// Remove the session from sessionVec.
index--;
sessionVec.remove(index);

String[] line = sessionToRemove[16].split(" ");
messageId = line[1];
// two cases: session type plug or session type socket

if(sessionToRemove[0].equals("session_type: plug")) {
// create (on the basis of this session) message of the form
/*
type: source_service_session_close_response
message_id: o
sub_type: source_service_to_agent
status: [status code]
*/

// " ~ " is line separator in the message
messageToSend = "type: source_service_session_close_response
~ message_id: " + messageId + " ~ sub_type:
source_service_to_agent ~ status: 200";

     }; // end of if

if(sessionToRemove[0].equals("session_type: socket")) {

// create (on the basis of this session) message of the form
/*
type: dest_service_session_close_response
message_id: o
sub_type: dest_service_to_agent
status: [status code]
*/

// " ~ " is line separator in the message
messageToSend = "type: dest_service_session_close_response
~ message_id: " + messageId + " ~ sub_type:
dest_service_to_agent ~ status: 200";

     }; // end of if

// Send the message to Agent
outToAgent.print(messageToSend);

// display the message
// System.out.println(messageToSend);
```

```
}; // end of if


// ---------
// (2) receiving messages from Agent via inFromAgent, and processing them:
//  (2.1) session_response
//  (2.2) session_close_request
//    (2.2.1) source_service_session_close_request
//    (2.2.2) destination_service_session_close_request
//  (2.3) graceful_shutdown_request
//  (2.4) health_control_request


// ----- declaration of objects

String messageFromAgent = inFromAgent.readLine();

String[] messageInLines = messageFromAgent.split(" ~ ");

// display the message
for (String row : messageInLines)
{
System.out.println(row + "\n");
};

String[] line = messageInLines[1].split(" ");
messageId = line[1];


// (2.1) session_response

// format of this message
/*
type: session_response
message_id: n
sub_type: agent_to_service
source_service_name: A
source_service_instance_id: i
dest_service_name: B
dest_service_instance_id: j
dest_socket_name: S
dest_service_instance_network_address: NA_j
dest_socket_port: k
status: [status code]
*/

if (messageInLines[0].equals("type: session_response")) {

String[] line6 = messageInLines[6].split(" ");
String destServiceInstanceId = line6[1];
```

```
String[] line8 = messageInLines[8].split(" ");
String destServiceInstanceNetworkAddress = line8[1];

String[] line9 = messageInLines[9].split(" ");
String destSocketPort = line9[1];

// Get the session with "session_request_id: " + messageId from sessionVec

// resetting variables for the next while loop
sessionToSet = null;
// or set all elements of sessionToSet  to empty string  ""
int index = 0;
boolean check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if( sessionVec.get(index)[3].equals("session_request_id: " +
messageId ) ){
check = false;
sessionToSet = sessionVec.get(index);
     };
index++;
}; // end of while loop

if (check.equals(false)){

// modify sessionToSet
sessionToSet[1] = "session_status: pendingOn";

sessionToSet[12] = "dest_service_instance_id: " +
destServiceInstanceId;

sessionToSet[11] = "dest_service_instance_network_address: " +
destServiceInstanceNetworkAddress;

sessionToSet[14] = "dest_socket_port: " + destSocketPort;

// modify sessionVec
index--;
sessionVec.set(index, sessionToSet);

};


// ---
// (2.2.1) source_service_session_close_request

// session type: plug , connection (A, (P,S), B)
// by default "session_status: on",
// format of this message
/*
type: source_service_session_close_request
```

```
message_id: o
sub_type: agent_to_source_service
source_service_name: A
source_service_instance_id: i
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_service_name: B
dest_service_instance_network_address: NA_j
dest_socket_name: S
dest_socket_port: k
dest_socket_new_port: l
*/

if(messageInLines[0].equals("type:
source_service_session_close_request") ){

String[] line7 = messageInLines[7].split(" ");
String sourcePlugPort = line7[1];
// recall that sourcePlugPort is also the session identifier

// resetting variables for the next while loop
sessionToSet = null;
// or set all elements of sessionToSet  to empty string  ""
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if( (sessionVec.get(index)[0].equals("session_type: plug")) &&
(sessionVec.get(index)[9].equals("source_plug_port: " +
sourcePlugPort) ) ){
check = false;
sessionToSet = sessionVec.get(index);
      };
index++;
}; // end of while loop

if (check.equals(false)){

index--;

// modify sessionToSet
sessionToSet[1] = "session_status: pendingClose";
sessionToSet[4] = "request_to_close: true";
sessionToSet[16] = "session_close_request_message_id: " +
messageId;

// modify sessionVec
sessionVec.set(index, sessionToSet);

      };
};
```

```
// ---
// (2.2.2) destination_service_session_close_request

// session type: socket , connection (C,(P,S), A)
// by default "session_status: on",

// format of this message is as follows
/*
type: dest_service_session_close_request
message_id: o
sub_type: agent_to_dest_service
dest_service_name: A
dest_service_instance_id: j
dest_service_instance_network_address: NA_j
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_socket_name: S
dest_socket_port: k
dest_socket_new_port: l
*/

if (messageInLines[0].equals("type:
dest_service_session_close_request")) {

String[] line11 = messageInLines[11].split(" ");
String destSocketNewPort = line11[1];

// Get the session form sessionVec with
// "session_type: socket"
// and
// "dest_socket_new_port: " + destSocketNewPort

// resetting variables for the next while loop
sessionToSet = null;
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){

if( (sessionVec.get(index)[0].equals("session_type: socket"))
&& (sessionVec.get(index)[15].equals("dest_socket_new_port: "
+ destSocketNewPort) ) ) {

check = false;
sessionToSet = sessionVec.get(index);
     };
index++;
}; // end of while loop
if (check.equals(false)){

index--;

// modify sessionToSet
sessionToSet[1] = "session_status: pendingClose";
sessionToSet[4] = "request_to_close: true";
sessionToSet[16] = "session_close_request_message_id: " +
messageId;

// modify sessionVec
sessionVec.set(index, sessionToSet);
          }
};


// -----
// (2.3) graceful_shutdown_request

// format of this message is as follows
/*
type: graceful_shutdown_request
message_id: o
sub_type: agent_to_service_instance
service_name: A
service_instance_id: i
*/

if (messageInLines[0].equals(" type:
graceful_shutdown_request")) {

serviceInstanceStatus = "pendingOffOnRequest";
pendingOffOnReguestMessageId = messageId;
// see page  3 and 4 for the meaning of these two object above
};

//  (2.4) health_control_request

// format of this message
/*
type: service_instance_health_request
message_id: o
sub_type: agent_to_service_instance
service_instance_id: i
service_instance_network_address: NA_i
*/
// format of response
/*
type: service_instance_health_response
message_id: o
sub_type: service_instance_to_agent
service_instance_id: i
status: [status code]
```

```
*/

if(messageInLines[0].equals("type:
service_instance_health_request")){

serviceInstanceId = messageInLines[3].split(" ")[1];

String status = null;
// value 200 means OK., value 199 means overload related
// to the number of open, busy communication sessions

// " ~ " is line separator in the message
messageToSend = " type: service_instance_health_response" ~
message_id: " + messageId + " ~ sub_type:
service_instance_to_agent ~ service_instance_id: " +
serviceInstanceId + " ~ status: " + status;

// Send the message to Agent
outToAgent.print(messageToSend);

// display the message
// System.out.println(messageToSend);

};

}// end of the main while loop of class ServiceComAPI
```

2. Instructions to inject into the code of service A

2.1 Session establishing for connection (A, (P,S), B)

```
// recall that
// String thisServiceName;    // == "A"

// String serviceInstanceId;  //  == "i"

// String serviceInstanceNetworkAddress;  // == "NA_i"

// the code to be insert before client (plug) constructor
//Socket clientSocket = new Socket( , );

String sourcePlugName P;

// by default destination socket name is S, i.e. (P,S) denotes the communication protocol
String destSockeName = S;
String destServiceInstanceNetworkAddress = null;
String destSocketPort = null;
```

