Descripción de la Arquitectura del Sistema

Entorno de Simulación Robótico

Anthony Figueroa

pgsimrob@fing.edu.uy

Tutor

Gonzalo Tejera

Cotutores

Gustavo Armagno, Facundo Benavides, Serrana Casella

15 de octubre de 2007

Instituto de Computación

Facultad de Ingeniería - Universidad de la República

Montevideo - Uruguay

Índice

Índice de figuras

1. Introducción

1.1. Propósito

Este documento tiene como objetivo la presentación de la arquitectura general del proyecto desarrollado en el marco de este proyecto de grado. Se utilizan diferentes vistas para presentar diversos aspectos importantes del sistema de forma tal que se reflejan diversas decisiones importantes tomadas en el diseño del mismo.

1.2. Organización

En la sección 1.3 se presenta la representación de la arquitectura utilizada en este documento, explicando las características de las diferentes vistas que lo componen. capítulos 2 a 7 se describen y explican las diferentes vistas utilizadas para describir este sistema.

1.3. Representación de la Arquitectura

La arquitectura está representada por diferentes vistas utilizando la notación UML[UML]. Estas permiten visualizar, entender y razonar los elementos más significativos de la arquitectura y a su vez identificar las áreas de riesgo que requieren mayor detalle de elaboración.

La arquitectura del sistema se descompone en las siguientes dimensiones:

- Requerimientos: funcionales y no funcionales del sistema
- Elaboración: representación lógica del sistema y representación de tiempo de ejecución
- Implementación: vista de módulos implementados, potenciales escenarios de infraestructura y distribución de los módulos

La siguiente sección detalla las vistas de la arquitectura que serán utilizadas para cubrir las dimensiones mencionadas. La sección 1.3.2 presenta el framework arquitectónico utilizado.

1.3.1. Representación

La arquitectura de este sistema está representada siguiendo las recomendaciones del RUP [RUP]. Las vistas necesarias para especificar dicho sistema se indican a continuación:

- Vista de Restricciones: Describe restricciones tecnológicas, normativas, uso de estándares, entre otros, las cuales deben ser respetadas tanto por el proceso de desarrollo como por el producto desarrollado.
- Vista de QoS: Incluye aspectos de calidad y describe los requerimientos no funcionales del sistema.

- Vista Lógica: Describe la arquitectura del sistema presentando varios niveles de refinamiento. Indica los módulos lógicos principales, sus responsabilidades y dependencias.
- Vista de Procesos: Describe los procesos concurrentes del sistema
- Vista de Implementación: Describe los componentes de distribución construidos y sus dependencias.
- Vista de Distribución: Presenta aspectos físicos como topología, infraestructura informática, e instalación de ejecutables. Incluye además plataformas y software de base.

1.3.2. Framework arquitectónico

La arquitectura sigue el framework 4+1 presentado en [KRU]. Este framework define 4 vistas para la arquitectura en conjunto con los escenarios, y se presenta en la siguiente figura.

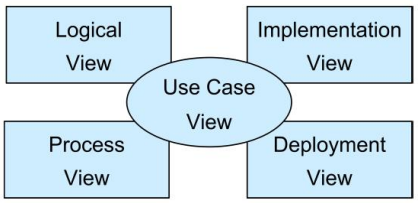

Figura 1: Framework 4+1

La correspondencia de las vistas utilizadas con las propuestas en el framework se presenta en la siguiente tabla.

Framework 4+1	Arquitectura
Vista de Casos de Uso	Vista de Restricciones y QoS
Vista Lógica	Vista Lógica
Vista de Procesos	Vista de Procesos
Vista de Implementación	Vista de Implementación
Vista de Distribución	Vista de Distribución

Figura 2: Correspondencia entre vistas

2. Vista de Restricciones

2.1. Alcance y duración del proyecto

El proyecto se desarrolla en el contexto de un proyecto de grado, por lo tanto existen restricciones intrínsecas a esta realidad. La duración del proyecto debería ser de 8 meses y la dedicación necesaria debería ser una persona con una dedicación de 15 horas semanales.

2.2. Plataforma de desarrollo

El sistema debe ejecutarse bajo cualquier distribución de Linux. Sin embargo, es deseable que las librerías que se utilicen sean portables de tal modo que el sistema sea fácilmente compilable en otros sistemas operativos sin mayores cambios.

2.3. Protocolos de comunicación

Una característica importante es que la comunicación con los agentes que se conecten al sistema se realice a través de un protocolo que sea soportado por una amplia variedad de lenguajes de comunicación. Además, el intercambio de mensajes entre el sistema y los agentes que se conectan a él, siguen una sintaxis definida en la documentación de este proyecto.

