UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

Diseño Software de una Arquitectura de Control de Robots Autónomos Inteligentes. Aplicación a un Robot Social.

Autor:
Rafael Rivas Estrada

Directores:
Miguel Ángel Salichs
Ramón Barber

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Leganés, Julio 2010

TESIS DOCTORAL

DISEÑO SOFTWARE DE UNA ARQUITECTURA DECONTROL DE ROBOTS AUTÓNOMOS INTELIGENTES. APLICACIÓN A UN ROBOT SOCIAL.

Autor: Rafael Rivas Estrada

Directores: Miguel Ángel Salichs, Ramón Barber

Firma del Tribunal Calificador:

Firma

Presidente:

Vocal:

Vocal:

Vocal:

Secretario:

Calificación:

Leganés, de de

Resumen

El objetivo principal de esta tesis es el diseño y desarrollo de un conjunto de herramientas para la realización de una arquitectura de control de un robot social. Esta realización debe permitir la apropiación de la tecnología necesaria para lograr el control efectivo y eficiente de los componentes lógicos y físicos que conforman un robot.

El componente principal de la arquitectura es una habilidad, definida en la arquitectura Automático-Deliberativa AD, desarrollada en el grupo de investigación RoboticsLab de la Universidad Carlos III de Madrid. En este trabajo se presenta los siguientes elementos: un sistema distribuido de emisión y captura de eventos, un sistema de memoria a corto plazo para la distribución de información, un tipo de dato abstracto que permite implementar habilidades, y un secuenciador capaz de interpretar en tiempo de ejecución secuencias de control.

Todos los componentes operan de forma débilmente acoplada, y pueden ejecutarse en uno o varios ordenadores interconectados mediante el protocolo TCP/IP. La operación concurrente y paralela de los componentes presentados abre la posibilidad de ejecutar comportamientos previamente establecidos o definidos en línea por un sistema sintetizador de secuencias de control.

Finalmente cabe destacar que los componentes han sido totalmente diseñados, desarrollados y probados en el robot social Maggie, y la integración de éstos permiten la construcción de la Arquitectura Automática Deliberativa de una manera modular, fácil de integrar y de depurar.

Abstract

The main objective of this thesis is the design and development of a set of tools for the realization of a control architecture of a social robot. This realization should allow the appropriation of technology for effective and efficient control of software and hardware components that make up a robot.

The main component of the architecture is a skill, as defined in the Automatic-Deliberative AD architecture, developed in the research group RoboticsLab Carlos III University of Madrid. In this paper, the following elements: a distributed system of emission and capture of events, a system of short-term memory for the distribution of information, an abstract data type that allows to implement skills, and a sequencer capable of playing time implementing control sequences.

All components operate in a loosely coupled, and can run on one or more computers interconnected by TCP\IP. Concurrent and parallel operation of the components presented opens the possibility of implementing pre-established behavior online or defined by a system of control sequences synthesizer.

Finally it is noted that the components have been fully designed, developed and tested in the social robot Maggie, and the integration of these allow the construction of the Automatic Architecture Deliberative in a modular, easy to integrate and debug.

a mi Madre y a la memoria de mi Padre

Agradecimientos

Al terminar una tesis de doctorado se acostumbra agradecer a aquellas personas o instituciones que han colaborado y muchas veces permitido la culminación de la misma. En este caso, quiero exponer que no es una costumbre, sino una necesidad de expresar mi agradecimiento, y a la vez, pedir disculpas ya que con estas palabras poco podré describir sus aportes. Mi deuda con las personas que nombraré a continuación sobrepasa su relación con esta tesis.

Al Dr. Don Miguel Ángel Salichs, Director de esta tesis, por ofrecerme la oportunidad y brindarme su apoyo en el ámbito académico, intelectual y sobre todo humano, en la asignación, desarrollo y la culminación de este trabajo.

Al Dr. Don Ramón Barber Castaño, Co-Director de esta tesis, por su amistad, tiempo y dedicación constante en la revisión de conceptos de robótica, así como en la ardua labor de la corrección de la memoria.

A la MSc. Doña Ana Corrales Paredes, por sus ideas, recomendaciones, por probar el software desarrollado en este trabajo.

Si me dieran a elegir un equipo de trabajo para desarrollar un tarea similar, sin pensarlo dos veces elegiría, el mismo una y otra vez, a los Robotiteros: Miguel Ángel Salichs, Ramón Barber, Ana Corrales, Javier Fernández, David Godoy, Elena Delgado, Álvaro Castro, María Malfaz, Rakel Pacheco y Fernando Alonso.

A la Universidad de Los Andes, especialmente a mis compañeros del Departamento de Computación, por darme la licencia de ausentarme cuatro años para vivir en Madrid y desarrollar este trabajo.

A los profesores del Diplomado en Estudios Avanzados de Ingeniería de Sistemas y Automática, de la Universidad Carlos III de Madrid, por orientarme en el área de Robótica.

Entre los requerimientos que han influido en la culminación de este trabajo se encuentran conocimientos en Programación, Sistemas Operativos, Redes de Ordenadores, Sistemas Distribuidos, por lo tanto, debo agradecer a mis alumnos de la U.L.A., ya que por sus exigencias que me ha obligado a elevar el nivel en todas estas áreas.

A mis compañeros del Departamento de Automática de la UC3M que me hicieron sentir como en casa.

A mi Madre, a mis hermanos, a mi hija Patricia, y a Nahir Barrios por ser siempre una fuente constante de ánimo que alimentó mi deseo de culminar este trabajo.

Va por todos ellos

Índice general

1. Introducción **15**

1.1. Motivación 16

1.2. Objetivo Principal 16

1.3. Objetivos secundarios 16

1.4. Organización del documento 17

2. Antecedentes teóricos sobre arquitecturas de control **19**

2.1. Arquitecturas de control de robots móviles 19

2.1.1. Aproximación deliberativa 20

2.1.2. Aproximación reactiva 23

2.1.3. Arquitecturas híbridas reactivas-deliberativas 26

2.2. Arquitectura Automática Deliberativa 29

2.2.1. Nivel deliberativo 30

2.2.2. Nivel automático 31

2.2.3. Tipos de memorias de la arquitectura AD 34

3. Descripción del problema **35**

3.1. Plataforma física inicial 35

3.1.1. Robots B21 36

3.1.2. Magellan Pro 37

3.2. Plataforma de programación inicial 37

3.2.1. Mobility 38

3.2.2. RFlex 40
3.3. Plataforma física actual 40
3.3.1. Maggie versión inicial 41
3.3.2. Maggie versión actual 41
3.4. Razones de migración 43

4. Estado del Arte 45

4.1. El Proyecto Orocos. 45
4.1.1. Objetivos. 46
4.1.2. Desarrollo del proyecto 46
4.1.3. Construcción de aplicaciones en Orocos. 48
4.1.4. Observaciones. 49
4.2. Proyecto Player & Stage. 50
4.2.1. Objetivos de diseño en Player. 50
4.2.2. Stage. 51
4.2.3. Gazebo. 53
4.2.4. Observaciones. 53
4.3. Proyecto CARMEN. 53
4.3.1. Uso de buenas prácticas en el diseño de CARMEN. 55
4.3.2. Navegación. 59
4.3.3. Localización y seguimiento. 59
4.3.4. Observaciones. 59
4.4. Microsoft Robotics Studio. 60
4.4.1. Observaciones. 62
4.5. Evolution Robotics ERSP. 62
4.5.1. Arquitectura de ERSP. 63
4.5.2. Núcleo de tecnologías. 64
4.5.3. Comportamientos. 65
4.5.4. Observaciones. 65
4.6. Proyecto CLARAty. 65

4.6.1. Capa funcional. 66

4.6.2. Capa de decisión. 68

4.6.3. Implementación. 70

4.6.4. Observaciones. 70

4.7. Skilligent. 70

4.7.1. Posibilidad de extensión. 71

4.7.2. Justificación. 71

4.7.3. Observaciones. 72

4.8. URBI. 72

4.8.1. Innovaciones en URBI. 73

4.8.2. Observaciones. 75

4.9. iRobots AWARE. 75

4.9.1. Observaciones. 75

4.10. Proyecto Open JAUS 76

4.10.1. Implementación. 76

4.10.2. Características. 77

4.10.3. Especificaciones de JAUS. 77

4.10.4. Observaciones. 78

4.11. Webots. 78

4.11.1. Características. 79

4.11.2. Observaciones. 79

4.12. Comparación de las alternativas vistas. 80

4.12.1. Criterios para la selección. 82

4.12.2. Observaciones finales. 83

5. Alternativas para la integración 85

5.1. Integración de procesos en un ordenador 86

5.1.1. Ficheros regulares con bloqueo. 86

5.1.2. Señales. 88

5.1.3. IPC System V 88

5.2. Integración de procesos en varios ordenadores 92
5.2.1. Sockets BSD 93
5.2.2. Llamadas a procedimientos remotos 94
5.2.3. Gestores de objetos 97
5.2.4. Web Services 102
5.2.5. Justificación de la selección 103

6. Diseño de componentes 107
6.1. Metodología usada 107
6.2. Primeras aproximaciones 110
6.3. Implementación de sensores y actuadores virtuales 113
6.3.1. Ejemplo de uso de los componentes básicos 116

7. Eventos y Memoria a Corto Plazo 121
7.1. Sistema de gestión de eventos 121
7.1.1. Emisión y subscripción de eventos 123
7.1.2. Servidores para la gestión de eventos 124
7.1.3. Ejemplo de envío y recepción de eventos 126
7.2. Sistema de memoria a corto plazo 128
7.2.1. Clases requeridas para usar la memoria a corto plazo . . . 129
7.2.2. Servidores requeridos 131
7.2.3. Ejemplo de uso de la memoria a corto plazo 134
7.3. Observaciones finales del capítulo 139

8. Habilidades en la arquitectura AD 141
8.1. Modelo de una habilidad 141
8.2. Implementación de una Habilidad 143
8.3. Ejecución de secuencias de habilidades 148
8.3.1. Requisitos de diseño 148
8.3.2. Versión 1.0 150
8.3.3. Versión 2.0 152

8.4. Editor y generador de secuencias 162

8.5. Observaciones finales del capítulo 163

9. Datos de desempeño de los principales componentes 165

9.1. Sistema de memoria a corto plazo 166

9.1.1. Lectura desde la memoria a corto plazo 166

9.1.2. Escritura en la memoria a corto plazo 173

9.2. Sistema de gestión de eventos 179

9.2.1. Transmisión entre pares en el mismo ordenador 179

9.2.2. Entre pares en ordenadores interconectados por la red interna del robot 188

9.2.3. Entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WIFI 194

9.2.4. Comparación entre los tres escenarios de prueba 203

10. Conclusiones 207

10.1. Aportes 208

10.2. Trabajos futuros 211

A. Modelos de sistemas de eventos discretos 213

A.1. Sistemas de eventos discretos 213

A.2. Máquinas de estados finitos 216

A.3. Redes de Petri 217

B. Ejemplo de una secuencia de habilidades 219

Referencias 222

Índice de figuras

2.1. Aproximaciones en el control de robots. 21
2.2. El robot Shakey. 22
2.3. Arquitectura de Shakey. 23
2.4. Arquitectura NASREM 24
2.5. Modelo basado en comportamientos reactivos. 25
2.6. Arquitectura híbrida reactiva-deliberativa 26
2.7. Arquitectura híbrida-deliberativa 27
2.8. Arquitectura AURA 28
2.9. Capas en la arquitectura AD 29
2.10. Arquitectura AD: Nivel deliberativo 31
2.11. Arquitectura AD: Nivel automático 32

3.1. Robot B21 36
3.2. Robot Magellan PRO 38
3.3. Estructura de una aplicación usando Mobility 39
3.4. Diseño inicial del robot Maggie 41
3.5. Parte superior del Magellan PRO 42
3.6. Robot Maggie 43

4.1. Logo de OROCOS, una formado por dos pinzas de robot. 46
4.2. Componentes de Orocos [1]. 47
4.3. Caja de herramientas de Tiempo Real de Orocos [1]. 48

4.4. Interfaz a componentes de control de Orocos [1]. 49
4.5. Simulación con Stage [2]. 52
4.6. Imagen de la interfaz de Gazebo[3]. 53
4.7. Arquitectura de 3 capas usada por Carmen[4] 54
4.8. Interfaz gráfica en Carmen[5] 56
4.9. Editor de parámetros en Carmen[5] 58
4.10. Entorno de programación visual de Microsoft [6] 60
4.11. Ejemplo de bloque de construcción del VPL de Microsoft[6] . . . 61
4.12. Imagen de una simulación 3D con el MSRS[6] 62
4.13. Arquitecturas de capas de ERSP[7] 63
4.14. Diagrama de bloques de vSLAM en ERSP[7] 64
4.15. Arquitectura de CLARAty [8] 66
4.16. Capa funcional en CLARAty [8] 67
4.17. Jerarquía de Clases en CLARAty [8] 68
4.18. Capa de decisión en CLARAty [8] 69
4.19. Diagrama de la arquitectura software Skilligent[9]. 71
4.20. Interfaz de usuario del URBI Studio[10]. 73
4.21. Topología del Sistema en JAUS [11] 77
4.22. Etapas en la programación de un robot usando Webots [12] . . . 78
4.23. Simulación multiagente usando Webots [12] 80

5.1. Coordinación entre procesos por bloqueo de región 87
5.2. Cola en el núcleo del sistema operativo 89
5.3. Cola de mensajes con prioridad 89
5.4. Región de memoria compartida por tres procesos 91
5.5. Conjunto de semáforos 92
5.6. Modelo de una llamada a un procedimiento local 94
5.7. Diagrama temporal de llamada a un procedimiento remoto. . . . 95
5.8. Modelo de una llamada a un procedimiento remoto 96

5.9. Modelo de una llamada a un método de un objeto remoto usando RMI . . . 99

5.10. Categorías de interfaces en CORBA . . . 101

6.1. Fases del desarrollo de cada componente . . . 108

6.2. Arquitectura de un componente básico . . . 114

6.3. Diagrama de la clase CmoveBase . . . 115

7.1. Propagación de un evento . . . 122

7.2. Diagrama de la clase CEventManager . . . 123

7.3. Diagrama de la clase CServEvent . . . 124

7.4. Diagrama de la clase CServEventRmt . . . 125

7.5. Diagrama de la clase CItem . . . 130

7.6. Diagrama de la clase CmemCortoPlazo . . . 131

7.7. Servidores de memoria a corto plazo . . . 132

7.8. Arquitectura del funcionamiento local . . . 133

8.1. Estados de una Habilidad . . . 142

8.2. Diagrama de la clase abstracta CHabilidad . . . 145

8.3. Caso de uso para el secuenciador . . . 149

8.4. Secuenciador y habilidades . . . 149

8.5. Diagrama de clases del Secuenciador v1.0 . . . 150

8.6. Diagrama UML de una Secuencia descrita en XML . . . 153

8.7. Ejemplo de función de evaluación . . . 155

8.8. Ejemplo de función de activación . . . 157

8.9. Diagrama Cero del flujo de datos en el secuenciador . . . 159

8.10. Autómata del secuenciador . . . 160

8.11. Diagrama de clases de la clase CSecuenciador versión 2.0 . . . 161

8.12. Interfaz del editor de secuencias . . . 163

8.13. Edición de los atributos un Lugar . . . 164

9.1. Duración del proceso de lectura de datos de tamaño 2 Bytes . . . 167

9.2. Duración del proceso de lectura de datos de tamaño 512 Bytes . 168

9.3. Duración del proceso de lectura de datos de tamaño 16 KByte . 169

9.4. Duración del proceso de lectura de datos de tamaño 64 KByte . 170

9.5. Duración promedio del proceso de lectura 170

9.6. Desviación promedio del proceso de lectura 171

9.7. Duración del proceso de escritura de datos de tamaño 2 Bytes . . 173

9.8. Duración del proceso de escritura de datos de tamaño 512 Bytes 174

9.9. Duración del proceso de escritura de datos de tamaño 16 KByte 174

9.10. Duración del proceso de lectura de datos de tamaño 64 KByte . 175

9.11. Duración promedio del proceso de escritura 176

9.12. Desviación promedio del proceso de escritura 177

9.13. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 20 mseg. 180

9.14. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 40 mseg 181

9.15. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 60 mseg. 182

9.16. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 80 mseg 183

9.17. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 100 mseg 184

9.18. Comportamiento del retardo promedio en la detección de eventos entre pares en el mismo ordenador 185

9.19. Desviación estándar del retardo promedio en detección de eventos entre pares en el mismo ordenador 186

9.20. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 200 mseg 188

9.21. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 250 mseg 189

9.22. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 350 mseg 189

9.23. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 450 mseg 190

9.24. Comportamiento del retardo promedio en la detección de eventos entre pares en ordenadores conectados a la misma LAN 191

9.25. Desviación estándar del retardo promedio en detección de eventos entre pares en ordenadores conectados a la misma LAN 192

9.26. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 300 mseg 195

9.27. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 350 mseg 196

9.28. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 400 mseg 197

9.29. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 450 mseg 198

9.30. Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 500 mseg 199

9.31. Retardo promedio en la detección de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WIFI 200

9.32. Desviación estándar en el retardo de la detección de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WIFI 201

9.33. Retardo promedio en la detección de eventos en los tres escenarios de prueba 204

9.34. Desviación estándar entre los tres escenarios de prueba 205

A.1. Funcionamiento de una máquina de estados finitos. 216

A.2. Ejemplo de una máquina de estados finitos. 217

A.3. Ejemplo de la estructura de una red de Petri 218

B.1. Diagrama de estados de la secuencia obstaclesavoidance. 219

B.2. Red de Petri de la secuencia obstaclesavoidance. 220

Índice de Tablas

4.1. Resumen comparativo entre las plataformas software descritas . . 81

9.1. Resultados de las pruebas para las lecturas en memoria a corto plazo 171

9.2. Tiempos mínimos y máximos registrados en la lectura de datos de memoria compartida 172

9.3. Resultados de las pruebas para las escrituras en la memoria a corto plazo 177

9.4. Tiempos mínimos y máximos registrados en la colocación de datos de memoria compartida 178

9.5. Resultados de las pruebas para el sistema de eventos detectados entre pares en un mismo ordenador 187

9.6. Tiempos mínimos y máximos registrados en la detección de eventos entre pares en un mismo ordenador 187

9.7. Resultados de las pruebas para el sistema de eventos entre pares en ordenadores en la misma LAN 193

9.8. Tiempos mínimos y máximos registrados en la detección de eventos entre pares en la misma LAN 193

9.9. Resultados de las pruebas para el sistema de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WiFi 202

9.10. Tiempos mínimos y máximos registrados en la detección de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WiFi . 202

9.11. Resultados de las pruebas para los tres tipos de escenarios de operación 206

1

Introducción

Uno de los principales objetivos de la robótica es lograr la incorporación de robots en la cooperación de las diferentes actividades del quehacer de los humanos. El rango de aplicaciones posibles va desde los robots industriales, robots que tengan la capacidad de usar herramientas, aplicaciones de medicina, de búsqueda de objetos específicos, y llegando a ser usados en el área de entretenimiento, asistencia o simple compañía de personas con discapacidades o sin éstas. Existe un amplio rango de dispositivos físicos que pueden incorporarse en un robot, y otra gran variedad de algoritmos y componentes lógicos que pueden ser incorporados para extender las diferentes funcionalidades de un robot.

