Desarrollo de Entornos Virtuales Inteligentes Basados en el Meta-Modelo MAM5

Jaime Andres Rincon Arango

Directores:

Dr. Carlos Carrascosa Casamayor
Dra. María Emilia García Marqués

Trabajo Final del Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Departamento de Sistemas Informáticos y Computación Valencia,
Julio 2014

Dedicado a
mis padres,
a Sol y a Itzael

Agradecimientos

En primer lugar quiero agradecer a mis padres por darme su apoyo incondicional en el transcurso de mis estudios. También quiero agradecer a Sol, por la paciencia y la comprensión que ha tenido. Y a mis dos tutores, que sin ellos este trabajo no hubiese sido posible.

Resumen

Los Entornos Virtuales Inteligentes (Intelligent Virtual Environment –IVE-) son herramientas de simulación y entretenimiento que se hacen cada vez más populares y permiten a particulares y empresas diseñar entornos de simulación que emulen características del mundo real. Estos entornos virtuales simulan un mundo real, teniendo en cuenta restricciones físicas con el objetivo de ofrecer un gran realismo que maximice, dentro de lo posible la sensación de inmersión de los usuarios. Para ello, en esta simulación deben permitir la interacción de estos usuarios con objetos y entidades que pueblen dicho entorno virtual. Estas entidades virtuales pueden estar dotadas de una cierta inteligencia que incremente la sensación de inmersión de los usuarios. Una de las técnicas que se puede utilizar para modelar estas entidades inteligentes son los Sistemas Multi-agente (Multi-Agent System –MAS-).

Este tipo de aplicaciones se encuentran entre las más demandadas hoy en día, no sólo por ser la clave para los juegos multi-usuario, tales como *World Of Warcraft* (con más de 7 millones de usuarios en 2013); sino también como la base de redes sociales virtuales como *Second Life* (con 36 millones de cuentas creadas en sus 10 años de historia). Estas aplicaciones plantean un gran reto debido al elevado número de entidades y objetos que podrían llegar a simular. Por este motivo es necesario contar con herramientas que faciliten su diseño y que a su vez sean altamente escalables. Además, deben ser capaz de adaptarse a los cambios, no sólo de la cantidad de entidades, sino también de las necesidades de los usuarios. La tecnología utilizada en la actualidad para desarrollar este tipo de productos, carece de elementos que faciliten la adaptación y la gestión del sistema. Tradicionalmente, este tipo de aplicaciones utilizan el paradigma cliente / servidor, pero debido a sus características, un enfoque distribuido, como el de los Sistemas Multi-Agente, permitiría el desarrollo de componentes que evolucionen de forma autónoma.

El presente trabajo plantea dar una solución a estas problemáticas, creando una herramienta que ayude al modelado, programación y simulación de IVEs. Para ello se ha creado $JaCalIVE$ (Jason Cartago to implement Inteligent Virtual Environments), que proporciona un método para desarrollar IVEs, junto con una plataforma de apoyo para su ejecución ($JaCalIVE$ Framework). $JaCalIVE$ se basa en el meta-modelo MAM5, que describe un método para diseñar *IVEs*. MAM5 a su vez se basa en el meta-modelo $A\&A$ el cual describe los entornos en donde habitan los Sistema Multi-Agente. Además se introduce un motor de simulación física, que permite el desarrollo de IVEs más realistas para el usuario, pudiéndose simular restricciones físicas que afectarían el comportamiento de todas las entidades que habitan el IVE.

Abstract

Intelligent Virtual Environments (IVE) are simulation and entertainment tools that are becoming more and more popular. They allow individuals and enterprises to design simulation envrionments that model real life features. These IVEs simulate a real world, taking into account physical restrictions in order to offer big realism that maximizes as far as possible the user's feeling of immersion. In order to achieve that, the simulation must allow the interaction between users, objects and entities populating the virtual environment. These virtual entities can have some intelligence that increases user's immersion feeling. One of the most used techniques to model these intelligent entities is Multi-Agent Systems.

This kind of application are among the most demanded ones, not only for being the key for multi-user games like World of Warcraft (with more than 7 millions of users in 2013), but also as the foundation of virtual social networks like Second Life (with 36 millions of users in its 10 years of history). These applications have to tackle with a huge number of entities and objects to simulate. Because of that it is important to have tools that facilitate the design and implementation of this kind of systems. Moreover, they must be able to adapt to changes not only in the amount of entities but also in the user needs. Current technology used for developing this kind of systems lacks of elements facilitating the adaptation and management of the system. Tradicionally, this kind of applications uses the client / server paradigm, but due to its features, a distrubuted approach, like Multi-Agent Systems, will allow the developing of components evolving autonomously.

The present work proposes a tool that deals with these open issues helping the modelling, programming and simulation of IVEs. This tool is called JaCalIVE (Jason Cartago to implement Intelligent Virtual Environments), and provides a method to develop IVEs along with an execution platform to simulate them. JaCalIVE is based on the MAM5 meta-model that describes a method to design IVEs based on A&A. Moreover, it introduces a physical simulation engine allowing realistic IVEs that includes physical restrictions that bound the behaviour of the entities populating the IVE.

Índice general

Capítulo 1

Introducción

Un entorno virtual (Virtual Environment ó VE) es una representación digital de un mundo, que puede tener un parecido al mundo real o ser un mundo imaginario [1]. En un VE los usuarios tienen la posibilidad de interactuar con los objetos que se encuentran dentro del mismo, brindándole la sensación de estar inmerso dentro de este mundo. Los VEs surgen en la década de los 60 [2], como una forma de transportar a los usuarios a mundos diferentes al real. Estos entornos despertaron una gran atención en diferentes industrias, debido a la gran aplicabilidad que podían ofrecer. Industrias como la de los videojuegos, que mueve millones de euros al año, vieron el verdadero potencial que tienen estas aplicaciones. Por otro lado, las empresas dedicadas a la creación de simuladores, se percataron de la posibilidad de utilizar los VEs como herramientas de entrenamiento para actividades poco convencionales, como el entrenamiento militar, el manejo de maquinaria pesada o la medicina.

Con los VEs se puede construir mundos digitales, que se asemejen o no al mundo real. Sin embargo, requieren de elementos de visualización avanzados para brindar a los usuarios un elevado nivel de inmersión. Esta inmersión es lograda gracias a los altos niveles de detalles que poseen hoy en día los nuevos motores gráficos, ya que permiten diseñar aspectos comunes del mundo real de manera sofisticada y realista, como el texturizado de los objetos, la ilumninación, las sombras o los mismos personajes. Además, es necesario contar con otros elementos, como los motores físicos, que que soporten el nivel de inmersión proporcionado por estos motores gráficos. Los motores físicos se encargan de las leyes físicas (detección de colisiones, gravedad o rozamiento) que rigen al VE, de manera que la simulación de los objetos puede ser dinámica o estática, permitiendo maximizar el realismo.

No obstante los VEs requieren de otro elemento que eleve el nivel de inmersión.Este elemento es, sin lugar a duda, la inteligencia artificial (IA), ya que puede dotar cierto grado de inteligencia a los objetos o personajes que habitan el VE, de manera que no es necesario su control por parte de los usuarios. Estas entidades son solo personajes que entran en contacto con el usuario y se les conoce como personajes no jugador (Non-player Character ó NPC); poseen una IA que les otorga un comportamiento autónomo.

Por lo tanto, un IVE es un espacio 3D que proporciona al usuario una colaboración, simulación e interacción con las entidades, gracias al modelo de IA que utilizan, incrementando aún más el nivel de inmersión que el usuario busca.

> "Un MAS es un sistema informático que está situado en algún entorno y que es capaz de ejecutar una acción autónoma para cumplir con sus objetivos de diseño".
>
> Michael Wooldridge [3]

Por otro lado, un MAS agrupa una cantidad definida de agentes de forma local o distribuida, que puede ser utilizada para abordar problemas muy complejos. Esta característica distribuida llama la atención ya que es posible tener diferentes agentes interactuando en un mismo entorno y ejecutándose en máquinas distintas, incluso en dispositivos móviles. Por ejemplo, diferentes investigaciones han utilizado el entorno 3D de *Second Life* [4, 5], como una forma de representar y visualizar sus entidades autónomas. Además de las aplicaciones en videojuegos, los MAS y los IVEs se pueden utilizar como herramientas en aplicaciones de robótica [6] o simulación [7].

Todas las entidades que se pueden encontrar dentro de los IVEs no tienen por qué ser agentes. De acuerdo a la definición de Wooldridge, un agente es sólo la parte inteligente del IVE (la mente), pero toda mente necesita tener un cuerpo para interactuar dentro del entorno. Por lo tanto, es necesario un meta-modelo para realizar esta diferenciación. El meta-modelo llamado $A\&A$ [8, 9] podría ser el indicado para esta tarea ya que permite distinguir dos tipos de entidades: los agentes y los artefactos. Los agentes implementan o encapsulan la inteligencia y los artefactos representan cuerpos de estos agentes. Siguiendo la misma metodología de separar la mente del cuerpo, también se encuentra un meta-modelo que está basado en el meta-modelo $A\&A$ y este distingue que entidades tendrán una representación virtual (visualización en algún motor gráfico). Multi Agent Model For Intelligent Virtual Environments (MAM5) [10] (Multi-Agent Model For Intelligent Virtual) es un meta-modelo para el diseño de IVEs, y permite al desarrollador separar del IVE, los agentes que tendrán una representación física.

MAM5 clasifica en dos grupos las entidades que se encuentran en el IVE: el primer grupo hace referencia a las entidades que poseen un cuerpo dentro del IVE y el segundo grupo son aquellas entidades que no poseen dicha representación. Actualmente no hay una plataforma para diseñar IVEs con estas características, por ello en este trabajo se presenta $JaCalIVE$, como una herramienta que ayude a los desarrolladores a modelar, programar y mantener los IVEs. $JaCalIVE$ es un framework que se basa en los meta-modelos $A\&A$, *MAM5*, los cuales diferencian claramente entre agentes y artefactos, con o sin representación virtual.

$JaCalIVE$ ha incorporado un motor físico que otorga a las entidades que habitan el IVE, las restricciones dinámicas y estáticas necesarias para realizar simulaciones

avanzadas. Los IVEs requieren de un sistema de visualización y $JaCalIVE$ no incorpora un motor 3D propio, sin embargo de esta forma el desarrollador tendrá libertad de utilizar cualquier motor gráfico, sea 3D ó 2D, que se ejecute en un entorno web, dispositivos móviles o aplicaciones de ordenador. Con todas esta características se obtiene un framework que incorpora todo lo necesario para la creación y mantenimiento de IVEs de forma rápida y eficaz.

1.1. Contexto

Este trabajo se ha realizado dentro del Grupo de Investigación de Tecnología Informática-Inteligencia Artificial (GTIA) del Departamento de Sistemas Informáticos y Computación (DSIC), en el marco de la línea de investigación de MAS. Además, se ha divulgado a través de dos congresos internacionales, realizados en la ciudad de Salamanca en el mes de Junio de 2014:

13th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)[11]

9th International Conference on Hybrid Artificial Intelligence (HAIS) [12]

El trabajo que se propone, contribuye al proyecto iHAS: Sociedades Humano-Agente: Diseño, Formación y Coordinación, financiado por el Ministerio de Ciencia y Economía con referencia TIN2012-36586-C03-01, PROMETEOII/2013/019. Este proyecto es la base de mi tesis doctoral que estará financiada a través de un contrato predoctoral otorgado por la Universidad Politecnia de Valencia, con referencia P2013-01276.

Capítulo 2

Objetivos

Este proyecto se formula a partir de la metodología MAM5 [10] sobre la integración de Sistemas Multi-Agente (MAS) con Entornos Virtuales Inteligentes (IVEs). El objetivo global de MAM5, es el de facilitar al desarrollador el diseño, creación y la posterior representación gráfica de entornos virtuales basados en el modelo enunciado anteriormente.

Para conseguir este objetivo global, se deben realizar los siguientes pasos:

1. Crear un método de especificación y representación de los datos que forman un IVE, así como un proceso que lea y analice estos datos a través de un formato interoperable basado en estándares.

2. Implementar un proceso gestor, llamado $JaCalIVEFrameWork$, que permita la simulación de IVEs y que, además, se encargue de la creación y mantenimiento de todos sus elementos.

El objetivo concreto del presente trabajo se centrará, en cumplir lo establecido en el segundo punto. Para lograr lo propuesto en el punto dos del objetivo global, se planteó la ejecución de los siguientes objetivos específicos:

1. Construir una infraestructura que permita la creación y gestión de agentes y artefactos, integrados dentro de un IVE.

2. Dotar al IVE de un adecuado realismo físico, como la simulación de gravedad, rozamiento, velocidad y cualquier otra restricción física.

3. Establecer un API que permita la independencia de la visualización con respecto al entorno.

4. Definir un método mediante el cual el diseñador de IVEs pueda definir su sistema.

Capítulo 3

Estado del Arte

En este capítulo se introducirán conceptos relacionados con Sistemas Multi-Agente, metodologás para el modelado de entornos, en especial la de $A\&A$ y las diferentes plataformas que permiten la construcción de agentes inteligentes. Dado que este trabajo se enmarca dentro del campo de los Entornos Virtuales, este capítulo también incluye secciones sobre Entorno Virtuales, simulación, Entornos Virtuales Inteligentes y por último se detallará el meta-modelo MAM5 en el cual se basa este trabajo.

3.1. Entornos Virtuales (VE)

Un Entorno Virtual (VE) es una representación digital de un mundo[1], y puede ser parecido al mundo real o un mundo imaginario. En estos mundos virtuales, los usuarios pueden interactuar y sentir que se encuentran dentro del VE. La interacción puede suceder entre un usuario y el mundo virtual o entre varios usuarios conectados a este mundo virtual e incluso interactuar entre ellos. Es por esto que este tipo de aplicaciones son muy utilizadas en videosjuegos[13] y en simulación[14].

3.2. Interfaz Humano Maquina

Además de las aplicaciones en simulación y videojuegos, los IVEs ofrecen la posibilidad de crear una interacción entre el usuario final y el propio IVE. Esta interacción se puede establecer a través de herramientas de interfaz humano-maquina, de manera que el usuario es capaz de interactuar con las entidades que habitan el IVE. $JaCalIVE$ es una plataforma que le permite al desarrollador integrar estas herramientas y crear aplicaciones con un alto nivel de inmersión. A continuación se describen algunas interfaces humano-maquina. En la Figura 3.1 se observa la clasificación de estas interfaces de acuerdo a la antigüedad y el nivel de adopción.

A partir de esta clasificación se seleccionaron las interfaces de mayor interés para la creación de una interacción entre el IVE y el usuario:

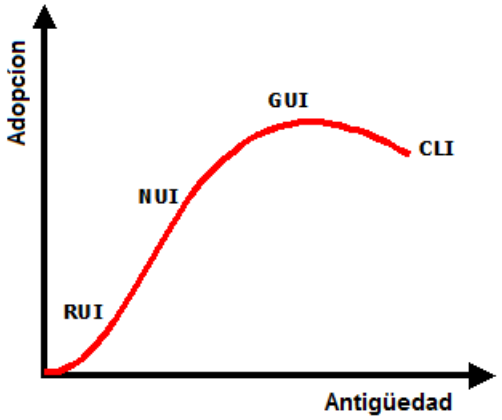

Figura 3.1: Clasificación Interfaz Humano Maquina

1. Interfaz Natural de Usuario (Natural User Interfaces - NUI).
2. Interfaz de Usuario para Realidad Virtual (Reality User Interface - RUI).
3. Interfaz Cerebro Computadora (Brain Computer Interface - BCI).

3.2.1. Interfaz Natural de Usuario (Natural User Interfaces - NUI)

Las Interfaces de Usuario Natural (NUI) posibilitan al desarrollador crear IVEs donde los movimientos gestuales del usuario, realizados con las manos o los pies, son utilizados como mando de control de alguna entidad que habite el IVE. En el mercado existen un gran número de dispositivos que proporcionan este tipo de interfaz, como los *touch, multi touch, gesture, voice o face recognition*, o incluso los *Smartphones, tablets, Wii o Kinect*. De estos dispositivos quizás el más utilizado es el *Kinect*; es capaz de capturar los movimientos para ser convertidos en órdenes. Si el desarrollador lo integra en el diseño de IVEs, permitiría al usuario una interacción que con otros dispositivos no podría realizar. El *Kinect* de Microsoft que se observa en la Figura 3.2 cambió completamente el concepto de interacción humano-máquina. Este dispositivo, inicialmente pensado para videojuegos, marcó la diferencia en la forma de jugar, pasando de los mandos con botones y joystick a controlar los personajes utilizando el cuerpo. El *Kinect* se ha estado utilizando como instrumento de captura de gestos corporales para navegación [15], reconocimiento de gestos de manos [16] y cualquier otro tipo de aplicación con la necesidad de una interacción natural.

3.2.2. Interfaz de Usuario para Realidad Virtual (Reality User Interface - RUI)

Como se describió en la sección 3.2.1, las NUI permiten interactuar con aplicaciones utilizando dispositivos como el kinect o el multi touch. Y aunque este tipo interfaz faculta la interacción con entornos 3D, no permite realizar visualizaciones.

Figura 3.2: Kinect Xbox.
(Fuente: http://www.xbox.com/es-ES/Kinect)

La realidad virtual y realidad aumentada son tecnologías pioneras en el campo de la visualización y en entornos de simulación; permiten a los usuarios estar inmersos en mundos virtuales. La NASA fue uno de los primeros centros en utilizar esta tecnología , como se observa en la Figura 3.3.

Figura 3.3: REalidad Virtual de la NASA, pionero Scott Fisher.
(Fuente: http://warrenrobinett.com/nasa/index.html)

La desventaja de estos dispositivos en sus inicios, era el elevado costo y su gran tamaño. Sin embargo en los últimos años ha sido posible acceder a este tipo de dispositivos a precios módicos y formas más ergonómicas, adquiriendo un valor significativo en las representaciones virtuales. La *Interfaz de Usuario para Realidad Virtual (Reality User Interface - RUI)*, permite al usuario visualizar información que puede ser superpuesta en el mundo real como es el caso de la realidad aumentada o la creación de un mundo virtual distinto al real. Por ejemplo, estas dos formas de visualización han supuesto verdaderos intereses comerciales con el anuncio de las Google Glass como se observa en la Figura 3.4. Este dispositivo permite visualizar la información de un smart phone en una pequeña pantalla y podría ser utilizado como una forma de visualización de información proveniente de un IVE. Este tipo de interfaz está a pocos meses de llegar al mercado y ya varias empresas están ofreciendo los dispositivos en pre-orden. Destacan marcas comerciales como Oculus 3D[1] (Figura 3.5) o Sulon[2] (Figura 3.6).

[1] http://www.oculusvr.com/
[2] http://sulontechnologies.com/

Figura 3.4: Google Glass.
(Fuente: https://www.google.com/glass/start/)

Estos dispositivos poseen APIs para algunos de los motores gráficos que se han mencionado anteriormente.

Figura 3.5: Oculus 3D DK2.
(Fuente: http://www.oculusvr.com/dk2/)

Figura 3.6: Sulon Development Kit.
(Fuente: http://sulontechnologies.com/)

3.2.3. Interfaz Cerebro Computadora (Brain Computer Interface - BCI)

La tecnología de los *Interfaz Cerebro Computadora (Brain Computer Interface - BCI)* ha crecido considerablemente en los últimos años, con dispositivos portátiles para la captura de señales electroencefalográficas (EEG). Esta interfaz captura y procesa las señales cerebrales con el fin de que los desarrolladores creen aplicaciones que permitan interactuar con ordenadores, tablets o smartphones. Estos dispositivos no invasivos son comercializados a través de empresas como *Emotiv*[3] (Figura 3.8) o *NeuroSky*[4] (Figura 3.7).

[3] http://www.emotiv.com/
[4] http://www.neurosky.com/

Figura 3.7: NeuroSkiy Mind Wave.
(Fuente: http://store.neurosky.com/)

Esta empresa promocionan sus dispositivos para diversión e investigación, siendo la investigación uno de los campos más utilizados. Así, se puede encontrar investigaciones relacionada con el uso de *NeuroSky* para detectar los niveles de atención[17], el reconocimiento de emociones con *Emotiv EPOC* [18] o en video juegos [19].

Figura 3.8: Emotiv EEG.
(Fuente: http://emotiv.com/about/media/)

3.3. Simulación

Como se mencionó anteriormente, un VE también puede ser utilizado como un entorno de simulación y como aplicación es interesante porque permite el entrenamiento en un ambiente seguro. Este tipo de entornos se están implementando en simulaciones avanzadas en medicina [20] ó robótica [21]. Para que un VE tenga la facultad de sumergir al usuario dentro de este entorno, debe contar un alto nivel en el detalle gráfico, implementar una simulación física y una Inteligencia Artificial (AI). Hoy en día es posible encontrar motores gráficos con un alto nivel de detalle que están diseñados principalmente para el desarrollo de videojuegos. La industria de los videos juegos invierte millones de euros al año con el fin de mejorar estos motores, permitiendo un nivel de detalle muy alto, un texturizado muy realista y una gran gama de iluminación ó sombras. Por ejemplo, en la Figura 3.9 se muestra

una imagen del terreno diseñado en Unity3D[5]. *Cryengine*[6] y *Unreal UDK* de Epic Games[7], como se observa en las Figuras 3.10 y 3.11 respectivamente, son otro ejemplo de otros tipos de motores gráficos con alto nivel de gráficos y que actualmente se usan para el diseño comercial de videojuegos multi-plataforma.

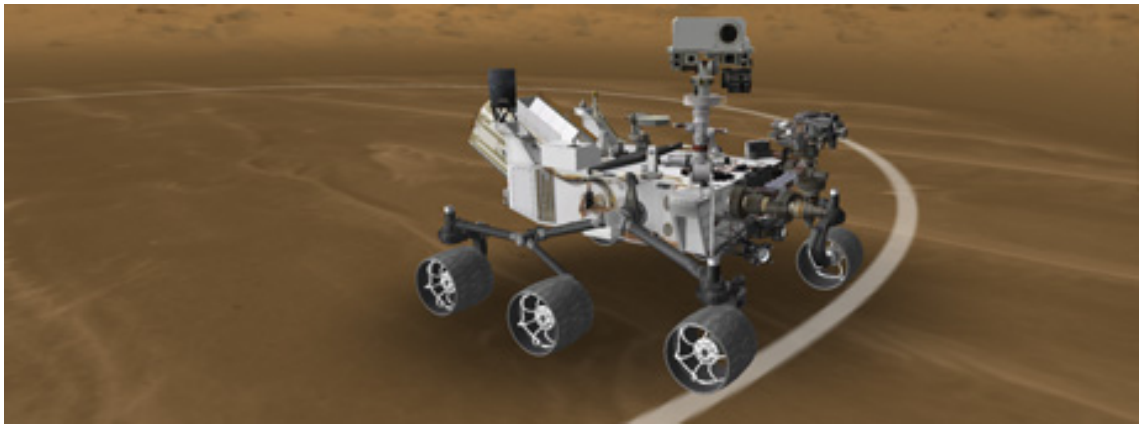

Figura 3.9: Motor Gráfico Unity 3D.
(Fuente:http://unity3d.com/company/sim/profiles)

Figura 3.10: Motor Gráfico Cryengine.
(Fuente: http://www.crytek.com/cryengine/cryengine3/overview)

Cabe recordar que también es necesario un motor físico que dé soporte real a esta potencia gráfica. Aunque estos motores gráficos cuentan con su propio motor de simulación física, existen otros motores físicos que pueden ser utilizados en diferentes aplicaciones de simulación, como ODE (Open Dynamics Engine)[8], Bullet (Real-Time

[5]http://unity3d.com/es/unity/quality/terrains
[6]http://www.cryengine.com/
[7]http://epicgames.com/
[8]http://www.ode.org/

Figura 3.11: Motor Gráfico Unreal UDK.
(Fuente: https://www.unrealengine.com/blog/unreal-engine-4-rivalry-demo-debuts-at-google-io)

Physics Simulation)[9] y *JBullet* (Java port of Bullet Physics Library)[10].

3.4. Sistema Multi-Agente (MAS)

Un Agente es una entidad virtual inteligente, capaz de interactuar en un entorno y modificarlo según sean sus objetivos. Es capaz de realizar tareas autónomas y tomar decisiones, de acuerdo a las percepciones que adquiere del entorno. Dichas percepciones forman parte del conocimiento que el agente tiene del lugar donde se encuentra, y son adquiridas a través de sensores o métodos externos que le otorgan información relacionada con el entorno. La información es utilizada por el agente, para ejecutar una acción que esté acorde con sus objetivos. Estos objetivos son, básicamente, los que le proporcionan el comportamiento cognitivo con respecto al entorno, como se muestra en la Figura 3.12.

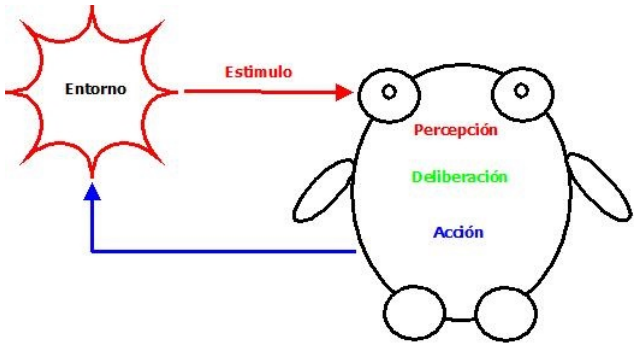

Figura 3.12: Agente

Por otro lado, un MAS agrupa una cantidad definida de agentes de forma local o distribuida. Esta filosofía se utiliza para abordar problemas muy complejos. Si se conoce que un agente posee unas características especiales y tiene unos objetivos claramente definidos, y se le asocia otro agente con diferentes características pero

[9] http://bulletphysics.org/wordpress/
[10] http://jbullet.advel.cz/

con los mismos objetivos, se podrá resolver entonces un problema, teniendo en cuenta las habilidades de cada uno de los agentes. La solución propuesta es gracias a que todo agentes tiene dentro de su estructura un comportamiento social, que le permite comunicarse con otros agentes ubicados incluso en máquinas distintas, otorgando a los MAS el carácter distribuido.

Cuando el diseñador quiere modelar el entorno, busca que el diseño tenga un alto grado de realismo, tanto desde el punto de vista de inteligencia como de gráficos. Por lo tanto, debe tener en cuenta que dentro de un IVE es necesario diferenciar qué entidades son inteligentes y cuáles no lo son. Las entidades inteligentes están representadas por los agentes, que poseen características sociales, proactividad y aprendizaje. Sin embargo, existe la posibilidad de modelar entornos con entidades que no posean estas características. A estas entidades no inteligentes se les conoce como artefactos. Son entidades estáticas que sólo se comportan como herramientas que pueden utilizar los agentes. La metodología que realiza esta distinción es la llamada *A&A* (Agents & Artifacts) [8]. Esta metodología plantea que un agente no está solo dentro de un entorno, por el contrario, un agente está en contacto con otros agentes y con otras entidades. Los artefactos dentro de un entorno representan la estructura física[9], con la que se soporta la complejidad del entorno, permitiendo a los MAS tener una serie de herramientas para interactuar con el entorno. Estas herramientas pueden ser creadas por los agentes y ser destruidas y reutilizadas por los mismos. De igual forma, pueden ser utilizadas por otros agentes con el fin de cumplir los objetivos planteados.