```
for( String[] plugConfig : plugConfigVec ) {
           if( plugConfig[0].equals(sourcePlugName) &&
plugConfig[1].equals(destSockeName) ){

destServiceName = plugConfig[2]; // denoted B

          }

};
```

```
// network address of an instance of B, and port of its socket S are needed
// Create new session for connection (A, (P,S), B)

String sessionRequestId = getNewMessageId(){;

String[] plugSession =
{
"session_type: plug",
"session_status: pendingRequest",
"session_id: ",
"session_request_id: " + sessionRequestId,
"request_to_close: false",
"source_service_name: " + thisServiceName,
"source_service_instance_network_address: " +
     serviceInstanceNetworkAddress,
"source_service_instance_id: " + serviceInstanceId,
"source_plug_name: " + sourcePlugName,
"source_plug_port: ,
"dest_service_name: " + destServiceName,
"dest_service_instance_network_address: ",
"dest_service_instance_id: ",
"dest_socket_name: " + destSockeName,
"dest_socket_port: ",
"dest_socket_new_port: ",
"session_close_request_message_id: "
};

// some values of the parameters are unspecified

int index = sessionVec.size();

// add the above session to the vector
sessionVec.add(plugSession);
// note that now sessionRequestId serves as the identifier of this session in the vector
// and sessionVec.get(index) is equal to plugSession

boolean pendingRequest = true;
String[] sessionToSet = new String[16];

// wait for the response from Manager via Agent
TimeUnit.SECONDS.sleep(1);
```

```
while (pendingRequest){

if(

sessionVec.get(index)[3].equals("session_request_id: " +
sessionRequestId)
&&
sessionVec.get(index)[1].equals("session_status: pendingOn")

 ) {

sessionToSet = sessionVec.get(index);

String[] line11 = sessionToSet[11].split(" ");
destServiceInstanceNetworkAddress = line11[1];

String[] line13 = sessionToSet[13].split(" ");
destSocketPort = line13[1];

pendingRequest = false;
      };

}; // end of while loop

int k = Integer.parseInt(destSocketPort);
Socket clientSocket = new
Socket(destServiceInstanceNetworkAddress, k);
int m = clientSocket.getLocalPort();
      // getLocalPort() returns the local port that the socket is bound to.
int l = clientSocket clientSocket.getPort();
      // getPort()  returns the remote port of the socket (i.e. other side address)

String sl = Integer.toString(l);
String sm = Integer.toString(m);

// modify the session
sessionToSet[1] = "session_status: onToCofirm";
sessionToSet[2] = "session_id: " + sm;
sessionToSet[9] = "source_plug_port: " + sm;
sessionToSet[15] = "dest_socket_new_port: " + sl;

// modify  sessionVec
// note that the position of this session in the vector may change in the meantime
if (sessionVec.get(index)[3].equals("session_request_id: " +
sessionRequestId) {

sessionVec.set(index, sessionToSet);

}else{
//  find  theIndex  such that
// (sessionVec.get(theIndex)[3].equals("session_request_id: " +
sessionRequestId)
// then sessionVec.set(theIndex, sessionToSet);
     };

}; // end of if
```

2.1.1 Code for connection (C, (P1,S1), A) from the point of view of instance i of service A

```
// recall that
// String thisServiceName; // = "A";

// String serviceInstanceId; // = "i";

// String serviceInstanceNetworkAddress; // = "NA_i";
//
// vector created from [configuration of sockets]
// Vector<String[]> serverSocketConfigVec = new Vector<String[]>();
//
// an example
// String[] serverSocketConfig =
// {
// "socket_name: S1",
// "port_number: 12564"};

String plugName = P1;
String socketName = S1;
String serverSocketPortNumber;

for(String[] serverSocketConfig : serverSocketConfigVec) {
           if(serverSocketConfig[0].equals(socketName) ) {

serverSocketPortNumber = serverSocketConfig[1];

};

// convert port number to integer
int m = String.valueOf(portNumber);

ServerSocket welcomeSocket = new ServerSocket(m);
Socket socket = welcomeSocket.accept();
// usually, this socket is passed as a parameter to a thread

// get port of plug P1, and network address of the instance of service C
String sourceServiceInstanceNetworkAddress =
socket.getInetAddress().getHostAddress();

String sourcePlugPort = Integer.toString(socket.getPort());

// get network address of this instance i of service A and port of this socket
```

```
String destSocketNewPort =
Integer.toString(socket.getLocalPort());

String destServiceInstanceNetworkAddress =
socket.getLocalAddress().getHostAddress();

// Create a new session (of type socket)
String[] socketSession =
{
"session_type: socket",
"session_status: on",
"session_id: " + sourcePlugPort,
"session_request_id: ",
"request_to_close: false",
"source_service_name: ",
"source_service_instance_network_address: " +
sourceServiceInstanceNetworkAddress,
"source_service_instance_id: ",
"source_plug_name: " + plugName, // P1
"source_plug_port: " + sourcePlugPort,
"dest_service_name: " + thisServiceName,
"dest_service_instance_network_address: " +
destServiceInstanceNetworkAddress,
"dest_service_instance_id: " + serviceInstanceId,
"dest_socket_name: " + socketName, // S1
"dest_socket_port: " + serverSocketPortNumber,
"dest_socket_new_port: " + destSocketNewPort,
"session_close_request_message_id: "
}

// note that some values of the parameters are unspecified,
// add the above session to the vector

sessionVec.add(socketSession);

// no confirmation to Manager via agent is needed
```

2.2 Closing a communication session

```
// typical piece of code associated with a Socket object

try {

// let socket, input and output streams be already constructed

// get the port number of this socket
String k = Integer.toString(socket.getLocalPort());
```

```
// three are three possible cases of the closing:
        // closing session on the request of Manager via Agent,
        // resulting from business logic of the service,
        // and when the TCP connection was broken
// let us start with the first case
// get the session from sessionVec corresponding to this socket

// resetting variables for the next while loop
sessionToSet = null;
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){

if(
( sessionVec.get(index)[4].equals("request_to_close: true"))
&&
(
( sessionVec.get(index)[0].equals("session_type: socket")) &&
(sessionVec.get(index)[15].equals("dest_socket_new_port: " +
k)) )
||
( (sessionVec.get(index)[0].equals("session_type: plug")) &&
(sessionVec.get(index)[9].equals("source_plug_port: " + k)) )
) {

check = false;
sessionToSet = sessionVec.get(index);
     };
index++;
}; // end of while loop

if (check.equals(false)){

index--;

// modify sessionToSet
sessionToSet[1] = "session_status: off";

// modify sessionVec
sessionVec.set(index, sessionToSet);

// close input and output streams, and then socket
socket.close;
};

// ---
// the second case: closing session resulting from business logic of the service A
// it is reasonable to introduce (as part of the business logic of this service) a waiting time limit
// (timeout) for incoming and outgoing messages. After this idle time, the session should be closed
```

```
//close input and output streams, and then socket
socket.close;
// then, get the session from  sessionVec  corresponding to this  socket
// resetting variables for the next while loop
sessionToSet = null;
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){

if(

((sessionVec.get(index)[0].equals("session_type: socket")) &&
(sessionVec.get(index)[15].equals("dest_socket_new_port: " +
k)) ) ||
((sessionVec.get(index)[0].equals("session_type: plug")) &&
(sessionVec.get(index)[9].equals("source_plug_port: " + k))
) {

check = false;
sessionToSet = sessionVec.get(index);
     };
index++;
}; // end of while loop

if (check.equals(false)){

index--;

// modify  sessionToSet
sessionToSet[1] = "session_status: off";

// modify  sessionVec
sessionVec.set(index, sessionToSet);
       };
} // end of  try  of the socket constructor
// the third case: session closing resulting from exceptions (broken TCP connection)

     catch (IOException | InterruptedException e) {
            e.printStackTrace();
// note that the code below is almost the same as for the closing resulting from
     // the business logic of service A
// get the session from  sessionVec  corresponding to this  socket
// resetting variables for the next while loop
sessionToSet = null;
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){

if(
((sessionVec.get(index)[0].equals("session_type: socket")) &&
(sessionVec.get(index)[15].equals("dest_socket_new_port: " +
k)) ) ||
((sessionVec.get(index)[0].equals("session_type: plug")) &&
(sessionVec.get(index)[9].equals("source_plug_port: " + k))
) {

check = false;
sessionToSet = sessionVec.get(index);
     };
index++;
}; // end of while loop

if (check.equals(false)){

index--;