La integración de los componentes físicos y lógicos de un robot es uno de los requerimientos fundamentales en las operaciones de las aplicaciones robóticas. El área de investigación en la cual se centra este trabajo es el de los sistemas de control para aplicaciones en robótica. Basado en las ideas planteadas en [13] se abre un camino para lograr el trabajo cooperativo entre los diferentes componentes de un robot. Se plantea una división jerárquica con el objetivo de integrar la ideas de arquitecturas reactivas y arquitecturas deliberativas, para conformar una arquitectura híbrida cuyo principal elemento básico de construcción[1] son las habilidades.

El presente trabajo se centra en el diseño, desarrollo e implantación de una arquitectura de software para aplicaciones robóticas, basada en el paradigma de la arquitectura Automática Deliberativa(AD) propuesta por el grupo de investigadores del RobiticsLab, de la Universidad Carlos III de Madrid, en Madrid, España.

[1] En este contexto a los elementos básicos de construcción se les asigna el nombre de constructos

1.1. Motivación

Hoy en día existen una gran variedad de herramientas de software para aplicaciones robóticas, pero la gran mayoría de estas son de código cerrado, dependientes del hardware o en bibliotecas que exigen el uso de paquetes completos. Las soluciones presentadas en este trabajo se pueden utilizar en conjunto o de manera individual y ser extendidas a plataformas gestionadas por sistemas operativos diferentes y lenguajes como Java o Python [14], entre otros.

1.2. Objetivo Principal

El objetivo principal de este trabajo es lograr una realización de la arquitectura Automática Deliberativa. Esta realización debe permitir la consecución de la tecnología necesaria para lograr el control efectivo y eficiente de los componentes lógicos y físicos que conforman un robot asistencial. En resumen el objetivo principal es:

> **Se debe diseñar y construir un sistema que sintetice[2] un sistema de control de robots móviles basado en una organización de componentes diseñados y construidos previamente, de tal forma que cumplan con las propuestas de la arquitectura de control AD**

El trabajo de investigación es desarrollado en la Universidad Carlos III de Madrid, específicamente dentro de la línea de investigación de arquitecturas de control de robots del RoboticsLab. La plataforma que es usada como plataforma de desarrollo es el Robot Maggie [15]. Además, como restricción, el núcleo central del sistema de control debe residir sobre el sistema operativo Linux. Sobre este sistema, y en vista de los requerimientos funcionales y temporales, se ha usado el lenguaje de programación C [16] y C++ [17] para los componentes de bajo nivel.

1.3. Objetivos secundarios

En todo trabajo de investigación, además del objetivo principal, se esperan resultados parciales, que proporcionen interesantes aportes o temas a investigar que coadyuven a lograr el alcance global del trabajo de investigación principal. Entre los objetivos secundarios se tienen:

- Una vez logrado el desarrollo propuesto se dispondrá de una plataforma de desarrollo para el inicio de nuevos proyectos en el grupo.

[2] Composición de un todo por la reunión de sus partes, fuente RAE

- Lograr el diseño y la realización de un sistema robótico multiproceso.
- Debido a que la organización de Maggie se basa en varios ordenadores interconectados por medio de una red de alta velocidad, el sistema a diseñar debe garantizar la interoperabilidad en un entorno multiprocesamiento distribuido.
- Se logrará diseñar y construir un sistema cuyos componentes principales hayan sido diseñados y construidos con herramientas de software libre, y sobre un sistema operativo de libre distribución, como lo es el sistema Linux.
- La mayoría de los componentes diseñados están débilmente acoplados, lo que garantiza un mantenimiento más fácil al poder intercambiar estos componentes por nuevas versiones de los mismos con mejoras, sin afectar el resto de componentes.
- Obtención de una plataforma para la realización de la arquitectura AD.

1.4. Organización del documento

El presente documento se inicia con una introducción al problema, así como una justificación y presentación del objetivo principal y objetivos secundarios.

En el Capítulo 2 se presenta el marco teórico, donde se comienza con la presentación de la definición de las estructuras de control robóticas. Se indican las corrientes clásicas de control en robótica como son las aproximaciones deliberativas y reactivas, y algunas realizaciones de estas arquitecturas. Posteriormente se llega a las estructuras de control híbridas que pretenden unir las ventajas de la aproximación deliberativa jerárquica y de la aproximación reactiva. Se finaliza con una breve descripción de la arquitectura AD propuesta por el RoboticsLabs para el uso en robots personales.

En el Capítulo 3 se describe el problema a resolver en el instante de iniciar el trabajo de investigación. Se inicia con una evaluación de la arquitectura de control presente en el robot. Se describe la plataforma inicial sus ventajas y desventajas de uso. El capítulo finaliza con una justificación a la realización del trabajo

En el Capítulo 4 se presenta el estado del arte, con todas las alternativas presentes y disponibles en el momento de iniciar el trabajo. Se hace una comparación entre las altertivas posibles que justifiquen la selección de cualquiera de las herramientas robóticas disponibles. Las plataformas de desarrollo estudiadas son Orocos, Player & Stage, Carmen, Microsoft Robotics Studio, ERSP, Claraty, Skilligent, URBI, iRobots Aware, Open JAUS, y Webots.

En vista de las diferentes alternativas disponibles y tomando en consideración que existe la posibilidad de desarrollar una propia, el Capítulo 5 se presentan las alternativas de integración disponibles para la realización del trabajo. Las herramientas presentadas en esta sección servirán para la construcción de un

software intermedio que permitirá lograr un nivel mayor de abstracción de las funcionalidades, facilidad en el mantenimiento y permitir la extensión del sistema en el futuro para lograr una mayor adecuación a las necesidades actuales y las que posiblemente surjan en el desarrollo del proyecto.

Una vez definida la plataforma a usar y los servicios requeridos para alcanzar el objetivo principal, en el Capítulo 6 presente la metodología usada y el diseño y construcción de los actuadores y sensores virtuales.

El Capítulo 7 Se inicia con el sistema de gestión de eventos, y el sistema de memoria a corto plazo. Además se presentan ejemplos de uso de cada una de estas implementaciones.

El diseño e implementación de una Habilidad es presentado en el Capítulo 8. Esta clase modela todas las habilidades posibles sin detallar ninguna específicamente, ésta sirve como mecanismo de abstracción, y gracias a su fácil extensión, es posible adecuarla a necesidades futuras o en fase de desarrollo. El capítulo termina con la presentación del secuenciador, encargado de la activación y desactivación de las habilidades. Sus requisitos de diseño permiten la ejecución secuencial, concurrente y paralela de habilidades operando de forma cooperativa en el robot.

En vista de la importancia del núcleo de componentes básicos, en el Capítulo 9 se presentan datos de desempeño del sistema distribuido de memoria a corto plazo y del sistema de gestión de eventos. El objetivo del capítulo es presentar los rangos de operación confiable de los componentes desarrollados.

En el Capitulo 10, se presentan las conclusiones y los aportes del trabajo presentado.

En los Apéndices se agrega información básica y necesaria para el uso de las herramientas, así como un ejemplo de una secuencia de habilidades en formato XML.

Se finaliza el documento con la Bibliografía citada.

2

Antecedentes teóricos sobre arquitecturas de control

Revisando la historia, se encuentra que la idea de crear un hombre artificial tiene sus orígenes en la antigua Grecia. En la mitología griega se hace referencia a *Talus*, un gigante hecho de hoja de lata, quien protegía a Creta de los invasores. En este caso se pensó en un humano artificial con un fin estrictamente militar, ya que su objetivo era el de dar protección. En [18] se hace un análisis de la cercanía al objetivo de poder construir un *hombre artificial*. Con tal fin se establece un conjunto de características que debe poseer éste:

- Inteligencia similar a la de los humanos.
- Comunicación similar a la de los humanos.
- Posibilidad de traslación similar a la de los humanos.

Se debe agregar que debe **ser capaz de adaptarse al entorno y a los dispositivos u objetos usados por humanos** y, de esa manera, **lograr una integración a recursos artificiales creados para los humanos**.

Como requisito básico para comprender el alcance de este trabajo, es necesario revisar los aspectos concernientes a las arquitecturas de control de robots móviles, estudiar las diferentes tendencias y algunas realizaciones.

2.1. Arquitecturas de control de robots móviles

Una de las primeras definiciones sobre sistemas control en robots encontradas en la revisión bibliográfica, es la propuesta por Walter Jacobs en [19]. En esta publicación se describe que el comportamiento de un robot depende de la selección de una *estrategia*, donde ésta es emitida por un componente del sistema llamado *control*. W. Jacobs estudiaba la manera de solucionar un problema

presentado por John H. Munson en [20]. El objetivo era lograr al avance de un robot en un entorno dado, cuyo problema era la necesidad de obtener un modelo detallado o mapa del entorno. Pero en general, al aumentar la complejidad del entorno, el modelo sería siempre inexacto o incompleto. El sistema contenía un componente de navegación, encargado de generar las operaciones para lograr el desplazamiento deseado, o la solución del problema planteado. El sistema también poseía un componente ejecutivo, encargado de convertir las operaciones en acciones que produzcan el movimiento deseado. Sin embargo, un conjunto de acciones en el mundo real no siempre están reflejadas fielmente a como están reflejadas en el modelo, y se puede terminar en un punto de coordenadas diferente al deseado. En este caso, se debería agregar una nueva etapa donde se genere un nuevo plan en el cual se especifique las acciones necesarias para poder alcanzar la posición deseada a partir de las coordenadas actuales del robot.

Maja Matarić ofrece en [21] una definición mas extensa, donde indica que una arquitectura de robot es principalmente una manera de organizar un sistema de control y, adicionalmente, la arquitectura impone restricciones a la manera en la que el problema de control debe ser resuelto.

Otra definición sobre arquitectura de robots es proporcionada por Ronald Arkin en [22, p. 124], donde indica que la arquitectura de robot son "las especificaciones y sistemas programados que proveen las herramientas y lenguajes para la construcción de sistemas *basados en comportamientos*". Esta definición se ubica en un nivel de abstracción superior, ya que la visión de Arkin se fundamenta en las arquitecturas basadas en comportamientos como un mecanismo para lograr el diseño de plataformas, haciendo énfasis en un alto acoplamiento entre la captura de estímulos del entorno y de las acciones generadas como respuesta a esos estímulos.

Muchas técnicas y aproximaciones diferentes para el control robótico han sido desarrolladas. Existen dos aproximaciones básicas que han emergido como resultado de este desarrollo: interés en procesos deliberativos e interés en los procesos reactivos. Estas dos aproximaciones pueden distinguirse por el uso que se le da a los datos obtenidos mediante sensores, por el conocimiento global que se pueda tener del entorno, velocidad de respuesta, selección de decisiones y complejidad computacional.

En la figura 2.1 puede verse el esquema de estas dos aproximaciones que se comentarán a continuación.

2.1.1. Aproximación deliberativa

Un robot que emplea una aproximación deliberativa requiere un conocimiento relativamente completo sobre el entorno, y usa este conocimiento para calcular el resultado de sus acciones, una capacidad que le permite optimizar su desempeño relativo al modelo que posee del entorno. La aproximación deliberativa a menudo requiere que asuma como cierto el modelo que posee del entorno. Principalmente el modelo sobre el cual calcula las acciones a ejecutar debe ser: consistente, confiable y seguro. Si la información usada es inexacta o ha cambia-

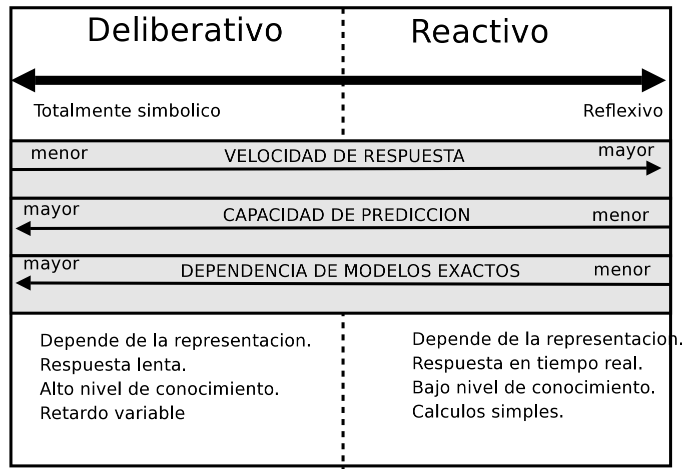

Figura 2.1: Aproximaciones en el control de robots.

do desde que fue obtenida, las acciones calculadas pueden ser incorrectas. Los modelos que representan el entorno son generalmente construidos a partir del conocimiento previo sobre el entorno y de los datos obtenidos por medio de los sensores.

Los sistemas basados en la aproximación deliberativa a menudo poseen las siguientes características:

- Son sistemas estructurados jerárquicamente con una clara e identificable división entre las diferentes funcionalidades que lo constituyen.
- El control y la comunicación ocurre de una manera predecible y predefinida, fluyendo desde los niveles superiores a los niveles inferiores de la jerarquía, con poco intercambio de ordenes de control en el sentido horizontal de la organización.
- Los niveles superiores de la estructura jerárquica proporcionan subobjetivos para los niveles subordinados.
- El ámbito de la planificación, espacial y temporal, cambia a medida que se desciende en la jerarquía. Los requerimientos de tiempo son mas cortos y las consideraciones espaciales son más locales en los niveles inferiores.
- Los modelos del entorno están basados en representaciones simbólicas

A continuación se presentan algunas arquitecturas que siguen en parte esta aproximación.

Figura 2.2: El robot Shakey.

Shakey

El robot Shakey [23] de SRI, mostrado en la figura 2.2[1], fue el prototipo de la arquitectura de robot utilizando la aproximación deliberativa. Esta aproximación implementada en Shakey está caracterizada por una estructura jerárquica (figura 2.3) en la cual cada capa provee sub-objetivos (o instrucciones explicitas) a las capas inferiores. Además, la arquitectura jerárquica incluye un modelo del entorno en el cual el robot se mueve. La percepción es usada para modificar y actualizar el modelo del entorno. De este modo, la acción a ejecutar es producida por el planificador a partir de la información obtenida del modelo. En otras palabras, el robot observa, piensa y luego se mueve. Porque el tiempo requerido para ejecutar las operaciones de planificación puede ser no despreciable, los robots que seguían esta aproximación en la época de Shakey estaban confinados a entornos estáticos en los cuales nada cambiaba mientras el proceso de observación se encontraba en ejecución. Esta restricción fue suavizada a medida que aumentaba las velocidades en los procesadores.

NASREM

NASREM [24] representó la culminación de un esfuerzo de mas de 15 años de investigación y desarrollo en el Instituto Nacional de Normas y Tecnologías de los Estados Unidos de América [25] en los sistemas de control a tiempo real para robots y máquinas inteligentes. La primera versión fue desarrollada por James Albus [26] para el desarrollo en investigación en laboratorios de robótica. Posteriormente fue extendida para robots en las áreas de manufactura. Las seis capas que caracterizan esta arquitectura son mostradas en la figura 2.4[2].