Para el desarrollo de aplicaciones en MAS, se pueden utilizar diversas plataformas con diferentes ventajas de diseño. Entre estas plataformas destacan las siguientes: *JADE*[11], *JADEX*[12]. También existe también un lenguaje de programación de agentes llamado *JASON*, que está basado en el modelo *AgentSpeak* y que a su vez implementa el modelo *BDI*. Algunas aplicaciones en las cuales se utilizó *JASON* como lenguaje de programación de agentes son: LEGORobotics[13][22], *JADEX*[23] y *JADE*[24, 25].

El lenguaje de desarrollo de Sistemas Multi-Agente de *JASON*, está basado en Java y utiliza una versión extendida del lenguaje *AgentSpeak* para definir el comportamiento de los agentes. *AgentSpeak* es un lenguaje abstracto basado en la arquitectura *BDI (Beliefs-Desires-Intentions)* [26], y se inspira en la lógica y la psicología que permite crear representaciones simbólicas de las creencias, deseos y las intenciones de los agentes. Las creencias son la información que el agente tiene acerca del entorno; es la información que el entorno entrega al agente. Esta información puede ser actualizada en cada instante de tiempo. Una desactualización de información hace que el agente tenga que realizar procesos deliberativos. Los deseos son

[11] http://jade.tilab.com/
[12] http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview
[13] http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com

las posibles acciones que el agente podría realizar. Y las intenciones representan las acciones que el agente ha decidido realizar y además, pueden ser metas que han sido delegadas al agente o pueden ser el resultado de un baremo de varias opciones.

No obstante en el diseño de entornos, los agentes no son los únicos modelados en este tipo de aplicaciones, ya que en un entorno también se representan los objetos que carecen de un comportamiento autónomo. Este grupo de objetos y según la metodología *A&A* mencionada anteriormente, se modela mediante Artefactos. Para modelar estas entidades y puedan interactúen con el entorno, se usa la plataforma *CArtAgO* [27][14].

3.5. Entornos Virtuales Inteligentes (IVEs)

Un Entorno Virtual Inteligente (IVE) [28], es un espacio 3D que proporciona al usuario una colaboración, simulación e interacción con las entidades, de forma que experimente un nivel elevado de inmersión. Esta inmersión se logra gracias a unos gráficos detallados, una física realista y una AI. Esta AI le otorga a las entidades, que no son controladas por un usuario, un comportamiento regido por las condiciones del entorno. Estos IVEs han despertado un gran interés comercial, ya que representan una nueva forma de comercializar productos. Como ejemplo de aplicaciones se puede mencionar Second Life[15] (en la Figura 3.13 se observa una captura del mundo virtual en Second Life). En Second Life existen tiendas virtuales donde los usuarios pueden comprar objetos y cambiar dinero real por dinero virtual. Estas aplicaciones despiertan un elevado interés comercial ya que brindan la oportunidad de hacer negocios con personas de todo el mundo. Como ejemplo de otras aplicaciones de los IVEs, también destacan Bluemars[16], Open Wonderland[17], Opensimulator[18] o Vastpark[19].

Los IVEs tambien permiten la creación de entorno de simulación avanzada [29][6][21], educación[30] y entretenimiento[4][31][32]. Es de especial interés el uso de este tipo de Entornos Virtuales en educación, como herramientas de diseño para realizar pruebas con Sistemas Multi-Agente [4, 13, 5].

3.6. Multi Agent Model For Intelligent Virtual Environments (MAM5)

Existen múltiples ejemplos de integración de MAS con IVEs [33, 34]. No obstante estos modelos plantean soluciones específicas que no permiten ser reutilizadas y tienen un nivel de escalabilidad de muy bajo. Por lo tanto, un IVE basado en MAS

[14] http://cartago.sourceforge.net/
[15] http://secondlife.com/
[16] http://www.bluemars.com/
[17] http://openwonderland.org/
[18] http://opensimulator.org
[19] http://www.vastpark.com/

Figura 3.13: Mundo en Second Life.
(Fuente: www.secondlife.com)

debe plantear soluciones distribuidas y reutilizables, de forma que el desarrollador tenga la libertad de modelar sus agentes, artefactos o añadir propiedades físicas (dinámicas y/o estáticas) al entorno. Además le debe permitir reutilizar las aplicaciones y escalar sorpresivamente su entorno. Para aprovechar las ventajas de una solución distribuida, es necesario entonces la utilización del meta-modelo MAM5[10].

MAM5 cumple con las condiciones anteriormente descritas. Este meta-modelo define completamente todos los elementos que habitan un IVE, permitiéndole al desarrollador definir la parte cognitiva (MAS), los objetos (Artefactos) con sus propiedades físicas y las leyes físicas que rigen el entorno. MAM5 a su vez divide en dos categorías el modelado de IVEs: En la primera, todos aquellos elementos que tendrán una representación virtual, es decir, todos aquellos que tengan un cuerpo dentro del IVE, y en la segunda, todos aquellos que no poseen una representación física dentro del IVE, como aquellos agentes que gestionan otros objetos como bases de datos o cuentas de usuario. La estructura del meta-modelo MAM5, se puede observar en la Figura 3.14.

De acuerdo al planteamiento presentado por MAM5, en el IVE interactuan entre si las entidades con representación virtual con las que no tiene esta representación.

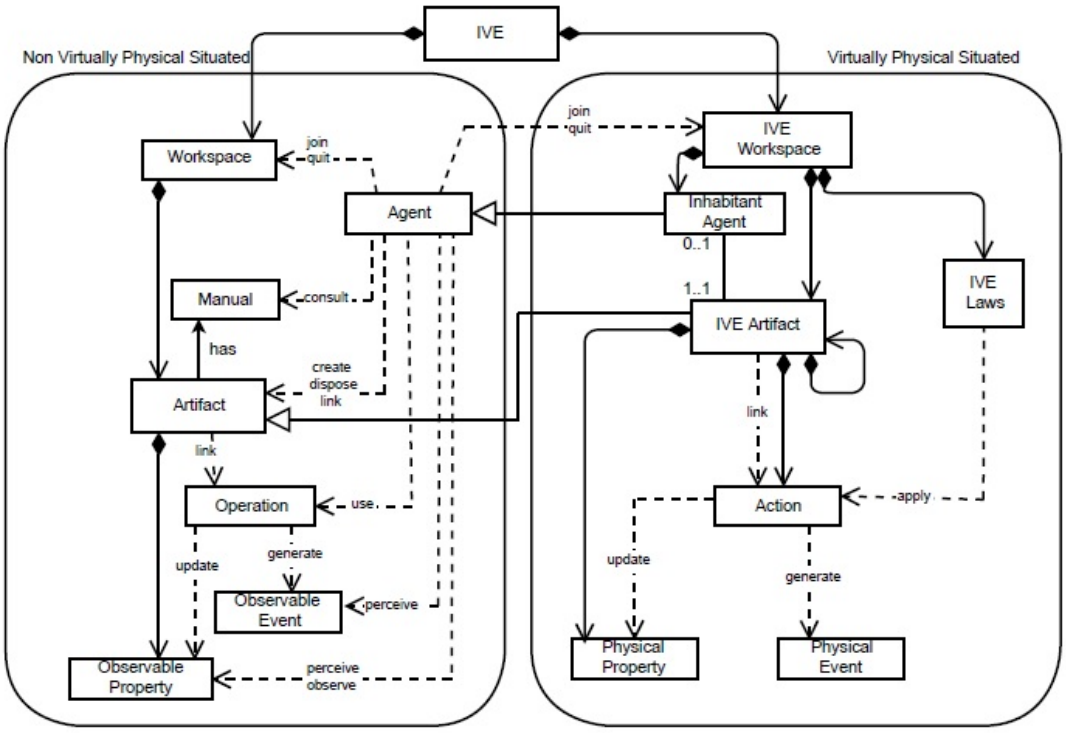

Figura 3.14: Meta-Modelo MAM5.

Este es un aspecto destacable ya que un agente que tenga una representación virtual puede comunicarse con un agente que no la tenga. Cabe aclarar también que MAM5 no incorpora dentro de su meta-modelo una etapa de renderizado, otorgándole al desarrollador la posibilidad de utilizar cualquier *render*, es decir cualquier motor gráfico 3D, lo que le confiere mayor grado de libertad a la hora de diseñar IVEs. Así el desarrollador puede diseñar diferentes formas de visualización ya sea 3D o 2D, para dispositivos móviles u ordenadores e incluso para Webs [20].

MAM5 plantea una clara separación entre la mente y el cuerpo de la entidades que habitan los IVEs. La mente estaría conformada por los agentes cuyos comportamientos se programan utilizando *JASON*. Los artefactos, modelados con la herramienta *CArtAgO*, permitirian a los agentes interactuar en el entorno. *CArtAgO* es una herramienta basada en el meta-modelo *A&A* [8], el cual define que dentro de un entorno existen dos tipos de entidades: los agentes como entidades autónomas y los artefactos como los objetos que están dentro del entorno. A continuación se detallaran los diferentes elementos y características de los dos grupos en los que se divide un IVE según MAM5.

3.6.1. Elementos No Representables Virtualmente

En este grupo están los elementos que no tienen representación virtual dentro del IVE, y comprende los siguientes items:

[20] http://jacalive.gti-ia.dsic.upv.es/Unity_Render/UnityRender.html

1. **Agentes:** Representan las entidades autónomas del sistema. En esta parte del meta-modelo estos agentes pueden gestionar objetos que no estén representados dentro del IVE. Estos agentes podrán gestionar cuentas de usuario, bases de datos, o cualquier otro elemento que no tenga representación virtual.
2. **Artefactos:** Son todos aquellos elementos no autónomos que no tienen una representación virtual dentro del IVE, tales como bases de datos, servidores web, etc.
3. **Workspaces:** Estos son los contenedores de los agentes y artefactos. Podrán existir diferentes *workspaces* con diferentes tipos de agentes y artefactos, lo que permitirá una agrupación de entidades dependiendo de sus características. Esto ayuda a tener *workspaces* estructurados.

3.6.2. Elementos Representables Virtualmente

Está compuesta por los elementos que poseen una representación virtual, es decir, las entidades que tendrán un cuerpo en 3D situados dentro del mundo virtual al renderizar el entorno. Estos elementos son los que otorgan al entorno un parecido al mundo real o crean un mundo completamente distinto, de acuerdo a las especificaciones del diseñador sobre las características que deben tener estos elementos. Cabe destacar, que este aspecto puede influir en el nivel de inmersión que se logre con el IVE, gracias a la interacción del mismo a través de las interfaces establecidas y mediadas por la simulación física que lo gobierne. A continuación se describen los elementos representables virtualmente, definidos por el meta-modelo MAM5.

1. **Inhabitant Agents:** Son los agentes que tienen una representación virtual dentro del entorno, es decir poseen un cuerpo con el cual interactuan con el entorno. Este cuerpo le ayudará a percibir el entorno donde encuentra y modificarlo.
2. **IVE Artifacts:**Los *IVE Artifacts* son objetos que poseen una representación 3D dentro de un mundo virtual. Esta representación puede ser el cuerpo de un agente, una herramienta o un objeto con el cual el agente interactuaría. Estos *artifacts* se caracterizan por poseer una serie de propiedades físicas que definen su relación con el mundo virtual, como la forma (información que se enviará a los *renders* para su visualización), la masa, el largo o el ancho.
3. **IVE Laws:** Son la leyes que rigen el mundo virtual. Estas leyes permitirán definir las restricciones físicas del entorno, indicando la gravedad, el índice de rozamiento u otra restricción física.
4. **IVE Workspaces:** Es donde se define la estructura del entorno, la topología del mapa, la disposición física de las entidades como los *IVE Artifacts* y *Inhabitant Agents* que aparezcan en él. Un *IVE Workspaces* incluye las leyes físicas (IVE laws), que afectan a todas las entidades que se encuentran dentro del lugar que se modela.

Capítulo 4

JaCalIVE

En la literatura existen aproximaciones que utilizan Sistemas Multi-Agente (MAS) como un paradigma para el modelado de IVEs. No obstante existen actualmente algunos problemas sin resolver, a la hora de realizar el diseño de IVEs,entre los que pueden destacar: baja generalidad y reutilización y soporte débil para el manejo de entornos abiertos y dinámicos, donde los objetos se crean y destruyen de forma dinámica. Este trabajo se enfrenta al reto de desarrollar una metodología para el desarrollo de IVEs, basándose en el meta-modelo MAM5. Esta metodología se ha denominado *JaCalIVE* (*JASON CArtAgO* implemented Intelligent Virtual Environment).

Para que la metodología *JaCalIVE* pueda realizar lo planteado, se vio la necesidad que crear una plataforma que la soporte, y se ha llamado *JaCalIVE* Framework. Esta plataforma se encarga de la ejecución de lo modelado por el desarrollador. *JaCalIVE* Framework a su vez integra *JASON* para el modelado de MAS, *CArtAgO* para el modelado de entorno y artefactos y *Jbullet* como motor de simulación física.

No obstante, lo que el desarrollador quiere al final es poder visualizar su IVE e interactuar con él, por lo que se ha desarrollado una forma de visualizar en 3D los IVEs, y además se estudió la posibilidad integrar algún tipo de interfaz que permita interactuar con los mismos. Para visualizar lo que el desarrollador ha diseñado, se plantean dos tipos de *renders*: El primero es un *render* del desarrollador y se encuentra dentro de *JaCalIVE* Framework , con el que se puede observar lo que se ha diseñado de una forma simple (promitivas basicas); el segundo *render* esta construido en *Unity 3D* y proporciona un nivel de detalle mayor que el anterior. De esta forma le permite al desarrollador visualizar estructuras más complejas y mejor detalladas.

La visualización no es la única forma de interactuar con un IVE ya que existen diferentes herramientas que permiten al desarrollador crear aplicaciones para que el usuario pueda interactuar con el IVE. De esta forma el IVE no solo es una he-

rramienta para la representación de información, sino también para la existencia de un flujo en doble sentido de información, es decir, que el usuario interactúe con el IVE y que a su vez el IVE interactúe con el usuario, de una forma muy natural y sin una comunicación explicita entre los dos. Por esta razón se realizó estudio de algunas herramienta de Interfaz Humano-Maquina como las descritas en la sección 3.2, de las cuales destacan algunas por su innovación y la posibilidad de crear aplicaciones para que los usuarios puedan interactuar con los agentes que habiten los IVEs.

En este capítulo se hablará, en primer lugar, sobre el método de desarrollo de IVEs, y en la segunda sección, se hablará sobre la plataforma que se encarga de dar soporte de ejecución, llamada $JaCalIVE$ Framework.

4.1. Método de Desarrollo de IVEs

El método propuesto para el desarrollo de IVEs llamado $JaCalIVE$, plantea que para el diseño de un IVE es necesario contar con tres partes: Modelado, Traducción del XML y Simulación. Estas tres partes se describen a continuación.

1. **Modelado**: El primer paso es el diseño del IVE. $JaCalIVE$ proporciona un XSD basado en el meta-modelo MAM5. Según él, un IVE puede estar compuesto por dos tipos diferentes de *Workspace*. Por un lado se modelan todas las entidades que tengan una representación virtual y por otro lado las entidades que no tengan dicha representación. En esta etapa se incluyen también la especificación de los agentes, los artefactos y las leyes físicas que rigen el IVE.

2. **Traducción del XML**: El segundo paso es la generación de código automático. En esta etapa se crean automáticamente los esqueletos de código para cada uno de los agentes y artefactos. $JaCalIVE$ cuenta con una aplicación que permite al desarrollador obtener la información almacenada dentro del archivo XML y traducirla a fragmentos de código *Java* o código *JASON*.

3. **Simulación:** Finalmente el último paso es la etapa de simulación. Los agentes y entidades generadas se ejecutan simulando el IVE sobre $JaCalIVE$ Framework. Está plataforma utiliza *JASON*, *CArtAgO* y *Jbullet*. *JASON* ofrece el soporte BDI a los agentes por lo que los agentes podrán tener una metodología basada en deseos, creencias e intenciones. *CArtAgO* ofrece a $JaCalIVE$ el modelado de entornos y de artefactos, permitiendo la creación de los objetos con los cuales los agentes interactuarían dentro del IVE. Y por último *Jbullet* se encarga de dar soporte a la simulación física del IVE. $JaCalIVE$ al estar basado en *JASON*, posee internamente un agente llamado *jacalive* y se encarga de supervisar y controlar el IVE.

En la Figura 4.1 se muestra los pasos descritos anteriormente, y que se deben seguir para crear un IVE de acuerdo al método propuesto.

Figura 4.1: Esquema General, *JaCalIVE*.

JaCalIVE es una herramienta versátil para el desarrollo de IVEs, ya que no solo permite al desarrollador crear los comportamientos de los agentes basados en *JASON*, sino que también le permite introducir otras herramientas de IA, para complementar los comportamientos de los agentes, como los algoritmos genéticos [35], los modelos de markov [36], clasificación [37, 38] o aprendizaje por refuerzo [39]. La introduccoión de estas herramientas de IA se debe principalmente a que el agente se comporta como un contenedor de estas herramientas. De esta forma el agente tendría más herramientas para realizar los procesos de razonamiento y la información provendría de las percepciones obtenidas del entorno.

4.1.1. Modelado

Para que el desarrollador agilice el diseño de IVEs, la primera etapa que *JaCalIVE* le otorga dentro del método propuesto es la del modelado. En esta etapa el desarrollador podrá plasmar su modelo de IVE utilizando el lenguaje XML. Este lenguaje le permite almacenar la información del IVE de forma legible. La estructura de este fichero XML, usado por el desarrollador para describir su IVE, viene definido por un esquema XSD que se realizó tomando como base el meta-modelo MAM5. Con este XSD se busca ayudar al desarrollador a analizar sintácticamente los archivos XML, impidiéndole escribir etiquetas incorrectas o que no estén dentro de la etapa de modelado.

El XSD utilizado para el modelado de IVEs se muestra en la Figura 4.2, en la cual se puede observar la clara separación de la parte no virtual de la virtual. Así mismo podemos observar las diferentes entidades que pueden conformar un IVE.

En esta etapa se le permite al diseñador describir el IVE a un nivel de abstracción elevado, sin crear una sola línea de código sobre el comportamiento de los agentes, artefactos o física del IVE. Únicamente plasmando su idea del IVE en términos del meta-modelo MAM5, y le permitirá realizar modificaciones de su modelo desde el XML. Es en esta etapa, el desarrollador plantea necesidades como el número de agentes con representación virtual (*Inhabitant Agent*), artefactos virtuales (*IVE_Artifacts*), agentes que no tendrán una representación virtual (Agentes) y qué leyes físicas quiere que intervengan en el IVE. Estas necesidades se plantean sin realizar ningún tipo de programación en JAVA, *JASON* o *CArtAgO*. Para demostrar cómo se crea un IVE, se ha creado un pequeño IVE de ejemplo llamado

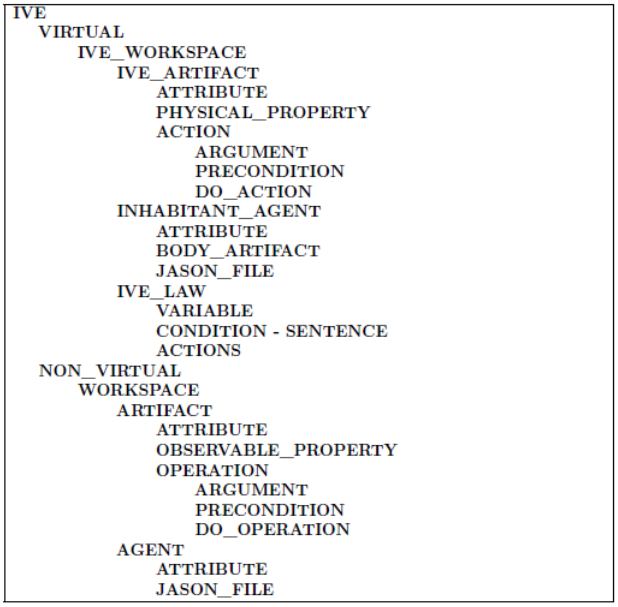

Figura 4.2: XSD utilizado para el analisis sintáctico del XML

starshipExplores con el cual se explicarán los procesos restantes. Un fragmento de código de este ejemplo se puede observar en la Figura 4.3, en la cual se crea el *workspace*, las entidades que lo habitarán (Agentes y Artefactos) y las leyes físicas que regirán el *workspace*.

```
<VIRTUAL>
  <IVE_WORKSPACE NAME="starshipExplorer_workspace">
    <IVE_ARTIFACTS>
      <ITEM NAME="Body_Starship"/>
      <ITEM NAME="Link_Artifact"/>
      <ITEM NAME="Unlink_Artifact"/>
    </IVE_ARTIFACTS>
    <INHABITANT_AGENTS>
      <ITEM NAME="starship"/>
    </INHABITANT_AGENTS>
    <IVE_LAWS>
      <ITEM NAME="Gravity"/>
    </IVE_LAWS>
  </IVE_WORKSPACE>
```

Figura 4.3: Modelado de un IVE en XML

Es importante resaltar que los *IVE Artifacts* poseen propiedades físicas que pueden ser de dos tipos:

1. **PERCEIVABLE**: Son aquellas propiedades a las cuales todos los agentes podrán acceder. Estas propiedades pueden ser: D *posición, largo, masa, forma, velocidad inicial, entre otras.*

2. **INTERNAL**: Son propiedades privadas que solo los agentes que las poseen podrán utilizar y percibir, como la *aceleración, la masa y el tamaño.*

En la Figura 4.4 se puede observar como es la configuración de dichas propiedades en el archivo XML.

```
<IVE_ARTIFACT NAME="Unlink_Artifact" LINKEABLE="false">
 <ATTRIBUTES/>
 <PHYSICAL_PROPERTIES>
  <PERCEIVABLE>

 <DOUBLE NAME = "position"> 8.0 8.0 5.1 </DOUBLE>
 <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
 <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
 <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>

 <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
 <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
 <STRING NAME = "shape"> concrete_wall_g </STRING>
 <STRING NAME = "sound"> none </STRING>
  </PERCEIVABLE>
  <INTERNAL>
 <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
 <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
 <DOUBLE NAME = "size"> 15.0 105.0 25.1 </DOUBLE>
  </INTERNAL>
 </PHYSICAL_PROPERTIES>
```

Figura 4.4: Propiedades PERCEIVABLE y INTERNAL

Dentro de un mismo IVE, es posible tener diferentes *workspaces virtuales* (delimitados por la etiqueta VIRTUAL). Es importante tenerlo en cuenta ya que $JaCalIVE$ contempla la creación de diferentes *workspaces*, en los cuales habitarán diferentes entidades con diferentes condiciones y restricciones físicas. Por lo tanto, se tendrán entidades que con un comportamiento particular en su workspace, pero al cambiar de workspace con otras condiciones su comportamiento se podrá ver afectado. Un ejemplo simple podría ser dos *workspace* en los cuales las gravedades son distintas, uno con una gravedad cero y el segundo con una gravedad como la de la tierra (-9.8) como se muestra la Figura 4.5.

```
 <IVE_LAW NAME="Gravity">
<DOUBLE NAME = "gravity"> 0.0 -9.8 0.0 </DOUBLE>
  <ACTIONS>
   <ITEM NAME="move"/>
  </ACTIONS>
 </IVE_LAW>
```

Figura 4.5: Configuración IVE_LAW

Es importante mencionar que, el desarrollador podrá agregar propiedades físicas como, el rozamiento, velocidades angulares o cualquier otro tipo de restricción físcia que soporte *Jbullet*. Una vez el desarrollador ha modelado su IVE, el siguiente paso es traducir este XML en esqueletos de código, los cuales podrán ser modificados, y de esta forma programar el IVE.

4.1.2. Traducción del XML

Una vez el desarrollador ha modelado el IVE en términos del meta-modelo MAM5, el siguiente paso es la creación automática de esqueletos de código que serán utilizados para crear los comportamientos de cada una de las entidades. Es en esta etapa

del diseño de IVEs, el desarrollador programaría el IVE, después de que *JaCalIVE* haya creado los primeros códigos y se ejecuten sin ningún error, es decir, podrá acceder a los diferentes archivos creados automáticamente por la herramienta, y añadir la programación de los comportamientos de los agentes, artefactos y los parámetros físicos necesarios para la simulación.

El generador automático de código extrae la información del XML y la convierte a clases de *JAVA* o archivos *JASON* (*.asl). En este momento el desarrollador inicia la programación del IVE que ha diseñado, creando los comportamientos de todas la entidades. Siguiendo con el ejemplo **starshipExplores**, en la Figura 4.6 se puede observar como es la construcción automática de un esqueleto de código, asociado a un *IVE Workspace* y un *IVE Artifact* dentro de él.

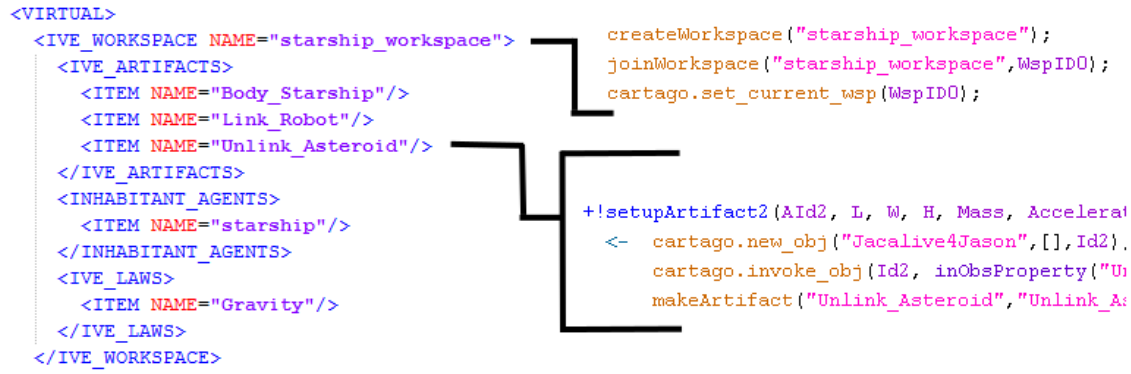

Figura 4.6: Creación de los Templates

4.1.3. Simulación

Esta es la última etapa del método que se propone en *JaCalIVE*. En la simulación del IVE, *JaCalIVE* utiliza como soporte las siguientes herramientas en las que se sustenta *JaCalIVE* Framework.

1. ***JASON***[1]: Es la herramienta encargada de la creación y gestión de los agentes que habitan el IVE. En *JASON* el desarrollador creará los comportamientos de los agentes, los planes y todo lo relacionado con la estructura de comunicación y acción del agente.

2. ***CArtAgO***[2]: Es la herramienta encargada de dar soporte a los *workspaces* en los que habitan todos los artefactos que se han modelado en el XML.

3. **Jbullet**[3]: Es la herramienta encargada de la simulación física y de controlar todas las restricciones físicas dentro del entorno como la gravedad, el rozamiento

[1] http://jason.sourceforge.net/wp/
[2] http://cartago.sourceforge.net/
[3] http://jbullet.advel.cz/

o la detección de colisiones.

