UNICAMP
Universidade Estadual de Campinas

FEEC
Faculdade de Engenharia Elétrica e de Computação

Aluno: Mateus Neves Barreto
R.A.: 142358
Disciplina: IA006
Professor: Ricardo R. Gudwin

Relatório – Aula 7

1 Atividade 1

Este tutorial visa a aplicação do conhecimento obtido em atividades anteriores no mundo virtual *"WorldServer3D"*. A atividade 1, aborda detalhadamente a análise dos códigos dos projetos: *"WorldServer3D", "DemoSOAR"* e *"WS3DProxy"*.
Analisando a classe *SimulationSOAR* presente no projeto *DemoSOAR,* pode-se observar que é utilizada uma chamada de sistema via *java* para identificar dois itens. O primeiro é a identificação do sistema operacional que a aplicação está sendo executado no momento. O segundo item, é a identificação da arquitetura que o sistema operacional (SO) está rodando, ou seja, mesmo que a estação de trabalho seja de 64 *bits* e o sistema operacional instalado seja de 32 *bits,* o retorno da arquitetura será referente a instalação do SO, Figura 1.

```
NativeUtils.setLibraryPath(".");
System.out.println("OS:" + System.getProperties().getProperty("os.name")
        + " Architecture:" + System.getProperties().getProperty("os.arch"));

String osName = System.getProperty("os.name").toLowerCase(Locale.ENGLISH);
String osArch = System.getProperty("os.arch").toLowerCase(Locale.ENGLISH);
```

Figura 1 – Identificando SO e arquitetura.

Um exemplo para a execução do código presente na Figura 1 pode ser o caso de uma estação de trabalho contendo um SO Linux Ubuntu de 64 *bits.* As variáveis *osName* e *osArch* irão conter respectivamente Linux e amd64.
Esta tarefa é necessária devido a necessidade da importação de arquivos para a configuração da interface de comunicação do *java* com o controlador SOAR. Identificado o SO e a arquitetura do mesmo, é possível carregar as bibliotecas e *dlls* específicas e necessárias para a criação desta interface de comunicação, de acordo com o SO e a arquitetura. Esta configuração é realizada utilizando a classe *NativeUtils.* Com a função *"setLibraryPath(".")"* é setado o diretório onde serão importados os arquivos (*libs, dlls e/ou só*). Os arquivos são importados através da função *"loadFileFromJar("path")"*, que os importa para o diretório, escolhido anteriormente, utilizando as classes padrões de manipulação de arquivos presentes na *java.io* respectivamente *InputStream* e *OutputStream,* Figura 2. Para o exemplo citado anteriormente, os caminhos dos arquivos importados seriam respectivamente: *"/linux64/libSoar.so", "/linux64/libJava_sml_ClientInterface.so"* e *"/soar-rules.soar",* Figura 3.

```
// Open and check input stream
InputStream is = NativeUtils.class.getResourceAsStream(path);
if (is == null) {
    throw new FileNotFoundException("File " + path
            + " was not found inside JAR.");
}

// Open output stream and copy data between source file in JAR
OutputStream os = new FileOutputStream(temp);
try {
    while ((readBytes = is.read(buffer)) != -1) {
        os.write(buffer, 0, readBytes);
    }
} finally {
```

Figura 2 – Modo de importação dos arquivos de configuração.

```
} else if (osName.contains("nix") || osName.contains("nux")) {
    if (osArch.contains("64")) {
        System.out.println("Linux 64 bits");
        NativeUtils.loadFileFromJar("/linux64/libSoar.so");
        NativeUtils.loadFileFromJar("/linux64/libJava_sml_ClientInterface.so");
    } else {
        System.out.println("Linux 32 bits");
        NativeUtils.loadFileFromJar("/linux32/libSoar.so");
        NativeUtils.loadFileFromJar("/linux32/libJava_sml_ClientInterface.so");
    }
} else {
```

Figura 3 – Importação dos arquivos de configuração.