// modify sessionToSet
sessionToSet[1] = "session_status: broken";

// modify sessionVec
sessionVec.set(index, sessionToSet);

}; // end of catch (IOException | InterruptedException e)
```

2.1 Service instance termination

```
// closing instance i of service A
// Check the object String serviceInstanceStatus before System.exit(0);
```

2.1.1 Termination resulting from business logic

```
serviceInstanceStatus = "pendingOff";

while( !serviceInstanceStatus.equals("pendingOffConfirmed") ){

//wait
TimeUnit.SECONDS.sleep(1);
}

// option: close all sessions, then

System.exit(0);
```

2.1.2 Graceful shutdown on request from Manager

```
if ( serviceInstanceStatus.equals("pendingOffOnReguest" ) ) {

serviceInstanceStatus = "pendingOffOnResponseToConfirm";

while( !serviceInstanceStatus.equals(
"pendingOffOnResponseConfirmed") ){

//wait
TimeUnit.SECONDS.sleep(1);
};

// option: close all sessions, then

System.exit(0);
};
```

3. Agent

/*
The basic functionalities:

- Registration to the Manager.
- Executing and killing a service instance upon request from the Manager.
- Forwarding messages from the Manager to service instances and vice versa.

*/

3.1. Service repository

```
// an example of service parameters structure in Agent's service repository
/*
String[] serviceParameters = {
"service_name: A",
"path_to_binary: /usr/bin/A",
"sockets: [sequence of socket names]",  //names are separated by ", "
"plugs: [sequence of plug names]" //names are separated by ", "
};

serviceRepositoryVec.add(serviceParameters);
*/

Vector<String[]> serviceRepositoryVec = new
Vector<String[]>();
```

3.2. Service instances

// an example of service instance parameters structure
/*

```
String[] serviceInstanceParameters = {
"service_name: A",
"service_instance_id: i",
"service_instance_network_address: NA_i",
"socket_configuration: [configuration of sockets]",
"plug_configuration: [configuration of plugs]"
};

serviceInstanceVec.add(serviceInstanceParameters);
*/

Vector<String[]> serviceInstanceVec = new Vector<String[]>();
```

3.3. Vector of running service instances

```
// a separate thread for any of the running service instances (as processes)
// parameters of the process constructor:

String serviceName; // == "A";
String pathToBinary; // == "/usr/bin/A";
String serviceInstanceId; // == "i";
String serviceInstanceNetworkAddress; // == "NA_i";
String socketConfiguration; // == "[configuration of
    sockets]";
String  plugConfiguration; // == "[configuration of plugs]";

// execute a service instance
ProcessBuilder pb = new ProcessBuilder(pathToBinary,
serviceName, serviceInstanceId, serviceInstanceNetworkAddress,
socketConfiguration, plugConfiguration); // "-arg1", "-arg2");

Process process = pb.start();
String id = serviceInstanceId;

// The parameter id  serves as the thread identifier in vector, and can be retrieved as follows
      // id = thread.getrunningServiceInstanceId();
      // it is needed for realizing Manager's request message of type
      // "type: hard_shutdown_request"

//new thread for this running service instance
ServiceInstanceThread  thread =  new
ServiceInstanceThread(process, id).start();

// add this tread to vector
runningServiceInstancesVector.add(thread);
```

```
// actually, this vector serves only to identify the service instance to be terminated by Agent
     // upon the request from Manager

Vector<ServiceInstanceThread> runningServiceInstancesVector =
new Vector<ServiceInstanceThread>();

        private static class ServiceInstanceThread extends Thread
{
         private Process serviceInstanceProcess;
         private String serviceInstanceId;

                public  ServiceInstanceThread(Process process, String
id) {
                        this.serviceInstanceProcess = process;
                        this.serviceInstanceId = id;

};

                public void run() {

//  the identifier of the thread where the service instance is running as a process
public static String runningServiceInstanceId =
serviceInstanceId;

public String getrunningServiceInstanceId(){
return runningServiceInstanceId;
};

// method: kill the process and stop the thread
public static void killServiceInstance()
        {
         // terminate process with destroy()
         serviceInstanceProcess.destroy();

         // allow process to exit gracefully with reasonable timeout
         TimeUnit.SECONDS.sleep(1);

         // kill it with destroyForcibly() if process is still alive
         serviceInstanceProcess.destroyForcibly();

         Thread.currentThread().interrupt();
         return;

        };  // the method to be called in the main Agent thread
        };  // end of public void run()
};
```

3.4. Registration to Manager

```
Socket AgentToManagerSocket = new Socket
(ManagerNetworkAddress, 55555);
// in ssmmp, the port number 55555 is fixed for establishing a communication session
        // between Agent and Manager

String agentNetworkAddress =
AgentToManagerSockey.getLocalAddress().getHostAddress();

PrintWriter outToManager = new
PrintWriter(AgentToManagerSocket.getOutputStream(), true);

BufferedReader inFromManager = new BufferedReader(new
InputStreamReader(AgentToManagerSocket.getInputStream()));

//  method: read message from Manager
public static fromManager(String message){
message = inFromManager.readLine()
};

String messageToManager = null;
String messageFromManager = null;
String responseToManager = null;
String messageId = null;
String[] messageInLines = null;

// method: send message to Manager
public static toManager(String message){
outToManager.print(message);
};

// send the registration message to Manager via outToManager
// format of this message is as follows
/*
type: initiation_request
message_id: [integer]
agent_network_address: [IPv6]
service_repository: [service name list]
*/

String serviceNameList;

for( String[] serviceParameters : serviceRepositoryVec) {
     String[] service = serviceParameters[0].split(" ");
     serviceNameList = serviceNameList + service[1] + ", ";
};

messageId = "38421";  //  randomly generated

// create the message
```

```
messageToManager = "type: initiation_request ~ message_id: " +
messageId + " ~ agent_network_address: " + agentNetworkAddress
+ " ~ service_repository: " + serviceNameList;

// display the message
// messageLines = messageToManager.split(" ~ ");
// for (String row : messageInLines){
// System.out.println(row + "\n")
// };

// send the message to Manger
toManager(messageToManager);

// get response from Manager
fromManager(messageFromManager);

// display the message
// messageInLines = messageFromManager.split(" ~ ");
// for (String row : messageInLines)
// {System.out.println(row + "\n")};

// check the Status of the response
// if OK, then continue, else HALT
```

3.5. Messages from Manager to Agent

```
// in separate thread

int index = 0;
boolean check = true;

 while(true) {
// read a message from inFromManager;

fromManager(messageFromManager);

messageInLines = messageFromManager.split(" ~ ");

// display the message
for (String row : messageInLines)
{ System.out.println(row + "\n") };

String[] line1 = messageInLines[1].split(" ");
String messageId = line1[1];

String messageType = messageInLines[0];

// depending on the type of this message
switch (messageType) {
```

```
case "type: execution_request":

// this message format is as follows
/*
type: execution_request
message_id: n
agent_network_address: NA_i
service_name: A
service_instance_id: i
socket_configuration: [configuration of sockets]
plug_configuration: [configuration of plugs]
*/

String[] line3 = messageInLines[3].split(" ");
String serviceName = line3[1];

String[] line4 = messageInLines[4].split(" ");
String serviceInstanceId = line4[1];

String[] line2 = messageInLines[2].split(" ");
String serviceInstanceNetworkAddress = line2[1];
// the same network address as for the Agent

String[] line5 = messageInLines[5].split(" ");
String socketConfiguration = line5[1];

String[] line6 = messageInLines[6].split(" ");
String plugConfiguration = line6[1];

// the last parameter needed to execute service instance is pathToBinary
// this parameter is retrieved from serviceRepositoryVec as follows