Estas herramientas que dan soporte a *JaCalIVE*, son las que se encargan de realizar la simulación del IVE, que el diseñador ha creado en los dos pasos anteriores. Pero *JaCalIVE* es una API que es independiente del *render* permitiéndole al desarrollador crear su propia forma de visualización.*JaCalIVE* incorpora dentro de él un entorno de visualización propio, que le permitiría al diseñador realizar una primera visualización, rápida pero con un bajo nivel de detalle. En el momento en el cual el desarrollador decide que lo que hacen sus entidades es lo correcto, se puede hacer un *render* más complejo utilizando otro tipo de herramienta más especializada. En la Figura 4.7 se puede observar como es el *render* que se incorpora dentro de la API.

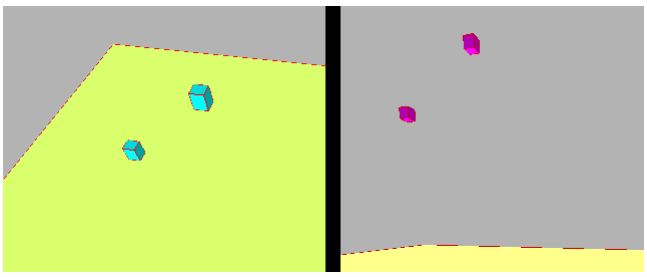

Figura 4.7: Render del Desarrollador

No obstante, se desarrolló un *render* utilizando Unity 3D, con el cual se visualizan las entidades del entorno con más nivel de detalle. Las entidades con representación en el mundo virtual pueden ser diseñadas utilizando herramientas avanzadas para la creació de modelos 3D (*SolidWorks, 3D-Max Studio, Blender*, etc.) compatibles con el motor gráfico que se valla a utilizar.

En la Figura 4.8 se observa el producto final del ejemplo **starshipExplores**, creado para explicar los pasos para el diseño de IVEs con la metodología descrita. Como se puede observar, los gráficos de estas dos imágenes tienen mayor nivel de detalle, permitiendole al desarrollador construir IVEs más realistas e inmersivos para el usuario.

JaCalIVE es una API que permite al desarrollador acoplar sus propias APIs, mejorando asi su IVE. Por ejemplo, se seleccionó una Interfaz Humano-Maquina para acoplarlo a *JaCalIVE* y comprobar la conexión. El dispositivo *Neurosky* se utilizó como dispositivo de prueba.

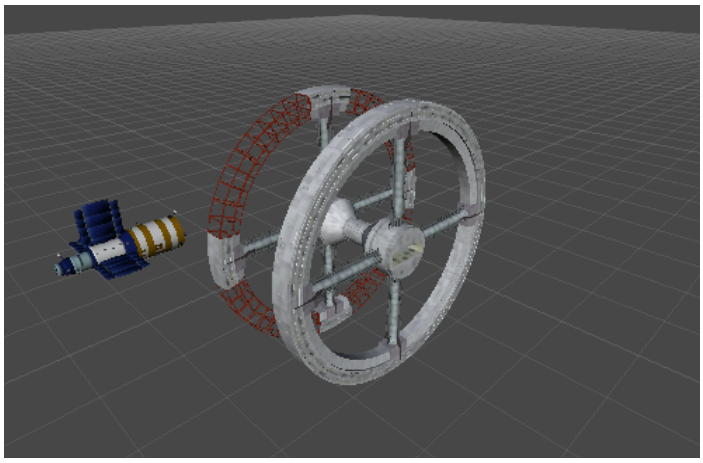

Figura 4.8: Naves Espaciales en Unity 3D

4.2. Plataforma de Soporte de Ejecución: *JaCalIVE* Framework

En este capítulo se explicará cómo funciona *JaCalIVE* Framework desde dentro, describiendo las diferentes herramientas que ayuda a que *JaCalIVE* funcione y cumpla con lo planteado. Además se hablará sobre la estructura interna de *Framework*, formado por las aplicaciones que permiten crear los esqueletos de código (*BuildJaCalIVE*), el agente que supervisa a todas la entidades que habitan el IVE creado (*AgenteJaCalIVE*) y la aplicación encargada de realizar el puente de comunicación entre *Jason* y *JaCalIVE* (*JaCalIVE4Jason*). En la Figura 4.9 se muestra el esquema de *JaCalIVE* Framework, en el cual se observan las herramienta que lo soportan junto con *JaCalIVE4Jason*.

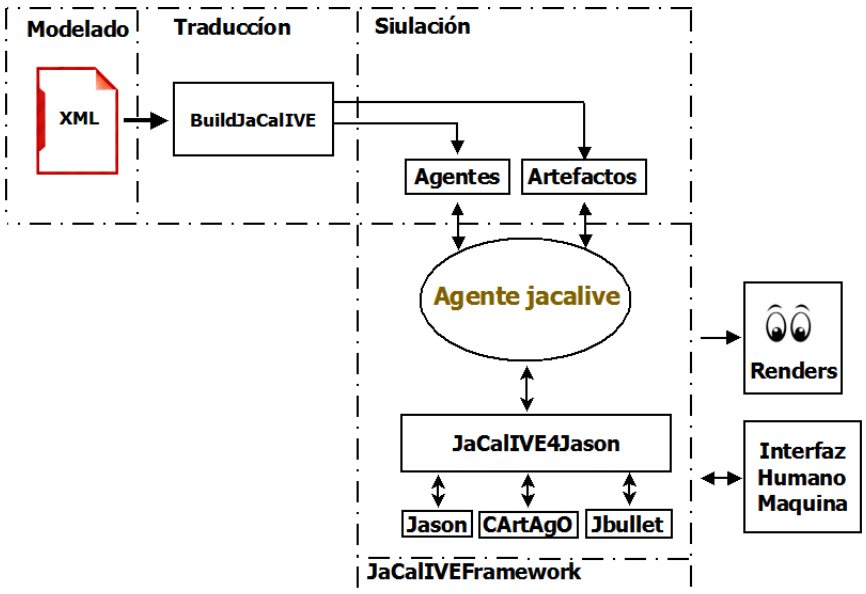

Figura 4.9: Esquema de *JaCalIVE* Framework

4.2.1. *BuildJaCalIVE*

Esta aplicación fue creada para interpretar la información que ha sido plasmada por el desarrollador, es decir, extrae todos los datos que se han escrito en el archivo XML y se encargada de realizar lo explicado en la etapa de 4.1.2; crea los esqueletos de código que serán completados por el desarrollador. La aplicación lee las etiquetas del XML, extrayendo los nombres, propiedades o los valores numéricos correspondientes que serán utilizados para la creación de los esqueletos de código. A diferencia del resto de partes de Framework, esta se ejecuta off-line antes de la ejecución del sistema, mientras que el resto forman el soporte de la fase de simulación. El esquema de esta aplicación se muestra en la Figura 4.10.

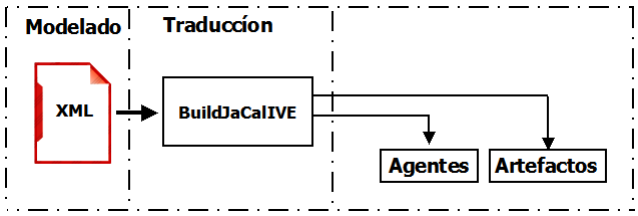

Figura 4.10: Esquema de *BuildJaCalIVE*

4.2.2. *Agentejacalive*

Este es el agente encargado de la supervisión de todas las entidades que habitan el IVE. *JaCalIVE* cuenta con este agente llamado *jacalive* para supervisar a los demás agentes y crear todos los artefactos que se han definido en la etapa de modelado. Si un agente quiere mover su cuerpo aplicando una fuerza sobre él y desplazarlo a una nueva posición, la posición obtenida por el agente tiene que ser enviada a *jacalive*. De esta forma *jacalive* puede validar la nueva posición ya que él conoce todo el entorno en el cual la entidad se está moviendo. Si un agente quiere cambia alguna restricción física en el momento de ejecución, este cambio será validado y realizado por *jacalive* el cual se muestra en la Figura 4.11. A demás de supervisar el flujo de información dentro del IVE, este agente se encarga también de enviar la información a los diferentes tipos de *renders*. Para ello se estableció una comunicación abierta, es decir que el desarrollador pudiese crear su propia estructura de envío de información al *render* y de esta forma se lograra una mayor versatilidad en el diseño de IVEs.

4.2.3. *Jacalive4Jason*

En la Figura 4.12, se muestra el diagrama de *Jacalive4Jason*. Esta herramienta es la encargada de crear el puente de comunicación entre *JaCalIVE* Framework, y el agente *jacalive*. Se vio la necesidad de crearla, ya que no existía una manera directa para comunicar el agente emphjacalive con las aplicaciones *Jason*, *CArtAgO* y *JBulet*. *Jacalive4Jason* inicializa todo el *Framework* y al mismo tiempo inicializa el entorno configurando parámetros como la *gravedad* o el nombre de los *workspaces*.

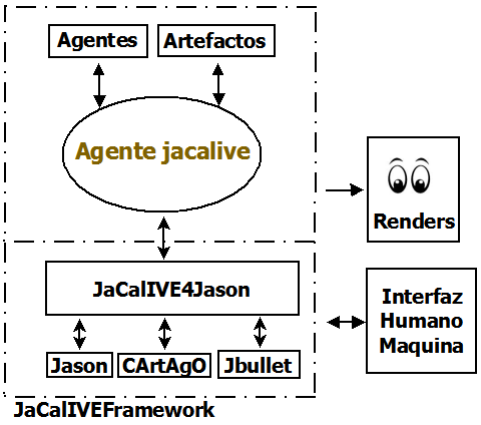

Figura 4.11: Agente jacalive

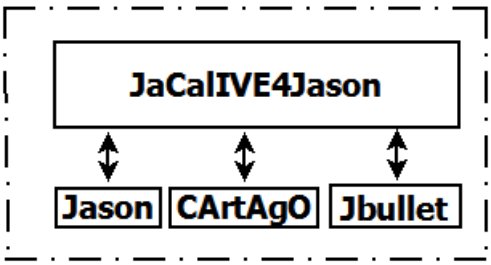

Figura 4.12: Jacalive4Jason

Capítulo 5

Ejemplos de Aplicación

En esta sección se muestran dos ejemplos de aplicación de $JaCalIVE$. El primer ejemplo de aplicación es una simulación en gravedad cero, utilizando modelos de asteriodes y naves espaciales como artefactos del entorno.

El segundo ejemplo de aplicación es una simulación de robótica modular, en la cual se pretende crear un entorno con robots modulares que podrán desplazarse por el entorno y asimilar nuevas partes para cambiar su forma.

5.1. Ejemplo Star Ship

El objetivo de este ejemplo es mostrar cómo se crea un IVE utilizando $JaCalIVE$. Se quiere crear un entorno simple con tres artefactos y un agente, los cuales estarán en el espacio, con sus respectivas restricciones físicas como, la ausencia de gravedad y un rozamiento nulo. Este ejemplo fue basado en el videojuego *Asteroids* (Figura 5.1), en el cual se plantea la aparición de asteriodes en el espacio y deben ser destruidos por el jugado. En este caso el jugador es reemplazado por un agente, pero de igual forma se pueden plantear algunas modificaciones con el fin de crear una interacción humano-agente.

5.1.1. Modelado

Siguiendo la metodología MAM5, se debe crear primero un modelo del IVE. Utilizando un editor de XML se define a qué parte del IVE van a pertenecer la entidades, es decir, si las entidades tendrán una representación virtual. Lo siguiente es dar nombre al workspace ya que en él habitaran todas las entidades y definir cada uno de los *workspaces* que existirán. En la parte virtual habrá un *workspace* para cada lugar con características física distintas, es decir que si se quiere realizar un modelado más detallado del juego, el desarrollador podría tener dos *workspaces*. El primero relacionado con el espacio exterior, el cual tendría una gravedad de cero y otras condiciones físicas, y el segundo *workspace* puede ser el interior de la nave, con sus respectivas restricciones físicas y artefactos.

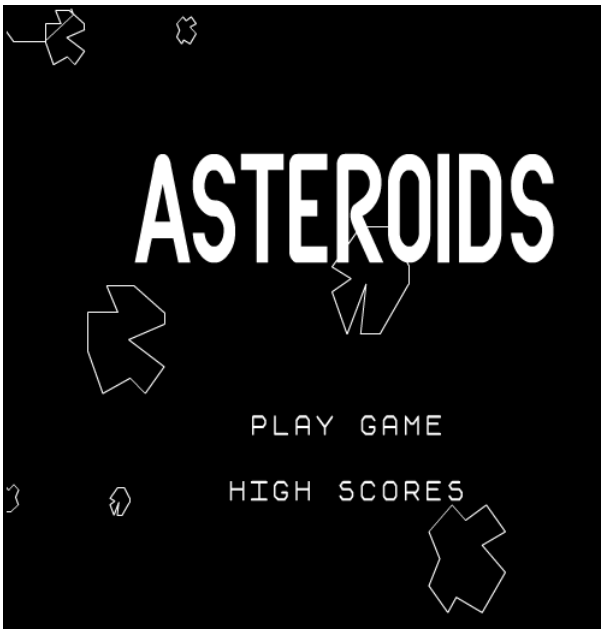

Figura 5.1: Juego Original de Asteriodes.
(Fuente: http://www.freeasteroids.org/)

Este ejemplo cuenta con tres artefactos, uno hace referencia a la nave espacial, el cual se le ha llamado $Body\_Artifact$ y los otros dos se les ha llamado $Unlink\_Asteroid1$ o $Unlik\_Asteroid2$. En la Figura 5.2 se muestra cómo quedaría el XML para este ejemplo. Es importante recalcar que para esta versión de $JaCalIVE$ Framework, los artefactos tienen que ir acompañado del prefijo *Link* y *Unlink*, ya que en versiones siguiente de $JaCalIVE$ Framework, será posible que una serie de artefactos se puedan adherir entre sí. De esta forma los artefactos que tengan el prefijo *Link* podrán unirse con otros que tenga el mismo prefijo y lo que tenga el *Unlink* no podrán hacerlo.

1. **Body_Artifact:** se declararía de la siguiente forma: *Body_Nombre_del_Artefacto*.

2. **Link_Artifact:** se declararía de la siguiente forma: *Link_Nombre_del_Artefacto*.

3. **Unlink_Artifact:** se declararía de la siguiente forma: *Unink_Nombre_del_Artefacto*.

Una vez teniendo claro en qué lugar habitaran las entidades y sabiendo cuantas se necesitan, lo siguiente es configurar las propiedades de cada una de ellas. En los $IVE\_Artifact$ es necesario que el desarrollador especifique algunas propiedades físicas del artefacto, como el ancho, largo o forma. Estas propiedades son definidas en dos partes: la primera son las propiedades observables y la segunda son las propiedades no observables. Las observables son aquellas que posee el artefacto y podrían ser vista por cualquier agente, mientras que las no observable son aquellas que solo el artefacto o agente asociados puede ver como se muestra en la Figura 5.3. Este mismo procedimientos se realiza para todos los IVE_Artifacts presentes dentro del IVE.

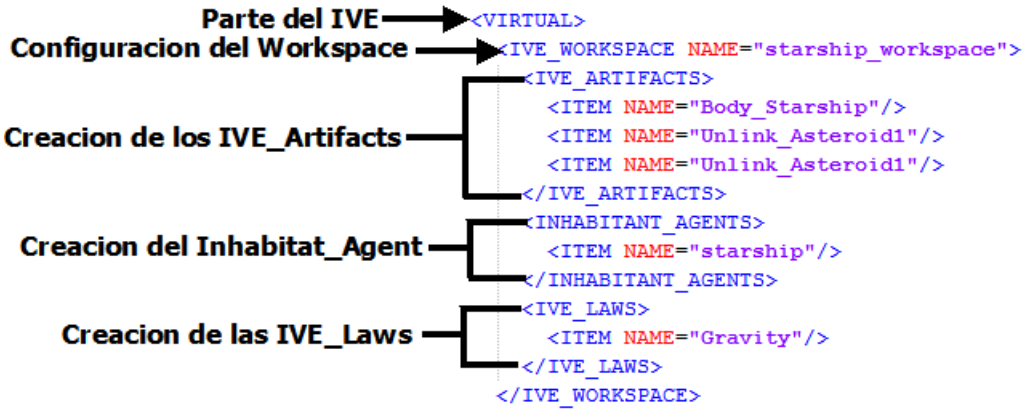

Figura 5.2: Creación del IVE *Workspace* y la entidades que lo habitan.

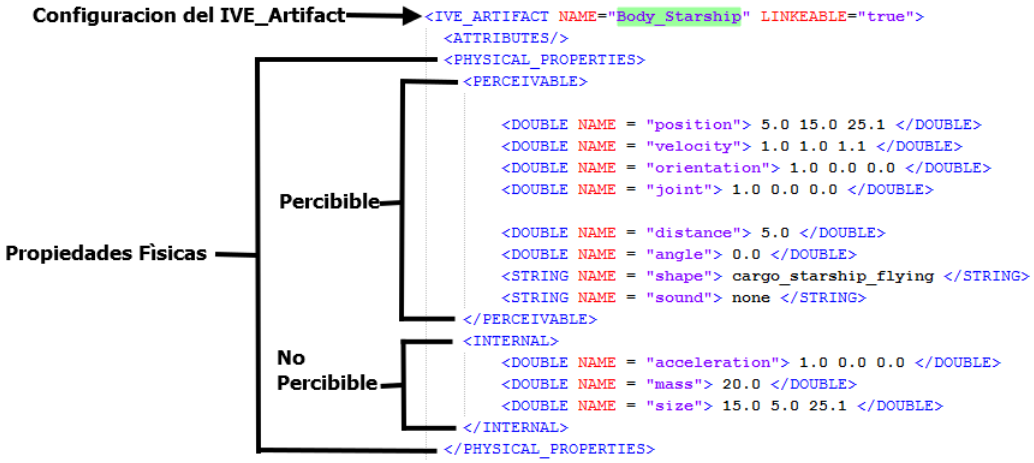

Figura 5.3: Configuración de un IVE_Artifact.

El siguiente paso es configurar el inhabitant_agent. Es primordial darle un nombre al agente y recordar que todos los nombre de los agentes tienen que ser en minúsculas. Figura 5.4. Lo siguiente es dar un nombre a los body_artifacts que tendrá el agente, con el cual se podrá saber cuántos body_artifacts tiene el mismo. Y por último se le dará un nombre al archivo de *JASON* que tendrá que ser igual a el nombre dado al agente. En la figura 5.4 muestra el proceso descrito anteriormente.

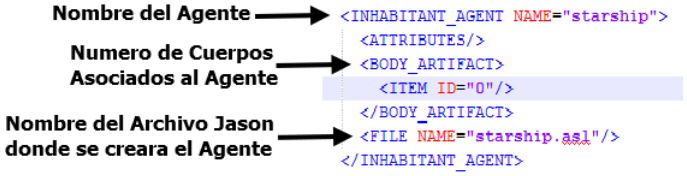

Figura 5.4: Configuración de un Inhabitant_Agent.

Para finalizar la etapa de modelado del IVE se debe configurar las restricciones físicas del entorno. En este ejemplo al ser una nave espacial, la restricción física a

utilizar es la gravedad y esta nula en todos los ejes (X, Y, Z), como se muestra en la Figura 5.5.

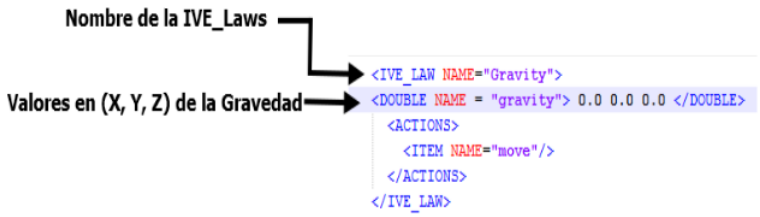

Figura 5.5: Configuración de una IVE_Law.

5.1.2. Traducción del XML

Una vez modelado el IVE, lo siguiente es crear, de forma automática, los templates los cuales serán completados por el diseñador. En el instante en el que se generan los templates, estáran listos para ser ejecutados por la herramienta *JASON*.

Cuando se tienen los archivos generados es solo ir a la herramienta de *JASON* y abrir el proyecto con extensión *.mas2j, y realizar la primera compilación. Solo se observará la ventana de *JASON*, ya que al no tener ningún comportamiento o *render* acoplado a $JaCalIVE$ Framework, no se podrá visualizar nada.

El siguiente paso es realizar la programación del IVE. Utilizando *JASON* se agregar el código a los templates que se han generado de forma automática. Un ejemplo de este código se puede ver en la Figura 5.6.

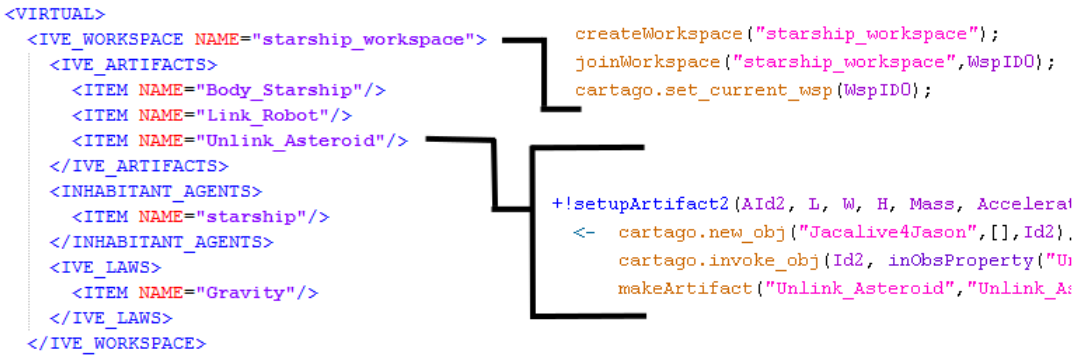

Figura 5.6: Generación de Código Automático.

En la Figura 5.7 se muestra un ejemplo de como se puede aplicar una velocidad en los 3 ejes de un IVE_Artifact, esto se verá luego reflejado en la etapa de simulación.

```
if (sBodyName.equals("Body_Robot")) {
    JacaLive.SetVelocityArtifacts(sBodyName, fVelox, fVeloz, fVeloy);

    fVelox = fVelox + 0.01f;
    fVeloy = fVeloy + 0.01f;
    fVeloz = fVeloz + 0.01f;

    if(fAngleY<1){
        fAngleY = fAngleY + 0.05f;
    }

    if(fVelox>3){
        fVelox = 0;
        fVeloy = 0;
        fVeloz = 0;
        fAngleY = 0;
    }
    signal("isIveAgent", "Body_Robot");
}
    PosRobotA.set(saOutPos);
```

Enviamos los parametros de velocidad a Jbullet

Incremento de la Velocidad en los 3 ejes.

Figura 5.7: Aplicando una velocidad a un IVE_Artifacts.

5.1.3. Simulación

El último paso para este ejemplo es la simulación, en el cual se ejecutará la herramienta *JaCalIVE* obteniendo una primera visaualización del IVE. Es en este punto se pueden observar todos los comportamientos que han sido programados en las etapas anteriores y se realiza el acople con el render usando *Unity 3D*, de modo que se pueden observar los objetos con un mayor nivel de detalle, como se muestra en la Figura .

```
if (sBodyName.equals("Body_Robot")) {
    JacaLive.SetVelocityArtifacts(sBodyName, fVelox, fVeloz, fVeloy);

    fVelox = fVelox + 0.01f;
    fVeloy = fVeloy + 0.01f;
    fVeloz = fVeloz + 0.01f;

    if(fAngleY<1){
        fAngleY = fAngleY + 0.05f;
    }

    if(fVelox>3){
        fVelox = 0;
        fVeloy = 0;
        fVeloz = 0;
        fAngleY = 0;
    }
    signal("isIveAgent", "Body_Robot");
}
    PosRobotA.set(saOutPos);
```

Enviamos los parametros de velocidad a Jbullet

Incremento de la Velocidad en los 3 ejes.

Figura 5.8: Ejemplo Naves Espaciales.

5.2. Ejemplo Robot Apodo

En este ejemplo se pretende mostrar como se puede utilizar $JaCalIVE$, en una simulación de robótica modular. Un robot modular es una estructura robótica compuesta por diferentes módulos y cada modulo representa una única entidad. Esta entidad tiene la capacidad de adherirse a otras estructuras similares o estructuras que posean la misma conexión de acople. Un ejemplo de un robot modular se muestra en la Figura 5.9.

Figura 5.9: Roombots. (Fuente: http://biorob.epfl.ch/cms/page-36376.html)

Una de las características principales de este tipo de robots, es la capacidad de modificar su forma, dependiendo del entorno en el que se encuentre. Si se logra tener la cantidad suficiente de módulos es posible crear estructuras complejas con comportamientos complejos. Una de las ventajas del uso este tipo de robots como ejemplo, es la posibilidad ver cada uno de los módulos como un agente. De esta forma se logra la descentralización de la estructura y se consigue una estructura distribuida, donde cada una de las entidades posee un comportamiento propio. Dada la peculiaridad de cambiar de forma de estos robots, en los últimos años se han diseñado diferentes plataformas que aprovechan esta propiedad, y es frecuente su uso en aplicaciones de búsqueda y rescate [40] y exploración espacial [6] La utilización de $JaCalIVE$ como herramienta de simulación de IVEs permitirían realizar simulaciones avanzadas para este tipo de robots. Se determinaría cuáles son las mejores configuraciones para determinados escenarios de aplicación real, lo que permitiría evaluar diferentes algoritmos de reconfiguración o estudio de comportamientos emergentes.

Por lo tanto, se pretende mostrar la utilidad de $JaCalIVE$ como herramienta de simulación en robótica y a continuación se describirán los pasos para la simulación de un módulo de este tipo de robots. Para realizar la simulación del robot modular es necesario saber por cuantos módulos estará compuesto. Para este ejemplo se establece que el robot contenga solo dos módulos. Un módulo izquierdo y un módulo derecho con una articulación en los tres ejes *(X, Y, Z)* como de muestran en las Figuras (5.10, 5.11, 5.12). Este eje de articulación dependerá de cómo se indique el desplazamiento del robot por el entorno.

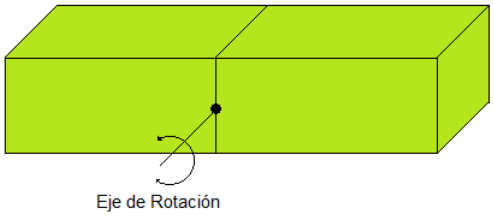

Figura 5.10: Eje de Articulación en X.

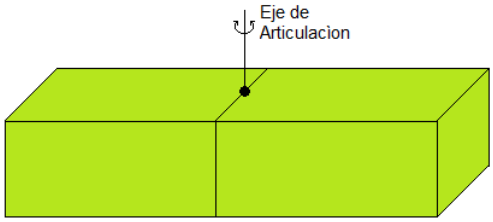

Figura 5.11: Eje de Articulación en Y.

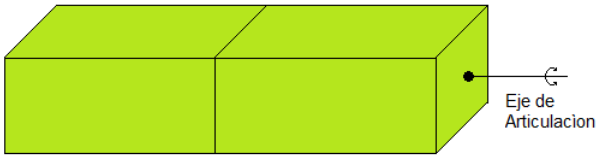

Figura 5.12: Eje de Articulación en Z.