Cada capa en la arquitectura tiene asignada ciertas funciones, que van desde el control en la capa más baja, hasta la planificación estratégica en la capa más alta. Es de hacer notar que las entradas a los sensores en cada capa superior son

[1] Imagen tomada de http://www.sri.com
[2] Imagen tomada de la página personal de James Albus

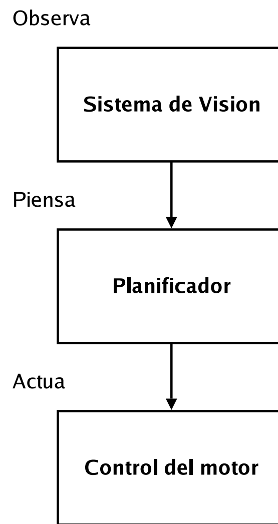

Figura 2.3: Arquitectura de Shakey.

usadas para actualizar el modelo del entorno. Esta arquitectura se ha diseñado para que solo las capas inferiores realicen tareas de control directo.

La planificación global es realizada en las capas superiores. Debido a que la planificación global se enfoca en metas a largo plazo, el tiempo requerido para completar esta tarea no está fuertemente limitado.

2.1.2. Aproximación reactiva

En el lado derecho de la figura 2.1 se representan los sistemas reactivos. Se puede definir que un sistema de control reactivo es una técnica que se caracteriza por la fuerte dependencia existente entre la información obtenida a través del sistema de percepción y las acciones que se generan. Típicamente, en el contexto del comportamiento de un motor, las respuestas en el sistema robótico se producen oportunamente ante cambios en la estructura del entorno.

A continuación se enumeran los conceptos comunes en los sistemas reactivos:

Comportamiento individual: un par estimulo/respuesta para un entorno específico es *modulado* por la atención y *determinado* por la intención.

Atención: los procesos de los recursos del sistema sensorial son priorizados y enfocados dependiendo del contexto actual del entorno.

Intención: determina el conjunto de comportamientos que debe ser activado basado en las metas y los objetivos.

Comportamiento emergente o inesperado el comportamiento global del robot es una consecuencia de la interacción entre los diferentes comportamientos individuales activos.

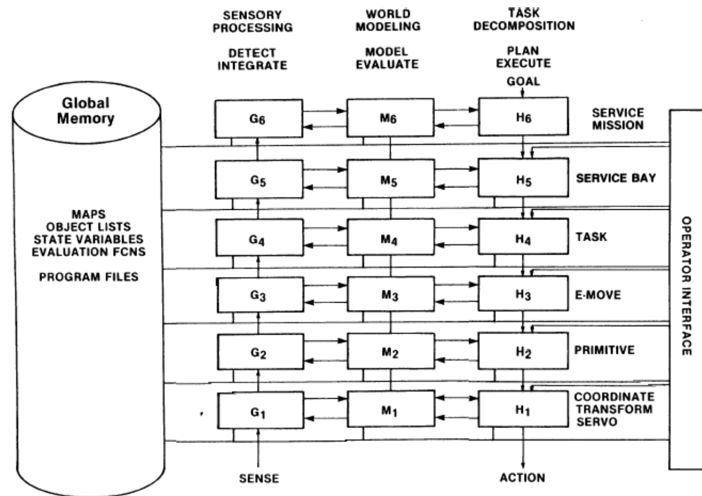

Figura 2.4: Arquitectura NASREM

Comportamiento reflexivo o reactivo puro: comportamiento que es generado por los lazos de control más cercanos a los sensores y actuadores, donde la información sensorial es no persistente. No se usan modelos del entorno.

Ubicación en un entorno real: el robot es un entidad situada y rodeada por un entorno real. Éste no se desenvuelve sobre representaciones abstractas de la realidad, ya que éste en sí es real.

Presencia: un robot tiene presencia física (tienes dimensiones y masa). Es una realidad espacial, por consecuencia sus interacciones dinámicas con el entorno no pueden ser simuladas fielmente.

Inteligencia emergente: Es posible que la inteligencia del robot puede incrementarse a partir de las interacciones de otros robots y sus entornos. Este concepto no aplica en robots en entornos aislados, sino que el conocimiento se obtiene a partir de la interacción entre robots que comparten condiciones comunes en su entorno.

Propuestas de Rodney Brooks

En el año de 1986 Rodney Brooks presenta en [27] una arquitectura para robots móviles, que se consideró revolucionaria para esa época. Su trabajo se originó a partir del paradigma: *observa-piensa-actúa* usado en el robot Shakey, discutido en su sección 2.1.1. Por el contrario Brooks planteó que la construcción de un modelo obtenido a partir de un entorno complejo podría ser un obstáculo para el éxito de la construcción de un robot móvil, además aseguró que el entorno que rodea al robot es el único modelo necesario para poder actuar perfectamente. En vez de usar una estructura vertical de observa-piensa-actúa, él sugirió una

descomposición horizontal en términos de comportamiento, como se muestra en el diagrama de la figura 2.5.

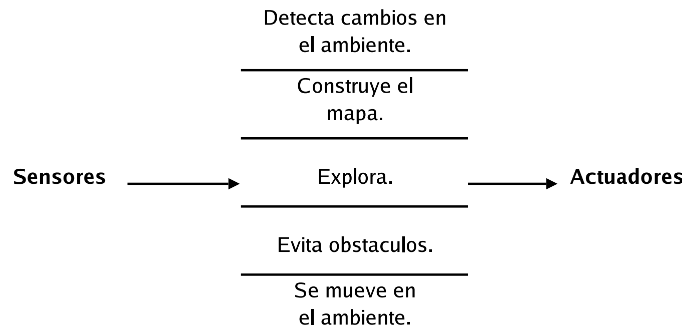

Figura 2.5: Modelo basado en comportamientos reactivos.

En las arquitecturas, basadas en el modelo de Brooks, todos los comportamientos trabajaban en paralelo, concurrentemente, y en general, de forma asíncrona. En las primeras versiones de esas arquitecturas, los comportamientos del robot fueron implementados usando máquinas de estado finitos que fueron *incómodas* de entender por los investigadores sin conocimiento de conceptos básicos de matemáticas discretas. En las siguientes versiones se describen los comportamientos en formas de reglas a las cuales el robot responde ante una entrada obtenida por medio de los sensores. Por ejemplo, en presencia de un obstáculo al frente del robot, los motores actúan para alejarse del obstáculo.

En el modelo de Brooks, los comportamientos complejos están subordinados[3] a los comportamientos más simples. Esto es, la exploración está subordinada a las maniobras para evitar obstáculos.

El diseño de robots basados en comportamientos subordinados a otros proporcionó un nuevo conjunto de herramientas de heurística, como las descrita por Maja Matarić en [28]. Un diseño basado en estas ideas debe cumplir con:

- Especificar la manera en la que el robot debe responder a los estímulos obtenidos del entorno.
- Descomponer el comportamiento del robot en acciones observables de ejecución independiente.
- Implementar estas acciones por medio de los actuadores del robot.

Además, la arquitectura de comportamientos subordinados debe limitar cuidadosamente la comunicación entre niveles, para poder lograr la coordinación entre los niveles superiores subordinados a los niveles inferiores. Por ejemplo, Maja Matarić en su tesis Doctoral [29] usa nuevos comportamientos para obtener mejores resultados en entornos de mayor complejidad. En sus trabajos el robot debía encontrar discos de madera(puck), los recogía (pickup) y los llevaba hasta un sitio específico (homming). Los comportamiento usados eran:

[3]Aproximación en castellano a la palabra en inglés *subsumption*.

- Desplazamiento exploratorio (wandering).
- Maniobras ante obstáculos (avoid).
- Recoger discos (pickup).
- Llevar discos a la meta (homming).

Es importante destacar que la arquitectura basada en comportamientos reactivos, introducida por Brooks, fue un cambio revolucionario en el área de arquitecturas de control. Desafortunadamente, las arquitecturas puramente reactivas no son apropiadas para algunas tareas complejas.

2.1.3. Arquitecturas híbridas reactivas-deliberativas

Las dos aproximaciones para el control de alto nivel de los robots han sido desarrolladas independientemente. La aproximación deliberativa fue la primera en ser propuesta, con el robot Shakey estando entre los primeros de esa generación. Las arquitecturas reactivas llegaron un poco mas tarde. Tal como se esperaba, solo era cuestión de tiempo para que los investigadores comenzaran a probar con sistemas híbridos que unieran algunas características de las dos aproximaciones. Entre los primeros estuvo Maja Matarić [28], que en su propuesta agregó una capa de planificación sobre múltiples capas reactivas. En la figura 2.6 se muestra la arquitectura propuesta.

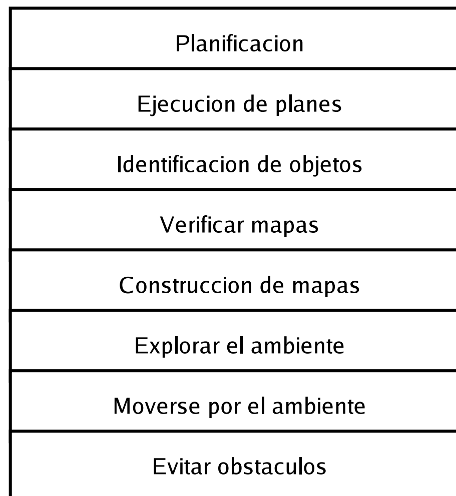

Figura 2.6: Arquitectura híbrida reactiva-deliberativa

En la figura 2.6 las cinco primeras capas inferiores corresponden a la arquitectura reactiva, y las dos capas superiores corresponden a la planificación y deliberación. El trabajo de Matarić fue rodeado de cierta polémica, ya que los defensores de la corriente reactiva estaban en contra de agregar una capa superior para la construcción del mapa, debido a que se suponía que el entorno en sí era el único mapa que hacia falta. En la figura 2.6 se muestra una arquitectura

híbrida donde las capas inferiores tienen prioridad sobre las capas superiores. Por lo tanto, el componente de evitar obstáculos tiene prioridad sobre la ejecución de un plan para alcanzar una meta distante.

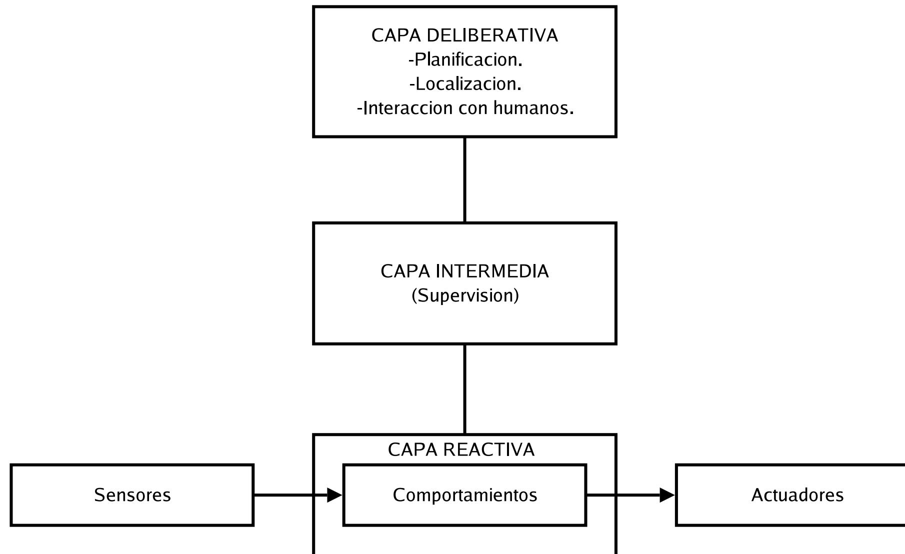

Figura 2.7: Arquitectura híbrida-deliberativa

En la figura 2.7 se muestra una arquitectura típica de tres capas. La capa inferior es la capa reactiva, en la cual los sensores y los actuadores están fuertemente acoplados. La capa superior provee el componente deliberativo, incluyendo la interacción con humanos, planificación, localización, y otras funciones cognitivas. La capa llamada supervisión es intermedia entre las otras dos.

Una revisión al modelo de tres capas involucra la estructuración jerárquica para la parte deliberativa de la arquitectura. Ésta es también llamada arquitectura de planificación jerárquica-deliberativa (la figura 2.8 muestra una estructura típica híbrida jerárquica-deliberativa), la cual subdivide el componente de planificación en secciones que dependen del horizonte de planificación tanto en tiempo como en espacio. La planificación a corto plazo se concentra en la consecución de objetivos en la vecindad del robot y éstas son alcanzables en un intervalo de *tiempo corto* (donde el llamado tiempo corto depende de una tarea en particular, la velocidad del robot, y la naturaleza del entorno). Claramente la planificación en corto plazo debe ser cumplida rápidamente y requiere un conocimiento preciso del entorno que rodea al robot.

Por otra parte, en el nivel superior, el planificador global se concentra con la estrategia general del robot. Debido a esto, se requiere una cantidad mayor de tiempo para obtener la estrategia general y ésta requiere de una precisión mayor. Los planificadores globales dependen de la precisión de los modelos del entorno para poder llevar a cabo la planificación. En cambio, el componente reactivo de la arquitectura basada en comportamientos, requiere un fuerte acoplamiento de la percepción y las acciones ejecutadas y estas deben trabajar de manera coordinada para poder actuar ante cambios en el entorno.

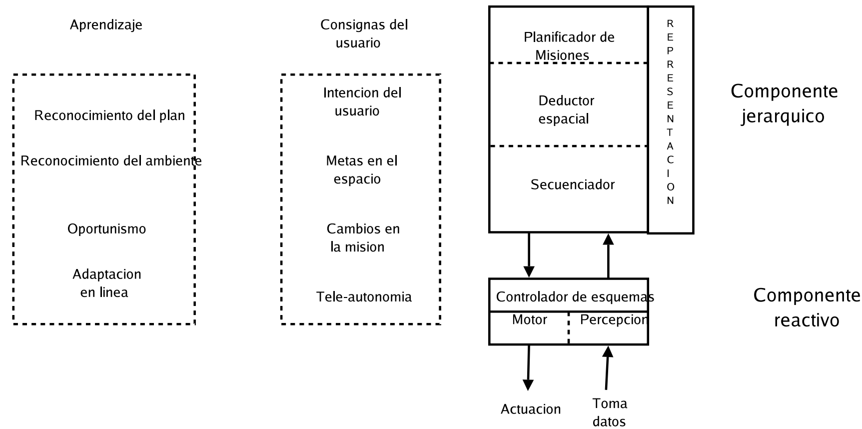

Figura 2.8: Arquitectura AURA

Varios investigadores han construido robots usando arquitecturas híbridas deliberativas-reactivas, entre éstas se puede mencionar:

- Aura (Autonomous Robot Architecture), desarrollada por Ronald Arkin [30].
- Atlantis, desarrollada por E. Gat, en NASA Jet Propulsion Laboratory 1992.
- Generic Robot Architecture, desarrollada por Noreils en 1993.
- Xavier, desarrollado por R Simmons, en 1994.
- 3T, desarrollada por R. Bonasso en 1997.
- Saphira, desarrollado por Konolige y Myers en 1996 en el SRI.
- TeamBots, desarrollado por Balch en 2000.
- BERRA (Behavior-based Robot Research Architecture), desarrollada por Lindström, Orebäck y Christensen en 2000.

En este capítulo se han descrito las principales corrientes en el desarrollo de sistema de control para robots. En la siguiente sección se describe la arquitectura desarrollada en el RoboticsLabs y publicada en [13].

En los últimos años, la robótica ha ido evolucionando, tratando de prestar servicios útiles a los hombres e integrándose cada vez más en su entorno. La interacción entre el hombre y los robots será tanto mayor cuanto más se parezcan los robots a los seres humanos, tanto en el aspecto físico como psíquico. En cuanto al aspecto físico es cada día mayor la cantidad de trabajos que aparecen en el

diseño y control de robots humanoides. Así lo muestran los trabajos de Laschi y Okumura [31, 32], donde se muestran robots con forma humana que intentan reproducir movimientos humanos como andar o arrodillarse. En el aspecto psíquico se intenta copiar las formas de razonar, navegar y aprender. También se intenta reproducir la estructura emocional humana e incluso se diseñan arquitecturas completas que intentan implementar los niveles de abstracción mental y afectiva del ser humano.

2.2. Arquitectura Automática Deliberativa

De acuerdo con las teorías de la psicología moderna [33], existen dos mecanismos mediante los que el hombre procesa la información percibida de su entorno: los procesos automáticos y los reflejos. Por procesos reflejos se entienden aquellos procesos que requieren de la capacidad de razonamiento o decisión. Los procesos automáticos apenas requieren de intervención de la atención. En el ser humano cabe, por lo tanto, diferenciar entre dos niveles de actividad mental: nivel Deliberativo y el nivel Automático. Basándose en estas ideas, en la arquitectura AD se establecen sólo dos niveles: Deliberativo y Automático. El nivel Deliberativo esta asociado con procesos reflexivos y el nivel Automático con procesos automáticos.

