Proyecto Forrest
Liga de Simulación 2D
RoboCup

Descripción de la Arquitectura

Versión 5.0

Histórico de Revisiones

Versión	Fecha	Resumen de cambios	Autor
1.0	01/03/2005	Versión Inicial	Grupo *Forrest*
2.0	05/04/2006	Se agrega la vista de procesos	Grupo *Forrest*
3.0	10/05/2006	Se agregan las vistas faltantes	Grupo *Forrest*
4.0	15/06/2006	Se completan las vistas Lógica, de procesos y de distribución	Grupo *Forrest*
5.0	01/08/2006	Se completan las secciones faltantes	Grupo *Forrest*

Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Índice

1. Introducción

El sistema *Forrest* es un equipo de la liga de simulación 2D de la *RoboCup* [LSIM]. Es un sistema de tiempo real que opera en un entorno dinámico y no determinista a través de un simulador de juego. Cuenta con 12 agentes de software independientes (once jugadores y un entrenador) con la capacidad de interactuar entre sí con el objetivo de ganarle al equipo contrario [CANd].

El presente documento tiene como propósito brindar una visión comprensible de la arquitectura del sistema *Forrest*. Para esto, se utilizan diferentes vistas en donde se ilustran los aspectos más relevantes del sistema y se capturan las decisiones más importantes que fueron tomadas con respecto a su arquitectura. Este documento profundiza principalmente en la vistas lógica y de procesos, incluyendo algunos elementos significativos de otras vistas.

El documento está organizado de la siguiente forma: en el capítulo 2 se presenta la Representación de la Arquitectura utilizada para el sistema y en los capítulos restantes se presentan la descripción de las vistas indicadas en el capítulo 2.

2. 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.

El sistema *Forrest* es considerado un sistema de tiempo real. Esta característica hace que la vista lógica y la vista de procesos sean las más relevantes.

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 2.2 presenta el framework arquitectónico utilizado.

2.1. Representación

La arquitectura del sistema está representada siguiendo las recomendaciones del RUP [RUP]. Las vistas necesarias para especificar al sistema son:

- *Vista de Restricciones*: Describe restricciones tecnológicas, normativas y uso de estándares, las cuales deben ser respetadas tanto por el proceso de desarrollo como por el producto desarrollado.
- *Vista 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.

Dadas las características de este sistema, las vista de Casos de Uso y la vista de Datos no fueron tomadas en cuenta ya que no son relevantes para la representación de la arquitectura.

2.2. 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 Fig. 1.

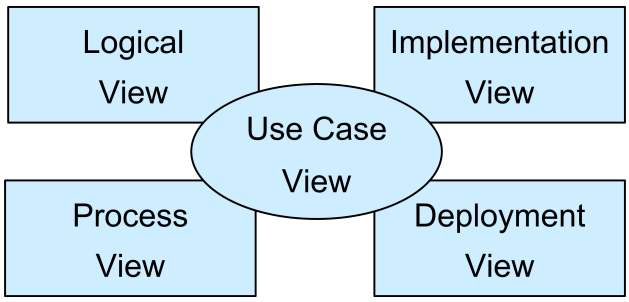

Figura 1: Framework 4+1.

La correspondencia de las vistas utilizadas con las propuestas en el framework se presenta en la tabla 1.

Framework 4+1	**Arquitectura**
Use Case View	Vista de Restricciones y QoS
Logical View	Vista Lógica
Process View	Vista de Procesos
Implementation View	Vista de Implementación
Deployment View	Vista de Distribución

Cuadro 1: Correspondencias entre vistas.

3. Vista de Restricciones

En esta vista se presentan las restricciones normativas y tecnológicas a las cuales está sujeto el producto desarrollado.

3.1. Normativas

Alcance y Duración del Proyecto

Dado que el sistema es desarrollado en el marco de un proyecto de grado, se cuentan con varias restricciones que deben ser tomadas en cuenta en el desarrollo del producto. Dos de las principales son:

- Duración del proyecto: 8 meses, con una posible extensión de 4 meses.
- Recursos Humanos: 3 recursos, 15 horas semanales c/u.

3.2. Plataforma de Desarrollo

Sistema Operativo

El sistema debe ejecutarse bajo cualquier distribución del sistema operativo Linux. En particular, en la última competencia oficial de la liga (año 2005), se utilizó la distribución *Gentoo*.

Lenguaje de Programación

En principio el sistema puede estar desarrollado en cualquier lenguaje de programación que pueda ejecutarse bajo el sistema operativo mencionado en el punto anterior.

Sistemas Existentes

El sistema *Forrest* debe interactuar con el simulador de juego oficial *Soccer Server* [CHE]. La versión a utilizar de este sistema es la 10.x.