3. Vista de QoS

3.1. Performance

3.1.1. Tiempos de respuesta

No existen tiempos de respuesta previamente establecidos para que se considere correcto el funcionamiento del sistema. Sin embargo, es deseable que los mismos sean minimizados de manera tal que el funcionamiento del sistema y la simulación propiamente dicha no se vean afectados

3.1.2. Carga

Existe un límite de carga que el sistema puede soportar que depende del hardware sobre el cual se está ejecutando. Puede ocurrir que los agentes periódicamente le envíen demasiados mensajes al sistema para que este realice acciones sobre el estado del modelo físico. Esto puede llevar al problema de que que el subsistema que controla el modelo físico no logre satisfacer los pedidos a una velocidad mayor que los que estos se producen, ocurriendo un clásico problema de "productor y consumidor".

4. Vista Lógica

4.1. Arquitectura del sistema

Básicamente, se pueden identificar dos grandes componentes en el sistema: *Communication* y *Simulator*. El primero, tienen como responsabilidad manejar todo lo necesario para llevar a cabo la comunicación del sistema con los agentes que eventualmente se conecten al sistema. Esto significa que es su responsabilidad mantener toda la información necesaria referente a los agentes externos y el envío y recepción de información. El otro componente, denominado *Simulator* es el responsable de modelar la simulación que se desea crear y gestionar su evolución. A continuación se puede ver un diagrama que incluye ambos componentes.

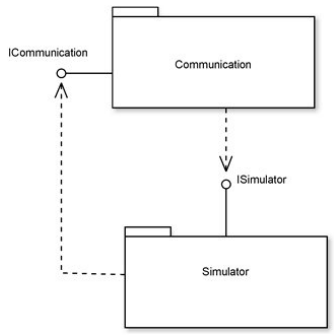

Figura 3: Arquitectura del sistema

4.2. Arquitectura Lógica

A continuación se presenta un refinamiento de la arquitectura del sistema, mostrando los subsistemas que componen su arquitectura lógica.

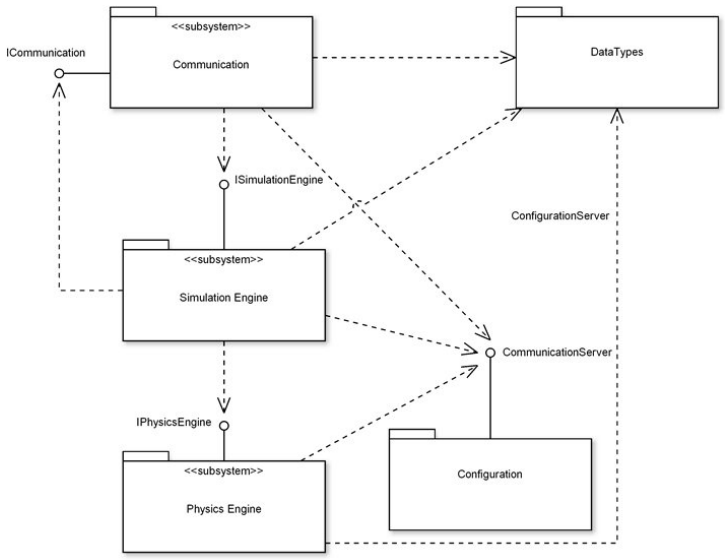

Figura 4: Arquitectura Lógica

El subsistema *Physical Engine* tiene la responsabilidad de encapsular todo el comportamiento referente al modelo físico de la simulación. El mismo expone una serie de métodos a través de su interfaz que permiten el control del modelo físico creado. A través de esta interfaz se inicializa el modelo físico, se crean y/o modifican los objetos físicos deseados, y se controla todos los demás aspectos de dicho modelo físico.

El subsistema *Simulation Engine* es el componente central del sistema, que coordina todos los demás subsistemas. Es su responsabilidad ejercer el control sobre el subsistema *Physics Engine*. Es su función administrar toda la información referente a los parámetros y modos en los cuales se está ejecutando la simulación y operar sobre el subsistema *Physics Engine* de forma coherente con dicha información. También mantiene las distintas entidades que componen al sistema y los distintos comandos que se pueden ejecutar para realizar alguna funcionalidad o realizar algún cambio en algún parámetro del sistema.

El subsistema *Communication* cumple la función de recibir y enviar información a todos los agentes que se conectan al sistema a través de este subsistema. Además es responsable de comunicarse con el subsistema *Simulation Engine* cuando esto sea necesario, para de esta manera notificarle de determinadas acciones que algún agente desea efectuar sobre la simulación. Actúa como un filtro descartando los mensajes que no deben ser procesados por el subsistema *Simulation Engine,* ya sea por estar mal formados o porque no le corresponde a dicho subsistema su procesamiento.