Una vez configurado el *IVE_Workspace*, donde habitaran las entidades y la cantidad necesaria para el IVE, y las propiedades físicas, observables y no observables (ejes de articulación, ancho, largo, algo, forma),el siguiente paso es la creación automática de los esqueletos de código.

5.2.1. Modelado

Siguiendo la metodología MAM5, lo primero es crear un modelo del IVE. Utilizando un editor de XML se define primero a que parte del IVE van a pertenecer la entidades, es decir, si las entidades tendrán o no una representación virtual. Lo siguiente es dar nombre al workspace donde habitaran todas las entidades; definir cada uno de los *workspaces* que existirán. En la parte virtual, habrá un *workspace* para cada lugar con características física distintas.

5.2.2. Traducción del XML

Un vez modelado el IVE, el siguiente paso es crear de forma automática los templates, los cuales serán completados por el diseñador. La definición del eje de articulación será desde el XML, bajo la etiqueta de *Joint* permitiendo elegir en que eje se quiere hacer la articulación como se muestra en la Figura 5.14. La articulación

```
<VIRTUAL>
  <IVE_WORKSPACE NAME="apodRobot_workspace">
    <IVE_ARTIFACTS>
      <ITEM NAME="BodyLeft"/>
      <ITEM NAME="BodyRight"/>
      <ITEM NAME="linkedArtifact"/>
      <ITEM NAME="unlinkedArtifact"/>
    </IVE_ARTIFACTS>
    <INHABITANT_AGENTS>
      <ITEM NAME="Robot"/>
    </INHABITANT_AGENTS>
    <IVE_LAWS>
      <ITEM NAME="Gravity"/>
    </IVE_LAWS>
  </IVE_WORKSPACE>
```

Figura 5.13: Creación del IVE *Workspace* y las entidades que lo habitan.

se puede elegir solo en un eje al tiempo, colocando el número "1.^{en} el eje en el que se quiere hacer la articulación.

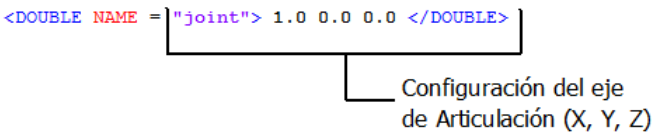

Figura 5.14: Eje de Articulación del Robot Apodo.

Una vez que se tienen los archivo generados es solo ir a la herramienta de *JASON* y abrir el proyecto con extension *.mas2j. y realizar la primera compilación. Solo saldrá la ventana de *JASON* sin ningún tipo de visualización 3D, ya que no se ha realizado ningún tipo de programación de comportamientos y restricciones físicas. El desplazamiento del robot modular a través del entorno, es similar al movimiento de las orugas o gusanos. Esta forma de desplazarse se asemeja a un oscilador sinusoidal mostrado en la ecuación 5.1, por lo que utilizando esta ecuación se calculan los ángulos con los cuales se realiza el movimiento de los módulos.

$$\gamma_j(t) = A_j \sin(\frac{2\pi}{\tau}t + \psi_j) + O_j \tag{5.1}$$

La Figura 5.15 muestra cómo es la codificación de un oscilador sinusoidal. Este código es el encargado de realizar el desplazamientos del robot por el entorno.

La Figura 5.16 muestra como es la señal de salida del oscilador sinusoidal para un solo modulo. Si existieran más módulos unidos la señal tendría que propagarse en función de la cantidad de cuerpos existentes.

```
//@OPERATION
public double servoMotorA(double Amplitud){
    dAngulo = Amplitud*Math.sin(2* dPI * in/iN);
    double Minimo = Math.min(dAngulo, doldAngulo);
    doldAngulo = dAngulo;
    in = (in + 1) % iN;
    //pack1.set(Double.toString(dAngulo));
    return dAngulo;
}
```

Figura 5.15: Generación de Código Automático.

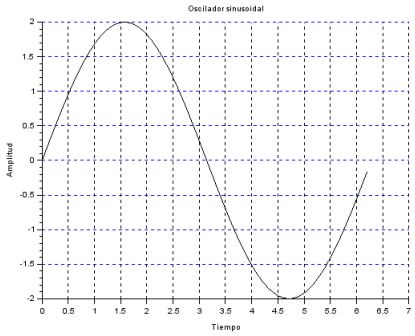

Figura 5.16: Oscilador Sinusoidal.

5.2.3. Simulación

En el proceso de simulación se ha creado un modulo 3D del robot y será visualizado por *Unity 3D* para tener una representación gráfica más realista. Este modelo se puede ver en la Figura 5.17, y es el mismo que se visualizará en los diferentes *renders*.

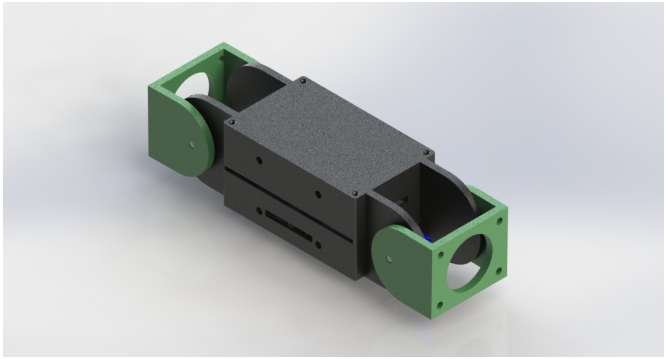

Figura 5.17: Generación de Código Automático.

Capítulo 6

Conclusiones y Trabajos Futuros

En este trabajo se ha diseñado e implementado una plataforma para el diseño de IVEs, basada en el meta-modelo MAM5, para definir un IVE en términos de agentes y artefactos. El desarrollo de este tipo de aplicaciones mediante esta herramienta permitió el modelado, programación y simulación de IVEs de forma más fácil y rápida. Además cabe destacar la naturaleza distribuida de los sistemas desarrollados con esta nueva herramienta, en contraposición con otras plataformas que solo proporcionan soluciones centralizadas. $JaCalIVE$ plantea un método de desarrollo de IVEs, iniciando con una etapa de modelado, en la cual el desarrollador plasma la idea del IVE, y le permite hacer cambios rápidos, aumentar la cantidad de agentes o artefactos desde un archivo XML. A su vez le brinda una herramienta que convierte la información almacenada en el archivo XML, en esqueletos de código para que el desarrollador programe con más facilidad. De esta forma el desarrollador podrá simular restricciones físicas como la gravedad, rozamiento, velocidad y cualquier otra restricción soportada por el motor físico.

Se construyeron dos tipos de *render* para visualizar todo lo que el desarrollador había diseñado, y observar como la simulación física restringe las acciones de la entidades que habitan el IVE. El primer *render* llamado *Render del Desarrollador*, permite observar todas las entidades que habitan el IVE en la etapa de simulación. Es un*render* muy básico que muestra primitivas simples como esferas, cajas o cilindros, indicando al desarrollador que son las entidades creadas en la etapa de modelado. No obstante se desarrolló otro render con *Unity 3D*, que permite visualizar con mayor detalle todas las entidades que habitan el IVE y crear representaciones en 3D o 2D, que serán ejecutadas en ordenadores, dispositivo móviles o en páginas web[1]. $JaCalIVE$ otorga al desarrollador una versatilidad para el diseño de IVEs. Parte de esta versatilidad radica en que el desarrollador puede conectar diferentes tipos de *renders* o algún tipo de tipo de Interfaz Humano-Maquina lo que permite al desarrollador mejorar la interacción entre el usuario y el IVE.

Este trabajo se ha divulgado a través de dos congresos internacionales, realizados en la ciudad de Salamanca en el mes de Junio de 2014:

[1] http://jacalive.gti-ia.dsic.upv.es/Unity_Render/UnityRender.html

13th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)[11]

9th International Conference on Hybrid Artificial Intelligence (HAIS) [12]

Además para aumentar el impacto de la divulgación, se construyó una página web web `http://jacalive.gti-ia.dsic.upv.es/`, donde se muestran los ejemplos y un *render* realizado con *Unity 3D*.

Atendiendo a lo desarrollado en el presente trabajo, se dispuso como base para la realización de una tesis doctoral, dentro del mismo grupo de investigación. El trabajo que se propone, se desarrollará para dar soporte al proyecto iHAS: Sociedades Humano-Agente: Diseño, Formación y Coordinación, financiado por el ministerio de ciencia y economía con referencia TIN2012-36586-C03-01, PROMETEOII/2013/019. De igual forma, mi trabajo estará financiado a través de un contrato predoctoral, otorgado por la Universidad Politécnica de Valencia, con referencia P2013-01276.

Como línea de investigación futura se plantea lo siguiente:

Utilizar $JaCalIVE$ como herramienta para creación de entornos virtuales para la simulación de agentes emocionales. Estos IVEs permitirían la introducción del usuario dentro de estas simulaciones, e incluso realizar simulaciones de emociones sociales humano-agente. Esta clase integración (humano-emociones-IVEs) podría ser útil en entornos ubicuos o inteligencia ambiental, de forma que la interacción con estos entornos sea más natural, es decir que el usuario haga parte del mismo entorno. Los Humanos entrarían en contacto con los agentes a través de distintas interfaces, como las descritas en la sección 3.2. A su vez, los agentes podrían interactuar con el mundo real, a través de una gama de actuadores (controles de temperatura o motores), o percibirlo mediante una variedad de sensores (Temperatura, presión o cámaras de video).

Bibliografía

[1] Ralph Schroeder. Defining virtual worlds and virtual environments. *Journal For Virtual Worlds Research*, 1(1), 2008.

[2] Stephen R. Ellis. What are virtual environments? *Computer Graphics and Applications, IEEE*, 14(1):17–22, 1994.

[3] Michael Woolridge and Michael J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[4] Surangika Ranathunga, Stephen Cranefield, and Martin K. Purvis. Interfacing a cognitive agent platform with a virtual world: a case study using second life. In *AAMAS*, page 1181–1182, 2011.

[5] Surangika Ranathunga, Stephen Cranefield, and Martin Purvis. Integrating expectation monitoring into jason: A case study using second life. 2010.

[6] Georgios Lidoris and Martin Buss. A multi-agent system architecture for modular robotic mobility aids. In *European Robotics Symposium 2006*, page 15–26, 2006.

[7] Vijay Kumar Mago and M. Syamala Devi. A multi-agent medical system for indian rural infant and child care. In *IJCAI*, page 1396–1401, 2007.

[8] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the *a&a* meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, May 2008.

[9] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Give agents their artifacts: the A&A approach for engineering working environments in MAS. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 150, 2007.

[10] A. Barella, A. Ricci, O. Boissier, and C. Carrascosa. MAM5: MultiAgent Model For Intelligent Virtual Environments. In *10th European Workshop on Multi-Agent Systems (EUMAS 2012)*, pages 16–30, 2012.

[11] J.A. Rincon, Carlos Carrascosa, and Emilia Garcia. Developing intelligent virtual environments using mam5 meta-model. In Yves Demazeau, Franco Zambonelli, JuanM. Corchado, and Javier Bajo, editors, *Advances in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*, volume

8473 of *Lecture Notes in Computer Science*, pages 379–382. Springer International Publishing, 2014.

[12] J.A. Rincon, Emilia Garcia, V. Julian, and C. Carrascosa. Developing adaptive agents situated in intelligent virtual environments. In Marios Polycarpou, AndréC.P.L.F. de Carvalho, Jeng-Shyang Pan, Michał Woźniak, Héctor Quintian, and Emilio Corchado, editors, *Hybrid Artificial Intelligence Systems*, volume 8480 of *Lecture Notes in Computer Science*, pages 98–109. Springer International Publishing, 2014.

[13] Caja Thimm. Virtual worlds: Game or virtual society? In Johannes Fromme and Alexander Unger, editors, *Computer Games and New Media Cultures*, pages 173–190. Springer Netherlands, January 2012.

[14] Om K. Gupta and Ray A. Jarvis. Using a virtual world to design a simulation platform for vision and robotic systems. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Yoshinori Kuno, Junxian Wang, Jun-Xuan Wang, Junxian Wang, Renato Pajarola, Peter Lindstrom, André Hinkenjann, Miguel L. Encarnação, Cláudio T. Silva, and Daniel Coming, editors, *Advances in Visual Computing*, number 5875 in Lecture Notes in Computer Science, pages 233–242. Springer Berlin Heidelberg, January 2009.

[15] Peter Dam, Priscilla Braz, and Alberto Raposo. A study of navigation and selection techniques in virtual environments using microsoft kinect®. In *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, page 139–148. Springer, 2013.

[16] Yi Li. Hand gesture recognition using kinect. In *2012 IEEE 3rd International Conference on Software Engineering and Service Science (ICSESS)*, pages 196–199, June 2012.

[17] Genaro Rebolledo-Mendez, Ian Dunwell, Erika A. Martínez-Mirón, María Dolores Vargas-Cerdán, Sara de Freitas, Fotis Liarokapis, and Alma R. García-Gaona. Assessing NeuroSky's usability to detect attention levels in an assessment exercise. In Julie A. Jacko, editor, *Human-Computer Interaction. New Trends*, number 5610 in Lecture Notes in Computer Science, pages 149–158. Springer Berlin Heidelberg, January 2009.

[18] Trung Duy Pham and Dat Tran. Emotion recognition using the emotiv EPOC device. In Tingwen Huang, Zhigang Zeng, Chuandong Li, and Chi Sing Leung, editors, *Neural Information Processing*, number 7667 in Lecture Notes in Computer Science, pages 394–399. Springer Berlin Heidelberg, January 2012.

[19] Raffaella Folgieri, Mattia G. Bergomi, and Simone Castellani. EEG-Based brain-computer interface for emotional involvement in games through music. In Newton Lee, editor, *Digital Da Vinci*, pages 205–236. Springer New York, January 2014.

[20] Ana Cláudia Melo Tiessi Gomes de Oliveira and Fátima de Lourdes dos Santos Nunes. Building a open source framework for virtual medical training. *J Digit Imaging*, 23(6):706–720, December 2010.

[21] Chih-Han Yu and Radhika Nagpal. A self-adaptive framework for modular robots in a dynamic environment: theory and applications. *The International Journal of Robotics Research*, 30(8):1015–1036, 2011.

[22] Andreas Schmidt Jensen. Implementing lego agents using jason. *arXiv preprint arXiv:1010.0150*, 2010.

[23] Konrad Steblovnik and Damjan Zazula. A novel agent-based concept of household appliances. *J Intell Manuf*, 22(1):73–88, February 2011.

[24] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with JADE. In Cristiano Castelfranchi and Yves Lespérance, editors, *Intelligent Agents VII Agent Theories Architectures and Languages*, number 1986 in Lecture Notes in Computer Science, pages 89–103. Springer Berlin Heidelberg, January 2001.

[25] Bertha Guijarro-Berdiñas, Amparo Alonso-Betanzos, Silvia López-López, Santiago Fernández-Lorenzo, and David Alonso-Ríos. A JADE-Based framework for developing evolutionary multi-agent systems. In Yves Demazeau, Juan Pavón, Juan M. Corchado, and Javier Bajo, editors, *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, number 55 in Advances in Intelligent and Soft Computing, pages 339–348. Springer Berlin Heidelberg, January 2009.

[26] Rafael H Bordini, Jomi Fred Hübner, and Michael J Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. J. Wiley, Chichester, England; Hoboken, NJ, 2007.

[27] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems III*, volume 4389 of *Lecture Notes in Computer Science*, chapter 4, pages 67–86. Springer Berlin Heidelberg, May 2007. 3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

[28] K.S. Hale and K.M. Stanney. *Handbook of Virtual Environments: Design, Implementation, and Applications*. Human Factors and Ergonomics. Taylor & Francis, 2002.

[29] Chih-han Yu and Radhika Nagpal. Distributed consensus and self-adapting modular robots. In *IROS-2008 workshop on Self-Reconfigurable Robots and Applications*, 2008.

[30] Álvaro Barbero, Mario Salvador González-Rodríguez, Juan de Lara, and Manuel Alfonseca. Multi-agent simulation of an educational collaborative web system. In *European Simulation and Modelling Conference*, 2007.

[31] Roberto Andreoli, Rosario De Chiara, Ugo Erra, and Vittorio Scarano. Interactive 3d environments by using videogame engines. In *Information Visualisation, 2005. Proceedings. Ninth International Conference on*, page 515–520, 2005.

[32] Catalina Roncancio and Eduardo Zalama. Modeling virtual agent behavior in a computer game to be used in a real enviroment. In *Trends in Practical Applications of Agents and Multiagent Systems*, page 623–630. Springer, 2010.

[33] Ruth Aylett and Michael Luck. Applying artificial intelligence to virtual reality: Intelligent virtual environments. *APPLIED ARTIFICIAL INTELLIGENCE*, 14:3–32, 2000.

[34] Jeehang Lee, Vincent Baines, and Julian Padget. Decoupling cognitive agents and virtual environments. In Frank Dignum, Cyril Brom, Koen Hindriks, Martin Beer, and Deborah Richards, editors, *Cognitive Agents for Virtual Environments*, number 7764 in Lecture Notes in Computer Science, pages 17–36. Springer Berlin Heidelberg, January 2013.

[35] A. Kazemi, M. H. Fazel Zarandi, and S. M. Moattar Husseini. A multi-agent system to solve the production–distribution planning problem for a supply chain: a genetic algorithm approach. *Int J Adv Manuf Technol*, 44(1-2):180–193, September 2009.

[36] Graçccaliz Pereira Dimuro, Antônio Carlos da Rocha Costa, Luciano Vargas Gonçccalves, and Alexandre Hubner. Interval-valued hidden markov models for recognizing personality traits in social exchanges in open multiagent systems. 2008.

[37] Jeff Orkin and Deb Roy. Semi-automated dialogue act classification for situated social agents in games. In *Agents for games and simulations II*, page 148–162. Springer, 2011.

[38] Michał Woźniak, Manuel Graña, and Emilio Corchado. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16:3–17, March 2014.

[39] Juan A. Garcá-Pardo and Carlos Carrascosa. Social welfare for automatic innovation. In *Multiagent System Technologies*, page 29–40. Springer, 2011.

[40] Juan Gonzalez-Gomez, Javier Gonzalez-Quijano, Houxiang Zhang, and Mohamed Abderrahim. Toward the sense of touch in snake modular robots for search and rescue operations. In *Proc. ICRA 2010 Workshop "Modular Robots: State of the Art*, page 63–68, 2010.

Apéndice A

Anexo I: Código del Ejemplo Star Ship

En el presente anexo se darán a conocer los códigos correspondientes a los ejemplos planteados en la sección .

A.1. Modelado del IVE en XML

A continuación se da a conocer el código XML, en el cual se modelo el IVE.

Listing A.1: Test

```
<?xml version="1.0" encoding="UTF-8"?>
<IVE NAME="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="generalStructure.xsd">

  <VIRTUAL>
    <IVE_WORKSPACE NAME="starship_workspace">
      <IVE_ARTIFACTS>
        <ITEM NAME="Body_Starship"/>
        <ITEM NAME="Link_Robot"/>
        <ITEM NAME="Unlink_Asteroid"/>
      </IVE_ARTIFACTS>
      <INHABITANT_AGENTS>
        <ITEM NAME="starship"/>
      </INHABITANT_AGENTS>
      <IVE_LAWS>
        <ITEM NAME="Gravity"/>
      </IVE_LAWS>
    </IVE_WORKSPACE>

    <IVE_ARTIFACT NAME="Body_Starship" LINKEABLE="true">
      <ATTRIBUTES/>
      <PHYSICAL_PROPERTIES>
        <PERCEIVABLE>

          <DOUBLE NAME = "position"> 5.0 15.0 25.1 </DOUBLE>
          <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
          <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
          <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>

          <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
          <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
          <STRING NAME = "shape"> cubic </STRING>
          <STRING NAME = "sound"> none </STRING>
```

```
34          </PERCEIVABLE>
35          <INTERNAL>
36           <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
37           <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
38           <DOUBLE NAME = "size"> 15.0 5.0 25.1 </DOUBLE>
39          </INTERNAL>
40        </PHYSICAL_PROPERTIES>
41        <ACTIONS>
42          <ACTION NAME="move">
43            <ARGUMENTS>
44              <PARAMETER NAME="newX" TYPE="DOUBLE" />
45              <PARAMETER NAME="newY" TYPE="DOUBLE" />
46              <PARAMETER NAME="newZ" TYPE="DOUBLE" />
47            </ARGUMENTS>
48            <DO_ACTION>
49            <ASSIGN>
50                  <OPERAND>
51                    <ELEMENT_PROP NAME="SELF" PROPERTY="position">
52                      <INDEX> "x" </INDEX>
53                    </ELEMENT_PROP>
54                  </OPERAND>
55                  <OPERAND>
56                    <PARAMETER NAME="newX" TYPE="DOUBLE" />
57                  </OPERAND>
58            </ASSIGN>
59            </DO_ACTION>
60            <DO_ACTION>
61            <ASSIGN>
62                  <OPERAND>
63                    <ELEMENT_PROP NAME="SELF" PROPERTY="position">
64                      <INDEX> "y" </INDEX>
65                    </ELEMENT_PROP>
66                  </OPERAND>
67                  <OPERAND>
68                    <PARAMETER NAME="newY" TYPE="DOUBLE" />
69                  </OPERAND>
70            </ASSIGN>
71            </DO_ACTION>
72            <DO_ACTION>
73            <ASSIGN>
74                  <OPERAND>
75                    <ELEMENT_PROP NAME="SELF" PROPERTY="position">
76                      <INDEX> "z" </INDEX>
77                    </ELEMENT_PROP>
78                  </OPERAND>
79                  <OPERAND>
80                    <PARAMETER NAME="newZ" TYPE="DOUBLE" />
81                  </OPERAND>
82            </ASSIGN>
83            </DO_ACTION>
84
85            <DO_ACTION>
86              <EXEC_LINKED_OP LINKED_ART_ID="1" LINKED_FUNCTION="move">
87                <LINKED_ARGUMENTS>
88                <ELEMENT_PROP NAME="SELF" PROPERTY="position">
89                      <INDEX> "x" </INDEX>
90                </ELEMENT_PROP>
91                <ELEMENT_PROP NAME="SELF" PROPERTY="position">
92                      <INDEX> "y" </INDEX>
93                </ELEMENT_PROP>
94                <ELEMENT_PROP NAME="SELF" PROPERTY="position">
95                      <INDEX> "z" </INDEX>
96                </ELEMENT_PROP>
97              </LINKED_ARGUMENTS>
```

```
98                  </EXEC_LINKED_OP>
99              </DO_ACTION>
100
101             <PHYSICAL_EVENT NAME="" />
102         </ACTION>
103         </ACTIONS>
104     </IVE_ARTIFACT>
105
106     <IVE_ARTIFACT NAME="Link_Robot" LINKEABLE="true">
107         <ATTRIBUTES/>
108         <PHYSICAL_PROPERTIES>
109             <PERCEIVABLE>
110
111             <DOUBLE NAME = "position"> 20.0 1.0 75.1 </DOUBLE>
112             <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
113             <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
114             <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>
115
116             <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
117             <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
118             <STRING NAME = "shape"> cubic </STRING>
119             <STRING NAME = "sound"> none </STRING>
120             </PERCEIVABLE>
121             <INTERNAL>
122             <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
123             <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
124             <DOUBLE NAME = "size"> 35.0 55.0 85.1 </DOUBLE>
125             </INTERNAL>
126         </PHYSICAL_PROPERTIES>
127         <ACTIONS>
128             <ACTION NAME="move">
129                 <ARGUMENTS>
130                     <PARAMETER NAME="newX" TYPE="DOUBLE" />
131                     <PARAMETER NAME="newY" TYPE="DOUBLE" />
132                     <PARAMETER NAME="newZ" TYPE="DOUBLE" />
133                 </ARGUMENTS>
134                 <DO_ACTION>
135                 <ASSIGN>
136                         <OPERAND>
137                             <ELEMENT_PROP NAME="SELF" PROPERTY="position">
138                                 <INDEX> "x" </INDEX>
139                             </ELEMENT_PROP>
140                         </OPERAND>
141                         <OPERAND>
142                             <PARAMETER NAME="newX" TYPE="DOUBLE" />
143                         </OPERAND>
144                 </ASSIGN>
145                 </DO_ACTION>
146                 <DO_ACTION>
147                 <ASSIGN>
148                         <OPERAND>
149                             <ELEMENT_PROP NAME="SELF" PROPERTY="position">
150                                 <INDEX> "y" </INDEX>
151                             </ELEMENT_PROP>
152                         </OPERAND>
153                         <OPERAND>
154                             <PARAMETER NAME="newY" TYPE="DOUBLE" />
155                         </OPERAND>
156                 </ASSIGN>
157                 </DO_ACTION>
158                 <DO_ACTION>
159                 <ASSIGN>
160                         <OPERAND>
161                             <ELEMENT_PROP NAME="SELF" PROPERTY="position">
```

```
                        <INDEX> "z" </INDEX>
                    </ELEMENT_PROP>
                </OPERAND>
                <OPERAND>
                    <PARAMETER NAME="newZ" TYPE="DOUBLE" />
                </OPERAND>
            </ASSIGN>
            </DO_ACTION>
            <PHYSICAL_EVENT NAME="" />
      </ACTION>
      </ACTIONS>

    </IVE_ARTIFACT>

     <IVE_ARTIFACT NAME="Unlink_Asteroid" LINKEABLE="false">
      <ATTRIBUTES/>
      <PHYSICAL_PROPERTIES>
        <PERCEIVABLE>

          <DOUBLE NAME = "position"> 8.0 8.0 5.1 </DOUBLE>
          <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
          <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
          <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>

          <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
          <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
          <STRING NAME = "shape"> cubic </STRING>
          <STRING NAME = "sound"> none </STRING>
        </PERCEIVABLE>
        <INTERNAL>
          <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
          <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
          <DOUBLE NAME = "size"> 15.0 105.0 25.1 </DOUBLE>
        </INTERNAL>
      </PHYSICAL_PROPERTIES>
      <ACTIONS/>

    </IVE_ARTIFACT>


    <INHABITANT_AGENT NAME="starship">
      <ATTRIBUTES/>
      <BODY_ARTIFACT>
        <ITEM ID="0" />
        <ITEM ID="1" />
      </BODY_ARTIFACT>
      <FILE NAME="starship.asl" />
    </INHABITANT_AGENT>

    <IVE_LAW NAME="Gravity">
   <DOUBLE NAME = "gravity"> 0.0 -9.8 0.0 </DOUBLE>
      <ACTIONS>
        <ITEM NAME="move" />
      </ACTIONS>
    </IVE_LAW>

  </VIRTUAL>

</IVE>
```

A.2. Esqueletos de Código Creados Automáticamente

A continuación se da a conocer los código *asl* y *java* que se crearon automáticamente.

A.2.1. Proyecto mas2j

Este es el proyecto *Jason* que se crea en el proceso de traducción.

Listing A.2: Test

```
/* Jason Project */
MAS starshipdemo{
    infrastructure: Centralised
    environment: c4jason.CartagoEnvironment
    agents:
       jacalive agentArchClass c4jason.CAgentArch;
       starship_0 agentArchClass c4jason.CAgentArch;
    classpath:
       "../lib/JacaLiveFrameWork.jar";
       "../Cartago_Lib/cartago.jar";
       "../Cartago_Lib//c4jason.jar";
    aslSourcePath:
       "src/asl";
}
```

A.2.2. Códigos asl

Estos son los códigos *asl* creados en el proceso de traducción, el primero corresponde a el agente *jacalive*.