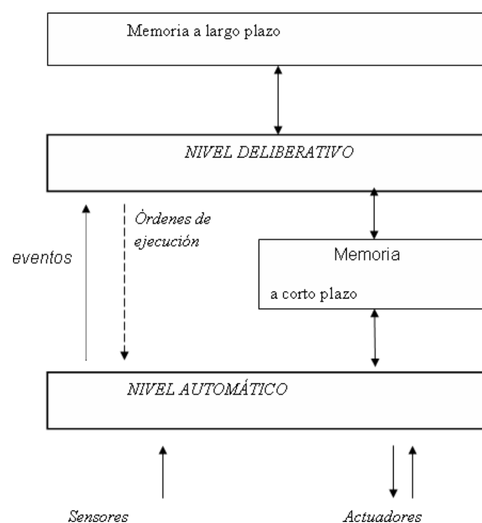

Figura 2.9: Capas en la arquitectura AD

Ambos niveles presentan una característica común: están formados por **habilidades** . Las habilidades son las diferentes capacidades para razonar, procesar información o llevar a cabo una acción. Estas habilidades son activadas por órdenes de ejecución producidas por otras habilidades o por un secuenciador, devolviendo datos y eventos a las habilidades o secuenciadores que las hayan activado. Esas habilidades son la base de la arquitectura AD. La figura 2.9 muestra un diagrama con la representación de la arquitectura AD.

La comunicación entre los niveles Deliberativo y Automático es bidireccional. Ambos niveles se comunican a través de la memoria a corto plazo y por medio de eventos, en la que se encuentra la información acerca del estado del robot, y de dos flujos de información:

- Flujo de eventos: La habilidades situadas en el nivel Automático notifican a las habilidades del nivel Deliberativo los eventos que generan, y éstas a su vez pueden recibir eventos de las habilidades deliberativas.

- Flujo de órdenes de ejecución: Las habilidades automáticas reciben órdenes de ejecución procedentes del nivel Deliberativo. Las habilidades deliberativas deciden qué habilidades del nivel Automático tienen que ser activadas o desactivadas en cada momento.

2.2.1. Nivel deliberativo

En el nivel Deliberativo se encuentran los módulos que requieren razonamiento o una capacidad de decisión. Estos módulos no producen respuestas inmediatas y necesitan procesar la información con la que trabajan para tomar decisiones. Estos módulos son los que forman las habilidades deliberativas y son activadas por otras habilidades o por el secuenciador principal, que será el encargado del correcto funcionamiento de estas habilidades. A diferencia del nivel Automático, las actividades del nivel Deliberativo se llevan a cabo secuencialmente y no es posible realizar más de una actividad deliberativa al mismo tiempo, tal y como ocurre en los humanos.

Por lo tanto, el nivel Deliberativo de esta arquitectura AD esta formada por las habilidades deliberativas y por el secuenciador principal, tal y como se muestra en la figura 2.10. Esta secuencia se establece a priori y va a marcar el comportamiento del robot y cómo éste va a proceder en las diversas situaciones.

Habilidades deliberativas

Las habilidades deliberativas son cada una de las capacidades de razonamiento y decisión con las que cuenta el sistema autónomo. Son las encargadas de manejar y modificar la memoria a largo plazo de la arquitectura, así como de manejar y secuenciar las habilidades automáticas. Como se ha comentado anteriormente, las habilidades deliberativas son manejadas y controladas por el Secuenciador Principal. A su vez, las habilidades del nivel Deliberativo son capaces de generar eventos, que comunican al Secuenciador Principal, para que éste decida las nuevas habilidades deliberativas a activar. Ejemplos de implementación de estas habilidades deliberativas son los planificadores, navegadores y los módulos que realizan el modelado del entorno [34].

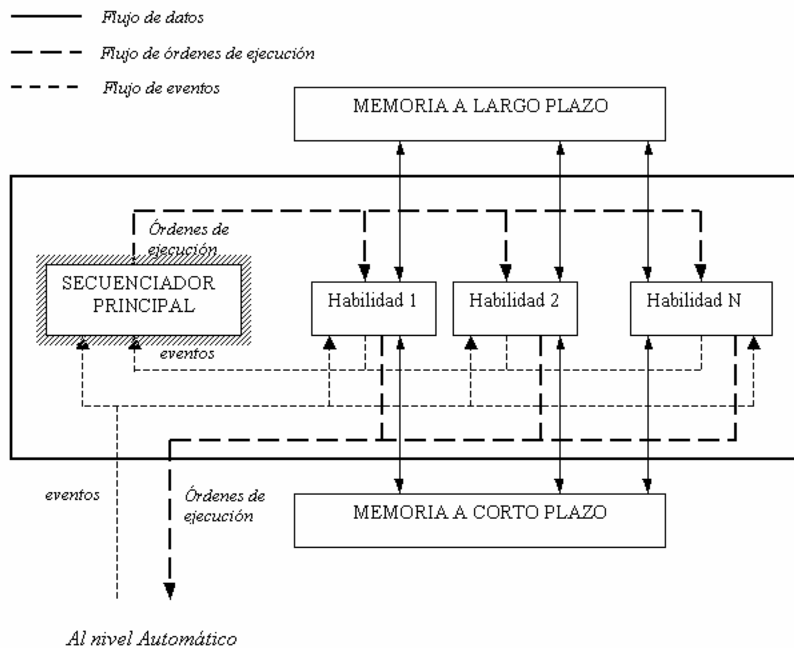

Figura 2.10: Arquitectura AD: Nivel deliberativo

2.2.2. Nivel automático

El nivel Automático se encarga del control a bajo nivel de los dispositivos de los que dispone el robot. Este nivel permite al robot tener la capacidad de reacción necesaria para responder rápidamente a cambios producidos en su entorno [13, 35]. La figura 2.11 muestra los elementos que constituyen este nivel:

- Sensores y actuadores virtuales: El nivel Automático se comunica con el hardware del robot a través de los sensores virtuales y los actuadores virtuales.
- Acciones reflejas: Son las respuestas involuntarias y prioritarias a los estímulos.
- Habilidades automáticas: Son las capacidades sensoriales y motoras del sistema.

El nivel Automático debe cumplir los siguientes requisitos:

- Los elementos que constituyen este nivel deben ser capaces de ejecutarse en paralelo y de interactuar unos con otros.
- Debe permitir añadir nuevos elementos sin afectar el funcionamiento de las otras, de manera que la arquitectura sea ampliable y dinámica.

- Debe permitir la generación de habilidades complejas a partir de las existentes. Una habilidad puede ser utilizada por varias habilidades complejas, lo que aporta flexibilidad a la arquitectura.

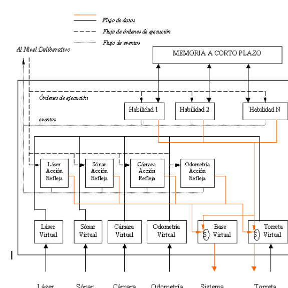

Figura 2.11: Arquitectura AD: Nivel automático

Sensores y actuadores virtuales

Los sensores y actuadores virtuales son los módulos que se comunican con el hardware del robot. Los módulos que necesiten acceder a datos proporcionados por los sensores o enviar comandos a los actuadores lo harán a través de ellos.

Los sensores virtuales, definidos para la Arquitectura AD en [36], proporcionan a las habilidades automáticas y a las acciones reflejas información leída de los sensores físicos del robot. Estos sensores informan sobre el entorno del robot (láser, sonar, micrófono, etc.), o su estado interno (odometría, etc.). Los sensores virtuales almacenan la información obtenida de los sensores físicos en un formato adecuado para ser usado por otros elementos.

Los actuadores virtuales envían comandos de movimiento procedentes de las habilidades automáticas o desde las acciones reflejas a los actuadores físicos del robot. Los actuadores virtuales permiten a las acciones reflejas tener prioridad para enviar comandos de movimiento. Cuando se produce una acción refleja, ésta envía comandos a los correspondientes actuadores inhibiendo el envío de comandos procedentes de las habilidades automáticas. Cuando la acción refleja desaparece, las habilidades automáticas pueden enviar de nuevo comandos a los actuadores. Un actuador no sólo recibe comandos de movimiento, también puede recibir comandos para informar sobre su estado interno: velocidad con la que se mueve o posición en la que se encuentra.

Acciones reflejas

Un reflejo es la forma más simple del comportamiento animal. Es una respuesta automática, rápida e involuntaria a un determinado estímulo imprevisto, interno o externo, y su duración depende directamente de la duración e intensidad del estímulo que la causa [13]. Las acciones reflejas son acciones rápidas y prioritarias y no implican la actuación del cerebro, aunque éste puede recibir

información sobre la localización y la naturaleza del estímulo. Una vez que el cerebro es consciente de que se ha producido una acción refleja, éste puede alterarla o inhibirla. De acuerdo con estas ideas, cada acción refleja está asociada con un sistema sensorial. Esto significa que habrá una acción refleja asociada al sonar, otra asociada al sistema de visión, al láser, etc. En la mayoría de los casos, las acciones reflejas se producen cuando la lectura de los sensores exceden un cierto umbral, como por ejemplo, si la lectura del sonar indica que el robot está situado a una distancia peligrosa de la pared, la acción refleja hará que le robot se aleje de ella a tanto mayor velocidad cuanto menor sea dicha distancia. La acción refleja está siempre leyendo los datos procedentes de los sensores. Si el dato que percibe excede un límite, la acción refleja entra en un estado reflejo. En este caso, la acción refleja se conecta con los sensores virtuales para mandarle comandos de movimiento, inhibiendo el envío de comandos procedentes de las habilidades automáticas. Al mismo tiempo, el nivel Deliberativo recibe notificación de la acción refleja. En ese momento, el nivel Deliberativo puede alterar o inhibir la acción refleja a través de las órdenes de ejecución. La acción refleja termina cuando el estímulo reflejo termina o cuando la acción refleja es inhibida por el nivel Deliberativo.

Habilidades Automáticas

Las habilidades automáticas se definen como la capacidad de procesar información procedente de los sensores virtuales y/o generar órdenes de ejecución sobre los actuadores virtuales. Estas habilidades son clasificadas como perceptivas, sensorimotoras y habilidades para el procesamiento de información. Las habilidades perceptivas interpretan la información percibida de los sensores, de las habilidades sensorimotoras o de otras habilidades perceptivas. Las habilidades sensorimotoras son aquellas que producen comandos de movimiento en función de la información percibida de los sensores, habilidades perceptivas o sensorimotoras. Las habilidades para el procesamiento de información no se comunican directamente con los sensores y actuadores.

Todas las habilidades automáticas tienen las siguientes características:

- Pueden ser activadas por habilidades situadas en el mismo nivel o en el nivel Deliberativo. Una habilidad sólo puede desactivar habilidades que ella misma ha activado previamente.
- Pueden almacenar los resultados en la memoria a corto plazo de forma tal que puedan ser utilizados por otras habilidades.
- Pueden generar eventos diversos y los notificarán únicamente a aquellas habilidades que los hayan pedido previamente.

Las habilidades automáticas sencillas de la arquitectura AD constituyen el substrato para la generación de habilidades complejas. Estas habilidades a su vez se pueden combinar para dar lugar a habilidades más complejas. El carácter modular de las habilidades permite utilizarlas fácilmente para construir jerarquías de habilidades con mayores niveles de abstracción. Las habilidades

automáticas no están organizadas a priori. Las habilidades se configuran dinámicamente en función de la tarea a realizar por el robot y de las condiciones del entorno. Con esto se consigue una mayor flexibilidad en los comportamientos del robot.

2.2.3. Tipos de memorias de la arquitectura AD

La arquitectura AD está formada por tres tipos de memoria: memoria sensorial, memoria a corto plazo y memoria a largo plazo. La memoria sensorial almacena información percibida directamente de los sensores, mantiene una detallada información sensorial y tiene un tiempo de vida corta. Esta memoria está contenida en cada uno de los sensores virtuales. Cuando el robot se mueve, ésta adquiere nueva información olvidando la información percibida anteriormente. Dependiendo de la importancia de esta información para el robot, ésta puede ser almacenada en la memoria a corto plazo. Esta información es almacenada en dicha memoria, de una forma más abstracta como resultado del procesado de la información. Sólo almacena información que sea útil al robot para ejecutar una determinada tarea. En la arquitectura AD, parte de esta información está contenida en las habilidades automáticas y deliberativas y otra es almacenada en la memoria a corto plazo, que es común para ambos niveles. En la memoria a largo plazo se almacena el conocimiento duradero. Este conocimiento puede venir del aprendizaje, del procesamiento de la información almacenada en la memoria a corto plazo o puede estar dada a priori. El robot usa este conocimiento para razonar o para tomar decisiones.

3

Descripción del problema

El grupo Roboticslab de la Universidad Carlos III de Madrid realiza su trabajo de investigación sobre plataformas comerciales y sobre desarrollos propios, esto trae como consecuencia la interacción con componentes físicos y lógicos provenientes de diferentes proveedores. En algunos casos se han presentado problemas de integración de componentes que requieren un esfuerzo extra para lograr un correcto funcionamiento entre los diferentes elementos que integran una aplicación.

En este capítulo se presentan las plataformas físicas y lógicas que han sido usadas en el presente trabajo. El contexto se ubica en el área de arquitecturas de control para robot autónomos. El trabajo se ha desarrollado directamente sobre plataformas físicas sin el uso de simuladores. Este último hecho conlleva a la aparición de situaciones que no se presentan en desarrollos sobre sistemas simulados básicos, como lo es la ocupación del robot por otros investigadores, periodos de mantenimiento y reparaciones, duración de las pruebas limitadas por la carga en las baterías, necesidad de un espacio físico para pruebas, trabajo extra en el caso de actualización del hardware y/o software, etc. Por otro lado, el disponer de un simulador hubiese permitido la realización de pruebas que por limitaciones físicas no se pueden realizar sobre robots reales.

3.1. Plataforma física inicial

En la etapa inicial de este trabajo se contaba con tres robots como plataforma de desarrollo y pruebas. Dos robots de la familia B21, y un robot Magellan Pro, ambos fabricados por la empresa iRobots Company. A continuación se describen estas plataformas de desarrollo.

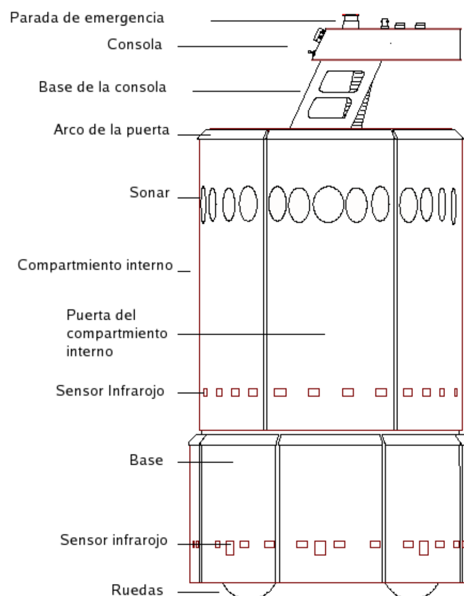

Figura 3.1: Robot B21

3.1.1. Robots B21

El B21 es un robot móvil diseñado para ser utilizado en investigación y desarrollo de aplicaciones en áreas interiores [37]. La figura 3.1 muestra una representación del robot B21. Este robot posee 4 ruedas que son conducidas y dirigidas de manera síncrona. La sección de la base del B21, la cual se extiende hasta la altura de 30 cm., gira de manera síncrona con respecto al ángulo de giro de las ruedas. La base contiene los motores, baterías, ruedas, interruptores de energía eléctrica y otros componentes referentes al movimiento y a la alimentación de energía del robot. Contiene sensores, ordenadores, equipos de comunicaciones de datos, la consola, indicadores, interruptores, así como otros equipos usados en detección, procesamiento y comunicaciones. La consola se encuentra en la parte superior del B21 y está compuesta por interruptores e indicadores que permiten de una manera rápida observar el estado general del funcionamiento del robot, así como activar/desactivar alguno de los componentes principales como lo son los ordenadores contenidos en el cuerpo y ciertos tipos de sensores como lo son las cámaras de vídeo, brújulas, altavoces, etc. Muchos de los B21 llevan dos ordenadores de sobremesa (sólo la CPU), pero también es posible usar un ordenador portátil para ser usado como interfaz del robot. Cada ordenador del B21 vendido por Real World Interface, Inc. [1] posee un sistema operativo de la familia Linux [2] y el software RWI Robotics Appli-

[1] Real World Interface, Inc. (RWII) es la empresa que fabricó y distribuyó los equipos B21
[2] Originalmente se usó la distribución Red Hat 6.1, www.redhat.com

cation Interface (RAI). Cuando hay múltiples ordenadores en el robot éstos son conectados por medio de un concentrador Ethernet IEEE 803.3.

Cada B21 es usualmente acondicionado para extender su funcionamiento con algún equipo especializado. En muchos casos el acondicionamiento es realizado por RWI en el momento en el que el robot comienza a ser ensamblado en la fábrica, en otros casos, es común que los usuarios finales agreguen algunas extensiones para tener su diseño particular de B21.

La mayor parte de las funciones que interactúan directamente con los dispositivos son manejados totalmente en la base, éstas incluyen la odometría y el nivel de las baterías. Además contiene hasta dos ordenadores principales, los sonars y los sensores de luz infrarrojo, los sistemas de distribución de energía, interruptores de energía, cableado, red de datos entre ordenadores.

La consola está colocada en la parte superior del cuerpo, al igual que el botón de activación de los ordenadores, indicadores del funcionamiento de los ordenadores por medio de LEDs, indicadores del funcionamiento de la red Ethernet, el punto de montaje de la antena, cámaras y cualquier otro dispositivo que pueda ser montado en este lugar. El cuerpo y la consola están físicamente unidas, sin embargo, la base es fácilmente separable.