// recall that
/*
String[] serviceParameters = {
"service_name: A",
"path_to_binary: /usr/bin/A",
"sockets: [sequence of socket names]",
"plugs: [sequence of plug names]"
};
serviceRepositoryVec.add(serviceParameters);
Vector<String[]> serviceRepositoryVec = new Vector<String[]>();
*/

index = 0;
check = true;

// determine the pathToBinary
```

```
while ( check && !(index.equals(serviceRepositoryVec.size()))
){

String row0 = serviceRepositoryVec.get(index)[0];

if( row0.split(" ")[1].equals(serviceName)){
check = false;

String row1 = serviceRepositoryVec.get(index)[1];
String pathToBinary = row1.split(" ")[1];
     };
index++;
}; // end of while loop

if (check.equals(false){
ProcessBuilder pb = new ProcessBuilder(pathToBinary,
serviceName, serviceInstanceId, serviceInstanceNetworkAddress,
socketConfiguration, plugConfiguration);

Process process = pb.start();

//new thread for this running service instance
ServiceInstanceThread  thread =  new
ServiceInstanceThread(process, serviceInstanceId).start();

// add this tread to vector
runningServiceInstancesVector.add(thread);

// a new element serviceInstanceParameters of serviceInstanceVec
// is constructed and added to the vector

String[] serviceInstanceParameters = {
"service_name: " + serviceName,
"service_instance_id: " + serviceInstanceId,
"service_instance_network_address: " +
    serviceInstanceNetworkAddress,
"socket_configuration: " + socketConfiguration,
"plug_configuration: " + plugConfiguration,
};

serviceInstanceVec.add(serviceInstanceParameters);

// send to Manager the response message "type: execution_response"
// format of this message is as follows
/*
type: execution_response
message_id: n
status: [status code]
*/
String responseToManager = "type: execution_response ~
message_id: " + messageId + " ~ status: 200";
}else{
String responseToManager = "type: execution_response ~
message_id: " + messageId + " ~ status: 300";
};

toManager(responseToManager);

break;

case "type: health_control_request":

// If the first line of the message is " type: agent_health_control_request",
/*  request message format
type: agent_health_control_request
message_id: o
sub_type: Manager_to_agent
agent_network_address: NA
*/

// response to Manager with message " type: agent_health_control_response"
/*  response message format, if Agent is OK
type: agent_health_control_response
message_id: o
sub_type: agent_to_Manager
agent_network_address: NA
status: [status code]
*/

String responseToManager = " type:
agent_health_control_response ~ message_id: " + messageId + "
~ sub_type: agent_to_Manager ~ agent_network_address: " +
agentNetworkAddress + " ~ status: 200";

toManager(responseToManager);

break;

case "type: hard_shutdown_request":

// the first line of the message is "type: hard_shutdown_request",
// get from the message: "service_instance_id: i" of service instance to be killed
/*  request message format
type: hard_shutdown_request
message_id: n
sub_type: Manager_to_agent
service_name: A
service_instance_id: i
*/
```

```
String[] line4 = messageInLines[4].split(" ");
String id = line4[1];

// find the thread in the vector and invoke killing of this service instance;
index = 0;
check = true;

while ( check &&
!(index.equals(runningServiceInstancesVector.size())) ){

if(
id.equals(runningServiceInstancesVector.get(index).getrunningS
erviceInstanceId()){
check = false;
runningServiceInstancesVector.get(index).killServiceInstance()
     };
index++;
}; // end of while loop

// the thread is already dead
// remove (???) this thread from the vector

index--;
runningServiceInstancesVector.remove(index);

// send response message
/*  response message format
type: hard_shutdown_response
message_id: n
sub_type: agent_to_Manager
service_instance_id: i
status: [status code]
*/

String responseToManager = "type: hard_shutdown_response ~
message_id: " + messageId + " ~ sub_type: agent_to_Manager ~
service_instance_id: " + id + " ~ status: 200";

toManager(responseToManager);

  break;

default:

// otherwise
// get service_instance_id from the message

String[] line4 = messageInLines[4].split(" ");
String ids = line4[1];

// Change the sub_type line of this message to
// "sub_type: agent_to_service_instance"

messageInLines[2] = "sub_type: agent_to_service";

String messageFromAgentToService;

for( String line : messageInLines ) {
    messageFromAgentToService = messageFromAgentToService +
line + " ~ " };

// find the handler for this service instance (with ids) in the vector handlerVector,
// created In the next subsection 3.6 ,
// and send the message to this instance
index = 0;
check = true;

while ( check && !(index.equals(handlerVector.size())) ){

if(
ids.equals(handlerVector.get(index).getServiceInstanceId()){
check = false;
handlerVector.get(index).
sendToServiceInstance(messageFromAgentToService) };
index++;
     }; // end of while loop

break;
    }//end of the switch

};//end of the while loop
```

3.6. Forwarding messages from service instances to Manager

```
// in ssmmp, the port number 55556 is fixed for communication between Agent and
// a service instance running on the same node as the Agent
// in separate thread
ServerSocket  AgentServerSocket =  new ServerSocket(55556);

while (true){
ServiceInstanceHandler  handler =  new
ServiceInstanceHandler(AgentServerSocket.accept()).start();

handlerVector.add(handler);
};

Vector<ServiceInstanceHandler> handlerVector = new
Vector<ServiceInstanceHandler>();

    private static class ServiceInstanceHandler extends Thread
{
        private Socket clientSocket;
        private PrintWriter outToServiceInstance;
        private BufferedReader inFromServiceInstance;


        public  ServiceInstanceHandler(Socket socket) {
            this.clientSocket = socket;
        };

        public void run() {
           PrintWriter outToServiceInstance = new
PrintWriter(clientSocket.getOutputStream(), true);

public static void sendToServiceInstance(String message)
    {
      outToServiceInstance.print(message)

    };  // the method to be called in the main Agent thread

           BufferedReader inFromServiceInstance = new
BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

// read the first message from inFromServiceInstance
// to get the identifier of this service instance
String messageFromService = inFromServiceInstance.readLine();

// the format of this message is as follows
/*
type: session_request
message_id: n
sub_type: service_to_agent
source_service_name: A
source_service_instance_id: i
source_service_instance_network_address: NA_i
source_plug_name: P
dest_service_name: B
dest_socket_name: S
*/

String[] messageFromServiceInLines =
messageFromService.split(" ~ ");

// display the message
// for (String row : messageFromServiceInLines)
// {System.out.println(row + "\n")};

String[] line3 = messageFromServiceInLines[3].split(" ");
public static String serviceName = line3[1];

String[] line4 = messageFromServiceInLines[4].split(" ");
public static String serviceInstanceId = line4[1];
public String getServiceInstanceId(){
return serviceInstanceId;
};

// change the line "sub_type: service_to_agent"  to
// "sub_type: agent_to_Manager"
// send this message to Manager

messageFromServiceInLines[2] = "sub_type: agent_to_Manager";

String messageToManager;

for( String line : messageFromServiceInLines ) {
    messageToManager = messageToManager + line + " ~ " };

toManager(messageToManager);
        // this method is in the main Agent thread

// the main while loop of this thread
while(true) {

//  read the next message from inFromServiceInstance
// and change the line "sub_type: service_to_agent"  to
// "sub_type: agent_to_Manager"
// send this message to Manager

messageFromService = inFromServiceInstance.readLine();

messageFromServiceInLines = messageFromService.split(" ~ ");

// display the message
// for (String row : messageFromServiceInLines)
// {
// System.out.println(row + "\n");
// };

messageFromServiceInLines[2] = "sub_type: agent_to_Manager";

for( String line : messageFromServiceInLines ) {
    messageToManager = messageToManager + line + " ~ " };

toManager(messageToManager);

                };  // end of the while loop
     }  // end of public void run()
};  // end of the thread constructor