El subsistema *Configuration* tiene la responsabilidad de proveer una interfaz a los distintos subsistemas que componen al sistema que brinde valores para

distintos parámetros de configuración y por valores por defecto del sistema.

Todos los subsistemas ya mencionados utilizan tipos de datos definidos en el módulo *Datatypes*.

Los subsistemas *Configuration*, *Simulation Engine* y *Physics Engine* integran el componente denominado *Simulator* descripto en la arquitectura del sistema. El subsistema *Communication* tiene su correspondiente homónimo en la arquitectura del sistema.

4.3. Arquitectura de Módulos

A continuación se presenta la arquitectura de módulos del sistema, detallando y profundizando la información referente a cada uno de los módulos que componen dicho sistema.

No se presentan diagramas referentes al comportamiento dinámico de los módulos, sin embargo se explica el comportamiento esperado de cada uno de ellos y se complementa esta información con diagramas estáticos.

4.3.1. Physics Engine

Este módulo se encarga de manejar el modelo físico de la simulación. Es su responsabilidad proporcionar una interfaz para la creación, actualización y obtención de datos de distintos objetos físicos representables en el sistema, así como también proporcionar una interfaz para controlar dicho modelo físico.

El modelo físico, mantenido por este componente, esta basado en algún motor físico. Este motor físico puede ser cualquiera, desde que el componente que lo encapsule cumpla con la interfaz genérica que se propone. En este caso el motor físico utilizado es ODE y se encapsula el comportamiento referente al control del modelo realizado sobre dicha librería en un controlador que implementa la interfaz propuesta. Es importante notar que el sistema propuesto soporta los objetos físicos necesarios para crear robots y estructuras utilizables para la creación de entidades físicas que se correspondan a las necesarias para satisfacer los requerimientos presentes en el alcance del sistema, sin embargo los tipos de objetos soportados se pueden extender fácilmente, dando lugar a la creación de estructuras físicas más complejas.

A continuación se presenta un diagrama de clases correspondiente a este módulo.