El B21 ha sido usado intensamente como soporte a las pruebas de las diferentes líneas de investigación de RoboticsLab , entre sus últimos usos como robot de ensayo se puede mencionar [38], [36] y [39] entre otras pruebas desarrolladas en RoboticsLab.

3.1.2. Magellan Pro

El RWI Magellan Pro de iRobot Company (antiguamente RWI) es la generación siguiente a los B21, se caracteriza básicamente por su tamaño, su red de campo interna gestionada a través de un sistema propietario sobre rFlex y por estar equipado con un ordenador empotrado con sistema operativo Linux pre-instalado, ver fig. 3.2.

Posee una base circular diámetro de 40.6 cm y 24.5 cm de altura. El sistema sensorial permite obtener información de 360º a su alrededor ya que cuenta con 16 sensores sonars, 16 sensores IR y 16 sensores de contacto. Además, tiene una pantalla LCD para visualizar el diagnóstico y estado del sistema. Para el movimiento cuenta 2 motores DC de 24V y 2 ruedas motrices independientes, lo que le permite girar sobre su propio centro de gravedad.

Este equipo posee las características comunes en muchas de las plataformas

3.2. Plataforma de programación inicial

Los robots inicialmente disponibles son equipos adquiridos a fabricantes externos, estos equipos se caracterizaban por poseer software propio para su

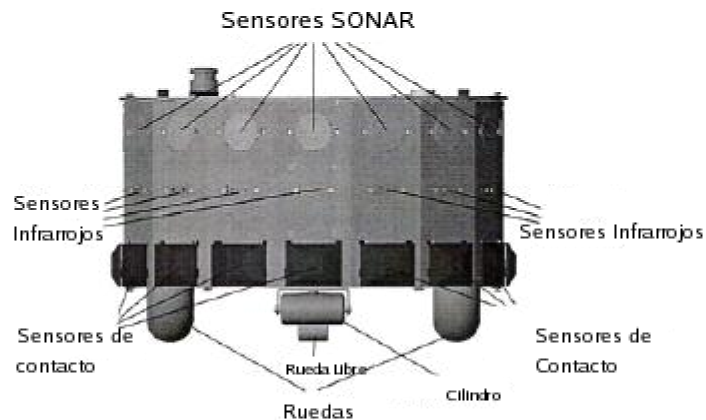

Figura 3.2: Robot Magellan PRO

utilización y experimentación. Este software se caracterizaba por envolver las interfaces directas al hardware de movimiento y detección. La plataforma de programación instalada en los robots disponibles al inicio era Mobility. Esta plataforma usaba los servicios de un gestor de objetos como mecanismo de integración y de comunicación entre procesos sobre un mismo ordenador o varios ordenadores.

3.2.1. Mobility

El software de integración Mobility es un conjunto de herramientas para desarrollo de sistemas distribuidos, orientado a objetos, que permite la construcción de software de control para sistemas de uno o varios robots.

Mobility consta de:

- Una colección de herramientas de desarrollo del software.
- Un software orientado a objetos para usar en el Robot.
- Un conjunto de módulos básicos de control.
- Un framework de clases orientados a objetos para simplificar desarrollo de código.

Mobility define el Mobility Robot Object Model (MROM) diseñado bajo el paradigma Common Object Request Broker Architecture 2 (CORBA 2.0), ver fig. 3.3, con especificaciones en un lenguaje de definición de interfaz estándar soportado por muchos lenguajes de programación y plataformas de cómputo. El Mobility Class Framework usado por Mobility complementa el MROM y simplifica enormemente el proceso de desarrollo de aplicaciones por medio de la reutilización de elementos software básicos del sistema y quedando solo por

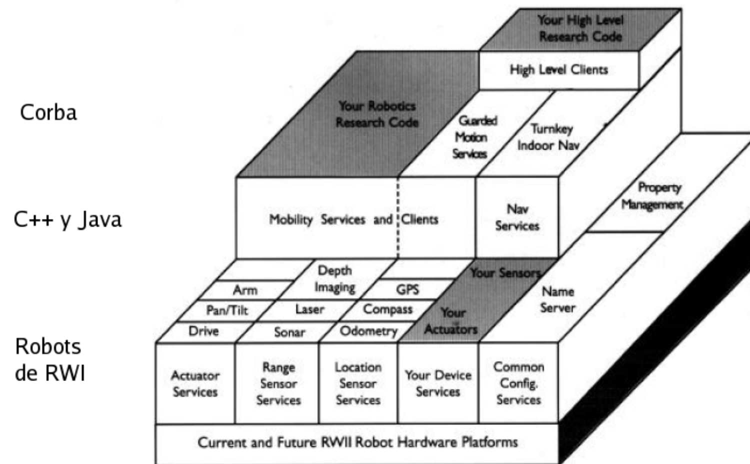

Figura 3.3: Estructura de una aplicación usando Mobility

parte del desarrollador el implementar sus ideas (tomado del manual producto ver [40]).

El Robot Object Model define un *sistema de robot* como un conjunto de objetos distribuidos organizados jerárquicamente. Cada objeto es una unidad software separada con identidad, interfaces y un estado propio. El conjunto de objetos representan una abstracción de todo el robot, incluyendo sensores, actuadores, comportamientos, procesos de percepción y almacenamiento de datos. Los Objetos proveen un modelo flexible de un sistema de robot que puede ser reconfigurado con nuevo hardware, software, nuevos algoritmos y nuevas aplicaciones.

El Mobility Class Framework ofrece una imagen del Robot Object Model y a la vez permite encargarse en gran parte del trabajo laborioso concerniente a la programación del software del robot. Por ser orientado a objetos, es posible la especialización en nuevas clases del Mobility Class Framework. Así se puede agregar fácilmente nuevos componentes software para la representación de sensores, actuadores, comportamientos procesos de percepción y tipos de datos al sistema. Mobility también permite extender al propio Robot Object Model por medio de la definición de nuevas interfaces usando el IDL de CORBA 2. Debido al hecho que todos los objetos del entorno de Mobility soporta un conjunto de interfaces comunes, éstas reciben el nombre de componentes del sistema.

Mobility soporta los lenguajes de programación C++ y Java, pero la versión 1.1 solo soporta el lenguaje C++. El conjunto de herramientas de Mobility opera sobre el kernel de Linux 2.x. Mobility también es compatible con Java 1.1 y usa Java para habilitar el uso de interfaces para la configuración, gestión, pruebas y visualización del sistema del robot mientras éste está en pleno funcionamiento.

3.2.2. RFlex

El sistema rFlex es el encargado de gestionar los dispositivos físicos empotrados de los robots de la familia Magellan Pro. Este sistema posee una interfaz visual propia para indicar al usuario el estado de los dispositivos, así como el envío de comando por medio de menúes de selección. El Magellan Pro posee los siguientes componentes:

- Módulo de posición, que gestiona la odometría y control de los motores diferenciales de traslación.
- Módulo de sonars, que se encarga de controlar el disparo/recepción de las señales acústicas para la detección de objetos físicos.
- Módulo de sensores de infra-rojos, al igual que los sonars se encarga de la detección de objetos físicos por medio de diodos infra-rojo.
- Módulo de sensores de contacto, detectan si un sensor de contacto ha sido pulsado.
- Módulo de potencia, encargado de supervisar la carga de las baterías del robot y en caso de baja carga iniciar el proceso de apagado automático del robot.

Este sistema se comunica con el ordenador empotrado del robot por medio de un puerto serio RS-232-C, lo que permite su interacción desde un programa construido con herramientas externas a las entregadas por el fabricante. Para la construcción de un manejador externo que interactue con rFlex se debe conocer los comandos, parámetros y respuestas, desafortunadamente este manual no ha estado disponible. A pesar de esto, algunos grupos han desarrollado herramientas utilizando técnicas de ingeniería inversa, como el caso de los manejadores de rFlex construidos por Toby Collet para el proyecto Player/Stage [41, 42].

3.3. Plataforma física actual

Uno de los objetivos planteados del grupo de investigación fue el diseño de un nuevo robot cuyo sistema de control que siguiese el paradigma AD. El objetivo era crear un robot para soportar la investigación en el área de robots personales e interacción humano robot. El robot base debería disponer de la capacidad de movimiento, detección de obstáculos y sobre todo poder soportar nuevos dispositivos como son sensores de tacto, sensores de sonido, visión, requeridos para desarrollar habilidades de interacción más cercana al ser humano. Antes de comenzar la construcción de la parte superior del robot se estudiaron diferentes prototipos de la apariencia física, pensando en que muchos de sus usuarios serían niños. Los prototipos del nuevo robot han sido resumidos en [43]. Después de discusiones técnicas y de aspecto externo se tomó la decisión de construir el prototipo Maggie mostrado en la figura 3.4.

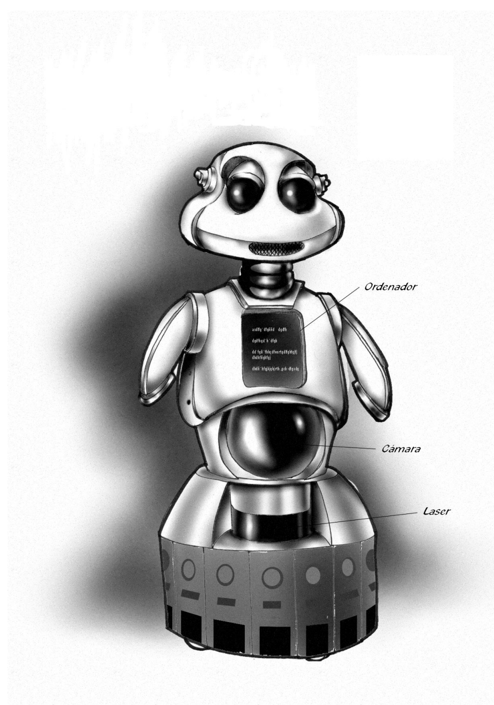

Figura 3.4: Diseño inicial del robot Maggie

3.3.1. Maggie versión inicial

En la versión inicial de Maggie se utiliza el robot Magellan Pro como robot base. La decisión de la selección del robot base se justifica en el tamaño de mismo, ya que permitía la modificación de la parte superior del Magellan Pro para poder colocar lo que sería el cuerpo del robot. La parte superior de este robot, ver fig. 3.5 está diseñada para colocar dispositivos como Láser, GPS, cámaras de vídeo, etcétera. El principal inconveniente de dicha selección es que las versiones de Mobility instaladas en el Magellan Pro ya no contaban con el soporte del fabricante, por lo que la actualización y el soporte de nuevo hardware/software no estaban garantizados. Por otro lado, la documentación de las herramientas de desarrollo de software no estaban disponibles. Por lo que fue necesario la utilización del manejador rFlex suministrado por el proyecto Player/Stage.

Esta versión del robot estuvo en uso por casi 12 meses, ya que fue necesario corregir el sistema de potencia para que soportase mayores requerimientos de corriente eléctrica para alimentar dispositivos adicionales.

3.3.2. Maggie versión actual

Las guías del diseño de la segunda versión trata de mantener el aspecto externo del robot, pero se realiza un cambio profundo en los componentes hardware

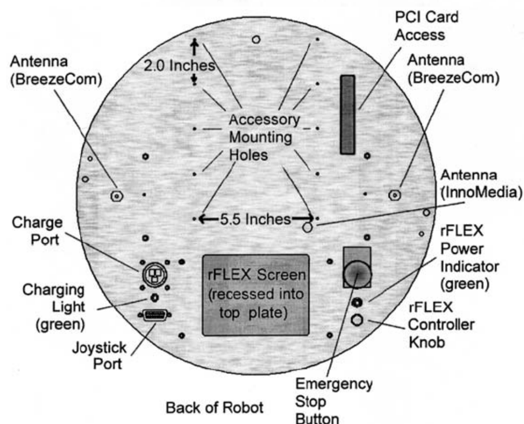

Figura 3.5: Parte superior del Magellan PRO

y software.

Entre las principales novedades se descarta el uso de un robot base fabricado por terceros, por lo tanto, se decide la construcción de la base en el mismo roboticslab, la descripción detallada de los componentes se encuentra en [44].

Se puede decir que el robot Maggie[45] es un robot personal desarrollado en RoboticsLabs, su principal misión es servir de soporte a la investigación sobre interacciones Humano - Robot. Desde el punto de vista hardware el robot cuenta con:

- Un ordenador portátil para procesamiento de imágenes y audio.
- Un ordenador portátil para control de motores, láser, sonars y software de control.
- Un ordenador portátil del tipo tablet PC con pantalla táctil.
- Concentrador Ethernet con interfaces 802.2 y 802.11.
- Chapa exterior de fibra de vidrio con sensores de tacto incorporados.
- Sensores de temperatura, sensor láser, cámara de vídeo digital, micrófonos, altavoces.
- Motores y servomecanismos para movimientos de brazos y cabeza.
- Motores de traslación por medio de ruedas diferenciales.
- Sistema de control de potencia y apagado de emergencia.

La figura 3.6 muestra la apariencia hoy en día de Maggie. Los desarrollos actuales de hardware están completos, por lo que se espera que su aspecto exterior no varíe significativamente.

Figura 3.6: Robot Maggie

El desarrollo de este proyecto de investigación proveerá a Maggie de un sistema de control de robot capaz de soportar los proyectos de investigación para la cual fue diseñada y construida.

3.4. Razones de migración

La línea original de desarrollo se basó en la idea de usar la plataforma Mobility como herramienta de trabajo. Como se ve en la descripción de Mobility, éste ofrecía una gran variedad de servicios para el desarrollo de arquitecturas de control de robots móviles y era soportado por lenguajes como C++ y JAVA. Otro punto a favor de Mobility es el usar como gestor de objetos a CORBA, que en los primeros años de esta década se percibía como un estándar de facto, ver [46].

Desafortunadamente el punto en contra de Mobility es su carácter privado. Es un software soportado por una empresa, y además todo el código era cerrado. Cuando la empresa iRobots decide abandonar el producto, los grupos de investigación y desarrollo que utilizaban Mobility pierden el soporte.

CORBA también ha experimentado un retroceso en los últimos años, a pesar de ser considerada la plataforma más confiable a final de los 90, ahora pierde terreno al observarse cada día más entornos débilmente acoplados donde la

tecnología de los Web Services toma cada vez mayor auge.

Antes de iniciar este trabajo se hizo un primer intento que consistía en descartar Mobility y gestionar todo el desarrollo por medio CORBA usando un diseño de clases propio inspirado en Mobility. Desafortunadamente la dependencia entre Mobility y el hardware de iRobots impedía hacer esta separación de una manera sencilla. Se requería una plataforma software independiente del hardware, y hardware construido para ser usado por medio de software abierto.

Los dos hechos anteriores llevaron a pensar seriamente en la búsquedas de alternativas que se ajusten a un desarrollo de una arquitectura de control basada en componentes software no privado. Es importante también pensar en la integración con nuevas tecnologías emergentes basadas e integradas por sistemas dispersos interconectados por redes de ordenadores.

En el siguiente capítulo se estudian las alternativas para el desarrollo de la arquitectura de control de robots personales.

4

Estado del Arte

En este capítulo se presentan las herramientas más recientes para el desarrollo de software de robots personales. Estas herramientas se caracterizan en que la información de éstas está disponible al público. La información e imágenes presentadas en este capítulo han sido obtenida de artículos, ponencias y páginas Web que hacían referencia a cada una de las herramientas.

Todas las herramientas se han desarrollado en proyectos impulsados por corporaciones privadas, mixtas o grupos de investigadores. Se caracterizan por contener un entorno de desarrollo, una biblioteca de componentes con funcionalidades especificas y un paradigma particular de diseño y desarrollo.

Algunas presentan una orientación para actividades específicas, como lo es la navegación, simulación, visión artificial, aplicaciones militares, aprendizaje, etc, otras en cambio son de uso general. Unas han sido implementadas bajo licencias que permiten su libre distribución, otras en cambio tienen licencias de uso privado que deben ser adquiridas. La mayoría pretenden ser usadas en robots comerciales e integrarse con otras herramientas ya disponibles.

El orden de presentación es aleatorio y no indica su relevancia. Al final del capítulo se presenta un resumen de los proyectos presentados.

4.1. El Proyecto Orocos.

El proyecto Open Robot Control Software (Orocos) es una de las principales alternativas para construir aplicaciones de control de robots y máquinas automáticas. Sus orígenes datan de Diciembre de 2002. El objetivo principal fue construir un conjunto de bibliotecas software orientado a los sistemas de control para robots en general.

La razón principal de la iniciación del proyecto, se debió a las pocas alternativas de *tecnología abierta* disponibles, en ese momento, para el desarrollo de arquitecturas de control para robots. La propuesta se difundió por medio de

Figura 4.1: Logo de OROCOS, una formado por dos pinzas de robot.

la lista de correo de la Red Europea de Robótica (EURON). Después de los primeros intercambios de opiniones entre los miembros interesados se produjo el primer documento formal conocido como IST Project Fact Sheet [47]. En este documento se describen los objetivos y la especificación del proyecto inicial. Éstos son mostrados a continuación.