Protocolo de comunicación

La comunicación entre los sistemas *Forrest* y *Soccer Server* se realiza a través del protocolo de comunicación UDP/IP. Además, para el envío y recepción de comandos se debe utilizar el protocolo de comandos definido en [CHE].

Características de los equipos informáticos a utilizar

Bajo las condiciones de una competencia oficial[1], el sistema *Forrest* contará con 3 PCs para distribuir la carga de sus 12 procesos. Cada uno de estos equipos cuentan con un procesador Xeon 3.4Ghz y 1Gb de memoria. El simulador de juego se ejecuta en un equipo distinto a estos 3 PCs.

Durante la duración del proyecto se utilizará un único equipo P4 con 512 MB de memoria RAM y 2.8 GHz de velocidad de procesador, utilizando la plataforma Linux Susse 9.2.

3.3. Diseño

Utilización del Framework UvA Trilearn 2003

Durante la etapa de análisis del proyecto se tomó la decisión de utilizar la distribución básica del equipo *UvA Trilearn 2003* [CANb], [DEB]. El mismo ofrece la implementación de varios módulos necesarios para el desarrollo de un equipo de esta liga. La distribución ya ofrece una arquitectura base la cual debe ser tomada en cuenta en el desarrollo del sistema.

[1] Datos relevados hasta la competencia de Osaka 2005.

4. Vista de QoS

4.1. Procesos del sistema

Una de las restricciones más importantes impuestas por el simulador de juego es referido a los procesos con los que debe contar el sistema *Forrest*. Cada uno de los 11 jugadores del equipo y el *Coach* deben ser procesos del sistema independientes, donde bajo ningún concepto pueden comunicarse directamente entre ellos. La única forma permitida de comunicación es a través del simulador de juego, el cual cuenta con varias restricciones acerca de la comunicación entre ellos agentes [CHE].

4.2. Performance

Tiempo de respuesta

El simulador de juego, en condiciones de competencia, avanza con intervalos de tiempo discretos de 100 ms de duración llamados ciclos. Al termino de cada ciclo el simulador actualiza el estado del juego con las acciones que le fueron enviadas por los jugadores de ambos equipos. Para que la acción de un jugador sea ejecutada en el ciclo *t*, debe llegar al simulador durante el transcurso del ciclo *t-1*. Si la acción llegara en otro ciclo, podría ser descartada o ejecutada sobre un estado de juego distinto al originalmente pensado. En dicho caso el resultado de aplicarla es impredecible.

Por lo dicho anteriormente, se requiere que el proceso de toma de decisiones de un agente no demore más de 100 ms.

Carga

En el contexto donde se desarrolla este trabajo, únicamente se cuenta con una maquina en donde se deberán ejecutar los 26 procesos de un partido de forma concurrente (dos equipos, el simulador y el monitor de visualización). Por este motivo, durante el diseño de los agentes es necesario controlar el uso de los recursos del sistema, como CPU y memoria, para evitar posibles degradaciones en la performance del equipo.

5. Vista Lógica

Esta vista presenta tres niveles de arquitectura para el sistema. Cada nivel corresponde a un refinamiento del nivel anterior. La organización de la vista lógica se presenta en la Fig. 2.

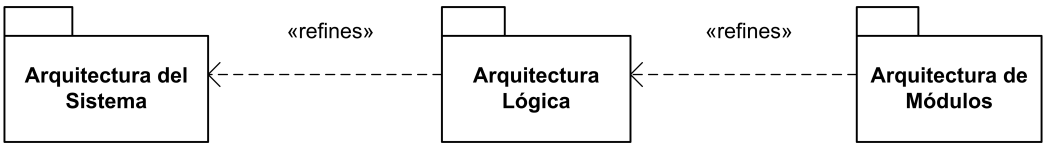

Figura 2: Organización de la vista lógica.

Al momento de definir la arquitectura del sistema *Forrest* existen varios aspectos que son influyentes en la obtención de la solución encontrada. Algunos de estos son:

- *Jugadores autónomos*: dadas las restricciones impuestas por el simulador de juego, se requiere que cada jugador del equipo sea implementado con un proceso independiente que perciba y actue en su entorno únicamente por medio del simulador de juego.

- *Objetivo en común*: dadas las características del juego (fútbol) los 11 jugadores del equipo tienen un objetivo en común: *tratar de ganar el partido*.
- *Cooperación*: para que el equipo cumpla con su objetivo se requiere que los jugadores cooperen entre si.

Estos aspectos, entre otros, hacen que el equipo *Forrest* sea considerado un sistema multi-agente en donde cada jugador es un agente autónomo.