Listing A.3: Test

```

!jacalive.


+!jacalive: true
<- !configArtifact.
+!configArtifact: true
<-
// Inicializacion de las Variables del Mundo
       Gx = 0.0;// Gravedad en X
       Gy = -9.8;// Gravedad en Y
       Gz = 0.0;// Gravedad en Z
       Friction = 0;       Bgravity = true;       U = 0.5;// Indice de Rozamiento
       Ts = 0.01;// Steep
       M = 1.0;// Numero de Artefactos linkeables
       M1 = 1.0;// Numero de Artefactos NO linkeables
       N = 1.0;// Numero de Agentes
       Width = 500;
       Height = 500;
       Length = 500;


       createWorkspace("starship_workspace");
       joinWorkspace("starship_workspace",WspID0);
       cartago.set_current_wsp(WspID0);
```

```
26 /* Concatenamos los nombres de de todos los Work Space */
27       NamesWorkSpaces ="starship_workspace";
28        TamaLinkArti = 1;
29        TamaUnLinkArti = 1;
30        TamaBodyArti = 1;
31        cartago.new_obj("Jacalive4Jason",[],Id);
32        cartago.invoke_obj(Id,initJacaLive(apodRobot_workspace, 1,TamaLinkArti,
              TamaUnLinkArti,TamaBodyArti, Gx, Gy, Gz, Width, Height, Length,
              Bgravity, Friction));
33       //AId, L, W, H, Mass, AccelerationX, AccelerationY, AccelerationZ, Sound,
              Shape, Angle, Distance, JointX, JointY, JointZ, OrientationX,
             OrientationY, OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py,
             Pz
34
35
36        !setupArtifact2(AId2, 15.0,5.0,25.1,20.0,1.0,0.0,0.0,none,cubic
              ,0.0,5.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.1,5.0,15.0,25.1);
37
38        !setupArtifact1(AId1, 35.0,55.0,85.1,20.0,1.0,0.0,0.0,none,cubic
              ,0.0,5.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.1,20.0,1.0,75.1);
39
40        !setupArtifact0(AId0, 15.0,105.0,25.1,20.0,1.0,0.0,0.0,none,cubic
              ,0.0,5.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.1,8.0,8.0,5.1);
41        Body_X = [5.0];
42        Body_Y = [15.0];
43        Body_Z = [25.1];
44        Link_X = [20.0];
45        Link_Y = [1.0];
46        Link_Z = [75.1];
47        ULink_X = [8.0];
48        ULink_Y = [8.0];
49        ULink_Z = [5.1];
50
51
52        //Creo el mampa
53
54     // Configuro los artefactos de el mundo, estos son los artfactos que se
           uniran a los agentes.
55
56     // Configuro los cuerpos de los agentes
57     for (.range(I,0,1-1) ) {
58        .nth(I, Body_X, XA1);
59         .nth(I, Body_Y, YA1);
60        .nth(I, Body_Z, ZA1);
61        cartago.invoke_obj(Id,bodyObj(XA1, YA1, ZA1, I));
62 }
63
64     for (.range(I,0,1-1) ) {
65        .nth(I, Link_X, XA1);
66        .nth(I, Link_Y, YA1);
67        .nth(I, Link_Z, ZA1);
68        cartago.invoke_obj(Id,linkArtifact(XA1, YA1, ZA1, I));
69 }
70
71   for (.range(I,0,1-1) ) {
72        .nth(I, ULink_X, XA1);
73        .nth(I, ULink_Y, YA1);
74        .nth(I, ULink_Z, ZA1);
75        cartago.invoke_obj(Id,uLinkArtifacr(XA1, YA1, ZA1, I));
76 }
77
78     /* Get all the names of the agents, the result is a list, which is stored in
           Name */
79     .all_names(Name);
```

```
80    /* Get the size of the list, there is discounting the few agents JACALIVE */
81    .length(Name, Tama);
82    /*I walk the list of names of the agents and sent him a message whose
          conternido is a 1,
83  *  which tells agents incien sending their ONFIGURATION parameters to be sent
        to the render.
84  */
85  for(.range(I,0,Tama-1)){
86        .nth(I,Name,X);
87        .send(X, achieve, value(1), 500);
88    }
89    .wait(2000);
90    /*Coordino el envio de informacion de los agentes*/
91    .broadcast(tell, value(1));
92  .wait(1000).
93
94
95
96        +!setupArtifact2(AId2, L, W, H, Mass, AccelerationX, AccelerationY,
              AccelerationZ, Sound, Shape, Angle, Distance, JointX, JointY, JointZ
              , OrientationX, OrientationY, OrientationZ, VelocityX, VelocityY,
              VelocityZ, Px, Py, Pz): true
97        <- cartago.new_obj("Jacalive4Jason",[],Id2);
98           cartago.invoke_obj(Id2, inObsProperty("Unlink_Asteroid",L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle
                , Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz));
99           makeArtifact("Unlink_Asteroid","Unlink_Asteroid_class",[L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle
                , Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz ], AId2)
                .
100
101
102       +!setupArtifact1(AId1, L, W, H, Mass, AccelerationX, AccelerationY,
              AccelerationZ, Sound, Shape, Angle, Distance, JointX, JointY, JointZ
              , OrientationX, OrientationY, OrientationZ, VelocityX, VelocityY,
              VelocityZ, Px, Py, Pz): true
103       <- cartago.new_obj("Jacalive4Jason",[],Id1);
104          cartago.invoke_obj(Id1, inObsProperty("Link_Robot",L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz));
105          makeArtifact("Link_Robot","Link_Robot_class",[L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz ], AId1)
                .
106
107
108       +!setupArtifact0(AId0, L, W, H, Mass, AccelerationX, AccelerationY,
              AccelerationZ, Sound, Shape, Angle, Distance, JointX, JointY, JointZ
              , OrientationX, OrientationY, OrientationZ, VelocityX, VelocityY,
              VelocityZ, Px, Py, Pz): true
109       <- cartago.new_obj("Jacalive4Jason",[],Id0);
110          cartago.invoke_obj(Id0, inObsProperty("Body_Starship",L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz));
111          makeArtifact("Body_Starship","Body_Starship_class",[L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz ], AId0)
                .
```

```
112
113
114 /*-------------------------------------------------------------------------*/
115 /* Asociamos los bodyartifact a los agentes, los agente envian un msg      */
116 /* para que el jacalive le adjudique un cuerpo                             */
117 /*-------------------------------------------------------------------------*/
118 +!associate(Dat)[source(Ag)] : true
119 <-     cartago.new_obj("Jacalive4Jason",[],Id1);
120           cartago.invoke_obj(Id1,artifactList(false), Pack_9);
121           cartago.invoke_obj(Id1, bodyAgent(Pack_9, Dat, Ag, Ag, 10, 0), Pack_3)
                ;
122           cartago.invoke_obj(Id1,agentId, Pack_4);
123           Pack_5 = [Pack_4];
124           .difference(Pack_5,["jacalive"],OutIdAgent);
125 /* This method cam be use to return the name body agent and your pos. */
126 .length(OutIdAgent, TamAg);
127           for(.range(I,0,TamAg-1)){
128              .nth(I, OutIdAgent, AgBdy);
129              cartago.invoke_obj(Id1,agentBodyArtifact(AgBdy), Pack_6);
130              cartago.invoke_obj(Id1,posAgentBodyArtifact(Pack_6), Pack_7);
131           }.
132 +!initEnviroment: true
133 <-
134        getAllNameLinkeArtifact(Pack_5);
135
136        .length(Pack_5, T_0);
137        for(.range(I_0,0,T_0-1)){
138           .nth(I_0,Pack_5,R);
139           initEnviromentLink(R, I_0);
140 }
141
142        getAllNameUnLinkeArtifact(Pack_6);
143
144        .length(Pack_6, T_1);
145        for(.range(I_1,0,T_1-1)){
146           .nth(I_1,Pack_6,R1);
147           initEnviromentUnLink(R1, I_1);
148        }
149
150        getAllNameAgent(Pack_8);
151        .length(Pack_8, T_2);
152        for(.range(I_2,0,T_2-1)){
153           .nth(I_2,Pack_8,R2);
154           initEnviromentAgent(R2, I_2);
155        }
156 .
157 +!jacalivLoop(Dat)[source(Ag)] : true
158 <-
159
160        cartago.new_obj("Jacalive4Jason",[],Id2);
161        .my_name(N);
162        .all_names(Name);
163        .difference(Name,[N],Xlp);
164        .length(Xlp,X);
165        cartago.invoke_obj(Id2,agentBodyArtifact(Ag), Pack_5);
166        cartago.invoke_obj(Id2,posAgentBodyArtifact(Pack_5), Pack_6);
167        .concat(Pack_5," ",Pack_6,Pack_7);
168        .term2string(Pack_7,NetData);
169        cartago.invoke_obj(Id2,sendRender(NetData), Pack_8);
170        //Place its code of communication with agents and artefacts here :)
171     .
```

Este es el código correspondiente al agente.

Listing A.4: Test

```
1
2
3 !starship0.
4
5
6 +!starship0: true
7 <-
8 .print("A").
```

A.2.3. Códigos Java

A continuación se da a conocer los código *Java* los cuales se crearon automáticamente en el proceso de traducción, el primer código es el encargado de realizar el puente entre *Jason* y *JaCalIVE*.

Listing A.5: Test

```
1 import jason.environment.grid.Location;
2 import cartago.*;
3 import JacaLiveFrameWork.JacaLive;
4 import cartago.OPERATION;
5 import cartago.CartagoNode;
6 import cartago.ArtifactConfig;
7 import cartago.CartagoWorkspace;
8 import cartago.WorkspaceKernel;
9 import jason.architecture.AgArch;
10 import java.util.logging.Level;
11 import java.util.logging.Logger;
12 import jason.infra.centralised.CentralisedAgArch;
13 import java.awt.Color;
14 import java.io.IOException;
15 import java.io.PrintWriter;
16 import java.net.DatagramSocket;
17 import java.util.ArrayList;
18 import java.util.Arrays;
19 import java.util.Enumeration;
20 import java.util.Hashtable;
21 import java.util.Iterator;
22 import java.util.LinkedList;
23 import java.util.List;
24 import java.util.StringTokenizer;
25 import java.util.logging.Level;
26 import java.util.logging.Logger;
27 import cartago.CartagoException;
28 import java.net.URL;
29 import java.net.URLClassLoader;
30 import java.util.StringTokenizer;
31 import org.lwjgl.LWJGLException;
32
33 public class Jacalive4Jason extends JacaLive {
34    CartagoWorkspace cartWork;
35    ArtifactConfig ConfigArti;
36    CartagoNode Node;
37    WorkspaceKernel WorkSpaKernel;
38    java.util.Set<String> list = null;
39    private int itama = 0;
40    private static Hashtable<String, AgentId> hIdAgent;
```

```
41    static String[] args1 = new String[15];
42    static JacaLive jcaIve;
43    private String sTokA = null;
44    private static String[] saTokA = new String[6];
45    private StringTokenizer tokenA;
46    //private String[] saOutName;
47
48    int iPosiX = 0;
49    int iPosiY = 0;
50    int iPosiZ = 0;
51    String sPosiX = "";
52
53    public Jacalive4Jason() {
54       hIdAgent = new Hashtable<String, AgentId>();
55    }
56
57    public void initJacaLive(String sName, int numWsP, int TamaLinkArti,
58                      int TamaUnLinkArti, int TamaBodyArti, double Gx,
59                      double Gy, double Gz, int Wmap, int Hmap, int Lmap,
60                      boolean bGravity, float fGroundFriction, String Floor)
                            throws Exception {
61       /** initialize de Nodos, Workspace etc **/
62       Node = CartagoNode.getInstance();
63       /** Config Artifact **/
64       ConfigArti = new ArtifactConfig();
65       /** Create the workspace **/
66       cartWork = Node.createWorkspace(sName);
67
68       WorkSpaKernel = cartWork.getKernel();
69       AgArch userAgArch = new AgArch();
70       int iId = 0;
71       System.out.println("-----------------------------------------------------------");
72       ClassLoader cl = ClassLoader.getSystemClassLoader();
73       URL[] urls = ((URLClassLoader) cl).getURLs();
74       for (URL url : urls) {
75          System.out.println(url.getFile());
76       }
77       System.out.println("-----------------------------------------------------------");
78       args1[0] = sName;
79       args1[1] = Integer.toString(numWsP);
80       args1[2] = Integer.toString(TamaLinkArti);
81       args1[3] = Integer.toString(TamaUnLinkArti);
82       args1[4] = Integer.toString(TamaBodyArti);
83       args1[5] = Double.toString(Gx);
84       args1[6] = Double.toString(Gy);
85       args1[7] = Double.toString(Gz);
86       args1[8] = Integer.toString(Wmap);
87       args1[9] = Integer.toString(Hmap);
88       args1[10] = Integer.toString(Lmap);
89       args1[11] = Boolean.toString(bGravity);
90       args1[12] = Double.toString(fGroundFriction);
91       args1[13] = Floor;
92       main(args1);
93       iniWsp(Node, ConfigArti, cartWork, WorkSpaKernel, userAgArch);
94    }
95
96    public void bodyObj(double iBodyX, double iBodyY, double iBodyZ, int index)
          {
97       setBodyObj((int) iBodyX, (int) iBodyY, (int) iBodyZ, index);
98    }
99
100   public void linkArtifact(double iLinkX, double iLinkY, double iLinkZ,
101         int index) {
102      setLinkArtifacr((int) iLinkX, (int) iLinkY, (int) iLinkZ, index);
```

```
103     }
104
105     public void uLinkArtifacr(double iULinkX, double iULinkY, double iULinkZ,
106             int index) {
107         setULinkArtifacr((int) iULinkX, (int) iULinkY, (int) iULinkZ, index);
108     }
109
110     public String agentBodyArtifact(String id) {
111         return getAgentBodyArtifact(id);
112     }
113
114     public String posAgentBodyArtifact(String body) {
115         return getPosAgentBodyArtifact(body);
116     }
117
118     public String getposLinkArtifact(String link) {
119         return getPosLinkArtifact(link);
120     }
121
122     public String getposUnLinkArtifact(String unlink) {
123         return getPosUnLinkArtifact(unlink);
124     }
125
126     public Boolean sendRender(String data) throws IOException {
127         return sendRenders(data);
128     }
129
130     public void inObsProperty(String Name, double L, double W, double H,
131             double Mass, double AccelerationX, double AccelerationY,
132             double AccelerationZ, String Sound, String Shape, double Angle,
133             double Distance, int JointX, int JointY, int JointZ,
134             double OrientationX, double OrientationY, double OrientationZ,
135             double VelocityX, double VelocityY, double VelocityZ, double Px,
136             double Py, double Pz) {
137
138         Object[] param = { Name, L, W, H, Mass, AccelerationX, AccelerationY,
139                 AccelerationZ, Sound, Shape, Angle,
140                 Distance, JointX, JointY, JointZ,
141                 OrientationX, OrientationY, OrientationZ,
142                 VelocityX, VelocityY, VelocityZ, Px,
143                 Py, Pz };
144         inputObsProperty(param);
145     }
146
147     public String artifactList(boolean flag) {
148         String Temp = null;
149         String Temp1 = null;
150         String[] outArti = getArtifactList(flag);
151         for (int i = 0; i < outArti.length; i++) {
152             Temp += outArti[i] + ",";
153         }
154         String[] defaultArtifact = { "console", "node", "blackboard",
155                 "workspace", "manrepo", "Jacalive" };
156         String Algo = compareArrays(outArti, defaultArtifact);
157         int tama = Algo.length() - 2;
158         Temp1 = Algo.substring(4, tama);
159         return Temp1;
160     }
161
162     private String compareArrays(String[] array1, String[] array2) {
163         boolean b = true;
164         String Temp = null;
165         if (array1 != null) {
166             for (int i = 0; i < array1.length; i++) {
```

```
167             if (array1[i] != "console" && array1[i] != "node"
168                   && array1[i] != "blackboard"
169                   && array1[i] != "workspace" && array1[i] != "manrepo"
170                   && array1[i] != "Jacalive") {
171                Temp += (char) 34 + array1[i] + (char) 34 + ",";
172             }
173          }
174       }
175       return Temp;
176    }
177
178    public String bodyAgent(String ArtifactNames, String data, String Ag,
179          String roll, int index, int Tama) throws CartagoException,
                LWJGLException {
180       String Temp = null;
181       String[] outBody = setBodyAgent(ArtifactNames, data, Ag, roll, index,
182             Tama);
183       for (int i = 0; i < outBody.length - 1; i++) {
184          Temp += outBody[i] + ",";
185       }
186       return Temp;
187    }
188
189    public String setLinlArtifact(String ArtifactNames) throws CartagoException,
          LWJGLException {
190       String Temp = null;
191       String[] saTemp = ArtifactNames.split(",");
192       int iTempTama = saTemp[0].length();
193       String sTempNames = saTemp[0].substring(1, (iTempTama-1));
194
195       int iInicio = sTempNames.indexOf("_");
196       String sTempArti = sTempNames.substring(0, iInicio);
197
198       String[] outLinkArti = setPosLinkArtifact(ArtifactNames, sTempNames,
              sTempArti);
199       for (int i = 0; i < outLinkArti.length - 1; i++) {
200          Temp += outLinkArti[i] + ",";
201       }
202       return Temp;
203    }
204
205    public String setUnLinlArtifact(String ArtifactNames) throws
          CartagoException, LWJGLException {
206       String Temp = null;
207       String[] saTemp = ArtifactNames.split(",");
208       int iTempTama = saTemp[1].length();
209       String sTempNames = saTemp[1].substring(1, (iTempTama-1));
210
211       int iInicio = sTempNames.indexOf("_");
212       String sTempArti = sTempNames.substring(0, iInicio);
213
214       System.out.println(ArtifactNames);
215
216       String[] outLinkArti = setPosUnLinkArtifact(ArtifactNames, sTempNames,
              sTempArti);
217       for (int i = 0; i < outLinkArti.length - 1; i++) {
218          Temp += outLinkArti[i] + ",";
219       }
220       return Temp;
221    }
222
223    public void render() throws LWJGLException{
224       RenderToJaCalIVE();
225    }
```

```
226
227     public String agentId() {
228        String Temp = null;
229        String Temp1 = null;
230        String[] outArti = getAgentId();
231        for (int i = 0; i < outArti.length - 1; i++) {
232           Temp += outArti[i] + ",";
233        }
234        int tama = Temp.length();
235        Temp1 = Temp.substring(4, tama - 1);
236        return Temp1;
237     }
238
239
240     public String artiTokenizer(String sTokenizer) {
241        tokenA = new StringTokenizer(sTokenizer.substring(1));
242        String sTokB = "";
243        String Temp = null;
244        String Temp1 = null;
245
246        int iJj = 0;
247        do {
248           sTokA = tokenA.nextToken();
249           saTokA[iJj] = sTokA;
250           sTokB += sTokA + ", ";
251           iJj++;
252        } while (tokenA.hasMoreTokens());
253
254        for (int i = 0; i < saTokA.length - 1; i++) {
255           Temp += saTokA[i] + ",";
256        }
257        int tama = Temp.length();
258        Temp1 = Temp.substring(4, tama - 1);
259        return Temp1;
260     }
261
262     public String checkPosition(String sPosi){
263        String[] sState = sPosi.split(",");
264        int iTama_0 = sState[0].length();
265        sPosiX = sState[0].substring(3, (iTama_0-2));
266        //String sPosiY = sState[1].substring(1, (sState[1].length()-1));
267        //String sPosiZ = sState[2].substring(1, (sState[2].length()-2));
268
269        iPosiX = Integer.parseInt(sPosiX);
270        System.out.println(sPosiX);
271
272        /*int iPosiX = Integer.parseInt(sPosiX);
273        int iPosiY = Integer.parseInt(sPosiY);
274        int iPosiZ = Integer.parseInt(sPosiZ);
275
276        System.out.println(iPosiX + " " + iPosiY + " " + iPosiZ);*/
277        //String sValue = JaCalIVE_CheckPosition(iPosiX, iPosiY, iPosiZ);
278        return null;
279     }
280
281     public void showMap(){
282        ShowMap();
283     }
284
285     public String getBodyAssociateToAgent(String id){
286        return getNameBodyAgent(id);
287     }
288
289     public String splitData(String Data, int index){
```

```
290         String[] sData = Data.split(",");
291         return sData[index];
292     }
293     //*************************************************************************
294     // If you want add your code, please begin hear
295     //*************************************************************************
296     public String getNameAg(String sVal){
297         String[] saOutName = sVal.split(",");
298         return saOutName[0];
299     }
300 
301     public String getXPos(String sVal){
302         String[] saOutName = sVal.split(",");
303         return saOutName[2];
304     }
305 
306     public String getYPos(String sVal){
307         String[] saOutName = sVal.split(",");
308         return saOutName[3];
309     }
310 
311     public String getZPos(String sVal){
312         String[] saOutName = sVal.split(",");
313         return saOutName[4];
314     }
315 
316     public String getAngle(String sVal){
317         String[] saOutName = sVal.split(",");
318         return saOutName[1];
319     }
320 }
```

Listing A.6: Test

```
1 
2 import cartago.*;
3 
4 @ARTIFACT_INFO(
5 outports = {
6 @OUTPORT(name = "out-Body_Starship_starship-Id0")
7 }
8 ) public class Body_Starship_class extends Artifact {
9 
10     // attributes
11 
12     // attributes and physical properties initialization
13     void init(double L, double W, double H, double Mass, double AccelerationX,
          double AccelerationY, double AccelerationZ, String Sound,  String Shape,
           double Angle, double Distance, int JointX, int JointY, int JointZ,
          double OrientationX, double OrientationY, double OrientationZ, double
          VelocityX, double VelocityY, double VelocityZ, double Px, double Py,
          double Pz){
14 
15      //Enter your code in this line :)
16 }
17 
18 }
```

Listing A.7: Test

```
1 
2 import cartago.*;
3 
```

```
4 @ARTIFACT_INFO(
5 outports = {
6 @OUTPORT(name = "out-Link_Robot_starship-Id1")
7 }
8 ) public class Link_Robot_class extends Artifact {
9
10    // attributes
11
12    // attributes and physical properties initialization
13    void init(double L, double W, double H, double Mass, double AccelerationX,
          double AccelerationY, double AccelerationZ, String Sound,  String Shape,
           double Angle, double Distance, int JointX, int JointY, int JointZ,
          double OrientationX, double OrientationY, double OrientationZ, double
          VelocityX, double VelocityY, double VelocityZ, double Px, double Py,
          double Pz){
14
15     //Enter your code in this line :)
16 }
17
18 }
```

Apéndice B

Anexo II: Código del Ejemplo Robot Apodo

En el presente anexo se darán a conocer los códigos correspondientes a los ejemplos planteados en la sección .

B.1. Esqueletos de Código Creados Automáticamente

A continuación se da a conocer los código *asl* y *java* que se crearon automáticamente.

B.1.1. Modelado del IVE en XML

A continuación se da a conocer el código XML, en el cual se modelo el IVE.

