

# Universidad Nacional de La Matanza

## Departamento de Ingeniería e Investigaciones Tecnológicas

# Trabajo práctico

# Sistemas multiagentes

# Tema Diabetes

Profesores:

Dr. Ierache, Jorge

Mg.Mangiarúa Nahuel

Ing. Montalvo Christian

Ing. Becerra Martín

Alumnos DNI

Comisión segundo cuatrimestre 2017

Objetivo TP

1. Conocer las diferentes dimensiones que componen a un sistema multiagente.
2. Construir un sistema multiagente.
3. Integrar diferentes algoritmos ML realizados en el Trabajo práctico 2.

Consigna TP

Seguir el tutorial para desarrollar el prototipo del sistema y completar las respuestas de las partes prácticas en el presente formulario. Para realizar el prototipo experimental se deberá utilizar la herramienta ​***JaCaMo framework***​.

Se debe presentar un archivo comprimido MAS\_nombre\_de\_grupo.rar que incluya:

1. El presente formulario completo, en formato pdf**.**
2. El tutorial entregado adjunto.
3. Código fuente del sistema multiagente representado en las **tres**​ **dimensiones** que provee el framework: ​**Jason, Cartago y Moise**.​

Notas de integración de modelos de algoritmos ML:

Agregar los archivos \*.model en la carpeta models del proyecto eclipse y agregar los nombres de los archivos en los vectores correspondientes treeModelNames y neuralNetworkModelPaths en la clase determineModelPath.

## Trabajo práctico Sistemas multiagentes

## Formulario a entregar

#### Consigna N°1

Hasta ahora vimos diferentes componentes de la arquitectura BDI: Creencias, deseos, intenciones, metas y planes.

1. Con el sistema corriendo, abrir el **mind inspector**, seleccionar al agente **measurement\_agent** e indicar que componente que no vimos hasta ahora se ve ahí. (Tip: a la izquierda de los elementos en verde indica que son).
2. Completar la siguiente tabla con las percepciones capturadas en el mind inspector que fueron originadas por el artefacto DatasetReader:

|  |  |
| --- | --- |
| Creencia | Descripción |
| current\_patient\_tuple([*6*,*148*,*72*,*35*,*0*,*33.6*,*0.627*,*50*]) |  |
|  | Número de tuplas del dataset cargadas |
| number\_of\_last\_tuple\_read(*0*) |  |

**Base de creencias**

En el mind inspector podemos inspeccionar la base de creencias de los agentes que seleccionemos. El programador puede agregar creencias en el cuerpo de un plan, a modo de una nota mental (como si fuese un recuerdo) si lo considera necesario:

+nombre\_creencia(dato1,dato2,dato3);

De esta manera veremos a la creencia creada en el inspector.

#### Consigna N°2

Agregar el siguiente plan en el agente measurement agent:

+current\_patient\_tuple(PatientDataTuple) : true

<-

.println(PatientDataTuple);

?number\_of\_last\_tuple\_read(NumberOfCurrentTuple);

!readNextPatientDataTuple(NumberOfCurrentTuple + 1).

Nota: ?number\_of\_last\_tuple\_read(NumberOfCurrentTuple) nos permite consultar si tenemos esa creencia cargada en la base de creencia del agente. La variable NumberOfCurrentTuple es cargada con el dato que guardamos en la misma.

a) Completar el siguiente plan descriptor del último plan agregado:

|  |  |
| --- | --- |
| Plan Description: |  |
| Evento(Trigger): |  |
| Context: |  |
| Uses data: |  |
| Procedure (Que hace, paso a paso): |  |

**Todavía no ejecutaremos** el sistema hasta que agreguemos planes para cumplir la meta: !readNextPatientDataTuple de la próxima consigna.

#### Consigna N°3

1. Desarrollar dos planes en el código fuente de measurement agent siguiendo los siguientes plan descriptors: Uno para continuar con la lectura de tuplas y el otro para imprimir un mensaje cuando finaliza sus tareas. Para más información sobre Jason ver la próxima sección de tips.