Al ser los 11 agentes jugadores del equipo de idénticas características, la arquitectura del sistema presentada en esta sección se corresponde con la arquitectura de un agente jugador. La participación del agente *Coach* en el equipo a quedado por fuera del alcance del proyecto por lo que su arquitectura no es considerada en este documento.

5.1. Arquitectura del Sistema

Como se mencionó en la sección 3.3, el sistema *Forrest* utiliza la distribución del equipo *UvA Trilearn* como base para la construcción de sus agentes. Esta decisión hace que los agentes adopten y utilicen su arquitectura.

La Fig. 3 muestra el estilo de arquitectura utilizada. La misma se encuentra organizada siguiendo el patrón de *arquitectura en capas verticales*. Esta descomposición permite dividir al sistema en progresivos niveles de abstracción en donde cada uno de ellos resuelve distintos problemas. Se le denomina *vertical* debido a que existe una única capa que tiene acceso al entorno del agente (sensores y actuadores). En lo que resta de la sección se describe brevemente el rol de cada una de las capas.

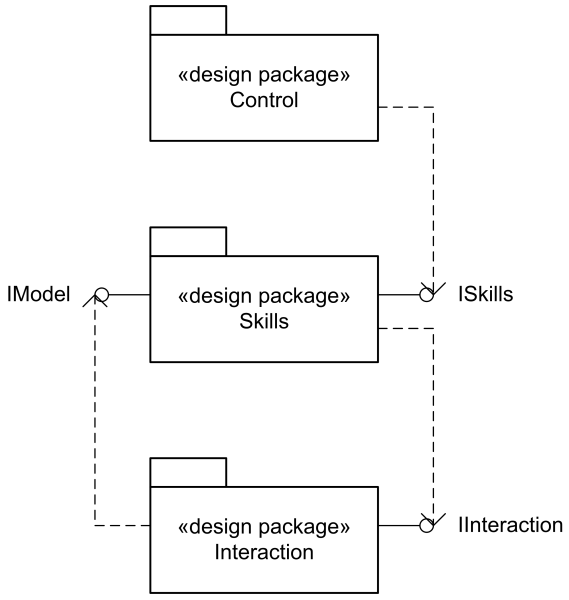

Figura 3: Arquitectura del sistema *Forrest*.

La capa *Interaction* es la encargada de resolver la comunicación con el simulador del juego, aislando el resto de las capas de los detalles de percibir y actuar en el entorno del agente. Para lograr una comunicación exitosa se debe utilizar el protocolo de comunicación del sistema *Soccer Server* [CHE].

La capa Skills es la encargada de modelar el estado del juego y de implementar las habilidades básicas que poseen los agentes.

La capa *Control* es la encargada de resolver el comportamiento de alto nivel del agente. Extrae del modelado de la realidad el estado actual del juego y mediante algún mecanismo de decisión (estrategia) elige la mejor acción a ejecutar en cada ciclo de simulación.

5.2. Arquitectura Lógica

En esta sección se presenta un refinamiento de la arquitectura del sistema. En la Fig. 4 se muestran los distintos módulos que la componen y como interactúan entre ellos.

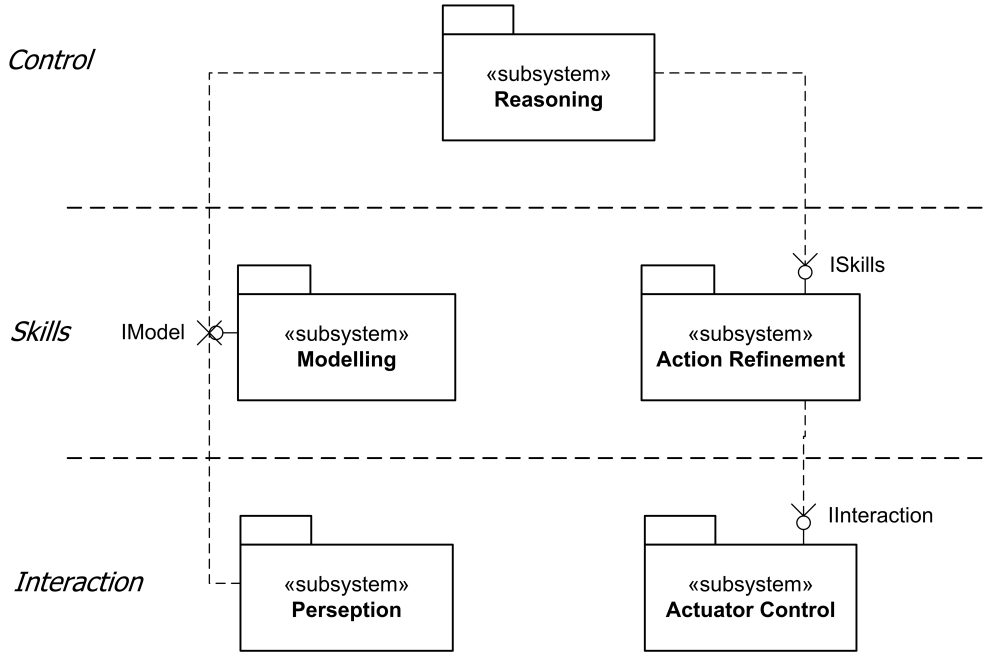

Figura 4: Arquitectura Lógica.