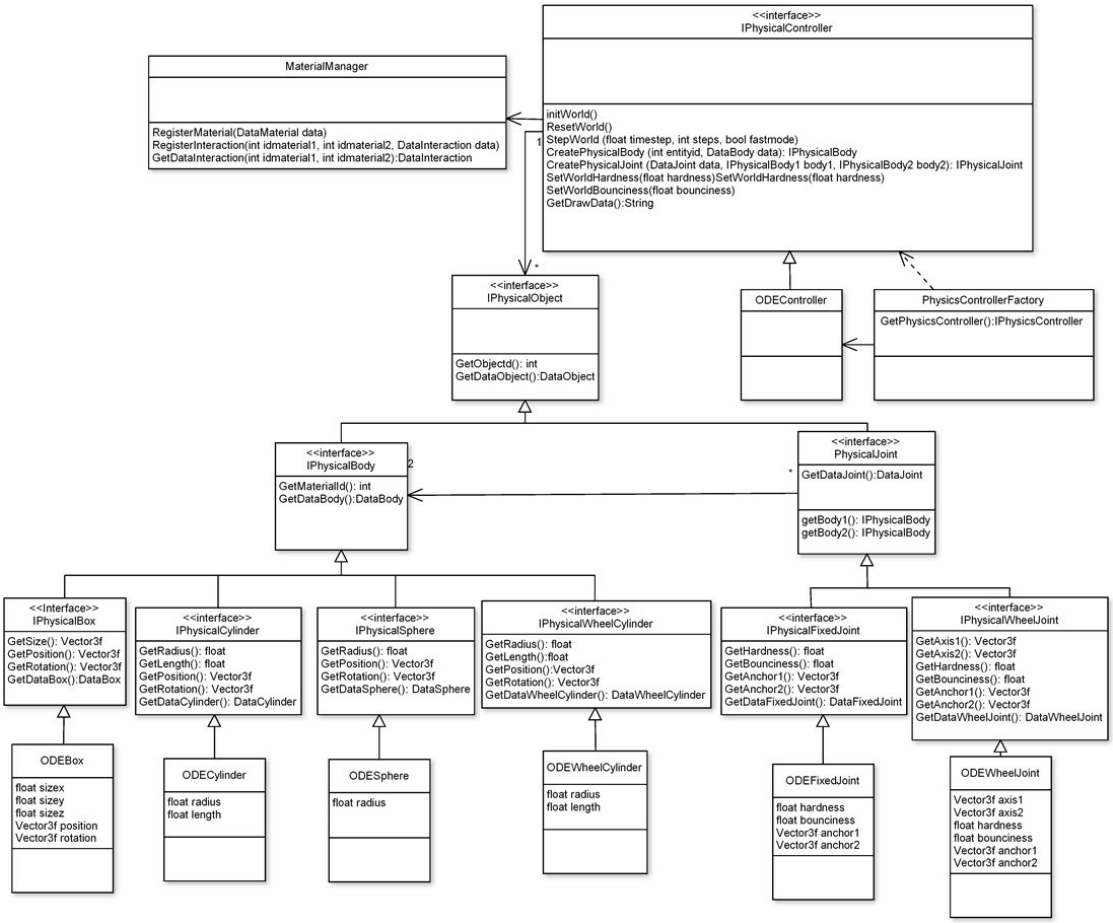

Figura 5: Módulo Physics Engine

La clase ODEController implementa la interfaz mediante la cual se accede a diversas funcionalidades provistas por este módulo. Dicha interfaz presenta operaciones necesarias para crear los diversos tipos de objetos físicos soportados. Estos objetos pueden ser cuerpos o empalmes, y a cada tipo de objeto diferente se le debe especificar un conjunto de características que definen sus aspectos físicos y comportamiento. Todos los cuerpos deben tener asignado un material y en caso de que este no sea especificado se le designará un material por defecto. Es conveniente la especificación de materiales para un flexible manejo del comportamiento de puntos de contacto. Existe un controlador de materiales, que dado dos materiales con un identificador dado, tiene la responsabilidad de informar distintos parámetros acerca del punto de contacto de dichos materiales. La información referente a los materiales presentes, sus identificadores, y

el comportamiento esperado para la interacción entre cada par de materiales debe ser registrado por algún agente conectado al sistema. Los materiales aportan facilidad de uso y flexibilidad, una correcta calibración de los mismos es importante para obtener el comportamiento deseado.

En caso que se desee modificar el motor físico subyacente, además de implementar un controlador específico para dicho motor, se deberán implementar las distintas clases que implementan las interfaces que se corresponden con los distintos objetos físicos soportados. Una vez hecho esto se podría con algunas modificaciones en el módulo, en caso de tener diversos motores físicos implementados, se podría incluso cambiar dicho motor físico en tiempo de ejecución, ya que este subsistema expone una interfaz física genérica hacia los otros subsistemas.

4.3.2. Configuration

Este módulo se encarga de registrar y mantener una serie de parámetros de configuración y personalización utilizados por el sistema. Estos parámetros son cargados desde un archivo cuando se inicia el sistema, y en caso de que un parámetro no esté referenciado en dicho archivo se le asigna un valor por defecto.

4.3.3. Simulation Engine

En este componente se concentra la inteligencia necesaria para la coordinación y sincronización de los distintos módulos del sistema y agentes ajenos al mismo. Mantiene una serie de entidades de alto nivel que corresponden a entidades de simulación, como robots, el campo de juego, etc. Cada una de estas clases encapsula cierto comportamiento específico a la misma y aporta un gran grado de flexibilidad al sistema, haciéndolo fácilmente extensible. Estas entidades son manejadas mediante la clase "EntityManager" que se encarga de su creación y acceso.

Cuando el módulo *Communication* recibe algún paquete que implique la ejecución de alguna funcionalidad relacionada a la simulación, este módulo se comunica con el módulo *Simulation Engine* enviándole un mensaje con la información necesaria. A cada ciclo de ejecución el controlador donde se ejecuta el control de la simulación verifica la existencia de nuevos mensajes del módulo de comunicación. En caso de que existan estos mensajes se decodifican y cada uno de ellos da lugar a la creación de un objeto de tipo DataRequest. Este objeto puede ser de dos tipos: "DataEntityRequest" o "DataControlRequest". Este objeto tiene asociado una clave de comando y un conjunto de objetos de tipo DataParameter. Estos objetos encapsulan datatypes que contienen información importante en la ejecución de la acción deseada.

Asociado a cada entidad de simulación existe un conjunto de clases que implementan la interfaz "ICommand". Estas clases contienen una operación cuyo

objetivo es ejecutar el código de determinado acción, obteniendo información de entrada a través de parámetros de tipo "DataParameter", o sea, cualquier tipo de información que se le desee pasar a esta operación debe tener este supertipo. Además, algunas instancias que implementan dicha interfaz no están asociadas a ninguna entidad, simplemente son comandos que se ejecutan para modificar alguna característica de la simulación, como tamaño del paso de simulación o el modo de ejecución (sincrónico o asincrónico, se explica en la sección 4.3.4). Cada entidad tiene acceso a sus distintos comandos, identificándolos a través de su clave. La clase encargada de efectuar la ejecución de los comandos asociados a entidades es el controlador denominado "EntityCommandManager", el cual obtiene la entidad involucrada, le solicita a esta el comando deseado y luego realiza la ejecución. La clase encargada de la ejecución de comandos no asociados a entidades se denomina "ControlCommandManager", que ubica los comandos a través de su clave y luego efectúa la ejecución. Esta forma de proceder se puede ver como una variante del patrón de diseño "Command".