4.1.1. Objetivos.

Los objetivos mostrados a continuación son los iniciales del proyecto Orocos, con una duración de 24 meses a partir de 01 de Septiembre de 2001.

- Producir una base funcional general para el software de control de un robot.
- Debe tener licencia *open source*, como por ejemplo la *GNU General Public License*.
- Ser independiente de las plataformas de cómputo. Por ejemplo debe poder funcionar en la mayoría de hardware, o ser portable de la manera más cómoda posible.
- Debe ser modular y distribuido. Como por ejemplo usar los estándares de CORBA para componentes distribuidos.
- Debe seguir estándares internacionales.
- Deber ser capaz de integrarse y reusar código abierto de proyectos complementarios relevantes. Por ejemplo, herramientas de visualización 3D, métodos numéricos, etc.

4.1.2. Desarrollo del proyecto

El proyecto tiene tres líneas principales: la identificación de los componentes principales y su interacción; el diseño e implementación de los componentes; documentación.

Es importante destacar que la meta del proyecto era crear una base funcional, y no el desarrollo completo de un sistema software para un robot.

El proyecto Orocos se ha dividido en varios sub-proyectos. El primero mantiene el nombre original de Orocos y se orienta hacia el software de control en tiempo real. El segundo llamado Orca[48], y un tercero llamado Orocos::SmartSoft[49] mantenido por un grupo de investigadores en Hochschule Ulm University of Applied Sciences, estos dos últimos no son orientados a aplicaciones en tiempo real.

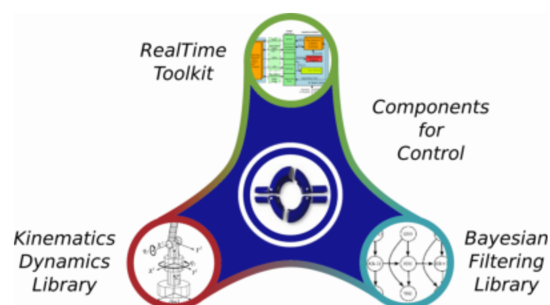

Figura 4.2: Componentes de Orocos [1].

En [1] se puede encontrar los componentes de sus líneas actuales de desarrollo. Éstas se caracterizan por cuatro bibliotecas principales en C++: la caja de herramientas (toolkit) para Tiempo Real, la biblioteca de la parte Cinemática y Dinámica, la biblioteca de software de filtros Bayesianos y la biblioteca de componentes de Orocos.

- La caja de herramientas para Tiempo Real (RTT) no es una aplicación por si misma, pero provee la infraestructura y las funcionalidades para construir aplicaciones robóticas en C++, destacando el uso en aplicaciones en tiempo-real e interactividad en línea basada en componentes.
- La biblioteca de componentes Orocos (OCL) provee componentes de control listos para usarse. Los componentes para la gestión y control del hardware pertenecen a esta biblioteca.
- La biblioteca de Cinemática y Dinámica es una biblioteca en C++ la cual permite calcular cadenas cinemáticas en tiempo real.
- La biblioteca de software de filtros Bayesianos proveen funcionalidades independientes para la inferencia de redes Bayesianas dinámicas. Por ejemplo, algoritmos basados en leyes de Bayes para procesar y estimar información, tales como filtros de Kalman, filtros de partículas, etc.

En [50] se detalla la información sobre aplicaciones construidas sobre la plataforma Orocos.

En los últimos años el proyecto Orocos ha permitido la creación de varios subproyectos. El primero mantiene el nombre original de Orocos y se orienta hacia el software de control en tiempo real. El segundo llamado Orca[48], y un tercero llamado Orocos::SmartSoft[49] mantenido por un grupo de investigadores en

Hochschule Ulm University of Applied Sciences, estos dos últimos proyectos no son orientados a aplicaciones en tiempo real.

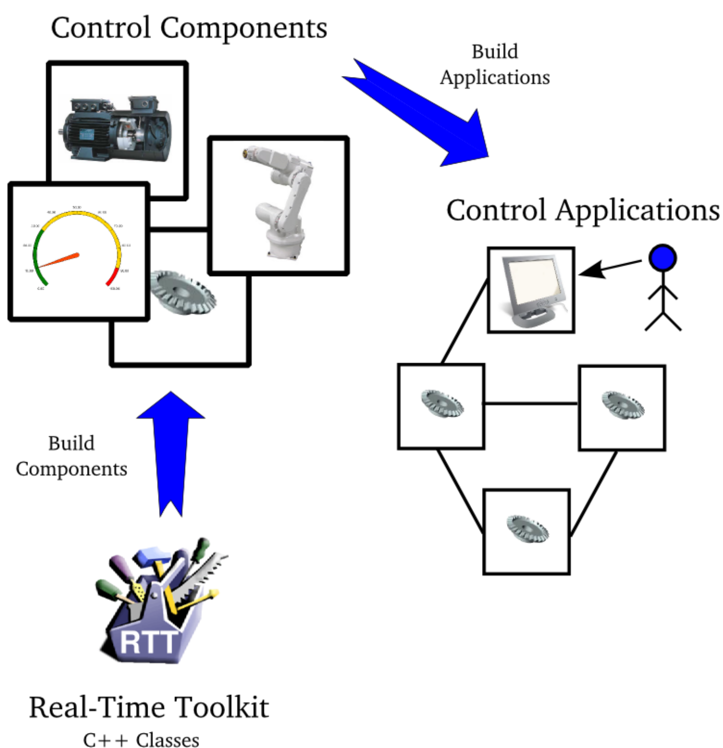

Figura 4.3: Caja de herramientas de Tiempo Real de Orocos [1].

4.1.3. Construcción de aplicaciones en Orocos.

Las aplicaciones en Orocos son compuestas por componentes software, los cuales forman un red de aplicación específica. Cuando se usa Orocos, se puede elegir los componentes predefinidos, los componentes aportados por la comunidad o construir componentes propios, usando la caja de herramientas de tiempo real.

Plantillas de aplicaciones.

Una *plantilla de aplicación* es un conjunto de componentes que operan de forma conjunta. Por ejemplo, la platilla de aplicación para control de movimiento contiene componentes para planificación de rutas, control de posición, acceso al hardware y reportes de datos.

Una plantilla de aplicación debe ser simple de manera que cualquier usuario de Orocos pueda usarla y modificarla.

Componentes de control.

Las aplicaciones son construidas usando *componentes de control* de Orocos. Los componentes de control son entidades distribuidas que poseen una interfaz orientada al control.

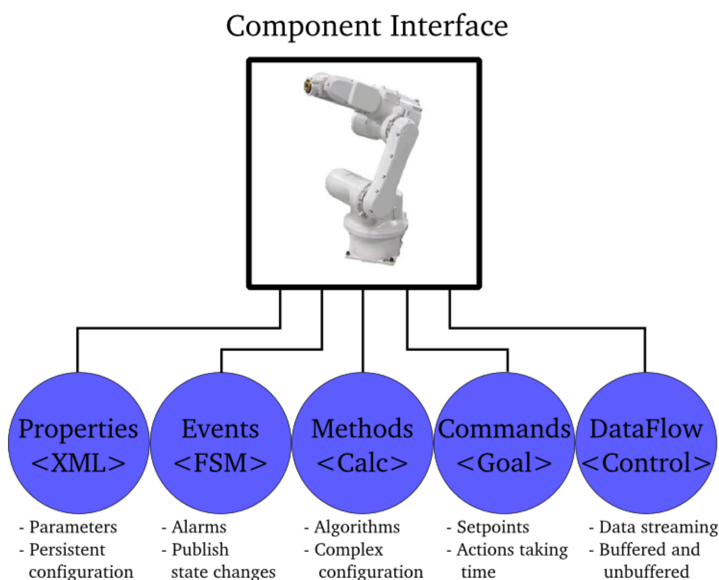

Figura 4.4: Interfaz a componentes de control de Orocos [1].

Un componente simple debe ser capaz de controlar una máquina totalmente, o ser una *pequeña* parte en una red de componentes, por ejemplo un interpolador o un componente cinemático. Los componentes son construidos con la caja de herramientas de Tiempo Real y opcionalmente hacen uso de otra biblioteca, como lo son las biblioteca de visión o la biblioteca de cinemática. Para configurar las aplicaciones se describen comúnmente las propiedades vía XML o usando métodos.

Hay cinco formas posibles de usar las interfaces de un componente de Orocos: a través de sus propiedades, eventos, métodos, comandos y flujos de datos, mostrado en la figura 4.4. Todas estas interfaces son opcionales.

4.1.4. Observaciones.

En [51] H. Bruyninckx y otros, indican que los objetivos de diseños de los componentes de control son: mínimo acoplamiento, separación entre estructura y funcionalidad y una interacción orientada a eventos. Es importante destacar el uso de CORBA como plataforma de middleware, específicamente la implementación conocida como TAO [52] que es una implementación de Corba para tiempo Real. Como mecanismos para modelar el comportamiento de los componentes usan máquinas de estados finitos, ver A.2.

4.2. Proyecto Player & Stage.

El proyecto *Player & Stage* es una plataforma que provee una abstracción para diferentes robots, se caracteriza por el uso de software libre, de hecho Player & Stage es software construido y distribuido bajo la licencia GNU General Public License. En las anuncios mostrados en su página web [53], indican que para Octubre de 2006 se habían registrado 60.000 descargas vía Internet. Este hecho les lleva a decir que el servidor Player es las interfaz de control más usada en el mundo. Sus herramientas de simulación Stage y Gazebo, son también ampliamente usadas.

En la primera publicación sobre Player[54], se le presenta como una herramienta para proveer de forma transparente y por medio de una red IEEE 802.11 acceso a todos los sensores y actuadores que conforman un robot. Los trabajos iniciales se hicieron sobre robots en el Robotics Research Labs de la Universidad del Sur de California. Todos los robots poseían una interfaz TCP/IP sobre una red inalámbrica 802.11, una configuración común en muchos laboratorios del mundo. La comunicación entre los robots en el laboratorio era fácil y económica, ya que soportaba el uso de sockets BSD [55]. Esto permitía que el mismo mecanismo que movía al robot desde el hardware local, lo pudiera hacer desde una estación de trabajo en la red, o desde Internet.

En la publicación antes mencionada se indican las tres principales razones para usar un robot-servidor basado en sockets:

Distribución. Un cliente tiene acceso a los sensores y actuadores desde cualquier lugar de la red. Los clientes pueden conectarse a múltiples servidores, y los servidores aceptan conexiones de múltiples clientes. Un solo programa puede controlar el comportamiento de varios robots. Diferentes programas pueden controlar diferentes comportamientos de un robot.

Independencia. Los clientes pueden ser escritos en cualquier lenguaje y sobre cualquier plataforma hardware que implemente sockets. El usuario puede elegir el lenguaje mas apropiado y el entorno para crear la aplicación.

Conveniencia. El servidor provee una interfaz abstracta única a todos los dispositivos en este. Los programas clientes se *suscriben* a un conjunto de dispositivos y especifican la frecuencia en que serán capturados los datos.

4.2.1. Objetivos de diseño en Player.

El Proyecto Player tiene objetivos muy claros, además, está orientado al desarrollo de aplicaciones sobre robots, cuya plataforma de cómputo es un ordenador único con sistema operativo Linux. Los dispositivos interconectados al robot deben tener manejadores disponibles en Player. Los manejadores de dispositivos disponibles son robots de la familia Pioneer, dispositivos láser vendidos de la marca Sick, modelos PLS y MLS, así como varias cámaras de vídeo, etc. Para obtener una lista actualizada de los dispositivos reconocidos por Player es

preferible visitar la página Web del Proyecto. Los objetivos de diseño de player se han publicado en [56]. A continuación se presenta un resumen de los mismos:

Interfaz con el cliente. Player es un servidor de dispositivos basados en BSD Sockets, que permite controlar una gran variedad de sensores y actuadores. Player se ejecuta en una máquina que es físicamente conectada a una colección de dichos dispositivos, y ofrece una interfaz por medio de un socket TCP a los clientes que desean controlar a estos. Los Clientes se conectan a Player y se comunican con los dispositivos por medio del intercambio de mensajes con player a través del socket TCP. Player trabaja de una forma similar a un servidor de dispositivos de UNIX. Al igual que otros servidores, Player puede soportar múltiples clientes, cada uno con diferentes servidores. Debido a que la interfaz externa de Player es un socket TCP, los programas clientes pueden ser escritos en cualquier lenguaje de programación que implementes los sockets TCP. En algunas aplicaciones es conveniente el uso de sockets UDP, se espera que las futuras versiones de player permitan el uso de este tipo de sockets.

Modelos de dispositivos. Para proveer una abstracción de una variedad de dispositivos, player sigue el modelo UNIX de tratar los dispositivos como ficheros. Esto permite el uso de primitivas básicas para usar un dispositivo, con las diferencia que no implementa las lecturas/escrituras con bloqueo. Eso permite que múltiples clientes pueden estar leyendo o colocando mensajes en los dispositivos.

4.2.2. Stage.

Stage[2] es un simulador de entornos de poblaciones de robots móviles, sensores y objetos en mapas de bits bidimensionales. Stage ha sido diseñado como herramienta de investigación en sistemas autónomos multiagentes, con el fin de proporcionar modelos de varios dispositivos que pretende comportarse como cualquier dispositivo con gran fidelidad. El usuario emplea los dispositivos de la misma manera que lo haría con Player, es decir, los clientes usan la misma interfaz con los modelos que la que usan con los dispositivos reales.

En [56] se describen los principales objetivos en el diseño de Stage:

- Posibilitar el desarrollo rápido de controladores sin la eventualidad de tratar directamente con robots reales.
- Permitir experimentos con robots sin tener acceso al hardware ni a entornos reales.
- Experimentar con nuevos dispositivos que pueden no haberse construido aun.

Stage fue específicamente diseñado para soportar la investigación en sistemas de múltiples robots. Cuando se programa y se experimenta con múltiples robots los beneficios del desarrollo rápido se multiplican. Stage permite realizar

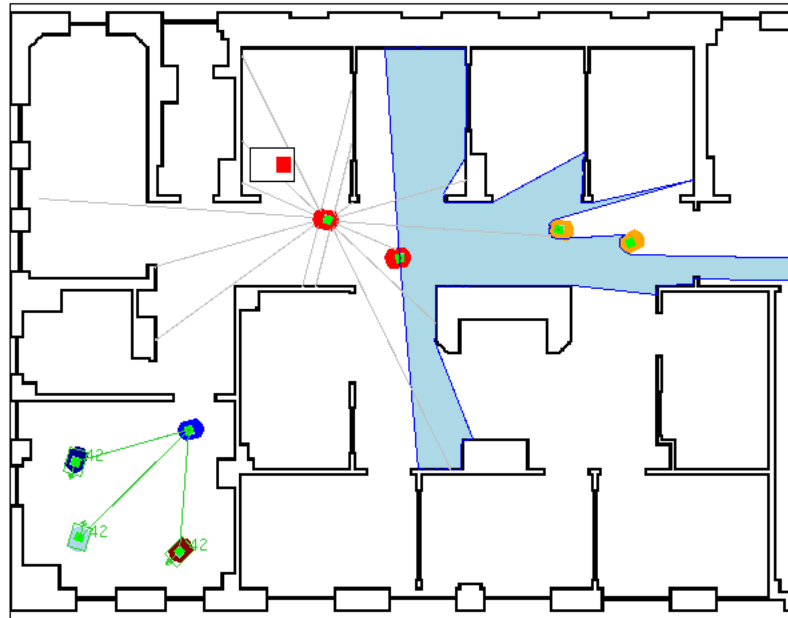

Figura 4.5: Simulación con Stage [2].

experimentos con un gran número de robots cuya compra y mantenimiento serían difícil de financiar. En la publicación antes mencionada se indican algunos aspectos que permiten a Stage su uso en sistemas de múltiples robots:

Buena fidelidad. Stage proporciona modelos computacionalmente económicos de gran variedad de dispositivos, cuyo comportamiento es fielmente simulado. Los modelos pocos exactos puede ser una gran desventaja cuando se diseñan controladores para robots, que luego deben incorporarse a robots reales. Las bajas demandas de cómputo pueden simular muchos dispositivos en ordenadores accesibles económicamente.

Escalabilidad lineal con respecto a la población. Todos los modelos de los sensores usan algoritmos que son independientes del tamaño de la población. Por lo tanto, los requerimientos computacionales de Stage aumentan linealmente conforme aumenta el numero de robots en la simulación.

Adaptabilidad Inicialmente, varios sensores y actuadores están disponibles en Stage. Los modelos, son menos complejos y más flexibles que cualquier pieza específica de hardware. Por lo que cada modelo es configurado para conseguir la mayor aproximación al dispositivo a simular (real o imaginario).

Interfaz con Player. Todos los sensores y actuadores son disponibles por medio de interfaces estándares de Player. Típicamente, los clientes no distinguen la diferencia entre los dispositivos reales en el robot y sus equivalentes simulados en Stage. Esto es debido a que Stage hereda la flexibilidad de de Player.

4.2.3. Gazebo.

Gazebo[3] es un simulador de múltiples robots para entornos externos. De la misma forma de Stage, éste es capaz de simular poblaciones de robots, sensores y objetos, pero en un entorno de tres dimensiones. Esto genera una realimentación más real en los sensores y mejores interacciones entre los objetos simulados. En la imagen 4.6 se muestra una imagen de la interfaz gráfica de Gazebo.