Los modulos *Perseption*, *Actuator Control*, *Modelling* y *Action Refinement* son provistos por el framework. En esta sección presenta una breve descripción de las responsabilidades de cada uno, pudiéndose encontrar información más detallada en [CANb], [DEB].

Perseption. Este módulo se encarga de manejar todos los mensajes recibidos del *Soccer Server.* Su principal cometido es parsear los mensajes recibidos y actualizar el modelo de la realidad.

Modelling. Este módulo se encarga de representar el entorno donde se desempeñan los agentes del sistema *Forrest*. Se debe mantener actualizada la ubicación de los 22 jugadores y la pelota dentro del campo de juego. Toma como entrada la información procesada por el módulo Perseption.

Reasoning. Este módulo se encarga de la toma de decisiones del agente, eligiendo la mejor acción dada una determinada situación. Obtiene del módulo *Modelling* la información relevante para establecer el estado actual del juego y en base a su estrategia decide la mejor acción a ejecutar en ese momento. Una vez tomada esta decisión se comunica con el módulo Action Refinement para ejecutar la acción.

Action Refinement. Este módulo se encarga de implementar todas las habilidades básicas de los agentes (dribble, pasar, patear, despejar, etc.). Según el tipo y configuración de la habilidad, el módulo debe realizar los cálculos necesarios y armar el comando que será enviado al simulador según el protocolo establecido.

Actuator Control. Este módulo se encarga de manejar todos los comandos que son enviados al *Soccer Server*. Recibe del módulo *Action Refinement* el comando a ejecutar y lo envía al simulador.

En [CANc] se presentó un estudio detallado acerca de los principales aspectos que deben ser tomados en cuenta en el desarrollo de un equipo de fútbol de la liga de simulación. A continuación se detallan cada uno de estos aspectos, explicando como son contemplados en la arquitectura planteada.

- *Comunicación con el entorno*: La comunicación con el sistema *Soccer Server* es resuelta en la capa *Interaction* por los módulos *Perseption* y *Actuator Control*.
- *Modelado de la realidad*: El módulo *Modelling* de la capa *Skills* resuelve todos los aspectos relacionados con el modelado de la realidad. Mantiene actualizado la posición y velocidad de todos los objetos del campo de juego y ofrece soporte al módulo de razonamiento para determinar el comportamiento del agente.
- *Sincronización*: Este problema es resuelto por los módulos *Perseption* y *Actuator Control.* El módulo *Actuator Control* es el encargado de almacenar los mensajes (comandos) que deben ser enviados al simulador. El módulo *Perseption* es encargado de detectar el comienzo y fin de cada ciclo. Habilitando y deshabilitando el envío de mensajes por parte del módulo *Actuator Control*.
- *Estrategia del equipo*: El comportamiento de alto nivel del agente es resuelto en la capa *Control* por el módulo *Reasoning.*
- *Habilidades*: Los comportamientos básicos de los jugadores son resueltos en la capa *Skills* por el módulo *Action Refinement*.

5.3. Arquitectura de Módulos

En esta sección se presenta la arquitectura de módulos del sistema *Forrest*. De todos los módulos del sistema, esta sección se enfoca en el módulo *Reasoning* por ser donde se diseña e implementa la estrategia del equipo.

Estrategia del equipo Forrest

La estrategia del equipo *Forrest* se basa en la combinación de las técnicas *grafos de coordinación* y *planificación en línea de dos niveles*.

Los *grafos de coordinación* es una técnica para la toma de decisiones en sistemas multi-agentes cooperativos, donde debe cumplirse que la función de utilidad global usada pueda ser escrita como una combinación lineal de utilidades locales. De esta manera el método determina la mejor acción que cada agente puede ejecutar en un ciclo, tal que en conjunto maximizan la utilidad global del equipo.

La técnica de *planeación en línea de dos niveles* plantea un método para la toma de decisiones de un agente, donde la evaluación de una acción no solo depende del estado actual del juego, sino también del estado futuro. En particular de la cantidad y calidad de las acciones a futuro que la decisión actual genere.