Listing B.1: Test

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <IVE NAME="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xsi:noNamespaceSchemaLocation="generalStructure.xsd">
4
5   <VIRTUAL>
6     <IVE_WORKSPACE NAME="apodRobot_workspace">
7       <IVE_ARTIFACTS>
8         <ITEM NAME="Body_Left" />
9         <ITEM NAME="Body_Right" />
10        <ITEM NAME="Link_Camera" />
11        <ITEM NAME="Unlink_Rock" />
12      </IVE_ARTIFACTS>
13      <INHABITANT_AGENTS>
14        <ITEM NAME="robot" />
15      </INHABITANT_AGENTS>
16      <IVE_LAWS>
17        <ITEM NAME="Gravity" />
18      </IVE_LAWS>
19    </IVE_WORKSPACE>
20
21    <IVE_ARTIFACT NAME="Body_Left" LINKEABLE="true">
22      <ATTRIBUTES/>
23      <PHYSICAL_PROPERTIES>
24        <PERCEIVABLE>
25
26          <DOUBLE NAME = "position"> 5.0 15.0 25.1 </DOUBLE>
```

```
27          <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
28          <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
29          <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>
30
31          <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
32          <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
33          <STRING NAME = "shape"> cubic </STRING>
34          <STRING NAME = "sound"> none </STRING>
35        </PERCEIVABLE>
36        <INTERNAL>
37          <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
38          <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
39          <DOUBLE NAME = "size"> 15.0 5.0 25.1 </DOUBLE>
40        </INTERNAL>
41      </PHYSICAL_PROPERTIES>
42      <ACTIONS>
43          <ACTION NAME="move">
44              <ARGUMENTS>
45                  <PARAMETER NAME="newX" TYPE="DOUBLE" />
46                  <PARAMETER NAME="newY" TYPE="DOUBLE" />
47                  <PARAMETER NAME="newZ" TYPE="DOUBLE" />
48              </ARGUMENTS>
49              <DO_ACTION>
50              <ASSIGN>
51                      <OPERAND>
52                          <ELEMENT_PROP NAME="SELF" PROPERTY="position">
53                              <INDEX> "x" </INDEX>
54                          </ELEMENT_PROP>
55                      </OPERAND>
56                      <OPERAND>
57                          <PARAMETER NAME="newX" TYPE="DOUBLE" />
58                      </OPERAND>
59              </ASSIGN>
60              </DO_ACTION>
61              <DO_ACTION>
62              <ASSIGN>
63                      <OPERAND>
64                          <ELEMENT_PROP NAME="SELF" PROPERTY="position">
65                              <INDEX> "y" </INDEX>
66                          </ELEMENT_PROP>
67                      </OPERAND>
68                      <OPERAND>
69                          <PARAMETER NAME="newY" TYPE="DOUBLE" />
70                      </OPERAND>
71              </ASSIGN>
72              </DO_ACTION>
73              <DO_ACTION>
74              <ASSIGN>
75                      <OPERAND>
76                          <ELEMENT_PROP NAME="SELF" PROPERTY="position">
77                              <INDEX> "z" </INDEX>
78                          </ELEMENT_PROP>
79                      </OPERAND>
80                      <OPERAND>
81                          <PARAMETER NAME="newZ" TYPE="DOUBLE" />
82                      </OPERAND>
83              </ASSIGN>
84              </DO_ACTION>
85
86              <DO_ACTION>
87                  <EXEC_LINKED_OP LINKED_ART_ID="1" LINKED_FUNCTION="move">
88                      <LINKED_ARGUMENTS>
89                      <ELEMENT_PROP NAME="SELF" PROPERTY="position">
90                              <INDEX> "x" </INDEX>
```

```
91                      </ELEMENT_PROP>
92                      <ELEMENT_PROP NAME="SELF" PROPERTY="position">
93                              <INDEX> "y" </INDEX>
94                      </ELEMENT_PROP>
95                      <ELEMENT_PROP NAME="SELF" PROPERTY="position">
96                              <INDEX> "z" </INDEX>
97                      </ELEMENT_PROP>
98                  </LINKED_ARGUMENTS>
99                  </EXEC_LINKED_OP>
100             </DO_ACTION>
101
102             <PHYSICAL_EVENT NAME=""/>
103     </ACTION>
104     </ACTIONS>
105 </IVE_ARTIFACT>
106
107 <IVE_ARTIFACT NAME="Body_Right" LINKEABLE="true">
108     <ATTRIBUTES/>
109     <PHYSICAL_PROPERTIES>
110       <PERCEIVABLE>
111
112         <DOUBLE NAME = "position"> 0.0 100.0 123.1 </DOUBLE>
113         <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
114         <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
115         <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>
116
117         <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
118         <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
119         <STRING NAME = "shape"> cubic </STRING>
120         <STRING NAME = "sound"> none </STRING>
121       </PERCEIVABLE>
122       <INTERNAL>
123         <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
124         <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
125         <DOUBLE NAME = "size"> 50.0 40.0 1.1 </DOUBLE>
126       </INTERNAL>
127     </PHYSICAL_PROPERTIES>
128     <ACTIONS>
129         <ACTION NAME="move">
130             <ARGUMENTS>
131                 <PARAMETER NAME="newX" TYPE="DOUBLE"/>
132                 <PARAMETER NAME="newY" TYPE="DOUBLE"/>
133                 <PARAMETER NAME="newZ" TYPE="DOUBLE"/>
134             </ARGUMENTS>
135             <DO_ACTION>
136             <ASSIGN>
137                     <OPERAND>
138                         <ELEMENT_PROP NAME="SELF" PROPERTY="position">
139                             <INDEX> "x" </INDEX>
140                         </ELEMENT_PROP>
141                     </OPERAND>
142                     <OPERAND>
143                         <PARAMETER NAME="newX" TYPE="DOUBLE"/>
144                     </OPERAND>
145             </ASSIGN>
146             </DO_ACTION>
147             <DO_ACTION>
148             <ASSIGN>
149                     <OPERAND>
150                         <ELEMENT_PROP NAME="SELF" PROPERTY="position">
151                             <INDEX> "y" </INDEX>
152                         </ELEMENT_PROP>
153                     </OPERAND>
154                     <OPERAND>
```

```
155                     <PARAMETER NAME="newY" TYPE="DOUBLE" />
156                 </OPERAND>
157             </ASSIGN>
158             </DO_ACTION>
159             <DO_ACTION>
160             <ASSIGN>
161                 <OPERAND>
162                     <ELEMENT_PROP NAME="SELF" PROPERTY="position">
163                         <INDEX> "z" </INDEX>
164                     </ELEMENT_PROP>
165                 </OPERAND>
166                 <OPERAND>
167                     <PARAMETER NAME="newZ" TYPE="DOUBLE" />
168                 </OPERAND>
169             </ASSIGN>
170             </DO_ACTION>
171             <PHYSICAL_EVENT NAME="" />
172     </ACTION>
173
174     </ACTIONS>
175
176 </IVE_ARTIFACT>
177
178  <IVE_ARTIFACT NAME="Link_Camera" LINKEABLE="true">
179     <ATTRIBUTES/>
180     <PHYSICAL_PROPERTIES>
181       <PERCEIVABLE>
182
183         <DOUBLE NAME = "position"> 20.0 1.0 75.1 </DOUBLE>
184         <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
185         <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
186         <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>
187
188         <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
189         <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
190         <STRING NAME = "shape"> cubic </STRING>
191         <STRING NAME = "sound"> none </STRING>
192       </PERCEIVABLE>
193       <INTERNAL>
194         <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
195         <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
196         <DOUBLE NAME = "size"> 35.0 55.0 85.1 </DOUBLE>
197       </INTERNAL>
198     </PHYSICAL_PROPERTIES>
199     <ACTIONS>
200         <ACTION NAME="move">
201             <ARGUMENTS>
202                 <PARAMETER NAME="newX" TYPE="DOUBLE" />
203                 <PARAMETER NAME="newY" TYPE="DOUBLE" />
204                 <PARAMETER NAME="newZ" TYPE="DOUBLE" />
205             </ARGUMENTS>
206             <DO_ACTION>
207             <ASSIGN>
208                 <OPERAND>
209                     <ELEMENT_PROP NAME="SELF" PROPERTY="position">
210                         <INDEX> "x" </INDEX>
211                     </ELEMENT_PROP>
212                 </OPERAND>
213                 <OPERAND>
214                     <PARAMETER NAME="newX" TYPE="DOUBLE" />
215                 </OPERAND>
216             </ASSIGN>
217             </DO_ACTION>
218             <DO_ACTION>
```

```
219             <ASSIGN>
220                 <OPERAND>
221                     <ELEMENT_PROP NAME="SELF" PROPERTY="position">
222                         <INDEX> "y" </INDEX>
223                     </ELEMENT_PROP>
224                 </OPERAND>
225                 <OPERAND>
226                     <PARAMETER NAME="newY" TYPE="DOUBLE" />
227                 </OPERAND>
228             </ASSIGN>
229             </DO_ACTION>
230             <DO_ACTION>
231             <ASSIGN>
232                 <OPERAND>
233                     <ELEMENT_PROP NAME="SELF" PROPERTY="position">
234                         <INDEX> "z" </INDEX>
235                     </ELEMENT_PROP>
236                 </OPERAND>
237                 <OPERAND>
238                     <PARAMETER NAME="newZ" TYPE="DOUBLE" />
239                 </OPERAND>
240             </ASSIGN>
241             </DO_ACTION>
242             <PHYSICAL_EVENT NAME="" />
243       </ACTION>
244       </ACTIONS>
245
246     </IVE_ARTIFACT>
247
248      <IVE_ARTIFACT NAME="Unlink_Rock" LINKEABLE="false">
249       <ATTRIBUTES/>
250       <PHYSICAL_PROPERTIES>
251         <PERCEIVABLE>
252
253           <DOUBLE NAME = "position"> 8.0 8.0 5.1 </DOUBLE>
254           <DOUBLE NAME = "velocity"> 1.0 1.0 1.1 </DOUBLE>
255           <DOUBLE NAME = "orientation"> 1.0 0.0 0.0 </DOUBLE>
256           <DOUBLE NAME = "joint"> 1.0 0.0 0.0 </DOUBLE>
257
258           <DOUBLE NAME = "distance"> 5.0 </DOUBLE>
259           <DOUBLE NAME = "angle"> 0.0 </DOUBLE>
260           <STRING NAME = "shape"> cubic </STRING>
261           <STRING NAME = "sound"> none </STRING>
262         </PERCEIVABLE>
263         <INTERNAL>
264           <DOUBLE NAME = "acceleration"> 1.0 0.0 0.0 </DOUBLE>
265           <DOUBLE NAME = "mass"> 20.0 </DOUBLE>
266           <DOUBLE NAME = "size"> 15.0 105.0 25.1 </DOUBLE>
267         </INTERNAL>
268       </PHYSICAL_PROPERTIES>
269       <ACTIONS/>
270
271     </IVE_ARTIFACT>
272
273     <INHABITANT_AGENT NAME="robot">
274       <ATTRIBUTES/>
275       <BODY_ARTIFACT>
276         <ITEM ID="0" />
277         <ITEM ID="1" />
278       </BODY_ARTIFACT>
279       <FILE NAME="apodRobotJason.asl" />
280     </INHABITANT_AGENT>
281
282     <IVE_LAW NAME="Gravity">
```

```
283     <DOUBLE NAME = "gravity"> 0.0 -9.8 0.0 </DOUBLE>
284         <ACTIONS>
285           <ITEM NAME="move" />
286         </ACTIONS>
287       </IVE_LAW>
288
289    </VIRTUAL>
290
291 </IVE>
```

B.1.2. Proyecto mas2j

Este es el proyecto *Jason* que se crea en el proceso de traducción.

Listing B.2: Test

```
1 /* Jason Project */
2 MAS robotdemo{
3     infrastructure: Centralised
4     environment: c4jason.CartagoEnvironment
5     agents:
6         jacalive agentArchClass c4jason.CAgentArch;
7         robot0 agentArchClass c4jason.CAgentArch;
8     classpath:
9         "../lib/JacaLiveFrameWork.jar";
10        "../Cartago_Lib/cartago.jar";
11        "../Cartago_Lib//c4jason.jar";
12    aslSourcePath:
13        "src/asl";
14 }
```

B.1.3. Códigos asl

Estos son los códigos *asl* creados en el proceso de traducción, el primero corresponde a el agente *jacalive*.

Listing B.3: Test

```
1
2 !jacalive.
3
4
5 +!jacalive: true
6 <- !configArtifact.
7 +!configArtifact: true
8 <-
9 // Inicializacion de las Variables del Mundo
10        Gx = 0.0;// Gravedad en X
11        Gy = -9.8;// Gravedad en Y
12        Gz = 0.0;// Gravedad en Z
13        Friction = 0;       Bgravity = true;       U = 0.5;// Indice de Rozamiento
14        Ts = 0.01;// Steep
15        M = 1.0;// Numero de Artefactos linkeables
16        M1 = 1.0;// Numero de Artefactos NO linkeables
17        N = 1.0;// Numero de Agentes
18        Width = 500;
19        Height = 500;
20        Length = 500;
```

```
21
22
23        createWorkspace("starship_workspace");
24        joinWorkspace("starship_workspace",WspID0);
25        cartago.set_current_wsp(WspID0);
26 /* Concatenamos los nombres de de todos los Work Space */
27       NamesWorkSpaces ="starship_workspace";
28        TamaLinkArti = 1;
29        TamaUnLinkArti = 1;
30        TamaBodyArti = 1;
31        cartago.new_obj("Jacalive4Jason",[],Id);
32        cartago.invoke_obj(Id,initJacaLive(apodRobot_workspace, 1,TamaLinkArti,
              TamaUnLinkArti,TamaBodyArti, Gx, Gy, Gz, Width, Height, Length,
              Bgravity, Friction));
33       //AId, L, W, H, Mass, AccelerationX, AccelerationY, AccelerationZ, Sound,
              Shape, Angle, Distance, JointX, JointY, JointZ, OrientationX,
             OrientationY, OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py,
             Pz
34
35
36        !setupArtifact2(AId2, 15.0,5.0,25.1,20.0,1.0,0.0,0.0,none,cubic
              ,0.0,5.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.1,5.0,15.0,25.1);
37
38        !setupArtifact1(AId1, 35.0,55.0,85.1,20.0,1.0,0.0,0.0,none,cubic
              ,0.0,5.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.1,20.0,1.0,75.1);
39
40        !setupArtifact0(AId0, 15.0,105.0,25.1,20.0,1.0,0.0,0.0,none,cubic
              ,0.0,5.0,1.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,1.1,8.0,8.0,5.1);
41        Body_X = [5.0];
42        Body_Y = [15.0];
43        Body_Z = [25.1];
44        Link_X = [20.0];
45        Link_Y = [1.0];
46        Link_Z = [75.1];
47        ULink_X = [8.0];
48        ULink_Y = [8.0];
49        ULink_Z = [5.1];
50
51
52        //Creo el mampa
53
54    // Configuro los artefactos de el mundo, estos son los artfactos que se
          uniran a los agentes.
55
56    // Configuro los cuerpos de los agentes
57    for (.range(I,0,1-1) ) {
58        .nth(I, Body_X, XA1);
59         .nth(I, Body_Y, YA1);
60        .nth(I, Body_Z, ZA1);
61        cartago.invoke_obj(Id,bodyObj(XA1, YA1, ZA1, I));
62 }
63
64    for (.range(I,0,1-1) ) {
65        .nth(I, Link_X, XA1);
66        .nth(I, Link_Y, YA1);
67        .nth(I, Link_Z, ZA1);
68        cartago.invoke_obj(Id,linkArtifact(XA1, YA1, ZA1, I));
69 }
70
71  for (.range(I,0,1-1) ) {
72        .nth(I, ULink_X, XA1);
73        .nth(I, ULink_Y, YA1);
74        .nth(I, ULink_Z, ZA1);
75        cartago.invoke_obj(Id,uLinkArtifacr(XA1, YA1, ZA1, I));
```

```
76 }
77
78    /* Get all the names of the agents, the result is a list, which is stored in
          Name */
79    .all_names(Name);
80    /* Get the size of the list, there is discounting the few agents JACALIVE */
81    .length(Name, Tama);
82    /*I walk the list of names of the agents and sent him a message whose
          conternido is a 1,
83  *  which tells agents incien sending their ONFIGURATION parameters to be sent
       to the render.
84  */
85 for(.range(I,0,Tama-1)){
86       .nth(I,Name,X);
87       .send(X, achieve, value(1), 500);
88    }
89    .wait(2000);
90    /*Coordino el envio de informacion de los agentes*/
91    .broadcast(tell, value(1));
92  .wait(1000).
93
94
95
96       +!setupArtifact2(AId2, L, W, H, Mass, AccelerationX, AccelerationY,
             AccelerationZ, Sound, Shape, Angle, Distance, JointX, JointY, JointZ
             , OrientationX, OrientationY, OrientationZ, VelocityX, VelocityY,
             VelocityZ, Px, Py, Pz): true
97       <- cartago.new_obj("Jacalive4Jason",[],Id2);
98          cartago.invoke_obj(Id2, inObsProperty("Unlink_Asteroid",L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle
                , Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz));
99          makeArtifact("Unlink_Asteroid","Unlink_Asteroid_class",[L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle
                , Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz ], AId2)
                .
100
101
102      +!setupArtifact1(AId1, L, W, H, Mass, AccelerationX, AccelerationY,
             AccelerationZ, Sound, Shape, Angle, Distance, JointX, JointY, JointZ
             , OrientationX, OrientationY, OrientationZ, VelocityX, VelocityY,
             VelocityZ, Px, Py, Pz): true
103      <- cartago.new_obj("Jacalive4Jason",[],Id1);
104         cartago.invoke_obj(Id1, inObsProperty("Link_Robot",L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz));
105         makeArtifact("Link_Robot","Link_Robot_class",[L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz ], AId1)
                .
106
107
108      +!setupArtifact0(AId0, L, W, H, Mass, AccelerationX, AccelerationY,
             AccelerationZ, Sound, Shape, Angle, Distance, JointX, JointY, JointZ
             , OrientationX, OrientationY, OrientationZ, VelocityX, VelocityY,
             VelocityZ, Px, Py, Pz): true
109      <- cartago.new_obj("Jacalive4Jason",[],Id0);
110         cartago.invoke_obj(Id0, inObsProperty("Body_Starship",L, W, H, Mass,
                AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz));
```

```
111          makeArtifact("Body_Starship","Body_Starship_class",[L, W, H, Mass,
                 AccelerationX, AccelerationY, AccelerationZ, Sound, Shape, Angle,
                  Distance, JointX, JointY, JointZ, OrientationX, OrientationY,
                 OrientationZ, VelocityX, VelocityY, VelocityZ, Px, Py, Pz ], AId0)
                 .
112
113
114 /*-------------------------------------------------------------------------*/
115 /* Asociamos los bodyartifact a los agentes, los agente envian un msg      */
116 /* para que el jacalive le adjudique un cuerpo                             */
117 /*-------------------------------------------------------------------------*/
118 +!associate(Dat)[source(Ag)] : true
119 <-    cartago.new_obj("Jacalive4Jason",[],Id1);
120          cartago.invoke_obj(Id1,artifactList(false), Pack_9);
121          cartago.invoke_obj(Id1, bodyAgent(Pack_9, Dat, Ag, Ag, 10, 0), Pack_3)
                 ;
122          cartago.invoke_obj(Id1,agentId, Pack_4);
123          Pack_5 = [Pack_4];
124          .difference(Pack_5,["jacalive"],OutIdAgent);
125 /* This method cam be use to return the name body agent and your pos. */
126 .length(OutIdAgent, TamAg);
127          for(.range(I,0,TamAg-1)){
128             .nth(I, OutIdAgent, AgBdy);
129             cartago.invoke_obj(Id1,agentBodyArtifact(AgBdy), Pack_6);
130             cartago.invoke_obj(Id1,posAgentBodyArtifact(Pack_6), Pack_7);
131          }.
132 +!initEnviroment: true
133 <-
134       getAllNameLinkeArtifact(Pack_5);
135
136       .length(Pack_5, T_0);
137       for(.range(I_0,0,T_0-1)){
138          .nth(I_0,Pack_5,R);
139          initEnviromentLink(R, I_0);
140 }
141
142       getAllNameUnLinkeArtifact(Pack_6);
143
144       .length(Pack_6, T_1);
145       for(.range(I_1,0,T_1-1)){
146          .nth(I_1,Pack_6,R1);
147          initEnviromentUnLink(R1, I_1);
148       }
149
150       getAllNameAgent(Pack_8);
151       .length(Pack_8, T_2);
152       for(.range(I_2,0,T_2-1)){
153          .nth(I_2,Pack_8,R2);
154          initEnviromentAgent(R2, I_2);
155       }
156 .
157 +!jacalivLoop(Dat)[source(Ag)] : true
158 <-
159
160       cartago.new_obj("Jacalive4Jason",[],Id2);
161       .my_name(N);
162       .all_names(Name);
163       .difference(Name,[N],Xlp);
164       .length(Xlp,X);
165       cartago.invoke_obj(Id2,agentBodyArtifact(Ag), Pack_5);
166       cartago.invoke_obj(Id2,posAgentBodyArtifact(Pack_5), Pack_6);
167       .concat(Pack_5," ",Pack_6,Pack_7);
168       .term2string(Pack_7,NetData);
169       cartago.invoke_obj(Id2,sendRender(NetData), Pack_8);
```

```
170         //Place its code of communication with agents and artefacts here :)
171     .
```

Este es el código correspondiente al agente.

Listing B.4: Test

```
1  !robot0.
2
3  +!robot0: true <- !configArtifact(Id).
4
5  +!configArtifact(Id): true
6  <-
7        XA = 20+5;
8        YA = 20+3;
9        ZA = 0;
10       makeArtifact("Motor","MotorRobot",[],Id).
11
12
13 +!value(X)[source(Ag)]: true
14 <- if(X=="1"){
15       !getBody;
16    }else{
17       !configArtifac;
18    }.
19
20 /*-------------------------------------------------------------------------*/
21 /*Accedo a los Atefactos y los envio al entorno virtual                    */
22 /*-------------------------------------------------------------------------*/
23 +!getBody
24 <-
25 /*-------------------------------------------------------------------------*/
26 /* comunicacion entre el agente robot y el manager, para que el manager le */
27 /* adjudique un cuerpo                                                     */
28 /*-------------------------------------------------------------------------*/
29    .send(jacalive, achieve, associate("Body"));
30    .wait(1000);
31    !robotALoop;
32    .
33
34 +!robotALoop: true
35 <- .my_name(N);                          // Nombre de el agente
36    moveRobot( 50,  50, 0, Angle, X, Y, Z);
37    .term2string(N, SName);
38    .term2string(Angle, Sangle);
39    .term2string(X,X1);
40    .term2string(Y,Y1);
41    .term2string(Z,Z1);
42    .concat(SName, " ", Sangle, " ", X1, " ", Y1, " ", Z1, " ", Ssend);
43    .send(jacalive, achieve, jacalivLoop(Ssend));
44    !robotALoop
45    .
```

B.1.4. Códigos Java

A continuación se da a conocer los código *Java* los cuales se crearon automáticamente en el proceso de traducción, el primer código es el encargado de realizar el puente entre *Jason* y $JaCalIVE$.

Listing B.5: Test

```
1  import jason.environment.grid.Location;
2  import cartago.*;
3  import JacaLiveFrameWork.JacaLive;
4  import cartago.OPERATION;
5  import cartago.CartagoNode;
6  import cartago.ArtifactConfig;
7  import cartago.CartagoWorkspace;
8  import cartago.WorkspaceKernel;
9  import jason.architecture.AgArch;
10 import java.util.logging.Level;
11 import java.util.logging.Logger;
12 import jason.infra.centralised.CentralisedAgArch;
13 import java.awt.Color;
14 import java.io.IOException;
15 import java.io.PrintWriter;
16 import java.net.DatagramSocket;
17 import java.util.ArrayList;
18 import java.util.Arrays;
19 import java.util.Enumeration;
20 import java.util.Hashtable;
21 import java.util.Iterator;
22 import java.util.LinkedList;
23 import java.util.List;
24 import java.util.StringTokenizer;
25 import java.util.logging.Level;
26 import java.util.logging.Logger;
27 import cartago.CartagoException;
28 import java.net.URL;
29 import java.net.URLClassLoader;
30 import java.util.StringTokenizer;
31 import org.lwjgl.LWJGLException;
32
33 public class Jacalive4Jason extends JacaLive {
34    CartagoWorkspace cartWork;
35    ArtifactConfig ConfigArti;
36    CartagoNode Node;
37    WorkspaceKernel WorkSpaKernel;
38    java.util.Set<String> list = null;
39    private int itama = 0;
40    private static Hashtable<String, AgentId> hIdAgent;
41    static String[] args1 = new String[15];
42    static JacaLive jcaIve;
43    private String sTokA = null;
44    private static String[] saTokA = new String[6];
45    private StringTokenizer tokenA;
46    //private String[] saOutName;
47
48    int iPosiX = 0;
49    int iPosiY = 0;
50    int iPosiZ = 0;
51    String sPosiX = "";
52
53    public Jacalive4Jason() {
54       hIdAgent = new Hashtable<String, AgentId>();
55    }
56
57    public void initJacaLive(String sName, int numWsP, int TamaLinkArti,
58                      int TamaUnLinkArti, int TamaBodyArti, double Gx,
59                      double Gy, double Gz, int Wmap, int Hmap, int Lmap,
60                      boolean bGravity, float fGroundFriction, String Floor)
                            throws Exception {
61       /** initialize de Nodos, Workspace etc **/
62       Node = CartagoNode.getInstance();
```

```
63        /** Config Artifact **/
64        ConfigArti = new ArtifactConfig();
65        /** Create the workspace **/
66        cartWork = Node.createWorkspace(sName);
67
68        WorkSpaKernel = cartWork.getKernel();
69        AgArch userAgArch = new AgArch();
70        int iId = 0;
71        System.out.println("-----------------------------------------------------------");
72        ClassLoader cl = ClassLoader.getSystemClassLoader();
73        URL[] urls = ((URLClassLoader) cl).getURLs();
74        for (URL url : urls) {
75           System.out.println(url.getFile());
76        }
77        System.out.println("-----------------------------------------------------------");
78        args1[0] = sName;
79        args1[1] = Integer.toString(numWsP);
80        args1[2] = Integer.toString(TamaLinkArti);
81        args1[3] = Integer.toString(TamaUnLinkArti);
82        args1[4] = Integer.toString(TamaBodyArti);
83        args1[5] = Double.toString(Gx);
84        args1[6] = Double.toString(Gy);
85        args1[7] = Double.toString(Gz);
86        args1[8] = Integer.toString(Wmap);
87        args1[9] = Integer.toString(Hmap);
88        args1[10] = Integer.toString(Lmap);
89        args1[11] = Boolean.toString(bGravity);
90        args1[12] = Double.toString(fGroundFriction);
91        args1[13] = Floor;
92        main(args1);
93        iniWsp(Node, ConfigArti, cartWork, WorkSpaKernel, userAgArch);
94     }
95
96     public void bodyObj(double iBodyX, double iBodyY, double iBodyZ, int index)
           {
97        setBodyObj((int) iBodyX, (int) iBodyY, (int) iBodyZ, index);
98     }
99
100    public void linkArtifact(double iLinkX, double iLinkY, double iLinkZ,
101          int index) {
102       setLinkArtifacr((int) iLinkX, (int) iLinkY, (int) iLinkZ, index);
103    }
104
105    public void uLinkArtifacr(double iULinkX, double iULinkY, double iULinkZ,
106          int index) {
107       setULinkArtifacr((int) iULinkX, (int) iULinkY, (int) iULinkZ, index);
108    }
109
110    public String agentBodyArtifact(String id) {
111       return getAgentBodyArtifact(id);
112    }
113
114    public String posAgentBodyArtifact(String body) {
115       return getPosAgentBodyArtifact(body);
116    }
117
118    public String getposLinkArtifact(String link) {
119       return getPosLinkArtifact(link);
120    }
121
122    public String getposUnLinkArtifact(String unlink) {
123       return getPosUnLinkArtifact(unlink);
124    }
125
```

```
126     public Boolean sendRender(String data) throws IOException {
127        return sendRenders(data);
128     }
129
130     public void inObsProperty(String Name, double L, double W, double H,
131           double Mass, double AccelerationX, double AccelerationY,
132           double AccelerationZ, String Sound, String Shape, double Angle,
133           double Distance, int JointX, int JointY, int JointZ,
134           double OrientationX, double OrientationY, double OrientationZ,
135           double VelocityX, double VelocityY, double VelocityZ, double Px,
136           double Py, double Pz) {
137
138        Object[] param = { Name, L, W, H, Mass, AccelerationX, AccelerationY,
139              AccelerationZ, Sound, Shape, Angle,
140              Distance, JointX, JointY, JointZ,
141              OrientationX, OrientationY, OrientationZ,
142              VelocityX, VelocityY, VelocityZ, Px,
143              Py, Pz };
144        inputObsProperty(param);
145     }
146
147     public String artifactList(boolean flag) {
148        String Temp = null;
149        String Temp1 = null;
150        String[] outArti = getArtifactList(flag);
151        for (int i = 0; i < outArti.length; i++) {
152           Temp += outArti[i] + ",";
153        }
154        String[] defaultArtifact = { "console", "node", "blackboard",
155              "workspace", "manrepo", "Jacalive" };
156        String Algo = compareArrays(outArti, defaultArtifact);
157        int tama = Algo.length() - 2;
158        Temp1 = Algo.substring(4, tama);
159        return Temp1;
160     }
161
162     private String compareArrays(String[] array1, String[] array2) {
163        boolean b = true;
164        String Temp = null;
165        if (array1 != null) {
166           for (int i = 0; i < array1.length; i++) {
167              if (array1[i] != "console" && array1[i] != "node"
168                    && array1[i] != "blackboard"
169                    && array1[i] != "workspace" && array1[i] != "manrepo"
170                    && array1[i] != "Jacalive") {
171                 Temp += (char) 34 + array1[i] + (char) 34 + ",";
172              }
173           }
174        }
175        return Temp;
176     }
177
178     public String bodyAgent(String ArtifactNames, String data, String Ag,
179           String roll, int index, int Tama) throws CartagoException,
                 LWJGLException {
180        String Temp = null;
181        String[] outBody = setBodyAgent(ArtifactNames, data, Ag, roll, index,
182              Tama);
183        for (int i = 0; i < outBody.length - 1; i++) {
184           Temp += outBody[i] + ",";
185        }
186        return Temp;
187     }
188
```