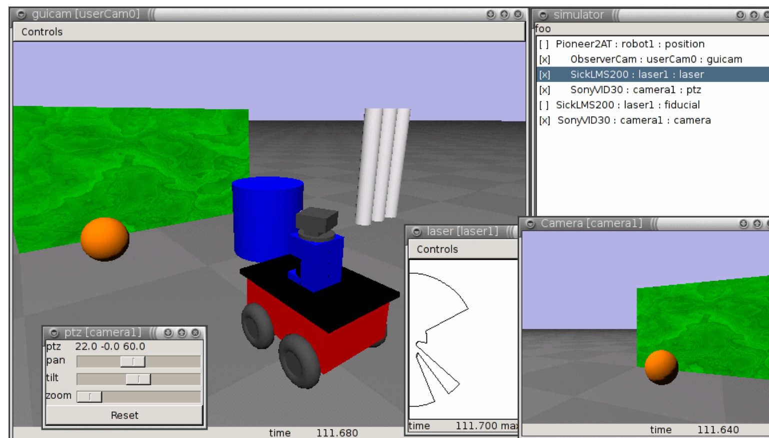

Figura 4.6: Imagen de la interfaz de Gazebo[3].

Entre las principales características de Gazebo se tienen:

- Simulación de los sensores estándar de un robot, incluyendo sonars, sensores láser, GPS y cámaras estéreo o monocular.
- Modelos de los robots Pioneer2DX, Pioneer2AT y SegwayRMP.
- Simulaciones realistas de objetos de cuerpo rígido. Los robots pueden empujar cosas, cogerlas y de forma general interactuar con el entorno simulado de una forma más real.

4.2.4. Observaciones.

El proyecto Player & Stage es tal vez la plataforma ideal para robots cuyas plataformas de cómputo usan sistema operativo Linux. Tienen una alta y activa comunidad dispuesta a compartir sus experiencias entre sus pares.

4.3. Proyecto CARMEN.

En [57] se presenta *The Carnegie Mellon Navigation (CARMEN) Toolkit*. El proyecto se inicia con el fin de proveer una interfaz consistente y un conjunto básico de primitivas, con el fin de soportar la investigación en robótica sobre

una amplia variedad de robots comerciales. Entre los objetivos principales del proyecto CARMEN se encuentra facilitar la incorporación de nuevos algoritmos sobre robots reales o simulados y para permitir compartir nuevos hallazgos y algoritmos entre diferentes instituciones. Los creadores de Carmen han hecho esfuerzos por permitir que el producto permita la inclusión de aportes investigadores colaboradores. Por esa razón, en el proyecto se ha elegido no adoptar estándares de forma estricta, pero se ha hecho un gran esfuerzo en usar las llamadas *buenas practicas de diseño.*

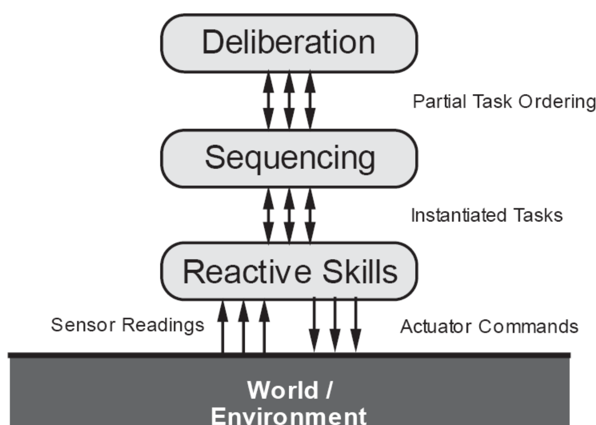

Figura 4.7: Arquitectura de 3 capas usada por Carmen[4]

CARMEN es un software modular, organizado en una arquitectura de tres capas,ver figura 4.7, propuesta en [4]. La capa base se encarga de la interacción y control del hardware. La capa base también se encarga de integrar la información proveniente del sensor de odometría, y además, de la detección de colisiones. Los módulos de la capa base de CARMEN pueden controlar varios robots comerciales incluyendo el modelos Scout y XR4000 de Nomadic Technologies, Pioneers de ActivMedia, ATRV y b21 de iRobots. CARMEN también provee una plataforma uniforme de simulación para todas las bases comerciales[1].

La capa de navegación implementa primitivas de navegación intermedia, incluyendo localización, seguimiento dinámico de objetos y planificación de movimientos. A diferencia de muchos otros sistemas de navegación, el control de movimiento no divide la planificación de alto nivel (estratégica) y la planificación de bajo nivel (táctica) para evitar obstáculos. Por el contrario, todo el control de movimiento de bajo nivel está integrado en un solo módulo. Esto permite una planificación de movimientos más confiable, a un costo computacional razonable.

Finalmente, la tercera capa es reservada para tareas a nivel de usuario, empleando primitivas de los servicios de la segunda capa. CARMEN también contiene una capa separada de componentes no autónomos, incluyendo módulos de despliegues visuales, editores, etc.

CARMEN está disponible a través de la página web de proyecto [58]. Este

[1]Información tomada de [4]

servicio ha permitido que sea usada por grupos de investigación alrededor del mundo en aplicaciones tales como robots asistenciales, exploración minera y sistemas de múltiples robots trabajando en forma cooperada.

4.3.1. Uso de buenas prácticas en el diseño de CARMEN.

CARMEN fue diseñada siguiendo tres objetivos:

Fácil de usar. El monto de tiempo necesario para aprender a como usar CARMEN debe ser pequeño. Las capacidades básicas de CARMEN deben ser fáciles de poner a funcionar en el robot.

Extensibilidad. Los nuevos programas de CARMEN deben ser fáciles de escribir. Los programas que conforman el núcleo de CARMEN deben servir como una sólida base para construir los capacidades de alto nivel. Debe ser fácil de reemplazar los módulos de CARMEN existentes.

Robustez. Los programas de CARMEN deben ser robustos ante una gran variedad de fallas, incluyendo fallas en las comunicaciones y fallas de otros programas.

Para alcanzar dichos objetivos los diseñadores de CARMEN se plantearon el uso de buenas prácticas. A continuación se resumen los siete aspectos de buenas practicas encontrados en CARMEN:

Modularidad.

CARMEN fue diseñado siguiendo una arquitectura software modular, en el cual cada principal funcionalidad se construye como un módulo separado. Los módulos se comunican unos con otros sobre un protocolo de comunicación llamado IPC, desarrollado por Reid Simmons en la Universidad Carnegie Mellon [5]. A pesar que la modularidad en algunos casos puede producir sistemas con niveles altos de consumo de ciclos de procesador, tiene ventajas importantes sobre los sistemas monolíticos:

Flexibilidad. Los robots usualmente presentan varias configuraciones. Un usuario puede mezclar diferentes bases de robots y sensores, simplemente ejecutando los módulos apropiados. Esto elimina la necesidad de acomodar todas las configuraciones hardware posible en un solo proceso.

Soporte de red. Todos los módulos no necesitan ser ejecutados en el mismo procesador. Procesar tareas de forma intensiva puede crear un desempeño bajo en el comportamiento del robot.

Confiabilidad. Si un modulo falla, esto no causa que el resto de módulos falle.

Extensibilidad. Es mucho mas fácil modificar componentes del robot, si cada componentes está autocontenido.

La modularidad incrementa la extensibilidad y el fácil uso, en que los diferentes módulos que ejecutan la misma función (por ejemplo, misma función para diferentes bases) deben converger a un única interfaz abstracta.

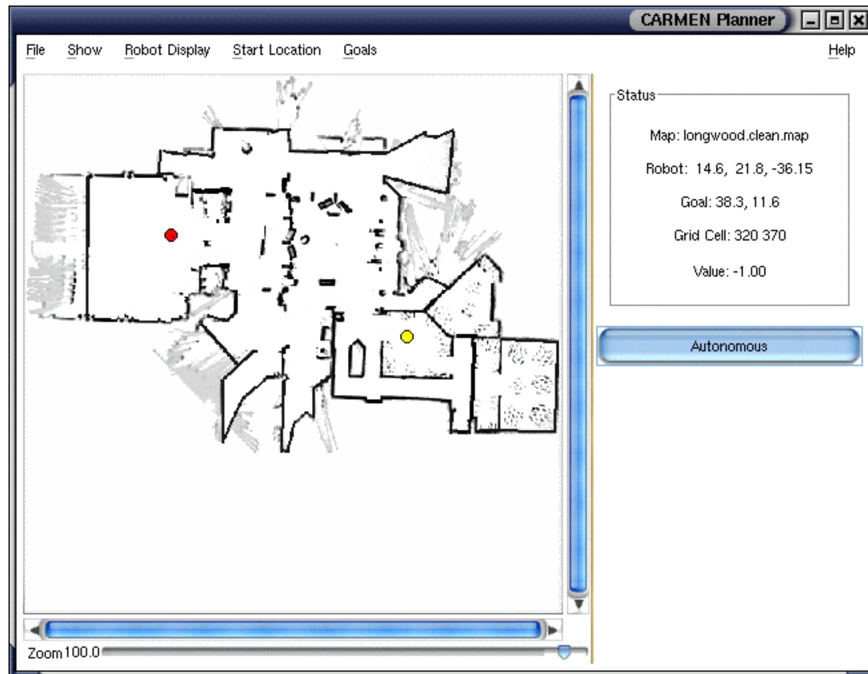

Figura 4.8: Interfaz gráfica en Carmen[5]

Núcleo compuesto por módulos simples.

Los diseñadores han creado un conjunto de módulos que conforman el núcleo central de CARMEN, éstos proveen un conjunto simple de primitivas para la navegación. Estas primitivas (control de la base, seguimiento y planificación de trayectorias) deben servir como un base consolidad para construir capacidades de mayor nivel. Muchos paquetes software para robots presentan múltiples funcionalidades en un mismo módulo. La aproximación de CARMEN es restringir fuertemente el número de funcionalidades en los módulos del núcleo.

Provee un pequeño conjunto de funciones en el conjunto de módulos del núcleo, diseñadas para alcanzar metas únicas, en módulos simples, que deben ser fáciles de entender y fáciles de realizar. La búsqueda de errores se incrementa proporcionalmente al tamaño de los módulos.

Separación del control de los despliegues gráficos.

Un importante principio de diseño de CARMEN es la separación del control del robot de los sistemas de despliegue de gráficos. De hecho se ejecutan en

procesos separados. La información es comunicada entre los módulos de control y los sistemas de despliegue usando protocolos estándar de comunicación.

Este principio de diseño es conocido como el *modelo vista/controlador*, y fue elegido por los desarrolladores de CARMEN para incrementar la transparencia en los módulos. Los despliegues gráficos embebidos pueden llevar a situaciones donde el estado del software es visible al usuario, pero inaccesible a otros programas. Esto no es posible, aunque fuese deseable, para poder tener una completa transparencia (por ejemplo, observar el contenido de todas las variables y estructuras de datos). Sin embargo, en CARMEN se encontró que separar los despliegues gráficos de los algoritmos implementados permite un grado mayor de transparencia. Esto asegura que la información de un módulo pueda estar disponible a todos los otros módulos. Adicionalmente usando el modelo vista/controlador permite los despliegues gráficos distribuidos. Un solo despliegue gráfico puede ser usado para desplegar el estado de múltiples robots, donde el despliegue de todos los robots esa distancia.

Comunicación abstracta.

Durante el desarrollo de CARMEN se empleó diferentes protocolos de comunicación entre diferentes módulos. Una lección de esta experiencia fue que los desarrolladores estaban expuestos a las diferentes peculiaridades de cada protocolo de comunicación. Una fuerte integración de los protocolos de comunicación y de los módulos del núcleo, hace que la actualización de los paquetes de comunicación a menudo sea una tarea inviable. Por esas razones se aseguró que todas las funcionalidades de comunicación en CARMEN estuviesen ocultas detrás de interfaces abstractas.

Todas las funciones generan tráfico extra en la comunicación entre módulos separados en procesos diferentes. Las llamadas a estas funciones debe ser transparentes a los protocolos que en realidad se están usando en dicha comunicación. Todos los módulos incluyen una biblioteca de interfaces separada que oculta el proceso de subscripción. En el momento de actualizar el protocolo de comunicación, solo el código en la interfaz de salida y en la interfaz de entrada debe ser cambiada. Todo el resto del código del robot debe permanecer sin cambios.

Interfaces abstractas al hardware.

CARMEN agrupa el hardware en bases de robots y sensores.

Interfaces a las bases. CARMEN soporta una amplia variedad de bases de robots comerciales. Desafortunadamente, estas bases difieren no solo en la forma, sino también de sus interfaces software. Algunos robots aceptan comandos para asignar la velocidad en las ruedas, otros en cambio, aceptan comandos de traslación y rotación. Algunos robots reportan la odometría, otros en cambio ejecutan la integración en el hardware y la reportan a partir de una posición arbitraria.

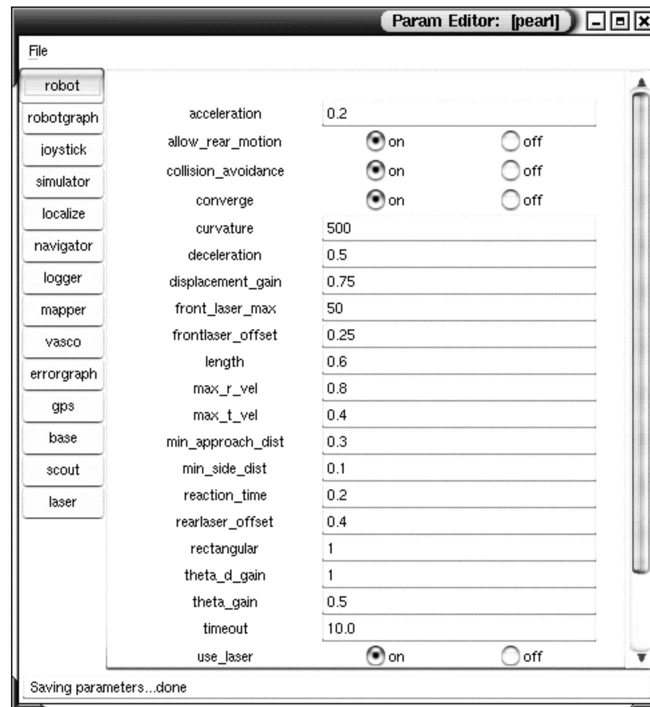

Figura 4.9: Editor de parámetros en Carmen[5]

Para asegurar la uniformidad entre plataformas, CARMEN soporta un conjunto estándar de funciones de interfaz, y ejecuta las transformaciones para la interfaz apropiada de la plataforma.

La principal desventaja es que algunas clases de movimientos no son soportadas, como rotaciones de la parte superior independientemente de la parte inferior.

Interfaces a lo sensores. Las interfaces a los sensores tratan de convertir las medidas de los sensores en un formato único, considerado por los desarrolladores como el más conveniente. Los datos son mantenidos con las menores transformaciones posibles. Los módulos modificaran los datos dependiendo del la funcionalidad a ejecutar.

Entre los sensores solo existen dos grupos de sensores: sensores de posición (GPS y sistemas de navegación inercial) y sensores de rango (sensores láser). Los datos de cada sensor están disponibles en todos los módulos por defecto. Cada dato de un sensor abstracto se componen de datos de odometría y datos del sensor de posición.

Coordenadas y unidades estandarizadas.

Un importante aspecto que a menudo es ignorado en las arquitecturas de robots son las concernientes a la forma de representar las coordenadas. CARMEN asume que todas las medidas se encuentran en el *Sistema Internacional*

de Unidades (SI) , en el cual todas las distancias están en metros y todas las medidas angulares en radianes.

Centralización de parámetros.

Para aumentar la robustez y la facilidad de uso, CARMEN provee un repositorio centralizado para manejar parámetros. El software de los robots móviles es comúnmente distribuido entre varios procesadores. Una punto de falla común es el conflicto entre valores de parámetros suministrados por fuentes diferente. Para garantizar que los valores de parámetros sean consistentes CARMEN carga los parámetros de un único archivo, permitiendo los cambios de los parámetros en tiempo de ejecución, eliminando la necesidad de reinicializar o recompilar los módulos.

4.3.2. Navegación.

CARMEN incluye detección y seguimiento de personas en base a mapas. Este seguimiento, es llevado a cabo por las diferencias entre las medidas actuales y las medidas esperadas de los sensores láser.

La navegación en CARMEN es un concepto difícil de estandarizar, es por esto que se intenta soportar muchos estilos diferentes de navegación. La actual implementación de la navegación es una realización del método conocido como *Konolige's local gradient descent planner* [59].

4.3.3. Localización y seguimiento.

El módulo de localización en CARMEN implementa una variación del método del algoritmo de *localización de MonteCarlo*[60]. El módulo acepta lecturas de odometría y del láser desde los módulos de la base y es capaz de estimar la posición del robot en un mapa en aproximadamente 100 mseg.

El uso de mapa en la navegación asume la existencia de un mapa exacto del entorno. CARMEN posee una implementación para construir mapas. Los mapas son construidos a partir de datos registrados de los sensores láser.