La estrategia del equipo *Forrest* combina ambas técnicas haciendo que la evaluación de la utilidad local de cada acción, necesaria para la resolución de los grafos de coordinación, sea definida utilizando el método de planeación en línea de dos niveles [CANb].

Diagrama de clases del módulo Reasoning

En la Fig. 5 se muestra el diagrama de clases del módulo *Reasoning.*

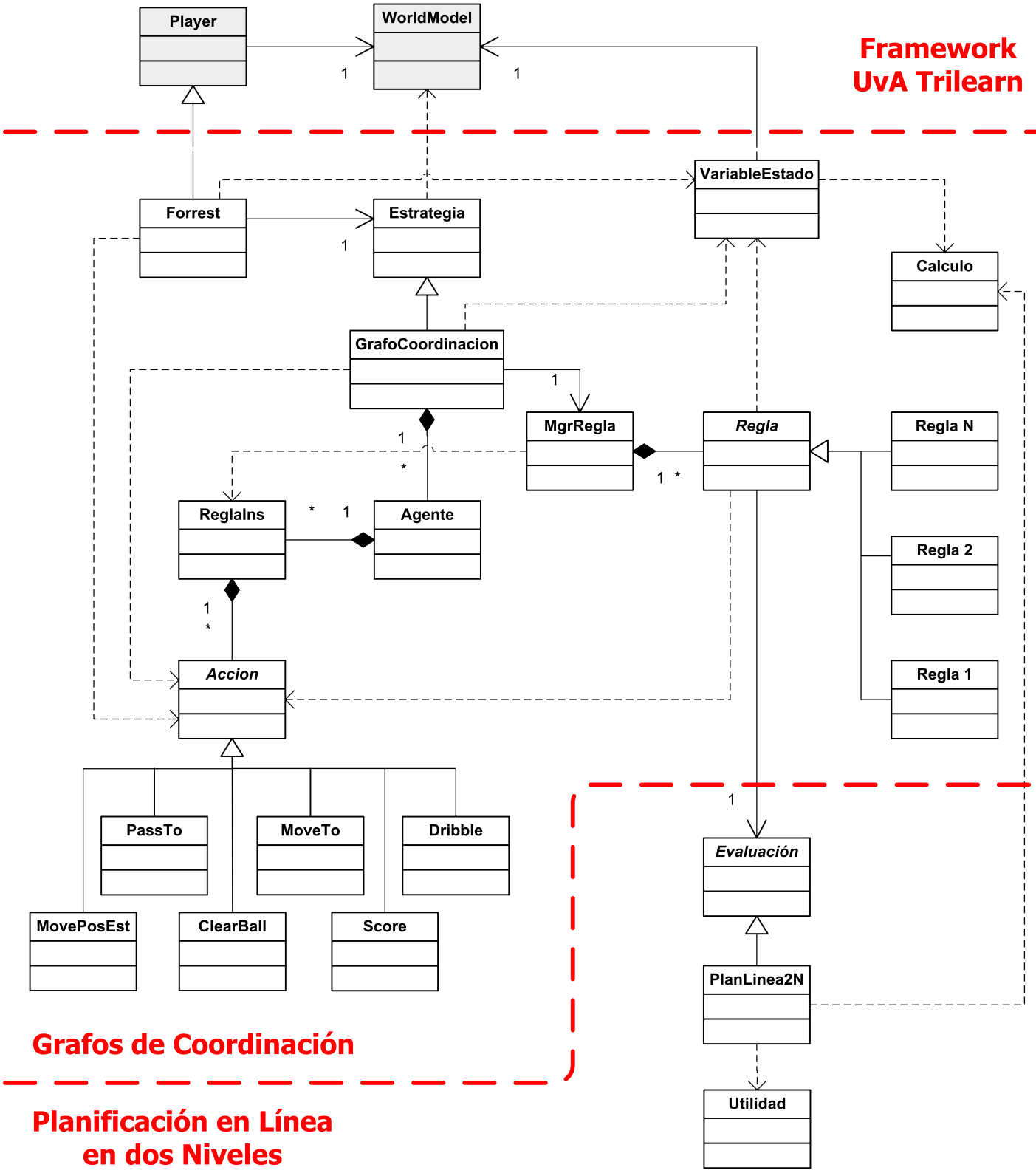

Figura 5: Diagrama de clases del módulo Reasoning.

Player. Representa al agente y contiene únicamente sus habilidades básicas: pasar la pelota, avanzar, marcar, despejar, etc. Esta clase es provista por el framework con el objetivo de que sea extendida para

la construcción de un nuevo agente.