En caso que el pedido de supertipo DataRequest, obtenido a partir de un mensaje recibido por el módulo, tenga como tipo a "DataEntityRequest", esto significa que el pedido contiene aparte de los parámetros asociados a la operación deseada y la clave de dicha operación, el identificador de una entidad de simulación. En este caso, el controlador de la simulación, que implementa la interfaz "ISimController", le delega el control al controlador "EntityCommandManager". Este controlador obtiene la instancia de entidad cuyo identificador es el deseado y a éste le solicita la instancia del comando que se desea ejecutar, luego procede a la ejecución del comando, pasándole los parámetros de supertipo DataParameter. En caso que el pedido sea de tipo "DataControlRequest", se delega el comando al controlador "ControlCommandManager" y este obtiene directamente el comando y lo ejecuta de manera análoga al caso anterior.

Casi todas las funcionalidades expuestas del sistema se traducen a un comando que la implementa. Los comandos se ejecutan en el intervalo entre ciclos de iteración, por lo que cuentan con todas las interfaces del sistema para efectuar cualquier tipo de acción con ellas, en particular los comandos de control ejecutan acciones sobre el controlador "SimulationEngineController" y sobre el controlador "IPhysicsEngineController". Los comandos de entidad además de estos controladores, efectúan acciones sobre la entidad de simulación asociadas a ellos, ya que cada comando de entidad está asociado a un objeto de tipo "ISimEntity", o sea, una entidad de simulación. Algunas ejemplos de funcionalidades que se llevan a cabo utilizando comandos son:

- Crear entidades
- Controlar motores de robots
- Guardar simulación en archivo
- Cargar simulación desde archivo
- Pasar a modo paso a paso: en este modo hay que indicar explícitamente

cada paso en la simulación que se desee realizar, ejecutando un comando específico.

- Pasar a modo normal: en este modo se avanza la simulación de manera normal.
- Mover entidades
- Cambiar la cantidad de pasos por iteración
- Cambiar el valor del parámetro de paso de tiempo
- Registrar materiales e interacciones

Es importante destacar que entre las características que se buscaron en la concepción del diseño del sistema, se priorizó la búsqueda de un diseño flexible y extensible. Resulta sencillo extender el sistema para crear nuevos tipos de entidades de simulación y/o agregarle funcionalidades al control de la simulación o de las entidades. Se podría incluso crear un nuevo tipo de entidad que sea un robot completamente genérico sin ningún tipo de restricciones estructurales más que lo que se encuentra por fuera de los límites de las funcionalidades del motor físico.

Su estructura de clases se puede ver a continuación.