Após cada importação é realizado o carregamento do arquivo importado para a execução do *java*, utilizando o comando *"System.load(temp.getAbsolutePath())"*.
Ainda na classe *SimulationSOAR* são realizadas as configurações para a execução da classe *SimulationTask* onde se inicializa a comunicação com o *WorldServer3D* utilizando as regras do SOAR. Em seguida é realizada a criação do avatar para ser utilizada pelas regras do operador SOAR e se inicia a classe *SoarBridge,* para a execução do controlador SOAR. Então a linha de execução inicia uma iteração de *loop* infinito para a executar as atividades propostas pelo código presente no arquivo *soar-rules.soar,* através da execução da função *runSimulation()* presente na classe *SimulationTask,* Figura 4.

```
SimulationTask simulationTask = new SimulationTask();
simulationTask.initializeEnviroment(Boolean.FALSE);
simulationTask.initializeCreatureAndSOAR(soarRulesPath, true,
        soarDebuggerPath, soarDebuggerPort);
// Run Simulation until some criteria was reached
Thread.sleep(3000);
while (true) {
    simulationTask.runSimulation();
    Thread.sleep(100);
}
```

Figura 4 – Inicia as atividades no *WorldServer3D*.

A classe *SimulationTask* possui um objeto, chamado *proxy*, instanciado com a classe *WS3DProxy*. Através do mesmo, é possível realizar operações no mundo virtual em execução do *WorldServer3D*, dentre as operações pode-se citar a criação da criatura (que retorno o objeto *Creature*) e adiquirir

("getar") o objeto *World*, o mundo virtual corrente, Figura 5.

```
c = proxy.createCreature(100, 100, 0);
w = proxy.getWorld();
c.start();
w.grow(1);
```

Figura 5 – Manipulação através do *WS3DProxy*.

A classe *SimulationTask* também possui um objeto para a comunicação com o controlador SOAR, o mesmo é declarado como *soarBridge* e instanciado com a classe *SoarBridge*. Utilizando este objeto é possível realizar a interação com controlador SOAR. Isto é realizado a partir do *loop* infinito presente na Figura 4, de modo que a cada iteração são realizadas as seguintes tarefas, Figura 6:

- Através do objeto *Creature* são adquiridos todos os estados da criatura no *WorldSever3D;*
- Possuindo as informações, são preparadas as estruturas a serem inseridas no controlador SOAR, informações de coisas (*things*) presentes no mundo virtual e informações referentes a criatura;
- Preparada as estruturas a serem inseridas no controlador SOAR, são enviadas através do objeto *soarBridge*;
- Após o envio para o controlador SOAR, são optidas as respostas do controlador;
- As respostas são processadas e tomadas enviando comandos ao mundo virtual.

```
public void runSimulation() throws SoarBridgeException, NullPoin
    if (soarBridge != null) {
        c.updateState();
        //Status currentStatus = c.getState();
        List<Thing> v = c.getThingsInVision();
        logger.info("Objetos no Ambiente: " + v.size());
        System.out.println("Objetos no Ambiente: " + v.size());
        for (Thing t : v) {
            logger.info(t.toString());
        }
        if (c != null) {
            // Prepare Creature To Simulation
            prepareAndSetupCreatureToSimulation(c, v);
            // Run simulation
            soarBridge.runSimulation();
            // Process Responde Commands
            processResponseCommands();
        } else {
            throw new NullPointerException("There are no creatur
        }
```

Figura 6 – Ciclo de execução.

A primeira chamada para a atualização é a da função presente na classe *Creature* (objeto c), *c.updateState()*. A atualização é realizada utilizando uma função estática da classe *CommandUtility*, utilizando a chamada *"CommandUtility.sendGetCreatureState(myselfName)"*, onde a variável *myselfName* é o ID da criatura. Através desta chamada é possível opter todas as informações como, informações da criatura e coisas em sua visão, Figura 7 e 8.

```
StringTokenizer st = CommandUtility.sendGetCreatureState(myselfName);

//////////////////Creature data:
if (!st.hasMoreTokens()) {
    logger.log(Level.SEVERE, "Error - myName missing!");
} else {
    command = st.nextToken();
    myselfName = command; //string
}
if (!st.hasMoreTokens()) {
    logger.log(Level.SEVERE, "Error - index is missing!");
} else {
    index = st.nextToken();
}
if (!st.hasMoreTokens()) {
```

Figura 7 – Atualizando informações sobre o mundo virtual – I.

```
public static synchronized StringTokenizer sendGetCreatureState(String robotNameID)
    String controlMessage = "getcreaturestate " + robotNameID;
    return sendCmdAndGetResponse(controlMessage);
}
```

Figura 8 – Atualizando informações sobre o mundo virtual – II.