// inFromServiceInstance.close(); outToServiceInstance.close();
// clientSocket.close();
```

4. Manager

/*

The basic functionality of Manager consists of:

- Sending requests to Agents to execute (or shut down) service instances.
- Response to requests from a service instance for parameters needed to open a new communication session.
- Requests to close a communication session.
- Updating data structures, if info messages (on closing sessions or terminating service instances) are received.

Let us start with the data structures of Manager. These are vectors of string arrays and are not persistent, i.e. they exist as long as the Manager is running. These data structures can be easily implemented as persistent databases. The code below can also be easily adapted to these databases.

*/

4.1. Abstract graph of CNApp

// abstract graph of CNApp (to be executed, managed, reconfigured and possibly terminated
// by the Manager) must be provided in advance
// Vector of connections represents abstract graph of CNApp;
// an example of a connection (A, (P,S), B) between vertex1 and vertex2
/*

```
String[] connection = {
"connection_id: id",
"vertex1: id1",
"service_name: A",
"plug: P",
"socket: S",
"service_name: B",
"vertex2: id2"
};
graphOfCNApp.add(connection);

*/
```

```
Vector<String[]> graphOfCNApp = new Vector<String[]>();
```

// although the order of the connections can be arbitrary in the vector,
// the first (in the partial order of the graph) connections of graphOfCNApp are of the form:
// (API Gateway, (P,S), B),
// the next elements are of the form: (B, (P1,S1), C), and so on

4.2. Service repository

// an example of service structure in Manager's service repository
/*

```
String[] service = {
```

```
"agent_network_address: NA_i",  // initially, NA_i is empty string
"service_name: A",
"sockets: [sequence of socket names]",
"plugs: [sequence of plug names]"
};

serviceVec.add(service);
*/

Vector<String[]> serviceVec = new Vector<String[]>();

// initially the vector serviceVec is constructed on the basis of graphOfCNApp
        // where service[0] = "agent_network_address: "
// note that the network address of the Agent corresponds to the fact that
        // an instance of this service can be execute by the Agent on the same host
```

4.3 Vector of Agents

```
// vector of agents
Vector<String[]> VectorOfAgents = new Vector<String[]>();

//  an example
/*
String[] newAgent = {
"agent_network_address: NA",
"health_control_response_time: ",  // the most recent response
};
VectorOfAgents.add(newAgent);
*/
```

4.4 Vector of service instance structures

```
// an example of service instance structure in Manager's vector of service instances
/*
String[] serviceInstance = {
"service_instance_network_address: NA_i",  // the same as the Agent's address
"service_name: A",
"service_instance_id: i",
"vertex_id: id",
"service_instance_status: on", // pendigOn, pendingOff, off
"socket_configuration: [configuration of sockets]",
"plug_configuration: [configuration of plugs]",
"service_instance_execution_request_message_id: ",
"service_instance_graceful_shutdown_request_message_id: ",
"service_instance_hard_shutdown_request_message_id: ",
"service_instance_health_request_message_id: ",
"health_control_response_time: ", // the most recent response
};
serviceInstanceVec.add(serviceInstance);
*/

Vector<String[]> serviceInstanceVec = new Vector<String[]>();
```

4.5 Vector of communication sessions

```
// new session for connection (A, (P,S), B); the format is the same as
        // in Section 1.   Java API for service
/*
String[] session =
{
"session_type: plug", // or socket
"session_status: pendingOnResponse",
// or: on, off, pendingOffResponse, broken
"session_id: m",
"session_request_id: n", // message_id is set here
"request_to_close: false",
"source_service_name: A",
"source_service_instance_network_address: NA_i",
"source_service_instance_id: i",
"source_plug_name: P",
"source_plug_port: m,
"dest_service_name: B",
"dest_service_instance_network_address: NA_j",
"dest_service_instance_id: j",
"dest_socket_name: S",
"dest_socket_port: k",
"dest_socket_new_port: l",
"session_close_request_message_id: "
}; // note that some values of the parameters are unspecified,
// add the above session to the vector
sessionVec.add(session);

*/

Vector<String[]> sessionVec = new Vector<String[]>();
```

4.6 Basic functionality of Manager

```
// in ssmmp, the port number 55555 is fixed for Manager to accept Agents, each one
        // in a separate thread
ServerSocket ManagerServerSocket = new ServerSocket(55555);

// the main while loop for processing connections to Agents
```

```
while (true){

Socket agentSocket = ManagerServerSocket.accept();

// the network address of the Agent
String agentId =
agentSocket.getInetAddress().getHostAddress();

AgentHandler  handler =  new AgentHandler(agentSocket,
agentId).start();

handlerVector.add(handler);

String[] newAgent = {
"agent_network_address: " + agentId,
"health_control_response_time: ", // the most recent response
};
VectorOfAgents.add(newAgent);

};

Vector<AgentHandler> handlerVector = new
Vector<AgentHandler>();

    private static class AgentHandler extends Thread {
        private Socket clientSocket;
        private String agentId;
        private PrintWriter outToAgent;
        private BufferedReader inFromAgent;


        public AgentHandler(Socket socket, String Id) {
            this.clientSocket = socket;
            this.agentId = Id;
        };

        public void run() {

           PrintWriter outToAgent = new
PrintWriter(clientSocket.getOutputStream(), true);

// method: send message to Agent
public static void toAgent(String message){
outToManAgent.print(message);
};

           BufferedReader inFromAgent = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

String idA = agentId;
// method: get  id (network address)  of Agent
```

```
public static String getAgentId(){return idA;};

int portNumberCounter = 54000;
// initial value, the next numbers are from the fixed range (from 54 000  to 55 554) separately
        // for each agent, and controlled only by Manager

public String getNewPortNumber(){

String port = String.valueOf(portNumberCounter);
portNumberCounter++;
return port;

};

// read the first registration message ("type: initiation_request") from Agent
// the format of this message is as follows
/*
type: initiation_request
message_id: [integer]
agent_network_address: [IPv6]
service_repository: [service name list]
*/

String messageFromAgent = null;
messageFromAgent = inFromAgent.readLine();

String messageInLines = null;
messageInLines = messageFromAgent.split(" ~ ");

// display the message
// for (String row : messageInLines){
// System.out.println(row + "\n") };

String messageId = null;
String agentNetworkAddress = null;
String serviceRepository = null;
String status = null;

messageId = messageInLines[1].split(" ")[1];

agentNetworkAddress = messageInLines[2].split(" ")[1];
// note that is the same as agentId

serviceRepository = messageInLines[3].split(" ")[1];

String[] serviceNames = serviceRepository.split(", ");

// update the vector serviceVec
// recall that initial elements of  serviceVec  are of the following form
/*
String[] service = {
"agent_network_address: ", // the value of this parameter is empty string
```

```
"service_name: A",
"sockets: [sequence of socket names]",
"plugs: [sequence of plug names]"
};
*/

String[] serviceToAdd = new String[4];

for( String name: serviceNames ){
     for(String[] service: serviceVec){
if( name.equals(service[1].split(" ")[1])
&& (service[0].split(" ")[1]).equals("")
){
serviceToAdd = service;
serviceToAdd[0] = agentNetworkAddress;

serviceVec.add(serviceToAdd);
          }
     }
};

// create the response to this message
// the response is of the following form
/*
type: initiation_response
message_id: [integer]
status: [status code]
*/

String messageToAgent;

messageToAgent = "type: initiation_response ~ message_id: " +
messageId + " ~ status: 200";

outToAgent.print(messageToAgent);

int index = 0;
boolean check = true;

// the while loop for processing next incoming messages from Agent
while(true){

// the following three objects are used to the end of the while loop
// read the next message from Agent
messageFromAgent = inFromAgent.readLine;

// split the message into lines
messageInLines = messageFromAgent.split(" ~ ");
// get message id
messageId = messageInLines[1].split(" ")[1];
```

```
// depending on the
String typeOfMessage = messageInLines[0];

// update the data structures, and/or response according to ssmmp:

switch(typeOfMessage){

      case "type: execution_response":

// the format of this message is as follows
/*
type: execution_response
message_id: n
status: [status code]
*/
// find out the appropriate service instance structure in serviceInstanceVec
      // with "service_instance_status: pendigOn"
      // and update structure

status = messageInLines[2].split(" ")[1];

// recall that the following service instance structure was created in Manager's vector of service
// instances, when Manager sent the request to Agent to execute this service instance
// see Section 4.10 Execution of a new instance of service