Plan Descriptors

|  |  |
| --- | --- |
| Description: | El plan solo debe invocar la operación del artefacto readTuple. Buscar el artefacto PatientDatasetReader e implementarla como las operaciones externas que implementamos anteriormente. |
| Evento (Trigger): | !readNextPatientDataTuple(NumberOfNextTuple) |
| Context: | number\_of\_tuples(NumberOfTuples) **&** NumberOfNextTuple < NumberOfTuples |
| Percepts: | - |
| Actions: | readTuple(NumberOfNextTuple) |
| Uses data: | Beliefs:  number\_of\_tuples(NumberOfTuples) |
| Produces data: | - |
| Goal: | Leer próxima tupla. |

|  |  |
| --- | --- |
| Description: | El plan solo debe imprimir el texto "I have finished my task". |
| Evento (Trigger): | !readNextPatientDataTuple(NumberOfNextTuple) |
| Context: | Buscar la creencia number\_of\_tuples, y escribir la condición que NumberOfNextTuple sea un índice que iguale la cantidad de tuplas. |
| Percepts: | - |
| Actions: | .println("I have finished my task"). |
| Uses data: | Beliefs:  number\_of\_tuples(NumberOfTuples) |
| Produces data: | - |
| Goal: | Imprimir mensaje de finalización de tareas. |

1. **Ejecutar el sistema y completar consigna agregando captura de pantallas** para mostrar los resultados obtenidos**.** Tiene que verse claramente **la impresión del encabezado** y otra captura con la **impresión que indica que finalizo su tarea**.

#### Consigna N°4

En el agente diabetes\_diagnosis\_vote\_judge desarrollar el siguiente plan. Ver los tips más abajo.

|  |  |
| --- | --- |
| Description: | Este plan maneja la solicitud para registrarse en la sesión para realizar el voto por el resultado del diagnóstico de diabetes positivo o negativo.  Agente que realiza el diagnóstico en el plan se llama VoterAgent en este plan. |
| Evento (Trigger): | +new\_voter\_registration[source(VoterAgent)] |
| Context: | - |
| Percepts: | - |
| Actions: | 1. Buscar la creencia number\_of\_voters 2. Actualizar number\_of\_voters incrementado su valor en uno. 3. Buscar mentor en la base de creencias. 4. Enviar al agente (VoterAgent) un mensaje informando quien es su mentor. 5. Enviar al agente VoterAgent la meta focus(diagnosis\_result\_depository). 6. Enviar a agente VoterAgent la meta focus(measure\_comm\_medium). |
| Uses data: | Beliefs:  number\_of\_voters(Number)  mentor(MentorAgent)  Contenido de mensajes:  mentor(MentorAgent)  focus(diagnosis\_result\_depository)  focus(measure\_comm\_medium) |
| Produces data: | - |
| Goal: | Registrar agente a la sesión de votación. |

#### Consigna N°5

**Agregar** estos dos planes en archivo asl del el agente diabetes\_diagnosis\_vote\_judge

+!start\_patient\_dataset\_reader: tuple\_reader\_agent(TupleReaderAgent) <-

.send(TupleReaderAgent,achieve,start\_patient\_data\_reader).

+!read\_next\_patient\_dataset\_tuple: tuple\_reader\_agent(TupleReaderAgent) <-

.send(TupleReaderAgent,achieve,read\_next\_patient\_data\_tuple).

**Completar** los descriptores simplificados correspondientes:

|  |  |
| --- | --- |
| Plan Description: |  |
| Evento(Trigger): |  |
| Context: |  |
| Uses data: |  |
| Incoming and Outgoing Messages | Ejemplo  Contenido\_mensaje (Agente\_origen -> agente\_fin) |
| (Mensajes aquí) | (Descripción de mensajes aquí) |
| Procedure (Que hace, paso a paso): |  |

|  |  |
| --- | --- |
| Plan Description: |  |
| Evento(Trigger): |  |
| Context: |  |
| Uses data: |  |
| Incoming and Outgoing Messages |  |
| Procedure (Que hace, paso a paso): |  |