```
189     public String setLinlArtifact(String ArtifactNames) throws CartagoException,
            LWJGLException {
190        String Temp = null;
191        String[] saTemp = ArtifactNames.split(",");
192        int iTempTama = saTemp[0].length();
193        String sTempNames = saTemp[0].substring(1, (iTempTama-1));
194
195        int iInicio = sTempNames.indexOf("_");
196        String sTempArti = sTempNames.substring(0, iInicio);
197
198        String[] outLinkArti = setPosLinkArtifact(ArtifactNames, sTempNames,
               sTempArti);
199        for (int i = 0; i < outLinkArti.length - 1; i++) {
200           Temp += outLinkArti[i] + ",";
201        }
202        return Temp;
203     }
204
205     public String setUnLinlArtifact(String ArtifactNames) throws
           CartagoException, LWJGLException {
206        String Temp = null;
207        String[] saTemp = ArtifactNames.split(",");
208        int iTempTama = saTemp[1].length();
209        String sTempNames = saTemp[1].substring(1, (iTempTama-1));
210
211        int iInicio = sTempNames.indexOf("_");
212        String sTempArti = sTempNames.substring(0, iInicio);
213
214        System.out.println(ArtifactNames);
215
216        String[] outLinkArti = setPosUnLinkArtifact(ArtifactNames, sTempNames,
               sTempArti);
217        for (int i = 0; i < outLinkArti.length - 1; i++) {
218           Temp += outLinkArti[i] + ",";
219        }
220        return Temp;
221     }
222
223     public void render() throws LWJGLException{
224        RenderToJaCalIVE();
225     }
226
227     public String agentId() {
228        String Temp = null;
229        String Temp1 = null;
230        String[] outArti = getAgentId();
231        for (int i = 0; i < outArti.length - 1; i++) {
232           Temp += outArti[i] + ",";
233        }
234        int tama = Temp.length();
235        Temp1 = Temp.substring(4, tama - 1);
236        return Temp1;
237     }
238
239
240     public String artiTokenizer(String sTokenizer) {
241        tokenA = new StringTokenizer(sTokenizer.substring(1));
242        String sTokB = "";
243        String Temp = null;
244        String Temp1 = null;
245
246        int iJj = 0;
247        do {
248           sTokA = tokenA.nextToken();
```

```
249             saTokA[iJj] = sTokA;
250             sTokB += sTokA + ", ";
251             iJj++;
252         } while (tokenA.hasMoreTokens());
253 
254         for (int i = 0; i < saTokA.length - 1; i++) {
255             Temp += saTokA[i] + ",";
256         }
257         int tama = Temp.length();
258         Temp1 = Temp.substring(4, tama - 1);
259         return Temp1;
260     }
261 
262     public String checkPosition(String sPosi){
263         String[] sState = sPosi.split(",");
264         int iTama_0 = sState[0].length();
265         sPosiX = sState[0].substring(3, (iTama_0-2));
266         //String sPosiY = sState[1].substring(1, (sState[1].length()-1));
267         //String sPosiZ = sState[2].substring(1, (sState[2].length()-2));
268 
269         iPosiX = Integer.parseInt(sPosiX);
270         System.out.println(sPosiX);
271 
272         /*int iPosiX = Integer.parseInt(sPosiX);
273         int iPosiY = Integer.parseInt(sPosiY);
274         int iPosiZ = Integer.parseInt(sPosiZ);
275 
276         System.out.println(iPosiX + " " + iPosiY + " " + iPosiZ);*/
277         //String sValue = JaCalIVE_CheckPosition(iPosiX, iPosiY, iPosiZ);
278         return null;
279     }
280 
281     public void showMap(){
282         ShowMap();
283     }
284 
285     public String getBodyAssociateToAgent(String id){
286         return getNameBodyAgent(id);
287     }
288 
289     public String splitData(String Data, int index){
290         String[] sData = Data.split(",");
291         return sData[index];
292     }
293     //************************************************************************
294     // If you want add your code, please begin hear
295     //************************************************************************
296     public String getNameAg(String sVal){
297         String[] saOutName = sVal.split(",");
298         return saOutName[0];
299     }
300 
301     public String getXPos(String sVal){
302         String[] saOutName = sVal.split(",");
303         return saOutName[2];
304     }
305 
306     public String getYPos(String sVal){
307         String[] saOutName = sVal.split(",");
308         return saOutName[3];
309     }
310 
311     public String getZPos(String sVal){
312         String[] saOutName = sVal.split(",");
```

```
313         return saOutName[4];
314     }
315
316     public String getAngle(String sVal){
317         String[] saOutName = sVal.split(",");
318         return saOutName[1];
319     }
320 }
```

Listing B.6: Test

```
1  import cartago.*;
2  import JacaLiveFrameWork.JacaLive;
3  import cartago.ARTIFACT_INFO;
4  import cartago.Artifact;
5  import cartago.ArtifactConfig;
6  import cartago.OPERATION;
7  import cartago.OUTPORT;
8  import cartago.ObsProperty;
9  import cartago.OpFeedbackParam;
10 import cartago.OperationException;
11
12 @ARTIFACT_INFO(
13 outports = {
14 @OUTPORT(name = "out-Body_Left_robot0-Id0")
15 }
16 ) public class Body_Left_Robot0_class extends Artifact {
17
18     // attributes
19
20     // physical properties
21     // physical properties
22     public ObsProperty L1;
23     public ObsProperty W1;
24     public ObsProperty H1;
25     public ObsProperty Shape1;
26     public ObsProperty Px1;
27     public ObsProperty Py1;
28     public ObsProperty Pz1;
29     public ObsProperty VelocityX1;
30     public ObsProperty VelocityY1;
31     public ObsProperty VelocityZ1;
32     public ObsProperty Distance1;
33     public ObsProperty Angle1;
34     public ObsProperty Sound1;
35     public ObsProperty AccelerationX1;
36     public ObsProperty AccelerationY1;
37     public ObsProperty AccelerationZ1;
38     public ObsProperty JointX1;
39     public ObsProperty JointY1;
40     public ObsProperty JointZ1;
41     public ObsProperty Mass1;
42
43     /*private double AX;
44     private double AY;
45     private double AZ;*/
46
47     int v = 0;
48
49     static String sX1 = null;
50     double Y1 = 0;
51     double Z1 = 0;
52
53     double DiffX = 0;
```

```
54    double DiffY = 0;
55    double DiffZ = 0;
56
57    boolean bFlag = true;
58    boolean bflag3 = true;
59    private boolean bflag4 = false;
60
61    float fTarguetX = 0;
62    float fTarguetY = 0;
63    float fX = 0;
64    float fY = 0;
65    float fGravedad = (float) 9.0;
66    float fMass = 0;
67
68    float fAngleY = 0;
69
70    Object XPos;
71    Object YPos;
72    Object ZPos;
73
74    String sPosiX = null;
75    String sPosiY = null;
76    String sPosiZ = null;
77
78    String str1 = null;
79    String str2 = null;
80    String str3 = null;
81
82    String[] saOutPos = new String[3];
83
84    float fVelox = 0f;
85    float fVeloy = 0f;
86    float fVeloz = 0f;
87
88    int conta = 0;
89
90    float[] fPosValue = new float[3];
91    public Body_Left_Robot0_class() {
92       /** Constructor empty **/
93    }
94
95       // attributes
96
97
98    // attributes and physical properties initialization
99    void init(double L, double W, double H, double Mass, double AccelerationX,
          double AccelerationY, double AccelerationZ, String Sound,  String Shape,
           double Angle, double Distance, int JointX, int JointY, int JointZ,
          double OrientationX, double OrientationY, double OrientationZ, double
          VelocityX, double VelocityY, double VelocityZ, double Px, double Py,
          double Pz){
100 System.out.println("Iniciando el Body Artifact... " + Px + " "
101             + Py + " " + Pz);
102
103       // attributes
104       // physical properties
105       defineObsProperty("L", L);
106       this.L1 = getObsProperty("L");
107
108       defineObsProperty("W", W);
109       this.W1 = getObsProperty("W");
110
111       defineObsProperty("H", H);
112       this.H1 = getObsProperty("H");
```

```
113
114         defineObsProperty("Px", Px);
115         this.Px1 = getObsProperty("Px");
116
117         defineObsProperty("Py", Py);
118         this.Py1 = getObsProperty("Py");
119
120         defineObsProperty("Pz", Pz);
121         this.Pz1 = getObsProperty("Pz");
122
123         defineObsProperty("VelocityX", VelocityX);
124         this.VelocityX1 = getObsProperty("VelocityX");
125
126         defineObsProperty("VelocityY", VelocityY);
127         this.VelocityY1 = getObsProperty("VelocityY");
128
129         defineObsProperty("VelocityZ", VelocityZ);
130         this.VelocityZ1 = getObsProperty("VelocityZ");
131
132         defineObsProperty("Distance", Distance);
133         this.Distance1 = getObsProperty("Distance");
134
135         defineObsProperty("Angle", Angle);
136         this.Angle1 = getObsProperty("Angle");
137
138         defineObsProperty("Sound", Sound);
139         this.Sound1 = getObsProperty("Sound");
140
141         defineObsProperty("AccelerationX", AccelerationX);
142         this.AccelerationX1 = getObsProperty("AccelerationX");
143
144         defineObsProperty("AccelerationY", AccelerationY);
145         this.AccelerationY1 = getObsProperty("AccelerationY");
146
147         defineObsProperty("AccelerationZ", AccelerationZ);
148         this.AccelerationZ1 = getObsProperty("AccelerationZ");
149
150         defineObsProperty("JointX", JointX);
151         this.JointX1 = getObsProperty("JointX");
152
153         defineObsProperty("JointY", JointY);
154         this.JointY1 = getObsProperty("JointY");
155
156         defineObsProperty("JointZ", JointZ);
157         this.JointZ1 = getObsProperty("JointZ");
158
159         defineObsProperty("Mass", Mass);
160         this.Mass1 = getObsProperty("Mass");
161
162         defineObsProperty("Shape", Shape);
163         this.Shape1 = getObsProperty("Shape");
164     }
165
166     // acciones
167     // getters & setters
168
169     // Metering
170     public void setL(float L) {
171         this.L1.updateValue(L);
172     }
173
174     public Object getL() {
175         return this.L1.getValue();
176     }
```

```
177
178     public void setW(float W) {
179         this.W1.updateValue(W);
180     }
181
182     public Object getW() {
183         return this.W1.getValue();
184     }
185
186     public void setH(float H) {
187         this.H1.updateValue(H);
188     }
189
190     public Object getH() {
191         return this.H1.getValue();
192     }
193
194     // Shape
195     public void setShape(String Shape) {
196         this.Shape1.updateValue(Shape);
197     }
198
199     public Object getShape() {
200         return this.Shape1.getValue();
201     }
202
203     // Pos
204     public void setPX(float PosX) {
205         this.Px1.updateValue(PosX);
206     }
207
208     public Object getPX() {
209         return this.Px1.getValue();
210     }
211
212     public void setPY(float PosY) {
213         this.Py1.updateValue(PosY);
214     }
215
216     public Object getPY() {
217         return this.Py1.getValue();
218     }
219
220     public void setPZ(float PosZ) {
221         this.Pz1.updateValue(PosZ);
222     }
223
224     public Object getPZ() {
225         return this.Pz1.getValue();
226     }
227
228     // Velocity
229     public void setVelocityX(float VeloX) {
230         this.VelocityX1.updateValue(VeloX);
231     }
232
233     public Object getVelocityX() {
234         return this.VelocityX1.getValue();
235     }
236
237     public void setVelocityY(float VeloY) {
238         this.VelocityY1.updateValue(VeloY);
239     }
240
```

```
241     public Object getVelocityY() {
242        return this.VelocityY1.getValue();
243     }
244
245     public void setVelocityZ(float VeloZ) {
246        this.VelocityZ1.updateValue(VeloZ);
247     }
248
249     public Object getVelocityZ() {
250        return this.VelocityZ1.getValue();
251     }
252
253     // Distan
254     public void setDistance(float Dist) {
255        this.Distance1.updateValue(Dist);
256     }
257
258     public Object getDistance() {
259        return this.Distance1.getValue();
260     }
261
262     // Angle
263     public void setAngle(float Ang) {
264        this.Angle1.updateValue(Ang);
265     }
266
267     public Object getAngle() {
268        return this.Angle1.getValue();
269     }
270
271     // Sound
272     public void setSound(String Soun) {
273        this.Sound1.updateValue(Soun);
274     }
275
276     public Object getSound() {
277        return this.Sound1.getValue();
278     }
279
280     // Aceleration
281     public void setAccelerationX1(float AccX) {
282        this.AccelerationX1.updateValue(AccX);
283     }
284
285     public Object getAccelerationX() {
286        return this.AccelerationX1.getValue();
287     }
288
289     public void setAccelerationY(float AccY) {
290        this.AccelerationY1.updateValue(AccY);
291     }
292
293     public Object getAccelerationY() {
294        return this.AccelerationY1.getValue();
295     }
296
297     public void setAccelerationZ(float AccZ) {
298        this.AccelerationZ1.updateValue(AccZ);
299     }
300
301     public Object getAccelerationZ() {
302        return this.AccelerationZ1.getValue();
303     }
304
```

```
305     // Joint
306     public void setJointX(float JointX) {
307        this.JointX1.updateValue(JointX);
308     }
309
310     public Object getJointX() {
311        return this.JointX1.getValue();
312     }
313
314     public void setAJointY(float JointY) {
315        this.JointY1.updateValue(JointY);
316     }
317
318     public Object getJointY() {
319        return this.JointY1.getValue();
320     }
321
322     public void setJointZ(float JointZ) {
323        this.JointZ1.updateValue(JointZ);
324     }
325
326     public Object getJointZ() {
327        return this.JointZ1.getValue();
328     }
329
330     // Mass
331     public void setMass(float mass) {
332        this.Mass1.updateValue(mass);
333     }
334
335     public Object getMass() {
336        return this.Mass1.getValue();
337     }
338
339     @OPERATION
340     public void stop(){
341        //while(true){}
342     }
343
344        /*
345      * Metodo que actualiza la pos de el artefacto izquierdo
346      */
347     @OPERATION
348     public void moveBodyB(String sBodyName, String Ang, String X, String Y,
349           String Z, String Index, OpFeedbackParam<String[]> PosRobotA, Object[]
                  Value) throws OperationException {
350
351        double dAngulo = Double.parseDouble(Ang);
352        double dX = Double.parseDouble(X);
353        double dY = Double.parseDouble(Y);
354        double dZ = Double.parseDouble(Z);
355
356        int iIndex = Integer.parseInt(Index);
357
358        if (sBodyName.equals("Body_Left")) {
359           //JacaLive.SetVelocityArtifacts(sBodyName, fVelox, fVeloz, fVeloy);
360
361           fVelox = fVelox + 0.01f;
362           fVeloy = fVeloy + 0.01f;
363           fVeloz = fVeloz + 0.01f;
364
365           if(fAngleY<1){
366              fAngleY = fAngleY + 0.05f;
367           }
```

```
368
369             if(fVelox>3){
370                 fVelox = 0;
371                 fVeloy = 0;
372                 fVeloz = 0;
373                 fAngleY = 0;
374             }
375             signal("isBodyLeft", "Body_Left");
376         }
377             PosRobotA.set(saOutPos);
378
379     }
380
381     @OPERATION
382     public void getPosBodyB(String agentName,OpFeedbackParam<String[]> PosRobotA
            ,
383                             OpFeedbackParam<Boolean> state) throws
                                OperationException {
384
385         if (agentName.equals("Body_Left")) {
386
387             // Update the ObserverProperty Pos
388             float[] fPosValue = JacaLive.GetPosArtifacts("Body_Left");
389             int ix = (int)fPosValue[0];
390             int iy = (int)fPosValue[1];
391             int iz = (int)fPosValue[2];
392
393             //System.out.println("Pos: " + " " + ix + " " + iy + " " + iz);
394
395             Px1.updateValue(ix);
396             Py1.updateValue(iy);
397             Pz1.updateValue(iz);
398
399             // Update the ObserverProperty Velocity
400             float[] fVeloValues = JacaLive.GetVelocityArtifacts();
401             VelocityX1.updateValue(fVeloValues[0]);
402             VelocityY1.updateValue(fVeloValues[1]);
403             VelocityZ1.updateValue(fVeloValues[2]);
404
405             // Send signal to agent JacaLive
406             signal("movePosX", ix);
407             signal("movePosY", iy);
408             signal("movePosZ", iz);
409
410             saOutPos[0] = String.valueOf(fPosValue[0]);
411             saOutPos[1] = String.valueOf(fPosValue[1]);
412             saOutPos[2] = String.valueOf(fPosValue[2]);
413
414
415             PosRobotA.set(saOutPos);
416             bflag4 = true;
417         } else {
418             bflag4 = false;
419         }
420         state.set(bflag4);
421     }
422
423 }
```

Listing B.7: Test

```
1
2 import cartago.*;
3 import JacaLiveFrameWork.JacaLive;
```

```
4  import cartago.ARTIFACT_INFO;
5  import cartago.Artifact;
6  import cartago.ArtifactConfig;
7  import cartago.OPERATION;
8  import cartago.OUTPORT;
9  import cartago.ObsProperty;
10 import cartago.OpFeedbackParam;
11 import cartago.OperationException;
12
13 @ARTIFACT_INFO(
14 outports = {
15 @OUTPORT(name = "out-Body_Right_robot0-Id2")
16 }
17 ) public class Body_Right_Robot0_class extends Artifact {
18    // attributes
19
20    // physical properties
21    // physical properties
22    public ObsProperty L1;
23    public ObsProperty W1;
24    public ObsProperty H1;
25    public ObsProperty Shape1;
26    public ObsProperty Px1;
27    public ObsProperty Py1;
28    public ObsProperty Pz1;
29    public ObsProperty VelocityX1;
30    public ObsProperty VelocityY1;
31    public ObsProperty VelocityZ1;
32    public ObsProperty Distance1;
33    public ObsProperty Angle1;
34    public ObsProperty Sound1;
35    public ObsProperty AccelerationX1;
36    public ObsProperty AccelerationY1;
37    public ObsProperty AccelerationZ1;
38    public ObsProperty JointX1;
39    public ObsProperty JointY1;
40    public ObsProperty JointZ1;
41    public ObsProperty Mass1;
42
43    /*private double AX;
44    private double AY;
45    private double AZ;*/
46
47    int v = 0;
48
49    static String sX1 = null;
50    double Y1 = 0;
51    double Z1 = 0;
52
53    double DiffX = 0;
54    double DiffY = 0;
55    double DiffZ = 0;
56
57    boolean bFlag = true;
58    boolean bflag3 = true;
59    private boolean bflag4 = false;
60
61    float fTarguetX = 0;
62    float fTarguetY = 0;
63    float fX = 0;
64    float fY = 0;
65    float fGravedad = (float) 9.0;
66    float fMass = 0;
67
```

```
68      float fAngleY = 0;
69
70      Object XPos;
71      Object YPos;
72      Object ZPos;
73
74      String sPosiX = null;
75      String sPosiY = null;
76      String sPosiZ = null;
77
78      String str1 = null;
79      String str2 = null;
80      String str3 = null;
81
82      String[] saOutPos = new String[3];
83
84      float fVelox = 0f;
85      float fVeloy = 0f;
86      float fVeloz = 0f;
87
88      int conta = 0;
89
90      float[] fPosValue = new float[3];;
91
92
93      public Body_Right_Robot0_class() {
94         /** Constructor empty **/
95      }
96
97      // attributes
98
99
100     // attributes and physical properties initialization
101     void init(double L, double W, double H, double Mass, double AccelerationX,
            double AccelerationY, double AccelerationZ, String Sound,  String Shape,
             double Angle, double Distance, int JointX, int JointY, int JointZ,
            double OrientationX, double OrientationY, double OrientationZ, double
            VelocityX, double VelocityY, double VelocityZ, double Px, double Py,
            double Pz){
102 System.out.println("Iniciando el Body Artifact... " + Px + " "
103             + Py + " " + Pz);
104
105        // attributes
106        // physical properties
107        defineObsProperty("L", L);
108        this.L1 = getObsProperty("L");
109
110        defineObsProperty("W", W);
111        this.W1 = getObsProperty("W");
112
113        defineObsProperty("H", H);
114        this.H1 = getObsProperty("H");
115
116        defineObsProperty("Px", Px);
117        this.Px1 = getObsProperty("Px");
118
119        defineObsProperty("Py", Py);
120        this.Py1 = getObsProperty("Py");
121
122        defineObsProperty("Pz", Pz);
123        this.Pz1 = getObsProperty("Pz");
124
125        defineObsProperty("VelocityX", VelocityX);
126        this.VelocityX1 = getObsProperty("VelocityX");
```

```
127
128         defineObsProperty("VelocityY", VelocityY);
129         this.VelocityY1 = getObsProperty("VelocityY");
130
131         defineObsProperty("VelocityZ", VelocityZ);
132         this.VelocityZ1 = getObsProperty("VelocityZ");
133
134         defineObsProperty("Distance", Distance);
135         this.Distance1 = getObsProperty("Distance");
136
137         defineObsProperty("Angle", Angle);
138         this.Angle1 = getObsProperty("Angle");
139
140         defineObsProperty("Sound", Sound);
141         this.Sound1 = getObsProperty("Sound");
142
143         defineObsProperty("AccelerationX", AccelerationX);
144         this.AccelerationX1 = getObsProperty("AccelerationX");
145
146         defineObsProperty("AccelerationY", AccelerationY);
147         this.AccelerationY1 = getObsProperty("AccelerationY");
148
149         defineObsProperty("AccelerationZ", AccelerationZ);
150         this.AccelerationZ1 = getObsProperty("AccelerationZ");
151
152         defineObsProperty("JointX", JointX);
153         this.JointX1 = getObsProperty("JointX");
154
155         defineObsProperty("JointY", JointY);
156         this.JointY1 = getObsProperty("JointY");
157
158         defineObsProperty("JointZ", JointZ);
159         this.JointZ1 = getObsProperty("JointZ");
160
161         defineObsProperty("Mass", Mass);
162         this.Mass1 = getObsProperty("Mass");
163
164         defineObsProperty("Shape", Shape);
165         this.Shape1 = getObsProperty("Shape");
166     }
167
168     // acciones
169     // getters & setters
170
171     // Metering
172     public void setL(float L) {
173         this.L1.updateValue(L);
174     }
175
176     public Object getL() {
177         return this.L1.getValue();
178     }
179
180     public void setW(float W) {
181         this.W1.updateValue(W);
182     }
183
184     public Object getW() {
185         return this.W1.getValue();
186     }
187
188     public void setH(float H) {
189         this.H1.updateValue(H);
190     }
```

```
191
192     public Object getH() {
193        return this.H1.getValue();
194     }
195
196     // Shape
197     public void setShape(String Shape) {
198        this.Shape1.updateValue(Shape);
199     }
200
201     public Object getShape() {
202        return this.Shape1.getValue();
203     }
204
205     // Pos
206     public void setPX(float PosX) {
207        this.Px1.updateValue(PosX);
208     }
209
210     public Object getPX() {
211        return this.Px1.getValue();
212     }
213
214     public void setPY(float PosY) {
215        this.Py1.updateValue(PosY);
216     }
217
218     public Object getPY() {
219        return this.Py1.getValue();
220     }
221
222     public void setPZ(float PosZ) {
223        this.Pz1.updateValue(PosZ);
224     }
225
226     public Object getPZ() {
227        return this.Pz1.getValue();
228     }
229
230     // Velocity
231     public void setVelocityX(float VeloX) {
232        this.VelocityX1.updateValue(VeloX);
233     }
234
235     public Object getVelocityX() {
236        return this.VelocityX1.getValue();
237     }
238
239     public void setVelocityY(float VeloY) {
240        this.VelocityY1.updateValue(VeloY);
241     }
242
243     public Object getVelocityY() {
244        return this.VelocityY1.getValue();
245     }
246
247     public void setVelocityZ(float VeloZ) {
248        this.VelocityZ1.updateValue(VeloZ);
249     }
250
251     public Object getVelocityZ() {
252        return this.VelocityZ1.getValue();
253     }
254
```

```
255     // Distan
256     public void setDistance(float Dist) {
257        this.Distance1.updateValue(Dist);
258     }
259
260     public Object getDistance() {
261        return this.Distance1.getValue();
262     }
263
264     // Angle
265     public void setAngle(float Ang) {
266        this.Angle1.updateValue(Ang);
267     }
268
269     public Object getAngle() {
270        return this.Angle1.getValue();
271     }
272
273     // Sound
274     public void setSound(String Soun) {
275        this.Sound1.updateValue(Soun);
276     }
277
278     public Object getSound() {
279        return this.Sound1.getValue();
280     }
281
282     // Aceleration
283     public void setAccelerationX1(float AccX) {
284        this.AccelerationX1.updateValue(AccX);
285     }
286
287     public Object getAccelerationX() {
288        return this.AccelerationX1.getValue();
289     }
290
291     public void setAccelerationY(float AccY) {
292        this.AccelerationY1.updateValue(AccY);
293     }
294
295     public Object getAccelerationY() {
296        return this.AccelerationY1.getValue();
297     }
298
299     public void setAccelerationZ(float AccZ) {
300        this.AccelerationZ1.updateValue(AccZ);
301     }
302
303     public Object getAccelerationZ() {
304        return this.AccelerationZ1.getValue();
305     }
306
307     // Joint
308     public void setJointX(float JointX) {
309        this.JointX1.updateValue(JointX);
310     }
311
312     public Object getJointX() {
313        return this.JointX1.getValue();
314     }
315
316     public void setAJointY(float JointY) {
317        this.JointY1.updateValue(JointY);
318     }
```

```
319
320     public Object getJointY() {
321        return this.JointY1.getValue();
322     }
323
324     public void setJointZ(float JointZ) {
325        this.JointZ1.updateValue(JointZ);
326     }
327
328     public Object getJointZ() {
329        return this.JointZ1.getValue();
330     }
331
332     // Mass
333     public void setMass(float mass) {
334        this.Mass1.updateValue(mass);
335     }
336
337     public Object getMass() {
338        return this.Mass1.getValue();
339     }
340
341     @OPERATION
342     public void stop(){
343        //while(true){}
344     }
345
346     /*
347      * Metodo que actualiza la pos de el artefacto izquierdo
348      */
349     @OPERATION
350     public void moveBodyA(String sBodyName, String Ang, String X, String Y,
351           String Z, String Index, OpFeedbackParam<String[]> PosRobotA, Object[]
                Value) throws OperationException {
352
353        double dAngulo = Double.parseDouble(Ang);
354        double dX = Double.parseDouble(X);
355        double dY = Double.parseDouble(Y);
356        double dZ = Double.parseDouble(Z);
357
358        int iIndex = Integer.parseInt(Index);
359
360        if (sBodyName.equals("Body_Right")) {
361           //JacaLive.SetVelocityArtifacts(sBodyName, fVelox, fVeloz, fVeloy);
362
363           fVelox = fVelox + 0.01f;
364           fVeloy = fVeloy + 0.01f;
365           fVeloz = fVeloz + 0.01f;
366
367           if(fAngleY<1){
368              fAngleY = fAngleY + 0.05f;
369           }
370
371           if(fVelox>3){
372              fVelox = 0;
373              fVeloy = 0;
374              fVeloz = 0;
375              fAngleY = 0;
376           }
377           signal("isBodyRight", "Body_Right");
378        }
379           PosRobotA.set(saOutPos);
380
381     }
```