/*
String[] serviceInstance = {
"service_instance_network_address: NA_i", // the same as the Agent's address
"service_name: A",
"service_instance_id: i",
"vertex_id: id",
"service_instance_status: pendigOn ",
"socket_configuration: [configuration of sockets]",
"plug_configuration: [configuration of plugs]",
"service_instance_execution_request_message_id: n",  // [7]
"service_instance_graceful_shutdown_request_message_id: ",
"service_instance_hard_shutdown_request_message_id: ",
"service_instance_health_request_message_id: ",
"health_control_response_time: ", // the most recent response
};
serviceInstanceVec.add(serviceInstance);
Vector<String[]> serviceInstanceVec = new Vector<String[]>();
*/
// find out this service instance structure in serviceInstanceVec
      // with serviceInstance[4] == "service_instance_status: pendigOn"
      // and  serviceInstance[7] ==
      // "service_instance_execution_request_message_id: n",
      // and update this structure

// resetting variables for the next while loop
String[] serviceInstanceToSet = new String[12];
index = 0;
check = true;

while ( check && !(index.equals(serviceInstanceVec.size()) ){

if(serviceInstanceVec.get(index)[7] ].split("
")[1].equals(messageId)) {

check = false;
serviceInstanceToSet = serviceInstanceVec.get(index);
      };
index++;
}; // end of while loop

if (check.equals(false)){

index--;

      if (status.equals("200"){

// modify serviceInstanceToSet
serviceInstanceToSet[4] = "service_instance_status: on";

// update serviceInstanceVec
serviceInstanceVec.set(index, serviceInstanceToSet);

      }else{
// remove from  serviceInstanceVec
serviceInstanceVec.remove(index);
      };
};

      break;

      case "type: session_request":

// format of this message
/*
type: session_request
message_id: n
sub_type: agent_to_Manager
source_service_name: A
source_service_instance_id: i
source_service_instance_network_address: NA_i
source_plug_name: P
dest_service_name: B
dest_socket_name: S
```

```
*/

String destServiceName = messageInLines[7].split(" ")[1];
String destSocketName = messageInLines[8].split(" ")[1];

// special method is required to determine appropriate port and network address
        // of instance of destination service;
        // this may require an execution of a new instance of destination service
// call the method from Section 4.7
String[] sessionParameters =
getParametersForNewSession(destServiceName, destSocketName);

String destServiceInstanceId = sessionParameters[0];

String destServiceInstanceNetworkAddress =
sessionParameters[1];

String destSocketPort = sessionParameters[2];

// sent response message of "type: session_response"
// the format of this message is as follows
/*
type: session_response
message_id: n
sub_type: Manager_to_agent
source_service_name: A
source_service_instance_id: i
dest_service_name: B
dest_service_instance_id: j
dest_socket_name: S
dest_service_instance_network_address: NA_j
dest_socket_port: k
status: [status code]
*/

messageToAgent = "type: session_response ~ message_id: " +
messageId + " ~ sub_type: Manager_to_agent ~ " +
messageInLines[3] + " ~ "
" + messageInLines[4] + " ~ " + messageInLines[5] + " ~ " +
messageInLines[6] + " ~ " + messageInLines[7] + " ~
dest_service_instance_id: " + destServiceInstanceId + " ~ "
messageInLines[8] + " ~ dest_service_instance_network_address: "
+ destServiceInstanceNetworkAddress + " ~ dest_socket_port: " +
destSocketPort + " ~ status: 200";

outToAgent.print(messageToAgent);

// create a new session and add to the session vector sessionVec
//see Section 4.5

String[] session =
{
```

```
"session_type: plug",
"session_status: pendingOnResponse",
"session_id: ",
"session_request_id: " + messageId,
"request_to_close: false",
messageInLines[3], // "source_service_name: A",
messageInLines[5],
// "source_service_instance_network_address: NA_i",
messageInLines[4], // "source_service_instance_id: i",
messageInLines[6], // "source_plug_name: P",
"source_plug_port: ,
messageInLines[7], // "dest_service_name: B",
"dest_service_instance_network_address: " +
destServiceInstanceNetworkAddress,
"dest_service_instance_id: " + destServiceInstanceId,
messageInLines[8], // "dest_socket_name: S",
"dest_socket_port: " + destSocketPort,
"dest_socket_new_port: ",
"session_close_request_message_id: "
}; // note that some values of the parameters are unspecified,

// add the above session to the vector
sessionVec.add(session);

     break;

     case "type: session_ack":

// format of this message
/*
type: session_ack
message_id: n
sub_type: agent_to_Manager
source_service_name: A
source_service_instance_id: i
source_plug_port: m
dest_socket_new_port: l
status: [status code]
*/
String sourcePlugPort = messageInLines[5].split(" ")[1];
// String destSocketNewPort = messageInLines[6];

// resetting variables for the next while loop
String[] sessionToSet = new String[];
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if(
sessionVec.get(index)[3].split(" ")[1].equals(messageId)
&&
sessionVec.get(index)[7].equals(messageInLines[4])
)
check = false;
sessionToSet = sessionVec.get(index);
     };
index++;
}; // end of while loop

if (check.equals(false)){
// modify sessionToSet
sessionToSet[1] = "session_status: on";
sessionToSet[2] = "session_id: " + sourcePlugPort;
sessionToSet[9] = messageInLines[5];
sessionToSet[16] = messageInLines[6];

index--;
// modify sessionVec
sessionVec.set(index, sessionToSet);

};
     break;

     case "type: source_service_session_close_info":

// format of this message
/*
type: source_service_session_close_info
message_id: n
sub_type: agent_to_Manager
source_service_name: A
source_service_instance_id: i
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_service_name: B
dest_service_instance_id: j
dest_service_instance_network_address: NA_j
dest_socket_name: S
dest_socket_port: k
dest_socket_new_port: l
status: [status code]
*/
// status is 111 if the closing is according to business logic, or 122 if the connection was broken
// get status (111 or 122) from the message
status = messageInLines[12].split(" ")[1];
// this value may be stored by Manager

// resetting variables for the next while loop
index = 0;
check = true;

// get the corresponding session in sessionVec
while ( check && !(index.equals(sessionVec.size())) ){
if(
sessionVec.get(index)[6].equals(messageInLines[9])
&&
sessionVec.get(index)[7].equals(messageInLines[4])
&&
sessionVec.get(index)[9].equals(messageInLines[7])
){
check = false;
     };
index++;
}; // end of while loop

if (check.equals(false)){
index--;
// remove from sessionVec
sessionVec.remove(index);
};

     break;

     case "type: dest_service_session_close_info":

// format of this message
/*
type: dest_service_session_close_info
message_id: o
sub_type: dest_service_to_agent
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_service_name: B
dest_service_instance_network_address: NA_j
dest_service_instance_id: j
dest_socket_name: S
dest_socket_port: k
dest_socket_new_port: l
status: [status code]
*/
// status is 111, if the closing is according to business logic, or 122 if the connection was broken
// get the corresponding session in sessionVec

// get status (111 or 122) from this message
status = messageInLines[12].split(" ")[1];
// this value may be stored by Manager

// resetting variables for the next while loop
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if(
sessionVec.get(index)[11].equals(messageInLines[7])
&&
sessionVec.get(index)[12].equals(messageInLines[8])
&&
sessionVec.get(index)[15].equals(messageInLines[11])
){
check = false;};
index++;
}; // end of while loop

if (check.equals(false)){
index--;
// remove from sessionVec
sessionVec.remove(index);
};

     break;

     case "type: source_service_session_close_response":

// format of this message
/*
type: source_service_session_close_response
message_id: o
sub_type: agent_to_Manager
status: [status code]
*/
// note that message_id: o is the same as in
session[16] == "session_close_request_message_id: o"
// and the messageId is uniquely determined by Manager

// get the corresponding session in sessionVec
// resetting variables for the next while loop
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if(
sessionVec.get(index)[16].equals("session_close_request_messag
e_id: " + messageId)
&&
sessionVec.get(index)[1].equals("session_status: on")
){
check = false;
};
index++;
}; // end of while loop

if (check.equals(false)){
index--;
// remove from sessionVec
sessionVec.remove(index);
};
```