#### Consigna N°6

1. En el agente diabetes\_diagnosis\_vote\_judge desarrollar el siguiente plan:

|  |  |
| --- | --- |
| Description: | Este plan maneja el cierre de la sesión de votación cuando no hay más tuplas con datos del paciente para leer. |
| Evento (Trigger): | +number\_of\_votes(CurrentNumberOfVotes) |
| Context: | vote\_session\_started **&** number\_of\_voters(NumberOfVoters) **&** CurrentNumberOfVotes == NumberOfVoters |
| Percepts: | - |
| Actions: | 1. Obtener los resultados de la votación con la operación indicada en la sección external action. 2. Imprimir por consola con .println los resultados que se obtuvieron con el formato indicado en la sección produces data. 3. Finalizar agregando la intención !read\_next\_patient\_dataset\_tuple para buscar la próxima tupla |
| Uses data: | Beliefs:  number\_of\_voters(NumberOfVoters)  Intentions:  !read\_next\_patient\_dataset\_tuple |
| Internal actions | .println(“Informe”) |
| External actions | getVotationResults(TupleNumber,PositiveVotes,NegativeVotes) |
| Produces data: | Informe: "Positive votes: ",PositiveVotes," Negative votes: ",NegativeVotes |
| Goal: | Cerrar la votación |

1. Por último, agregar este plan faltante:

+no\_tuples\_to\_read <-

.println("Votation session ended").

1. **Ejecutar el sistema** completo e imprimir **una captura de los resultados obtenidos**.

#### Consigna N° 7

Crear un agente doctor\_agent.asl siguiendo los siguientes descriptors:

**Agent Descriptor**

|  |  |
| --- | --- |
| Name: | doctor\_agent |
| Description: | El agente tiene la función de contar la cantidad de diagnósticos realizados y cuantos pacientes son casos positivos y negativos. |
| Cardinality: (min-max) | 1 - 1 |
| Initialisation: | Debe tener como intención inicial, alcanzar la meta !configure\_my\_result\_artefact. |
| Incoming and Outgoing Messages: | +partial\_diagnosis\_result.  diabetes\_diagnosis\_vote\_judge\_agent -> doctor\_agent |
| Goals: | !configure\_my\_result\_artefact  !final\_report |
| Included plans: | Implementar los siguientes plan descriptors para responder a los siguientes eventos:  +!configure\_my\_result\_artefact  +partial\_diagnosis\_result  +!final\_report |

**Plan Descriptors**

|  |  |
| --- | --- |
| Name: | Configuración del artefacto de doctor agent |
| Description: | En este plan se crea el artefacto “DoctorResultArtefact” para luego poder contar la cantidad de diagnósticos realizados. |
| Trigger: | !configure\_my\_result\_artefact |
| Context: | - |
| Percepts: | - |
| Actions: | makeArtifact, focus |
| Goal: | Crear y configurar el artefacto |
| Procedure: | 1. Crear el artefacto con el literal doctor\_result\_artefact y la ruta de archivo:   “diabetes\_mas.DoctorResultArtefact”   1. Hacer focus en él. |

|  |  |
| --- | --- |
| Name: | Agregar resultados de votación al artefacto: DoctorResultArtefact |
| Description: | El plan permite registrar la votación en el artefacto y decidir en base a los votos si es un caso positivo (Tiene diabetes) o negativo (No tiene diabetes). (Ver operaciones del artefacto). |
| Trigger: | +partial\_diagnosis\_result  (PatientTupleNumber,PositiveCases,NegativeCases) |
| Context: | - |
| Incoming and Outgoing Messages: | +partial\_diagnosis\_result.  (diabetes\_diagnosis\_vote\_judge\_agent -> doctor\_agent) |
| Percepts: | - |
| Actions: | addPartialDiagnosisResult(PositiveCases,NegativeCases) |
| Goal: | Definir por cantidad de votos si es un caso positivo o negativo (en operación de artefacto realiza esto). |
| Procedure: | 1. Agregar el resultado de la votación al artefacto. |

|  |  |
| --- | --- |
| Name: | Impresión de informe final |
| Description: | Este plan imprime el informe final por pantalla |
| Trigger: | +!final\_report |
| Context: | - |
| Incoming and Outgoing Messages: | - |
| Percepts: | - |
| Actions: | buildFinalReport(FinalReport), .println |
| Goal: | Mostrar el informe final de la sesión. |
| Procedure: | 1. Crear el informe final. 2. Imprimir informe en la consola general. |

#### Consigna N° 8

Agregar los siguientes elementos en el archivo xml de moisé:

* Role: *“final\_judge"* con cardinalidad: min 1 y max 1
* Goal: *“final\_report”* luego de la meta *close\_vote\_session*
* Misión: *"final\_judge\_mission"* con el goal agregado.
* Norma: *"norm\_final\_judge"* del tipo *"obligation"* uniendo el rol y misión creado.