WorldModel. Esta clase es provista por el framework y permite modelar el entorno donde opera el agente. Contiene información sobre el estado de todos los objetos del campo de juego: agentes, pelota y marcas de referencia. Esta información es utilizada por el agente para la toma de decisiones.

Forrest. Es una especialización de la clase Player. Representa al agente *Forrest* e implementa el flujo principal de toma de decisiones. Es responsable de instanciar la estrategia del equipo y comunicar la acción elegida al simulador en cada ciclo.

Estrategia. Es una clase abstracta y tiene como objetivo definir la estrategia del equipo *Forrest.* El método *estrategia* (invocado desde la clase Forrest) recibe una referencia del modelado de mundo y devuelve la mejor acción a ejecutar.

GrafosCoordinación. Es una especialización de la clase *Estrategia* e implementa el proceso principal de la técnica grafos de coordinación.

Agente. Esta clase contiene la información relevante del agente para la resolución del grafo de coordinación: rol del agente, colección de agentes padres, reglas instanciadas, acción óptima a ejecutar.

Regla. Es una clase abstracta que representa a cada una de las reglas utilizadas en la resolución del grafo de coordinación. Cada regla que se desee incluir debe ser una especialización de esta clase, en donde se especifiquen las variables de estado que se deben cumplir y las acciones que deben ejecutar los agentes involucrados. En la estrategia del equipo *Forrest* se definieron las reglas: Regla1, Regla2, Regla3, Regla4, Regla5, Regla6, Regla7, Regla8, Regla9, Regla10, Regla11 [CANb].

ReglaIns. Modela una regla instanciada. Se almacena la colección de agentes involucrados junto con sus acciones y el valor de evaluación de la regla según el estado del juego actual.

MgrRegla. Esta clase maneja la colección de reglas que son utilizadas para la resolución del grafo. Dado el agente y el estado del mundo les solicita a cada regla (clase Regla) que se instancie, devolviendo una colección de reglas instanciadas (clase ReglaIns).

Accion. Es una clase abstracta que permite modelar las acciones que pueden realizar los agentes *Forrest*. Cada acción debe ser una especialización de esta clase, en donde se definan los atributos necesarios para su ejecución. Los agentes consideran las siguientes acciones: *Intercept, PassTo, Score, MoveTo, MovePosEst, ClearBall* y *Dribble.*

VariableEstado. Modela las distintas variables de estado utilizadas para instanciar las reglas del grafo de coordinación.

Evaluacion. Esta clase es abstracta y define el método de evaluación de acciones utilizado por los grafos de coordinación.

PlanLinea2N. Es una especialización de la clase *Evaluacion* que implementa el método de evaluación de acciones a través de la técnica de planificación en línea de dos niveles.

Utilidad. Implementa la función de utilidad del equipo *Forrest*. Determina la utilidad que obtiene un jugador al estar posicionado en un punto *p* con la pelota en su poder. Este valor es utilizado para determinar la evaluación de una acción.

Calculo. Esta clase implementa métodos auxiliares que son utilizados por las clases mencionadas anteriormente.

6. Vista de Procesos

En el caso del fútbol de robots, existen tres tareas que debe realizar un agente: procesar la información de los sensores, razonar y actuar. Los equipos de esta liga son sistemas de tiempo real que cuentan con un tiempo finito para determinar su accionar (menos de 100 ms). El ejecutar estas tareas de forma secuencial genera un impacto negativo en la performance del equipo, principalmente por las demoras ocasionadas en las operaciones de I/O del sistema. Por este motivo, es necesario que el agente efectúe el razonamiento de alto nivel independientemente de las tareas de interacción con el entorno.

Una posible solución a este problema es la utilización de múltiples hilos de ejecución, donde cada uno se ejecuta de forma independiente utilizando el mismo espacio de memoria que el resto. El sistema *Forrest* presenta una arquitectura multi-hilada con el objetivo de permitir la ejecución de las tareas más relevantes de los agente de forma independiente.

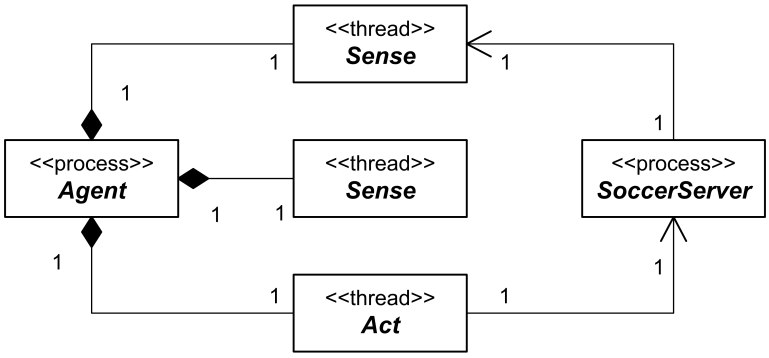

Figura 6: Arquitectura de procesos de un agente.