```
     break;

     case "type: dest_service_session_close_response":

// format of this message
/*
type: dest_service_session_close_response
message_id: o
sub_type: agent_to_Manager
status: [status code]
*/
// the flowing code is exactly the same as in the previous case
// note that message_id: o is the same as in
session[16] == "session_close_request_message_id: o"
// and the messageId is uniquely determined by Manager

// get the corresponding session in sessionVec
// resetting variables for the next while loop
index = 0;
check = true;

while ( check && !(index.equals(sessionVec.size())) ){
if(
sessionVec.get(index)[16].equals("session_close_request_messag
e_id: " + messageId)
&&
sessionVec.get(index)[1].equals("session_status: on")
){
check = false;
};
index++;
}; // end of while loop

if (check.equals(false)){
index--;
// remove from sessionVec
sessionVec.remove(index);
};

     break;

     case "type: graceful_shutdown_response":

// this message is of the following format
/*
type: graceful_shutdown_response
message_id: o
sub_type: agent_to_Manager
status: [status code]
*/
```

```
status = messageInLines[3].split(" ")[1];

// note that message_id: o is the same as in
// serviceInstance[8]
// "service_instance_graceful_shutdown_request_message_id: o",
// and the messageId was uniquely determined by Manager

// Recall that the following service instance structure was created in Manager's vector of service
// instances, when Manager sent the request to Agent to execute this service instance.
// See Section 4.10 Execution of a new instance of service
// and then, it was updated after receiving the positive response from the Agent

/*
String[] serviceInstance = {
"service_instance_network_address: NA_i", // the same as the Agent's address
"service_name: A",
"service_instance_id: i",
"vertex_id: id",
"service_instance_status: on",
"socket_configuration: [configuration of sockets]",
"plug_configuration: [configuration of plugs]",
"service_instance_execution_request_message_id: n",
"service_instance_graceful_shutdown_request_message_id: o",
"service_instance_hard_shutdown_request_message_id: ",
"service_instance_health_request_message_id: ",
"health_control_response_time: ", // the most recent response
};
serviceInstanceVec.add(serviceInstance);

Vector<String[]> serviceInstanceVec = new Vector<String[]>();
*/

// resetting variables for the next while loop
index = 0;
check = true;

while ( check && !(index.equals(serviceInstanceVec.size()) ){

if(serviceInstanceVec.get(index)[8].equals(messageId)) {

check = false;
      };
index++;
}; // end of while loop
```

```
if (check.equals(false)){

index--;

     if (status.equals("200"){

// remove from  serviceInstanceVec
serviceInstanceVec.remove(index);
     };
};


     break;

     case "type: hard_shutdown_response":

//format of this message is as follows
     /*
     type: hard_shutdown_response
     message_id: n
     sub_type: agent_to_Manager
     source_service_instance_id: i
     status: [status code]
     */
status = messageInLines[4].split(" ")[1];

// the code is the same as for
// case "type: graceful_shutdown_response":
// except the line
// if(serviceInstanceVec.get …
// where 8 is changed to 9

// resetting variables for the next while loop
index = 0;
check = true;

while ( check && !(index.equals(serviceInstanceVec.size()) ){

if(serviceInstanceVec.get(index)[9].equals(messageId)) {

check = false;
     };
index++;
}; // end of while loop

if (check.equals(false)){

index--;

     if (status.equals("200"){

// remove from  serviceInstanceVec
serviceInstanceVec.remove(index);
     };
};

     break;

     case "type: service_instance_termination_info":

// format of this message is as follows
     /*
     type: service_instance_termination_info
     message_id: n
     sub_type: agent_to_Manager
     service_name: A
     service_instance_id: i
     service_instance_network_address: NA_i
     */

// String serviceInstanceId = messageInLines[4].split(" ")[1];

// resetting variables for the next while loop
index = 0;
check = true;

while ( check && !(index.equals(serviceInstanceVec.size()) ){

if(serviceInstanceVec.get(index)[2].equals(messageInLines[4]))
{

check = false;
     };
index++;
};  // end of while loop

if (check.equals(false)){

index--;

// remove from  serviceInstanceVec
serviceInstanceVec.remove(index);

};

     break;

     case "type: service_instance_health_response":

// format of this message is as follows
     /*
     type: service_instance_health_response
     message_id: o
     sub_type: agent_to_Manager
```

```
     service_instance_id: i
     status: [status code]
     */

status = messageInLines[4].split(" ")[1];
// String serviceInstanceId = messageInLines[3].split(" ")[1];

Date date = new Date();
Timestamp ts = new Timestamp(date.getTime());
String healthControlResponseTime = ts.toString();

// find out  service instance structure in serviceInstanceVec
     // with messageId and serviceInstanceId
     //  and update this structure

// resetting variables for the next while loop
String[] serviceInstanceToSet = new String[12];
index = 0;
check = true;

while ( check && !(index.equals(serviceInstanceVec.size()) ){

if(
serviceInstanceVec.get(index)[2].equals(messageInLines[3])
&&
serviceInstanceVec.get(index)[10].split("
")[1].equals(messageId)
) {

check = false;
serviceInstanceToSet = serviceInstanceVec.get(index);
     };
index++;
};  // end of while loop

if (check.equals(false)){

index--;

     if (status.equals("200"){

// modify serviceInstanceToSet
serviceInstanceToSet[11] = "health_control_response_time: " +
healthControlResponseTime;

// update serviceInstanceVec
serviceInstanceVec.set(index, serviceInstanceToSet);
     }
};

     break;
```

```
      case "type: agent_health_control_response":


// format of this message is as follows
      /*
      type: agent_health_control_response
      message_id: o
      sub_type: agent_to_Manager
      agent_network_address: NA
      status: [status code]
      */

status = messageInLines[4].split(" ")[1];
// String agentNetworkAddress = messageInLines[3].split("
")[1];

// recall that
// vector of agents
//Vector<String[]> VectorOfAgents = new Vector<String[]>();

//  an example
/*
String[] newAgent = {
"agent_network_address: NA",
"health_control_response_time: ", // the most recent response
};
VectorOfAgents.add(newAgent);
*/

Date date = new Date();
Timestamp ts = new Timestamp(date.getTime());
String healthControlResponseTime = ts.toString();

// find out  agent  in VectorOfAgents
      // with  agentNetworkAddress
      //  and update this agent

// resetting variables for the next while loop
String[] agentToSet = new String[2];
index = 0;
check = true;

while ( check && !(index.equals(VectorOfAgents.size()) ){

if(
VectorOfAgents.get(index)[0].equals(messageInLines[3])
) {

check = false;
agentToSet = VectorOfAgents.get(index);
      };
index++;
}; // end of while loop

if (check.equals(false)){

index--;

      if (status.equals("200"){

agentToSet[1] = "health_control_response_time: " +
healthControlResponseTime;

VectorOfAgents.set(index, agentToSet);
      }
};

      break;

      //default:
      //
      //break;

      }; //  end of switch
}; // end of while loop for processing incoming messages from Agent

// inFromAgent.close();
// outToAgent.close();
// clientSocket.close();
     }; // end of the while loop of server socket
}; // end of Agent handler thread



//
// ------------------------------------------
     // tasks to be completed by Manager depending of its own business logic;:
          // send request to execute a service instance. It is already done, see Section 4.10

          // send request to terminate (gracefully) a service instance

/*
String[] serviceInstance = {
"service_instance_network_address: NA_i", // the same as the Agent's address
"service_name: A",
"service_instance_id: i",
"vertex_id: id",
"service_instance_status: on",
"socket_configuration: [configuration of sockets]",
"plug_configuration: [configuration of plugs]",
"service_instance_execution_request_message_id: n",
"service_instance_graceful_shutdown_request_message_id: ",
"service_instance_hard_shutdown_request_message_id: ",
"service_instance_health_request_message_id: ",
"health_control_response_time: ", // the most recent response
};
serviceInstanceVec.add(serviceInstance);