```
382
383     @OPERATION
384     public void getPosBodyA(String agentName,OpFeedbackParam<String[]> PosRobotA
            ,
385                             OpFeedbackParam<Boolean> state) throws
                                    OperationException {
386
387         if (agentName.equals("Body_Right")) {
388
389             // Update the ObserverProperty Pos
390             float[] fPosValue = JacaLive.GetPosArtifacts("Body_Right");
391             int ix = (int)fPosValue[0];
392             int iy = (int)fPosValue[1];
393             int iz = (int)fPosValue[2];
394
395             //System.out.println("Pos: " + " " + ix + " " + iy + " " + iz);
396
397             Px1.updateValue(ix);
398             Py1.updateValue(iy);
399             Pz1.updateValue(iz);
400
401             // Update the ObserverProperty Velocity
402             float[] fVeloValues = JacaLive.GetVelocityArtifacts();
403             VelocityX1.updateValue(fVeloValues[0]);
404             VelocityY1.updateValue(fVeloValues[1]);
405             VelocityZ1.updateValue(fVeloValues[2]);
406
407             // Send signal to agent JacaLive
408             signal("movePosX", ix);
409             signal("movePosY", iy);
410             signal("movePosZ", iz);
411
412             saOutPos[0] = String.valueOf(fPosValue[0]);
413             saOutPos[1] = String.valueOf(fPosValue[1]);
414             saOutPos[2] = String.valueOf(fPosValue[2]);
415
416
417             PosRobotA.set(saOutPos);
418             bflag4 = true;
419         } else {
420             bflag4 = false;
421         }
422         state.set(bflag4);
423     }
424 }
```

Listing B.8: Test

```
1  import cartago.*;
2  import cartago.AbstractWorkspacePoint;
3  import JacaLiveFrameWork.JacaLive;
4  import cartago.ARTIFACT_INFO;
5  import cartago.Artifact;
6  import cartago.ArtifactConfig;
7  import cartago.OPERATION;
8  import cartago.OUTPORT;
9  import cartago.ObsProperty;
10 import cartago.OpFeedbackParam;
11 import cartago.OperationException;
12
13 @ARTIFACT_INFO(
14 outports = {
15 @OUTPORT(name = "out-Link_Artifact-Id3")
16 }
```

```
17 ) public class Link_Artifact_class extends Artifact {
18 ArtifactConfig ArtiConf;
19
20    //float[] fPosValue = new float[3];
21    String[] saOutPos = new String[3];
22
23    float fVelox = 0f;
24    float fVeloy = 0f;
25    float fVeloz = 0f;
26    float fAngleY = 0f;
27
28    private boolean bflag4 = false;
29
30    // attributes
31
32    // physical properties
33    // physical properties
34    public ObsProperty L1;
35    public ObsProperty W1;
36    public ObsProperty H1;
37    public ObsProperty Shape1;
38    public ObsProperty Px1;
39    public ObsProperty Py1;
40    public ObsProperty Pz1;
41    public ObsProperty VelocityX1;
42    public ObsProperty VelocityY1;
43    public ObsProperty VelocityZ1;
44    public ObsProperty Distance1;
45    public ObsProperty Angle1;
46    public ObsProperty Sound1;
47    public ObsProperty AccelerationX1;
48    public ObsProperty AccelerationY1;
49    public ObsProperty AccelerationZ1;
50    public ObsProperty JointX1;
51    public ObsProperty JointY1;
52    public ObsProperty JointZ1;
53    public ObsProperty Mass1;
54    // attributes and physical properties initialization
55    void init(double L, double W, double H, double Mass, double AccelerationX,
56            double AccelerationY, double AccelerationZ, String Sound,
57            String Shape, double Angle, double Distance, int JointX,
58            int JointY, int JointZ, double OrientationX, double OrientationY,
59            double OrientationZ, double VelocityX, double VelocityY,
60            double VelocityZ, double Px, double Py, double Pz){
61
62          //Enter your code in this line :)
63          System.out.println("Iniciando el Linked Artifact... " + Px + " "
64             + Py + " " + Pz);
65
66          // attributes
67          // physical properties
68          defineObsProperty("L", L);
69          this.L1 = getObsProperty("L");
70
71          defineObsProperty("W", W);
72          this.W1 = getObsProperty("W");
73
74          defineObsProperty("H", H);
75          this.H1 = getObsProperty("H");
76
77          defineObsProperty("Px", Px);
78          this.Px1 = getObsProperty("Px");
79
80          defineObsProperty("Py", Py);
```

```
81          this.Py1 = getObsProperty("Py");
82
83          defineObsProperty("pz", Pz);
84          this.Pz1 = getObsProperty("pz");
85
86          defineObsProperty("VelocityX", VelocityX);
87          this.VelocityX1 = getObsProperty("VelocityX");
88
89          defineObsProperty("VelocityY", VelocityY);
90          this.VelocityY1 = getObsProperty("VelocityY");
91
92          defineObsProperty("VelocityZ", VelocityZ);
93          this.VelocityZ1 = getObsProperty("VelocityZ");
94
95          defineObsProperty("Distance", Distance);
96          this.Distance1 = getObsProperty("Distance");
97
98          defineObsProperty("Angle", Angle);
99          this.Angle1 = getObsProperty("Angle");
100
101         defineObsProperty("Sound", Sound);
102         this.Sound1 = getObsProperty("Sound");
103
104         defineObsProperty("AccelerationX", AccelerationX);
105         this.AccelerationX1 = getObsProperty("AccelerationX");
106
107         defineObsProperty("AccelerationY", AccelerationY);
108         this.AccelerationY1 = getObsProperty("AccelerationY");
109
110         defineObsProperty("AccelerationZ", AccelerationZ);
111         this.AccelerationZ1 = getObsProperty("AccelerationZ");
112
113         defineObsProperty("JointX", JointX);
114         this.JointX1 = getObsProperty("JointX");
115
116         defineObsProperty("JointY", JointY);
117         this.JointY1 = getObsProperty("JointY");
118
119         defineObsProperty("JointZ", JointZ);
120         this.JointZ1 = getObsProperty("JointZ");
121
122         defineObsProperty("Mass", Mass);
123         this.Mass1 = getObsProperty("Mass");
124
125         defineObsProperty("Shape", Shape);
126         this.Shape1 = getObsProperty("Shape");
127     }
128
129     // acciones
130     // getters & setters
131
132     // Metering
133     public void setL(float L) {
134        this.L1.updateValue(L);
135     }
136
137     public Object getL() {
138        return this.L1.getValue();
139     }
140
141     public void setW(float W) {
142        this.W1.updateValue(W);
143     }
144
```

```
145     public Object getW() {
146        return this.W1.getValue();
147     }
148
149     public void setH(float H) {
150        this.H1.updateValue(H);
151     }
152
153     public Object getH() {
154        return this.H1.getValue();
155     }
156
157     // Shape
158     public void setShape(String Shape) {
159        this.Shape1.updateValue(Shape);
160     }
161
162     public Object getShape() {
163        return this.Shape1.getValue();
164     }
165
166     // Pos
167     public void setPX(float PosX) {
168        this.Px1.updateValue(PosX);
169     }
170
171     public Object getPX() {
172        return this.Px1.getValue();
173     }
174
175     public void setPY(float PosY) {
176        this.Py1.updateValue(PosY);
177     }
178
179     public Object getPY() {
180        return this.Py1.getValue();
181     }
182
183     public void setPZ(float PosZ) {
184        this.Pz1.updateValue(PosZ);
185     }
186
187     public Object getPZ() {
188        return this.Pz1.getValue();
189     }
190
191     // Velocity
192     public void setVelocityX(float VeloX) {
193        this.VelocityX1.updateValue(VeloX);
194     }
195
196     public Object getVelocityX() {
197        return this.VelocityX1.getValue();
198     }
199
200     public void setVelocityY(float VeloY) {
201        this.VelocityY1.updateValue(VeloY);
202     }
203
204     public Object getVelocityY() {
205        return this.VelocityY1.getValue();
206     }
207
208     public void setVelocityZ(float VeloZ) {
```

```
209         this.VelocityZ1.updateValue(VeloZ);
210     }
211
212     public Object getVelocityZ() {
213         return this.VelocityZ1.getValue();
214     }
215
216     // Distan
217     public void setDistance(float Dist) {
218         this.Distance1.updateValue(Dist);
219     }
220
221     public Object getDistance() {
222         return this.Distance1.getValue();
223     }
224
225     // Angle
226     public void setAngle(float Ang) {
227         this.Angle1.updateValue(Ang);
228     }
229
230     public Object getAngle() {
231         return this.Angle1.getValue();
232     }
233
234     // Sound
235     public void setSound(String Soun) {
236         this.Sound1.updateValue(Soun);
237     }
238
239     public Object getSound() {
240         return this.Sound1.getValue();
241     }
242
243     // Aceleration
244     public void setAccelerationX1(float AccX) {
245         this.AccelerationX1.updateValue(AccX);
246     }
247
248     public Object getAccelerationX() {
249         return this.AccelerationX1.getValue();
250     }
251
252     public void setAccelerationY(float AccY) {
253         this.AccelerationY1.updateValue(AccY);
254     }
255
256     public Object getAccelerationY() {
257         return this.AccelerationY1.getValue();
258     }
259
260     public void setAccelerationZ(float AccZ) {
261         this.AccelerationZ1.updateValue(AccZ);
262     }
263
264     public Object getAccelerationZ() {
265         return this.AccelerationZ1.getValue();
266     }
267
268     // Joint
269     public void setJointX(float JointX) {
270         this.JointX1.updateValue(JointX);
271     }
272
```

```
273     public Object getJointX() {
274        return this.JointX1.getValue();
275     }
276
277     public void setAJointY(float JointY) {
278        this.JointY1.updateValue(JointY);
279     }
280
281     public Object getJointY() {
282        return this.JointY1.getValue();
283     }
284
285     public void setJointZ(float JointZ) {
286        this.JointZ1.updateValue(JointZ);
287     }
288
289     public Object getJointZ() {
290        return this.JointZ1.getValue();
291     }
292
293     // Mass
294     public void setMass(float mass) {
295        this.Mass1.updateValue(mass);
296     }
297
298     public Object getMass() {
299        return this.Mass1.getValue();
300     }
301
302     @OPERATION
303     public void stop(){
304        //while(true){}
305     }
306
307     @OPERATION
308     public void setVelocityLinkArtifact(String artifactName, String Ang, Object
            [] Value) throws OperationException {
309
310        double dAngulo = Double.parseDouble(Ang);
311        int LengthArtiName = artifactName.length();
312        String NameArti = artifactName.substring(1,LengthArtiName-1);
313
314        if (NameArti.equals("Link_Obj_Artifact")) {
315
316           // Set volocity to the Body
317           JacaLive.SetVelocityArtifacts("Link_Obj_Artifact", fVelox, fVeloz,
                   fVeloy);
318
319           //fVelox = fVelox + 0.01f;
320           //fVeloy = fVeloy + 0.01f;
321           //fVeloz = fVeloz + 0.01f;
322
323           if(fAngleY<1){
324              fAngleY = fAngleY + 0.01f;
325           }
326
327           if(fVelox>3){
328              fVelox = 0;
329              fVeloy = 0;
330              fVeloz = 0;
331              fAngleY = 0;
332           }
333           signal("isLinkArtifact", "Link_Obj_Artifact");
334
```

```
335         }
336
337     }
338
339     @OPERATION
340     void getPositionLinkArtifact(String artifactName,OpFeedbackParam<String[]>
            PosArtifact,
341                                  OpFeedbackParam<Boolean> state){
342         //ObsProperty prop = getObsProperty("pz");
343         //prop.updateValue(getPZ());
344         //signal("tick");
345         int LengthArtiName = artifactName.length();
346         String NameArti = artifactName.substring(1,LengthArtiName-1);
347
348         if (artifactName.equals("Link_Obj_Artifact")) {
349
350
351             // Update the ObserverProperty Pos
352             float[] fPosValue = JacaLive.GetPosArtifacts(artifactName);
353             int ix = (int)fPosValue[0];
354             int iy = (int)fPosValue[1];
355             int iz = (int)fPosValue[2];
356             Px1.updateValue(ix);
357             Py1.updateValue(iy);
358             Pz1.updateValue(iz);
359
360             // Update the ObserverProperty Velocity
361             float[] fVeloValues = JacaLive.GetVelocityArtifacts();
362             VelocityX1.updateValue(fVeloValues[0]);
363             VelocityY1.updateValue(fVeloValues[1]);
364             VelocityZ1.updateValue(fVeloValues[2]);
365
366             // Update Map
367             //JacaLive.UpdateMap("Link", ix, iz, iy);
368
369             // Send signal to agent JacaLive
370             //signal("movePosX", ix);
371             //signal("movePosY", iy);
372             //signal("movePosZ", iz);
373
374             //System.out.println("Link: " + fPosValue[0] + ", " + fPosValue[1] + "
                    , " + fPosValue[2]);
375
376             saOutPos[0] = String.valueOf(fPosValue[0]);
377             saOutPos[1] = String.valueOf(fPosValue[1]);
378             saOutPos[2] = String.valueOf(fPosValue[2]);
379
380             PosArtifact.set(saOutPos);
381             bflag4 = true;
382         }else {
383             bflag4 = false;
384         }
385         state.set(bflag4);
386     }
387
388 }
```

Listing B.9: Test

```
1
2
3 import cartago.*;
4 import cartago.AbstractWorkspacePoint;
5 import JacaLiveFrameWork.JacaLive;
```

```
6  import cartago.ARTIFACT_INFO;
7  import cartago.Artifact;
8  import cartago.ArtifactConfig;
9  import cartago.OPERATION;
10 import cartago.OUTPORT;
11 import cartago.ObsProperty;
12 import cartago.OpFeedbackParam;
13 import cartago.OperationException;
14
15 @ARTIFACT_INFO(
16 outports = {
17 @OUTPORT(name = "out-Unlink_Artifact-Id4")
18 }
19 ) public class Unlink_Artifact_class extends Artifact {
20 String[] saOutPos = new String[3];
21
22    //float[] fPosValue = new float[3];
23    float[] fVeloValues = new float[3];
24
25    float fAngleY = 0f;
26    float fVelox = 0f;
27    float fVeloy = 0f;
28    float fVeloz = 0f;
29    private boolean bflag4 = false;
30
31    // attributes
32
33    // physical properties
34    // physical properties
35    public ObsProperty L1;
36    public ObsProperty W1;
37    public ObsProperty H1;
38    public ObsProperty Shape1;
39    public ObsProperty Px1;
40    public ObsProperty Py1;
41    public ObsProperty Pz1;
42    public ObsProperty VelocityX1;
43    public ObsProperty VelocityY1;
44    public ObsProperty VelocityZ1;
45    public ObsProperty Distance1;
46    public ObsProperty Angle1;
47    public ObsProperty Sound1;
48    public ObsProperty AccelerationX1;
49    public ObsProperty AccelerationY1;
50    public ObsProperty AccelerationZ1;
51    public ObsProperty JointX1;
52    public ObsProperty JointY1;
53    public ObsProperty JointZ1;
54    public ObsProperty Mass1;
55    // attributes and physical properties initialization
56    void init(double L, double W, double H, double Mass, double AccelerationX,
57            double AccelerationY, double AccelerationZ, String Sound,
58            String Shape, double Angle, double Distance, int JointX,
59            int JointY, int JointZ, double OrientationX, double OrientationY,
60            double OrientationZ, double VelocityX, double VelocityY,
61            double VelocityZ, double Px, double Py, double Pz){
62
63          //Enter your code in this line :)
64         System.out.println("Iniciando el Un Linked Artifact... " + Px + " "
65             + Py + " " + Pz);
66
67         // attributes
68         // physical properties
69         defineObsProperty("L", L);
```

```
70          this.L1 = getObsProperty("L");
71
72          defineObsProperty("W", W);
73          this.W1 = getObsProperty("W");
74
75          defineObsProperty("H", H);
76          this.H1 = getObsProperty("H");
77
78          defineObsProperty("Px", Px);
79          this.Px1 = getObsProperty("Px");
80
81          defineObsProperty("Py", Py);
82          this.Py1 = getObsProperty("Py");
83
84          defineObsProperty("pz", Pz);
85          this.Pz1 = getObsProperty("pz");
86
87          defineObsProperty("VelocityX", VelocityX);
88          this.VelocityX1 = getObsProperty("VelocityX");
89
90          defineObsProperty("VelocityY", VelocityY);
91          this.VelocityY1 = getObsProperty("VelocityY");
92
93          defineObsProperty("VelocityZ", VelocityZ);
94          this.VelocityZ1 = getObsProperty("VelocityZ");
95
96          defineObsProperty("Distance", Distance);
97          this.Distance1 = getObsProperty("Distance");
98
99          defineObsProperty("Angle", Angle);
100         this.Angle1 = getObsProperty("Angle");
101
102         defineObsProperty("Sound", Sound);
103         this.Sound1 = getObsProperty("Sound");
104
105         defineObsProperty("AccelerationX", AccelerationX);
106         this.AccelerationX1 = getObsProperty("AccelerationX");
107
108         defineObsProperty("AccelerationY", AccelerationY);
109         this.AccelerationY1 = getObsProperty("AccelerationY");
110
111         defineObsProperty("AccelerationZ", AccelerationZ);
112         this.AccelerationZ1 = getObsProperty("AccelerationZ");
113
114         defineObsProperty("JointX", JointX);
115         this.JointX1 = getObsProperty("JointX");
116
117         defineObsProperty("JointY", JointY);
118         this.JointY1 = getObsProperty("JointY");
119
120         defineObsProperty("JointZ", JointZ);
121         this.JointZ1 = getObsProperty("JointZ");
122
123         defineObsProperty("Mass", Mass);
124         this.Mass1 = getObsProperty("Mass");
125
126         defineObsProperty("Shape", Shape);
127         this.Shape1 = getObsProperty("Shape");
128     }
129
130     // acciones
131     // getters & setters
132
133     // Metering
```

```
134     public void setL(float L) {
135        this.L1.updateValue(L);
136     }
137
138     public Object getL() {
139        return this.L1.getValue();
140     }
141
142     public void setW(float W) {
143        this.W1.updateValue(W);
144     }
145
146     public Object getW() {
147        return this.W1.getValue();
148     }
149
150     public void setH(float H) {
151        this.H1.updateValue(H);
152     }
153
154     public Object getH() {
155        return this.H1.getValue();
156     }
157
158     // Shape
159     public void setShape(String Shape) {
160        this.Shape1.updateValue(Shape);
161     }
162
163     public Object getShape() {
164        return this.Shape1.getValue();
165     }
166
167     // Pos
168     public void setPX(float PosX) {
169        this.Px1.updateValue(PosX);
170     }
171
172     public Object getPX() {
173        return this.Px1.getValue();
174     }
175
176     public void setPY(float PosY) {
177        this.Py1.updateValue(PosY);
178     }
179
180     public Object getPY() {
181        return this.Py1.getValue();
182     }
183
184     public void setPZ(float PosZ) {
185        this.Pz1.updateValue(PosZ);
186     }
187
188     public Object getPZ() {
189        return this.Pz1.getValue();
190     }
191
192     // Velocity
193     public void setVelocityX(float VeloX) {
194        this.VelocityX1.updateValue(VeloX);
195     }
196
197     public Object getVelocityX() {
```

```
198         return this.VelocityX1.getValue();
199     }
200 
201     public void setVelocityY(float VeloY) {
202         this.VelocityY1.updateValue(VeloY);
203     }
204 
205     public Object getVelocityY() {
206         return this.VelocityY1.getValue();
207     }
208 
209     public void setVelocityZ(float VeloZ) {
210         this.VelocityZ1.updateValue(VeloZ);
211     }
212 
213     public Object getVelocityZ() {
214         return this.VelocityZ1.getValue();
215     }
216 
217     // Distan
218     public void setDistance(float Dist) {
219         this.Distance1.updateValue(Dist);
220     }
221 
222     public Object getDistance() {
223         return this.Distance1.getValue();
224     }
225 
226     // Angle
227     public void setAngle(float Ang) {
228         this.Angle1.updateValue(Ang);
229     }
230 
231     public Object getAngle() {
232         return this.Angle1.getValue();
233     }
234 
235     // Sound
236     public void setSound(String Soun) {
237         this.Sound1.updateValue(Soun);
238     }
239 
240     public Object getSound() {
241         return this.Sound1.getValue();
242     }
243 
244     // Aceleration
245     public void setAccelerationX1(float AccX) {
246         this.AccelerationX1.updateValue(AccX);
247     }
248 
249     public Object getAccelerationX() {
250         return this.AccelerationX1.getValue();
251     }
252 
253     public void setAccelerationY(float AccY) {
254         this.AccelerationY1.updateValue(AccY);
255     }
256 
257     public Object getAccelerationY() {
258         return this.AccelerationY1.getValue();
259     }
260 
261     public void setAccelerationZ(float AccZ) {
```

```
262         this.AccelerationZ1.updateValue(AccZ);
263     }
264
265     public Object getAccelerationZ() {
266         return this.AccelerationZ1.getValue();
267     }
268
269     // Joint
270     public void setJointX(float JointX) {
271         this.JointX1.updateValue(JointX);
272     }
273
274     public Object getJointX() {
275         return this.JointX1.getValue();
276     }
277
278     public void setAJointY(float JointY) {
279         this.JointY1.updateValue(JointY);
280     }
281
282     public Object getJointY() {
283         return this.JointY1.getValue();
284     }
285
286     public void setJointZ(float JointZ) {
287         this.JointZ1.updateValue(JointZ);
288     }
289
290     public Object getJointZ() {
291         return this.JointZ1.getValue();
292     }
293
294     // Mass
295     public void setMass(float mass) {
296         this.Mass1.updateValue(mass);
297     }
298
299     public Object getMass() {
300         return this.Mass1.getValue();
301     }
302
303     @OPERATION
304     public void stop(){
305         //while(true){}
306     }
307
308         @OPERATION
309     public void setVelocityUnLinkArtifact(String UnartifactName, String Ang,
            Object[] Value) throws OperationException {
310
311         double dAngulo = Double.parseDouble(Ang);
312
313         int LengthUnArtiName = UnartifactName.length();
314         String NameUnArti = UnartifactName.substring(1,LengthUnArtiName-1);
315
316         if (NameUnArti.equals("Unlink_Wall_Artifact")) {
317
318             // Set volocity to the Body
319             JacaLive.SetVelocityArtifacts("Unlink_Wall_Artifact", fVelox, fVeloz,
                    fVeloy);
320
321             //fVelox = fVelox + 0.01f;
322             //fVeloy = fVeloy + 0.01f;
323             //fVeloz = fVeloz + 0.01f;
```

```
324
325         //if(fAngleY<1){
326             fAngleY = fAngleY + 0.01f;
327         //}
328
329         if(fVelox>3){
330             fVelox = 0;
331             fVeloy = 0;
332             fVeloz = 0;
333             fAngleY = 0;
334         }
335         signal("isUnLinkArtifact", "Unlink_Wall_Artifact");
336     }
337
338 }
339
340 @OPERATION
341 void getPositionUnLinkArtifact(String UnartifactName,OpFeedbackParam<String
        []> PosUnArtifact,
342                         OpFeedbackParam<Boolean> state){
343     //ObsProperty prop = getObsProperty("pz");
344     //prop.updateValue(getPZ());
345     //signal("tick");
346     int LengthUnArtiName = UnartifactName.length();
347     String NameUnArti = UnartifactName.substring(1,LengthUnArtiName-1);
348
349     if (UnartifactName.equals("Unlink_Wall_Artifact")) {
350
351         // Update the ObserverProperty Pos
352         float[] fPosValue = JacaLive.GetPosArtifacts(UnartifactName);
353         int ix = (int)fPosValue[0];
354         int iy = (int)fPosValue[1];
355         int iz = (int)fPosValue[2];
356         Px1.updateValue(ix);
357         Py1.updateValue(iy);
358         Pz1.updateValue(iz);
359
360         // Update the ObserverProperty Velocity
361         fVeloValues = JacaLive.GetVelocityArtifacts();
362         VelocityX1.updateValue(fVeloValues[0]);
363         VelocityY1.updateValue(fVeloValues[1]);
364         VelocityZ1.updateValue(fVeloValues[2]);
365
366         // Update Map
367         //JacaLive.UpdateMap("UnLink", ix, iz, iy);
368
369         // Send signal to agent JacaLive
370         //signal("movePosX", ix);
371         //signal("movePosY", iy);
372         //signal("movePosZ", iz);
373
374         //System.out.println("UnLink: " + fPosValue[0] + ", " + fPosValue[1] +
              ", " + fPosValue[2]);
375
376         saOutPos[0] = String.valueOf(fPosValue[0]);
377         saOutPos[1] = String.valueOf(fPosValue[1]);
378         saOutPos[2] = String.valueOf(fPosValue[2]);
379
380         PosUnArtifact.set(saOutPos);
381         bflag4 = true;
382     }else {
383         bflag4 = false;
384     }
385     state.set(bflag4);
```

```
386     }
387
388 }
```

Listing B.10: Test

```
1  import cartago.Artifact;
2  import cartago.OPERATION;
3  import cartago.OpFeedbackParam;
4  import cartago.OperationException;
5
6
7  public class MotorRobot extends Artifact {
8
9     // Var tipo int
10    int iN = 32;
11    int in = 0;
12    static int iHora = 0;
13     static int iMinutos = 0;
14     static int iSegundos = 0;
15    int iI = 0;
16    int iX = 0;
17    int iY = 0;
18    int iX2 = 0;
19    int iY2 = 0;
20    int ir1X = 0;
21    int ir1Y = 0;
22    int ir1Z = 0;
23
24    // Var tipo double
25    double dPI = 3.14159265359;
26    double dAngulo = 0;
27    double doldAngulo = 0;
28    double dKP = 8.3;
29    double dWMAX = 13.0 * dPI/9.0;
30    double doldPosX = 0;
31    double doldPosY = 0;
32    double dX1[];
33     double dY1[];
34    double dPD = -110;
35    double dSalida_A[];
36
37    // Var tipo boolean
38    boolean bflag0 = true;
39    boolean bflag1 = true;
40    boolean bflag2 = true;
41    boolean bflag3 = true;
42    boolean bflagTime = false;
43
44    // Constructores
45
46
47
48    @OPERATION
49    public void init(){
50       System.out.println("Ini Motor");
51    }
52
53    //@OPERATION
54    public double servoMotorA(double Amplitud){
55       dAngulo = Amplitud*Math.sin(2* dPI * in/iN);
56       double Minimo = Math.min(dAngulo, doldAngulo);
57       doldAngulo = dAngulo;
58       in = (in + 1) %iN;
```

```
59          //pack1.set(Double.toString(dAngulo));
60          return dAngulo;
61      }
62
63      @OPERATION
64      public void moveRobot(double x, double y, double z, OpFeedbackParam<Double>
            Angle, OpFeedbackParam<Integer> X_Pos, OpFeedbackParam<Integer> Y_Pos,
            OpFeedbackParam<Integer> Z_Pos){
65          double dAngulo = servoMotorA(50);
66          moveTowards((int)x, (int)y, (int)z);
67          X_Pos.set(50);
68          Y_Pos.set(50);
69          Z_Pos.set(10);
70          Angle.set(dAngulo);
71      }
72
73      private void moveTowards(int x, int y, int z) {
74          if (ir1X < x){
75              ir1X++;
76          }else if (ir1X > x){
77              ir1X--;
78          }
79
80          if (ir1Y < y){
81              ir1Y++;
82          }else if (ir1Y > y){
83              ir1Y--;
84          }
85
86          if(z>0){
87              if (ir1Z < z){
88                  ir1Z++;
89              }else if (ir1Z > z){
90                  ir1Z--;
91              }
92          }else{
93              ir1Z = z;
94          }
95      }
96  }
```