En la Fig. 6 se muestran los hilos que componen a un agente*: Sense*, *Think* y *Act*. El hilo *Sense* ejecuta el módulo Perseption, el hilo *Think* ejecuta los módulos *Modelling*, *Action Refinement* y *Reasoning*. Por último, el hilo *Act* ejecuta el módulo *Actuator Control*. El proceso *SoccerServer* representa al simulador de juego y se ejecuta independientemente del proceso del agente. Los componentes del agente encargados de establecer la comunicación con el simulador son *Sense* y *Act*. *Sense* recibe información desde el simulador y *Act* envía información hacia el simulador. Es importante destacar que el desarrollo de este trabajo se centra principalmente sobre el componente *Think*, por ser donde se ejecuta la estrategia del equipo del equipo. Todos los agentes del sistema utilizan la misma arquitectura de procesos.

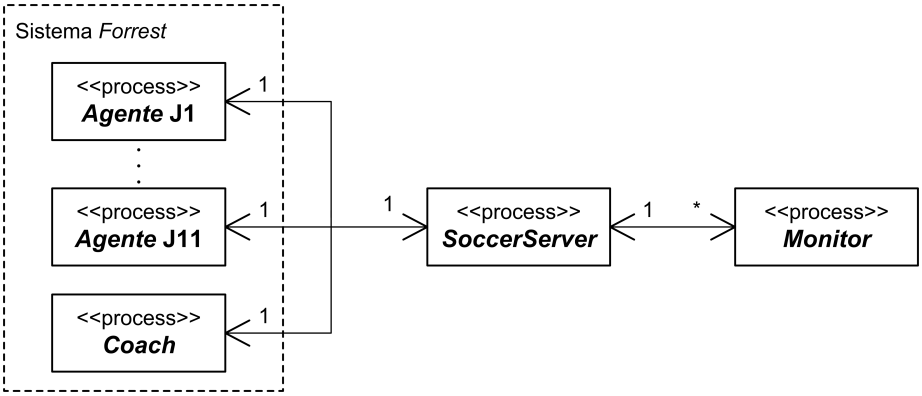

Figura 7: Arquitectura de procesos del sistema multi-agente *Forrest*.

En la Fig. 7 se muestran todos los procesos que componen al sistema *Forrest*. Existen once procesos agentes jugadores y un agente Coach. En la figura se pueden observar también los procesos *SoccerServer* y *Monitor* necesarios respectivamente para la ejecución y visualización de un partido.

7. Vista de Implementación

La vista de implementación muestra los distintos componentes de distribución construidos para el sistema. En la Fig. 8 se presentan estos componentes. Cada una de las capas y módulos presentados en la arquitectura lógica fueron desarrollados en un único componente de distribución. El sistema *Forrest* cuenta con dos tipos de agente: jugador y coach. Esto da lugar a la creación de un ejecutable por cada tipo de agente: *forrestJugador* y *forrestCoach.* Los componentes *player.conf* y *formation.conf* son archivos de configuración utilizados por los agentes jugadores para lograr su correcto funcionamiento. El primero contiene parámetros utilizados para establecer algunos de los comportamientos del agente (amplitud del cono de visión, distancia del marcado, etc.), mientras que el segundo contiene todas las formaciones de juego que puede adoptar el equipo dentro del campo de juego. El componente *start.sh* es utilizado para ejecutar al equipo completo, levantando a los 11 procesos jugadores y al proceso coach.

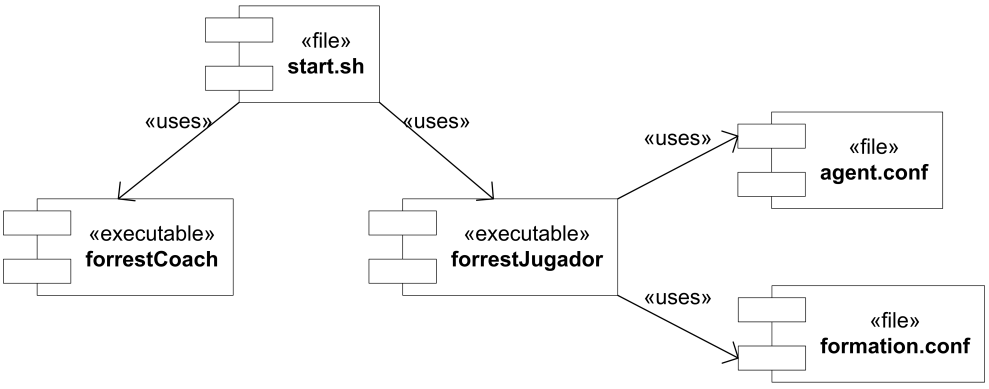

Figura 8: Componentes de implementación del sistema *Forrest*.