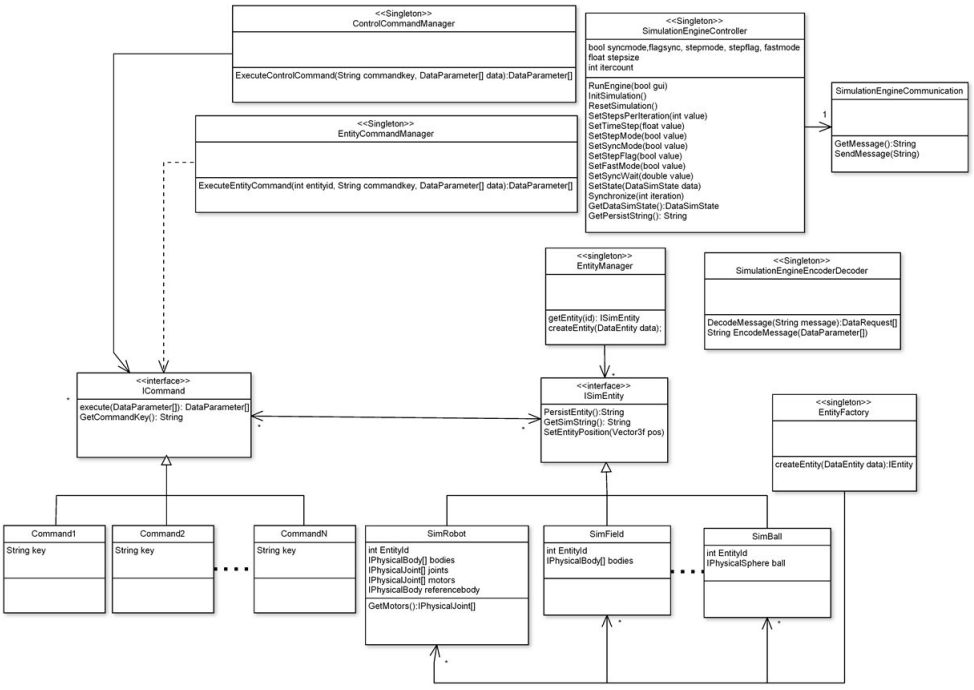

Figura 6: Módulo Simulation Engine

4.3.4. Communication

Este componente maneja todo lo referente a la comunicación del sistema con los agentes que se conectan a el. Básicamente escucha en un puerto por posibles mensajes que le puedan llegar, una vez que esto sucede, el mensaje se decodifica y si es necesario se ejecuta la operación correspondiente en el controlador, que a su vez realiza las acción es necesarias y en algunos casos envía mensajes a otros componentes o agentes. Se encarga de administrar la información referente a los distintos agentes conectados al sistema. Para esto cada vez que se registra un agente al sistema se crea la clase correspondiente al tipo de dicho agente, que implementa la interfaz IAgent. Cada tipo de agente tiene asociado una clase que implementa la interfaz IFilter. Dicha clase recibe información global del estado de la simulación y selecciona que información se le debe enviar a ese agente cada vez que se le va a enviar información acerca de la simulación.

En cada ciclo de ejecución de la simulación se debe hacer broadcast de la información referente al estado de la simulación. El componente de comunicación es notificado de esto por el componente *Simulation Engine* y en este momento a cada agente conectado al sistema, filtra la información a mandar a dicho agente utilizando su filtro asociado.

Existen dos modos de conexión de agentes: sincrónico y asincrónico. En la conexión asincrónica, se envía información a los agentes externos en cada ciclo de simulación y no se espera una respuesta de manera sincrónica. Para los agentes que se registran para comunicación en modo sincrónico, el sistema hace broadcast del estado de la simulación en cada ciclo de ejecución de la misma y espera por cierto tiempo la respuesta de todos los agentes que se disponen a conectarse de forma sincrónica. En caso de recibir todas las respuestas esperadas o que se agote el tiempo de espera dispuesto de antemano, se envía al módulo *Simulation Engine* un mensaje que lo notifica que se puede continuar la ejecución de la simulación. También se puede configurar el sincronismo en el contexto del controlador de tal modo que no se permita conexiones sincrónicas con agentes.

Se pueden recibir mensajes tanto de agentes como del componente *Simulation Engine.* Este módulo puede enviar mensajes o bien para especificar opciones de configuración o también para notificar del avance de un nuevo paso de simulación y comenzar el proceso de broadcast. Los agentes pueden enviar mensajes por diversos motivos, entre los cuales están registrarse en el sistema, pedir información acerca de otro agente conectado o realizar acciones sobre las entidades de simulación, como actualizar posiciones, rotaciones y motores.

A continuación se muestra un diagrama de clases correspondiente a este módulo.

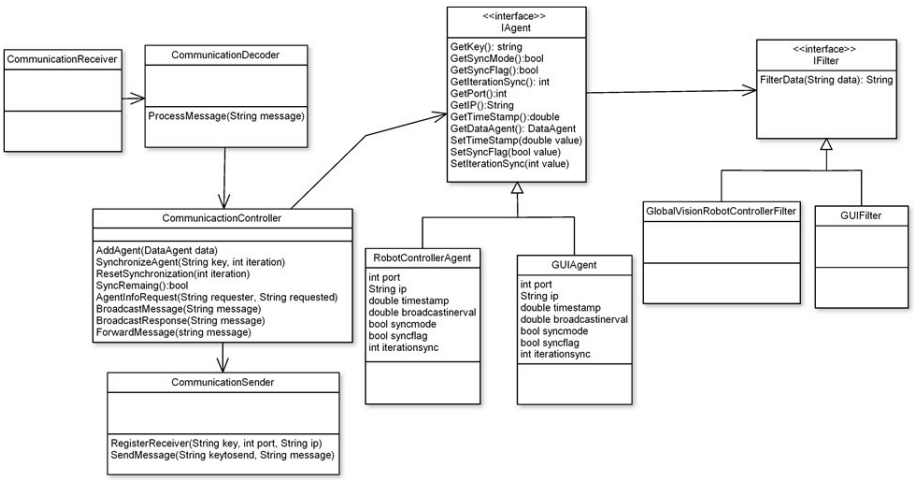

Figura 7: Módulo Communication

4.3.5. Datatypes

Este módulo contiene tipos de datos que son utilizados0 por todos los demás módulos del sistema. Los tipos de datos pertenecientes a este módulo son los siguientes:

- DataParameter: es una superclase utilizada como parámetro pasados a comandos.
- Vector3f: contiene tres atributos de tipo float. Es de tipo DataParameter.
- DataQuaternion: contiene cuatro atributos de tipo float. Es de tipo DataParameter.
- DataFloat: encapsula un dato de tipo float. Es de tipo DataParameter.
- DataInteger: encapsula un dato de tipo integer. Es de tipo DataParameter.
- DataDouble: encapsula un dato de tipo double. Es de tipo DataParameter.
- DataString: encapsula un dato de tipo String. Es de tipo DataParameter.
- DataBool: encapsula un dato de tipo booleano. Es de tipo DataParameter.
- DataMaterial: contiene información asociada a un material registrado en el sistema. Es de tipo DataParameter.
- DataMaterialInteraction: contiene información asociada a una interacción entre materiales registrados en el sistema. Es de tipo DataParameter.