**Descomentar** línea include("$jacamoJar/templates/org-obedient.asl") de doctor agent

**Agregar** un agente que implemente los planes del archivo asl creado en archivo **jcm.**

**No olvidar** de agregarlo en **la organización** en el mismo **archivo jcm.**

#### Consigna N° 9

1. Agregar un plan en el agente **diabetes\_diagnosis\_vote\_judge** que responda al commitment de la misión final\_judge\_mission y guardar una nota mental (creencia), final\_judge\_agent con el nombre del agente que se comprometió a cumplirla.
2. En el agente **diabetes\_diagnosis\_vote\_judge,** escribir un plan que cumpla el siguiente plan descriptor**:**

|  |  |
| --- | --- |
| Name: | Transmisión de resultado de votación de una tupla de datos |
| Description: | Cuando se termine la votación de una tupla de datos, se envían los resultados a doctor agent. |
| Trigger: | +number\_of\_votes(CurrentNumberOfVotes) |
| Context: | vote\_session\_started **&** number\_of\_agents\_subcribed(NumberOfAgents) **&** CurrentNumberOfVotes == NumberOfAgents |
| Procedure: | 1. Obtener de la base de creencias a doctor agent 2. Obtener los resultados de la votación con la operación getVotationResults como esta especificado en actions. 3. Enviar un mensaje a doctor agent con el contenido especificado en Messages 4. Ejecutar intención: !read\_next\_patient\_dataset\_tuple |
| Messages | partial\_diagnosis\_result(PatientTupleNumber,PositiveVotes,NegativeVotes) |
| Actions | getVotationResults(PatientTupleNumber,PositiveVotes,NegativeVotes); |

#### Consigna N° 10

**Ejecute sistema** y **adjunte capturas** de los resultados obtenidos. Se espera que el sistema multiagente muestre la tupla de datos correspondiente a cada paciente, los resultados de la votación de cada una y que imprima el informe final (síntesis) en el último paso.

## Cuestionario Final

1. Sintetice el propósito de cada dimensión (Jason, Cártago y Moisé) de la herramienta JaCaMo framework.
2. Explique cómo se lleva a cabo los conceptos de noción débil y fuerte de teoría de agentes en los agentes Jason. ¿Si una propiedad no se cumple como la llevaría a cabo?
3. Explique los componentes de un plan de un agente Jason.
4. ¿Qué tipos de acciones puede realizar un agente Jason? Explique para que sirve cada una. Por cada tipo de acción indique donde se reflejan las consecuencias de su ejecución.
5. Explique cómo se selecciona un plan para reaccionar a un evento cuando hay varias alternativas a elegir. ¿Qué pasaría si la condición de dos planes se cumplen a la vez?.
6. Explique con sus palabras que es un artefacto en Cártago. ¿Para qué se utiliza un artefacto en JaCaMo?
7. ¿Qué elementos vistos en la teoría se abstraen en un artefacto?
8. Describa las diferencias que observa sobre el ciclo de ejecución en la teoría de la arquitectura BDI y la implementación realizada en este trabajo práctico. Como llevaría a cabo el ciclo de ejecución de una arquitectura BDI como se encuentra en la teoría, con los elementos que conoce de JaCaMo.
9. Defina los elementos vistos en el modelo proporcionado por Moisé: Roles, grupos, metas, planes del equipo, misiones y normas. Sintetice en pocas palabras como se relacionan estos elementos para que un sistema multiagente cumpla con su meta principal.
10. ¿Conoce otras Herramientas para desarrollar sistemas multiagentes? Si conoce otras que ventajas/desventajas tienen cada una.