Vector<String[]> serviceInstanceVec = new Vector<String[]>();
*/

// serviceInstance[7] == n

// set
// serviceInstance[8] =
"service_instance_graceful_shutdown_request_message_id: n",

            // send request to terminate (gracefully) a service instance

/*
type: graceful_shutdown_request
message_id: n
sub_type: Manager_to_agent
service_name: A
service_instance_id: i
*/

// and serviceInstance[9] =
"service_instance_hard_shutdown_request_message_id: n";

            // send request to terminate (hardly) a service instance

/*
type: hard_shutdown_request
message_id: n
sub_type: Manager_to_agent
service_name: A
service_instance_id: i
*/



// send request to close a communication session
      /*
      type: source_service_session_close_request
      message_id: o
      sub_type: Manager_to_agent
      source_service_name: A
```

```
source_service_instance_id: i
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_service_name: B
dest_service_instance_network_address: NA_j
dest_socket_name: S
dest_socket_port: k
dest_socket_new_port: l
*/

/*
type: dest_service_session_close_request
message_id: o
sub_type: Manager_to_agent
dest_service_name: A
dest_service_instance_id: j
dest_service_instance_network_address: NA_j
source_service_instance_network_address: NA_i
source_plug_name: P
source_plug_port: m
dest_socket_name: S
dest_socket_port: k
dest_socket_new_port: l
*/

// send "type: health_control_request" to service instance
/*
type: service_instance_health_request
message_id: o
sub_type: Manager_to_agent
service_instance_id: i
service_instance_network_address: NA_i
*/


// send "type: health_control_request" to agent
/*
type: agent_health_control_request
message_id: o
sub_type: Manager_to_agent
agent_network_address: NA
*/
```

4.7 Parameters for new communication session

```
public String[] getParametersForNewSession(String
destServiceName, String destSocketName){

String destServiceInstanceId;

String agentNetworkAddress;
// load balancing or new instance of service B (destServiceName) for determining
```

// the network address of Agent where an instance of service B is running

```
String portNumber;
```
// port number of server socket S (`destSocketName`) of one of the instances of service B
// running by the Agent with `agentNetworkAddress`

```
String[] parameters = {
destServiceInstanceId,
agentNetworkAddress,
portNumber,
     };
return parameters;
};
```

4.8 getPortForNewInstance

```
public String getPortForNewInstance(String
agentNetworkAddress){

int index = 0;
boolean check = true;

while ( check && !(index.equals(handlerVector.size())) ){

if(handlerVector.get(index).getAgentId().equals(agentNetworkAd
dress)) {

check = false;
port = handlerVector.get(index).getNewPortNumber();
     };
index++;
}; // end of while loop

if (check.equals(false)){
return port;
}
else{return "0"};
```

4.9 getNewMessageId()

```
int messageIdCounter = 30000;

public String getNewMessageId(){

String messageId = String.valueOf(messageIdCounter);
messageIdCounter++;
return messageId;

};
```

4.10 Execution of a new instance of service

```
int serviceInstanceIdCounter = 0;

// called only by Manager
public String getNewServiceInstanceId(){

String id = String.valueOf(portNumberCounter);
// String id = Integer.toString(ServiceInstanceIdCounter);
ServiceInstanceIdCounter++;
return id;

};

// recall the vector serviceVec  of service structures
// an example of service structure in Manager's service repository
/*
String[] service = {
"agent_network_address: NA_i",  // initially, NA_i is empty string
"service_name: A",
"sockets: [sequence of socket names]",
"plugs: [sequence of plug names]"
};
serviceVec.add(service);

Vector<String[]> serviceVec = new Vector<String[]>();
*/

// initially the vector serviceVec is constructed on the basis of graphOfCNApp
    // where service[0] = "agent_network_address: "


// execute a new service instance
public static void executeServiceInstance(
String serviceName,
String vertexId,  // in the abstract graph of CNApp
String agentNetworkAddress,
){

// message to be sent to Agent if of the following form
/*
type: execution_request
message_id: n
agent_network_address: NA_i
service_name: A
service_instance_id: i
socket_configuration: [configuration of sockets]
plug_configuration: [configuration of plugs]
```

```
*/
// the following additional parameters of this message are needed

String messageId = getNewMessageId();
String serviceInstanceId = getNewServiceInstanceId();
String configurationOfSockets;
String configurationOfPlugs;

// first, get the socket names
String sequenceOfSocketNames;
int index = 0;
boolean check = true;
String[] service =new String[4];

while ( check && !(index.equals(serviceVec.size())) ){

if(
serviceVec.get(index)[0].equals("agent_network_address: " +
agentNetworkAddress)
&&
serviceVec.get(index)[1].equals("service_name: "
+ serviceName)
) {

check = false;
service = serviceVec.get(index);
     };
index++;
}; // end of while loop

if (check.equals(false)){

sequenceOfSocketNames = service[2].split(" ").[1];

String[] socketsNames = sequenceOfSocketNames.split(", ");

for (String socketName : socketsNames)
{
socketConfiguration = socketConfiguration + socketName + ", "
+ getPortForNewInstance(agentNetworkAddress) + " & ";
};

// get the connections in the abstract graph of CNApp with vertex1 == A  and vertexId
// recall that vector of connections represents abstract graph of CNApp
// an example of a connection   (A, (P,S), B) between vertex1 and vertex2
/*
String[] connection = {
"connection_id: id",
"vertex1: idA",
"service_name: A",
"plug: P",
"socket: S",
"service_name: B",
"vertex2: idB"
};
graphOfCNApp.add(connection);

Vector<String[]> graphOfCNApp = new Vector<String[]>();

*/

int index = 0;
String P;
String S;
String B;

while (!(index.equals(graphOfCNApp.size())) ){

if (
connection[1].equals("vertex1: " + vertexId)
&&
connection[2].equals("service_name: " + serviceName)
) {

P = connection[3].split(" ")[1];
S = connection[4].split(" ")[1];
B = connection[5].split(" ")[1];

configurationOfPlugs = configurationOfPlugs +  P + ", " + S +
", " + B + " & ";
index++;
};

// message to Agent

String messageToAgent = "type: execution_request ~ message_id:
" + messageId +  " ~ agent_network_address: " +
agentNetworkAddress + " ~ service_name: " + serviceName + " ~
service_instance_id: " + serviceInstanceId + " ~
socket_configuration: " + configurationOfSockets + " ~
plug_configuration: " + configurationOfPlugs;

// send message to Agent
int index = 0;
boolean check = true;

while ( check && !(index.equals(handlerVector.size())) ){

if(handlerVector.get(index).getAgentId().equals(agentNetworkAd
dress)) {

check = false;
handlerVector.get(index).toAgent(messageToAgent);
     };
index++;
}; // end of while loop

// create new service instance structure in serviceInstanceVec
     // with "service_instance_status: pendigOn"
     // and wait to response from Agent to be processed in the main while loop
// an example of service instance structure in Manager's vector of service instances

String[] serviceInstance = {
"service_instance_network_address: " + agentNetworkAddress,
"service_name: " + serviceName,
"service_instance_id: " + serviceInstanceId,
"vertex_id: " + vertexId,
"service_instance_status: pendigOn",
"socket_configuration: " + configurationOfSockets,
"plug_configuration: " + configurationOfPlugs,
"service_instance_execution_request_message_id: " + messageId,
"service_instance_graceful_shutdown_request_message_id: ",
"service_instance_hard_shutdown_request_message_id: ",
"service_instance_health_request_message_id: ",
"health_control_response_time: ", // the most recent response
};
serviceInstanceVec.add(serviceInstance);

}; // end of the method executeServiceInstance(---)
```

4.11 Business logic of Manager

- Send request to an Agent to execute an instance of API Gateway. The rest of CNApp, i.e. service instances are executed automatically according to SSMMP
- Monitoring (via Agent) performance of service instances including API Gateways
- Load balancing by service instance replication.
- Load balancing by replication of API Gateway using DNS: aliases and canonical names.
- Reconfiguration and replacement of service instances to other nodes.
- Recovery from failures is done automatically as long as instance of API Gateway is running.