- DataSimState: contiene diversos datos que se corresponden a diversos parámetros del estado del simulador. Es de tipo DataParameter.
- DataRequest: es un tipo de datos utilizado para modelar una solicitud de acción en el módulo *Simulation Engine*. Puede ser de tipo DataEntityRequest o DataControlRequest. Además contiene asociada la información de si requiere respuesta o no.
- DataEntityRequest: es de tipo DataRequest, se utiliza cuando se desea efectuar alguna acción asociada a una entidad de la simulación.
- DataControlRequest: es de tipo DataRequest, se utiliza cuando se desea efectuar alguna acción sobre los parámetros de la simulación.
- DataEntity: contiene datos sobre una entidad de simulación. Es una generalización de tipos de datos como DataRobot, DataBall y DataField.
- DataRobot: contiene información referente a una entidad de tipo robot. Entre otros datos contiene su identificador, posición y rotación. Es de tipo DataEntity.
- DataBall: contiene información acerca de una entidad de tipo pelota. Es de tipo DataEntity.
- DataField: contiene información acerca de una entidad del tipo que se asocia al campo de juego. Es de tipo DataEntity.
- DataAgent: contiene información acerca de un agente conectado o que desea conectarse al sistema.
- DataGUIAgent: contiene información acerca de un agente de tipo interfaz gráfica conectada al sistema.
- DataRobotControllerAgent: contiene información acerca de un agente de tipo controlador de robots conectada al sistema.
- DataObject: contiene datos acerca de un objeto de simulación. Puede ser de tipo DataBody o DataJoint.
- DataBody: contiene datos acerca de un cuerpo del modelo físico de la simulación. Es de tipo DataBody.
- DataJoint: contiene datos acerca de un empalme del modelo físico de la simulación. Es de tipo DataJoint.
- DataSphere, DataCylinder, DataWheelCylinder, DataBox: contienen datos acerca de un cuerpo determinado del modelo físico de la simulación. Son de tipo DataBody.
- DataFixedJoint, DataWheelJoint: contienen datos acerca de un empalme determinado del modelo físico de la simulación. Son de tipo DataJoint.

A continuación se pueden ver los distintos datatypes en un diagrama de clases.

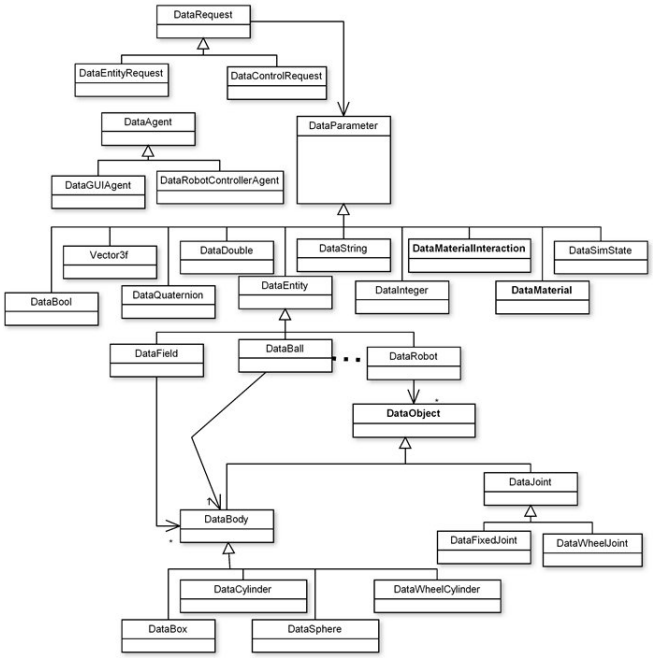

Figura 8: Módulo DataTypes

5. Vista de Procesos

El sistema debe comunicarse con procesos externos que constituyen los agentes que se conectan a él. Esto puede implicar una fuerte carga de mensajes que se reciben y se envían, por lo que esto puede consumir mucho tiempo de ejecución. Muchos de los mensajes que llegan al sistema desde agentes tienen como objetivo realizar una modificación en el modelo físico, pero otros no, como por ejemplo un pedido de información acerca de otro agente conectado. Por esta razón el filtrado de estos mensajes y la decodificación de los mismos no debe interrumpir la ejecución de la simulación propiamente dicha.