8. Vista de Distribución

La vista de distribución presenta la infraestructura necesaria para la instalación del 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 del sistema desde el punto de vista de procesos, es posible identificar tres tipos de nodos: *Forrest*, *Simulador* y *Monitor.* En la Fig. 9 se muestran estos nodos. El nodo *Simulador* contiene al simulador de juego *SoccerServer.* El nodo *Monitor* contiene al software de visualización del juego. Por último, el nodo *Forrest* contiene un instancia de los componentes de distribución *forrestJugador* y *forrestCoach* presentados en la vista de implementación.

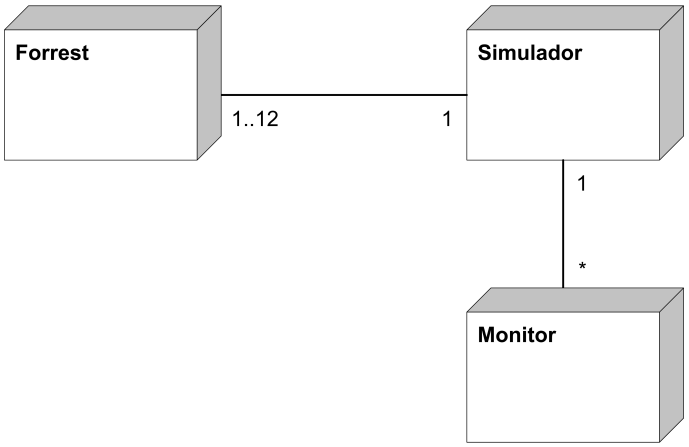

Figura 9: Arquitectura de distribución.

Como se explicó en la sección 6, el sistema cuenta con 12 agentes que se ejecutan de forma independiente. Esta característica permite distribuir la carga del sistema en varios nodos *Forrest* dependiendo de la infraestructura con la que se cuente. Las configuraciones posibles son varias, desde contar con un único nodo *Forrest* donde se ejecuten los 12 agentes, hasta llegar a la distribución máxima, donde cada agente es ejecutado en un nodo diferente. Durante las competencias oficiales, cada equipo cuenta con tres nodos de uso exclusivo para distribuir la carga de su sistema. Durante el transcurso de de este proyecto, se utilizará un único equipo donde deberán ejecutarse: los dos equipos, el simulador y el monitor de visualización.

Referencias

[CANa] Canales, R.; Casella, S.; Rodríguez, P. *"Reglas de la Liga - Reglas de la Competencia Osaka 2005"*. Tejera, G. (tutor). Trabajo de Grado. Facultad de Ingeniería. Instituto de Computación - Montevideo, Junio 2005.

[CANb] Canales, R.; Casella, S.; Rodríguez, P. *"Análisis del equipo UvA Trilearn - Distribución 2003"*. Tejera, G. (tutor). Trabajo de Grado. Facultad de Ingeniería. Instituto de Computación - Montevideo, Diciembre 2005.

[CANc] Canales, R.; Casella, S.; Rodríguez, P. *"Descripción de la Solución"*. Tejera, G. (tutor). Trabajo de Grado. Facultad de Ingeniería. Instituto de Computación - Montevideo, Abril 2006.

[CANd] Canales, R.; Casella, S.; Rodríguez, P. *"Especificación de Requerimientos"*. Tejera, G. (tutor). Trabajo de Grado. Facultad de Ingeniería. Instituto de Computación - Montevideo, Enero 2006.

[CHE] Chen, M.; Dorer, K.; Foroughi, E.; Heintz, F.; Huangy, Z.; Kapetanakis. S; Kostiadis, K; Kummeneje, J.; Murray, J.; Noda, I.; Obst, O.; Riley, P.; Steens, T., Wang, Y.; Yin, X. *"RoboCup SoccerServer: User Manual"*. Comunidad de RoboCup, Febrero 2003.

[DEB] De Boer, R; Kok, J.R. "*The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*" PhD tésis. Universidad de Amsterdam, Holanda 2002.

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

[LSIM] Liga simulada de RoboCup. Disponible en: http://sserver.sourceforge.net/index.html [Consulta: Agosto 2005].

[RUP] IBM Rational Unified Process. Disponible en: http://www.rational.com/rup [Consulta: Febrero 2006].

[UML] Unified Modeling Language. Disponible en: http://www.omg.org/uml [Consulta: Febrero 2006].