Por lo tanto, es de vital importancia que el sistema se ejecute en múltiples hilos para lograr independizar tareas de comunicación y de seguimiento de la simulación. Estos hilos se ejecutan concurrentemente compartiendo el mismo espacio de memoria.

El proceso principal está compuesto por dos hilos: Communication y Engine. El hilo Communication ejecuta el módulo lógico homónimo, encargándose por lo tanto de todas las tareas de comunicación con agentes externos. El hilo Engine recibe información del hilo Communication cuando se debe realizar alguna modificación en la simulación como resultado de la llegada de un mensaje enviado por un agente externo. El hilo Communication por su parte, recibe información del hilo Engine cuando es notificado de algún evento relevante para él o para algún agente externo que requiera notificación. Los demás procesos que actúan como agentes del sistema multiagente propuesto se comunican con el sistema únicamente a través del hilo Communication.

En la siguiente figura se puede ver un diagrama representando la vista de procesos del sistema.

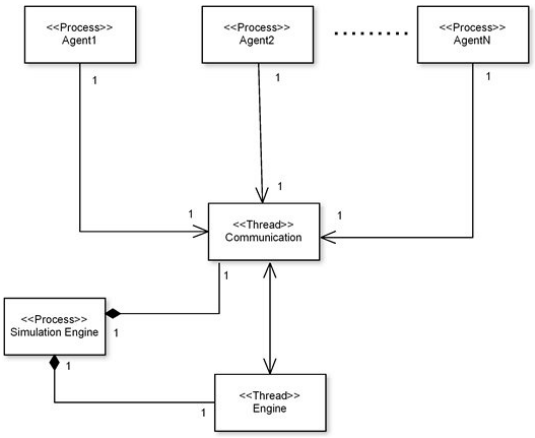

Figura 9: Vista de procesos

6. Vista de Implementación

La vista de implementación muestra los distintos componentes de distribución construidos para el sistema. Todos los componentes presentados en la arquitectura lógica están implementados en un único componente de distribución. Este componente hace uso de un archivo de configuración con algunos parámetros importantes en el sistema.

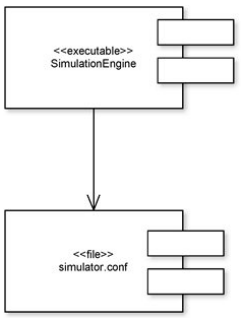

Figura 10: Vista de implementación

7. Vista de Distribución

La vista de distribución presenta la infraestructura necesaria para instalar el sistema. En esta sección se presenta la infraestructura tecnológica esperada y como se localizan en ella los componentes de distribución del sistema.

Considerando la distribución de la aplicación desde el punto de vista de los procesos, es posible identificar dos tipos de nodos: los agentes externos que se comunican al sistema y el nodo Simulation Engine. La comunicación entre estos dos tipos de nodos se realiza mediante el protocolo UDP, lo que implica flexibilidad en la distribución de los nodos. Es posible distribuir todos los nodos en una misma máquina o que cada nodo, o sea, el nodo Simulator Engine y cada uno de los agentes externos, se distribuyan en una máquina distinta.

Esta distribución es mostrada en la siguiente figura.

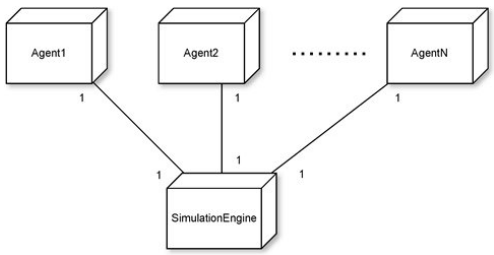

Figura 11: Vista de distribución

Referencias

[UML] Unified Modeling Language. Disponible en: http://www.omg.org/uml, Octubre 2007

[RUP] IBM Rational Unified Process. Disponible en: http://www.rational.com/rup, Octubre 2007

[KRU] Kruchten, P. "The "4+1" View Model of Software Architecture". Rational Software Corp. IEEE Software12(6), November 1995.

[RAF07] Anthony Figueroa, Requerimientos y Planificación, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2007.

[AF07] Anthony Figueroa, Estado del Arte, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2007.

[CAN] Canales, R.; Casella, S.; Rodríguez, P. "Proyecto Forrest Liga de Simulación 2D RoboCup - Descripción de la Arquitectura", Instituto de Computación, Facultad de Ingeniería, Uruguay, Junio 2005.

[ABR] Gustavo Armagno, Facundo Benavides, and Claudia Rostagnol, "Descripción de la Arquitectura del Sistema" Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República ,Uruguay, 2005.