4.3.4. Observaciones.

CARMEN es tal vez la plataforma de libre distribución mejor diseñada para aplicaciones de navegación. En un futuro es posible que la documentación completa esté disponible, pero dicho esfuerzo solo dependerá de la comunidad que sigue y apoya al proyecto.

4.4. Microsoft Robotics Studio.

Microsoft Robotics Studio (MSRS), es una plataforma de desarrollo para crear aplicaciones sobre un grupo de robots específicos, usando tecnología .NET, en ordenadores con sistema operativo Windows y Windows CE.

La empresa Microsoft previendo un crecimiento en los compradores, investigadores y aplicaciones militares en el área de la robótica, ha creado un equipo de desarrollo que garantice parte en este mercado. En la conferencia *Robo Business* de 2006, fue presentado el primer *community technical preview* de MSRS.

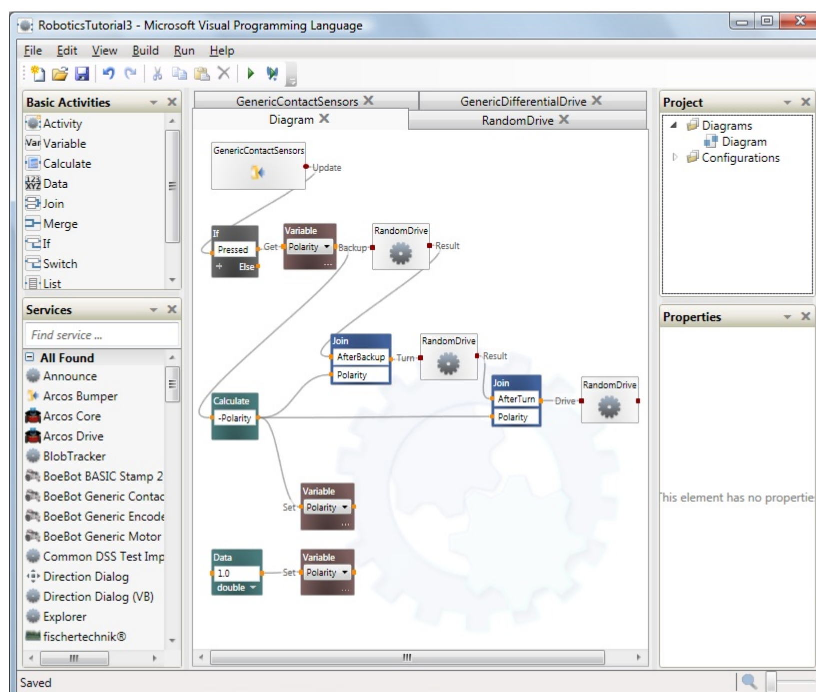

Figura 4.10: Entorno de programación visual de Microsoft [6]

MSRS es un entorno de programación visual. Este permite a los usuarios cortar y pegar componentes que representan comportamientos de bajo nivel y servicios, enlazando estos entre si, para crear aplicaciones robóticas. MSRS también posee un entorno de simulación visual de robots y su entorno, con simulaciones de fricciones, gravedad y otros factores. En el documento que resume las características y promueve el MSRS (obtenido en [6]), describe los beneficios de la plataforma:

Entorno de programación visual para crear aplicaciones. El *visual programming environment (VPL)* de Microsoft, ver figura 4.10, permite crear y depurar programas de una manera fácil. Solo se debe tomar y colocar los bloques que representan servicios y conectarlos entre si. Se puede tener un grupo de bloques ya interconectados, y reusarlo como si fuese un único bloque en cualquier lugar del programa.

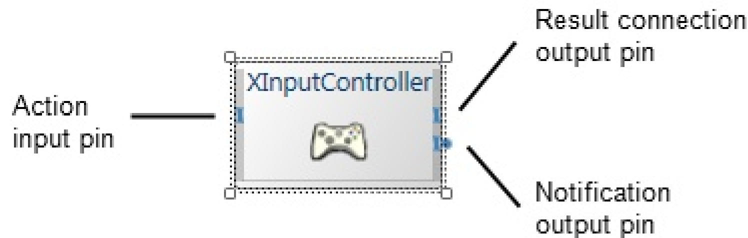

Figura 4.11: Ejemplo de bloque de construcción del VPL de Microsoft[6]

Entorno de simulación 3D. La simulaciones de aplicaciones sobre robots usan modelos 3D que reflejan la realidad con bastante fidelidad. El entorno de simulación virtual es soportado por un motor de simulación por la tecnología AGEIA PhysX, caracterizada por permitir modelos de entornos físicos de alta fidelidad.

Configuración para diferentes plataformas. La configuración de servicios es también por medio de un entorno visual.

Interacción con robots desde aplicaciones Web en Windows. La aplicaciones creadas permiten controlar el robot usando interfaces basadas en Windows o con un navegador Web compatible. Por ejemplo, por medio de HTML y Java Script se puede supervisar las cámaras y sensores del robot a través de la Web.

Permite la programación asíncrona. La biblioteca para el manejo de la concurrencia y coordinación en tiempo de ejecución (CCR) permite escribir programas que manejen la entradas asíncronas de múltiples sensores, así como también en envío de ordenes a los actuadores.

Modelos que permiten el reuso de software. Las funcionalidades de alto nivel (como los servicios de navegación) pueden usar fácilmente los servicios de bajo nivel (servicios de motores y sensores), proporcionando la reusabilidad del software y un mejor diseño para el manejo de fallas y actualización de las aplicaciones.

Posibilidad de extender las funcionalidades del MSRS. Es posible agregar o extender la funcionalidad del MSRS agregando nuevo servicios de software o hardware.

Permite la conexión entre el ordenador y el robot. La conexión entre los ordenadores de desarrollo de aplicaciones y el robot es proporcionado por el MSRS y la familia de sistemas operativos Windows actualmente soportados (Vista, CE, XP y Mobile). También es posible que las aplicaciones se ejecuten en robots cuyos procesadores tengan algunos de los sistemas operativo Windows antes mencionados.

Posibilidad de usar varios lenguajes. La aplicaciones pueden ser escritas también en varios lenguajes, siempre y cuando sean lenguajes de cuyos

Figura 4.12: Imagen de una simulación 3D con el MSRS[6]

compiladores sean compatibles con la plataforma Microsoft, incluyendo a Microsoft Visual Studio y Microsoft Visual Studio Express (C#, C++ y VB.NET), así como el lenguaje Microsoft IronPyton.

4.4.1. Observaciones.

En [61] S. Cherry presenta, según sus opiniones, las razones por las cuales Microsoft crea el proyecto MSRS. Para finalizar es importante destacar el esfuerzo que Microsoft esta haciendo para impulsar el MSRS, colocando a la disposición de futuros clientes una gran cantidad de documentación y tutoriales para familiarizarse con el MSRS.

4.5. Evolution Robotics ERSP.

Le empresa Evolution Robotics desarrolla soluciones en el área de robótica y además patrocina equipos para la automatización de la manufactura, con el fin de integrar las tecnologías de robótica en sus productos. Entre las soluciones ofrecidas se encuentra ERSP.

ERSP es una caja de herramientas software para el soporte de aplicaciones de visión, navegación y desarrollo de sistemas. ERSP está compuesto por varios módulos software unidos en una arquitectura software. Entre los módulos se encuentran manejadores de sensores de rango, comportamientos y tareas. Algunos de estos módulos están basados en algoritmos de visión y navegación. ERSP también proporciona interfaces a sus módulos que permiten extender el sistema

para que se ajuste a necesidades específicas de una aplicación.

4.5.1. Arquitectura de ERSP.

La arquitectura de ERSP consiste de varias capas que coordinan un rango de operaciones, que van desde tareas simple de movimiento de un simple motor, a tareas complejas tales como la navegación misma en entornos desconocidos. La arquitectura del sistema es modular, con interfaces bien definidas entre sus módulos y capas del sistema, ver figura 4.13.

La capa inferior es la capa de abstracción del hardware (HAL), la cual sirve de interfaz entre las aplicaciones y el hardware del robot. Los manejadores de dispositivos de la capa HAL, son los responsables de la interacción entre con los sensores y actuadores del robot. Un sensor puede ser una cámara o un sensor de luz infrarojo, y un actuador puede ser un motor en el sistema de movimiento. Esto significa que la capa HAL gestiona el control del robots con el mundo físico. Esta capa, también interactúa con los componentes de bajo nivel del sistema operativo.

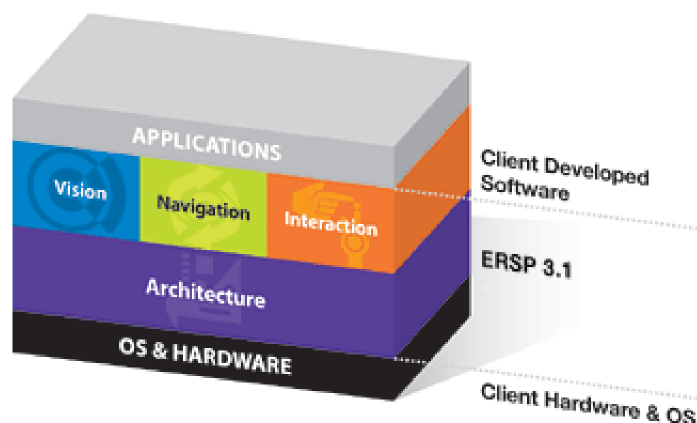

Figura 4.13: Arquitecturas de capas de ERSP[7]

La capa intermedia es la capa de ejecución de comportamientos[2] (BEL), la cual se comunica con los manejadores de sensores y actuadores por medio de la capa HAL. El bloque de construcción básico de la BEL son los comportamientos, los cuales toman decisiones basadas en la información obtenida a partir de los sensores. Los comportamiento cubren un amplio rango de funcionalidades , desde la lectura de datos en los sensores y uso de motores, hasta la implementación de operaciones y algoritmos matemáticos. Los comportamientos son altamente reactivos y son ubicados en lazos críticos de control.

La capa superior es la capa de ejecución de tareas (TEL). Ésta permite una programación del robot orientada a tareas. Mediante la secuenciación y combinación de tareas, propuesta por los diseñadores de ERSP, se puede crear

[2]Traducción de la palabra behavior, aplicadas a robots en vez de máquinas

planes flexibles para que un robot los lleve a cabo en vez de usar el procedimiento natural usando programación tradicional.

Adicionalmente a las tres capas mencionadas, ERSP incluye varias facilidades que, independientemente del tipo de robot, son útiles en aplicaciones en robótica. Muchas de estas facilidades están disponibles en la biblioteca principal del sistema, y otras están integradas en las correspondientes capas de la arquitectura. La caja de herramientas para imágenes, encapsula formatos estándares de imágenes, permitiendo la conversión, compresión, y otras transformaciones comunes. La biblioteca matemática provee una amplia variedad de herramientas para el cálculo, incluyendo operaciones de matrices y vectores, modelos gaussianos y lógica borrosa. Las herramientas de abstracción de plataforma simplifica el desarrollo cruzado entre plataformas cuando se usan servicios tales como el sistema de archivos, concurrencia de objetos y uso de la red. La funcionalidad de bitácoras provee un medio para depurar programas en ejecución.

4.5.2. Núcleo de tecnologías.

El núcleo de tecnologías se basa en dos funcionalidades principales visión y navegación. Las interfaz a la funcionalidades de visión permite el acceso a algoritmos de visión por ordenador. Estos algoritmos pueden analizar imágenes capturadas por la cámara para reconocimiento de objetos, detección de movimiento, o detección de piel para la detección de personas.

La interfaz a las funcionalidades de navegación provee mecanismos para controlar el movimiento del robot. Esta interfaz permite el acceso a los módulos para el uso conjunto con mapas, localización, exploración, planificación de movimiento, seguimiento de objetivos y teleoperación.

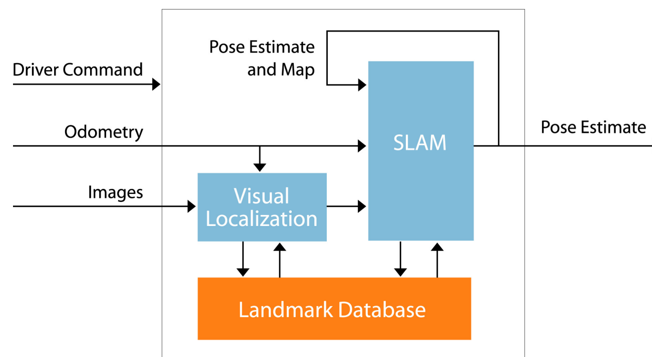

Figura 4.14: Diagrama de bloques de vSLAM en ERSP[7]

El algoritmo vSLAM de Evolution Robotics es el núcleo de la funcionalidad de navegación en ERSP. Éste permite de forma simultanea la localización y la actualización/creación de mapas usando una cámara de bajo costo y los encoders de las ruedas. La razón de usar una cámara Web en vez de un sensor láser u otro dispositivo de mayor costo, es que la tecnología vSLAM puede ofrecer esa reducción de costo. Con vSLAM, un robot puede operar en un entorno

desconocido, construir un mapa, y localizarse al mismo tiempo en ese mapa.

4.5.3. Comportamientos.

La capa de ejecución de comportamientos (BEL) de ERSP es un marco para el desarrollo de sistemas autónomos. Una aplicación usa el BEL para capturar datos de los sensores, tomar decisiones en base a esos datos, para luego tomar las apropiadas acciones.

El bloque principal de construcción es el comportamiento, definido como una unidad computacional que relaciona entradas con salidas. Las entradas y salidas en un comportamiento son llamados puertos. Cada puerto de salida tiene un número arbitrario de puertos de entrada. Un puerto es caracterizado por un tipo de dato, un tamaño de dato, una semántica, indicando la estructura y el dato que llega a través de un puerto. Los atributos del puerto determinan la validez de las conexiones, garantizando que el puerto de inicio y el puerto de destino tienen tipos compatibles.

Las cadenas de comportamientos conectados entre si forman redes. ERSP ejecuta todos los comportamientos en la red secuencialmente. En este modelo secuencial, la ejecución de un comportamiento se hace en dos etapas: primero el comportamiento recibe los datos de los puertos de entrada, hace los cálculos, y luego envía los datos resultantes a los puertos de salida.

Mientras que los comportamientos, son implementados por objetos en C++, una red de comportamiento puede ser implementada por medio de un programa en C++, aunque la forma mas fácil y estándar es por medio de un archivo XML.

4.5.4. Observaciones.

La información y documentación sobre ERSP ha sido obtenida de [7] de forma gratuita previo registro. En este sitio se ofrecen documentación, tutoriales y vídeos. Es importante destacar la gran cantidad de comportamientos predefinidos que aparecen en la guía del usuario.

La plataforma ERSP basa su apuesta en los comportamientos disponibles en su biblioteca de comportamientos, permitiendo al investigador colocar y seleccionar los comportamientos que mejor se ajuste a su aplicación.

4.6. Proyecto CLARAty.

En [8] se presenta la arquitectura *Coupled Layer Architecture for Robotic Autonomy (CLARAty)*. Esta fue diseñada para permitir un sistema software altamente modular para el control y la interacción con robots autónomos. El desarrollo de CLARAty ha sido patrocinado por el Jet Propulsion Laboratory, NASA Ames Research Center, Carnegie Mellon y la Universidad de Minnesota.

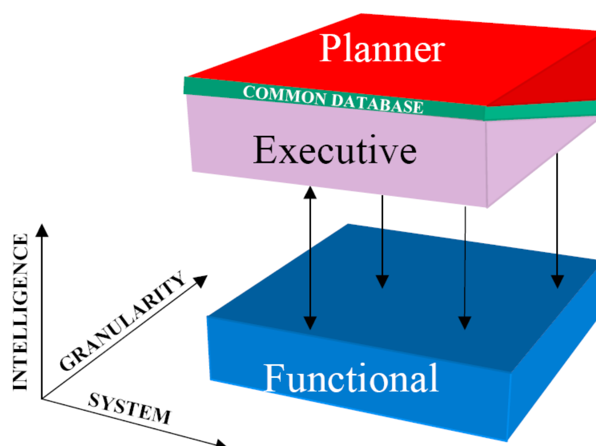

Figura 4.15: Arquitectura de CLARAty [8]

CLARAty propone una arquitectura de dos capas, mostrada en la figura 4.15. Según los autores, presenta dos ventajas principales: una representación explícita de la granularidad de los sistemas de capas con una tercera dimensión y una mezcla de técnicas declarativas y procedimentales para la toma de decisiones.

La adición de una dimensión para representar la granularidad permite una representación de la jerarquía del sistema en la capa funcional. En esta capa, una jerarquía orientada a objetos describe el sistema como una encapsulación de subsistemas, donde cada nivel de abstracción permite funcionalidades básicas. Por ejemplo, el comando "move" puede ser dirigido a un motor, a un manipulador, a un robot móvil, o a un grupo de robots. Para la capa de decisión, la granularidad relaciona las actividades con la línea del tiempo entre la creación y la ejecución. Debido a la naturaleza de la dinámica del sistema físico controlado por la capa funcional, hay una fuerte correlación entre el la granularidad del sistema y la granularidad temporal de la capa de decisión.

4.6.1. Capa funcional.

La capa funcional, mostrada en la figura 4.16, es una interfaz a todo el hardware del sistema y sus capacidades, incluyendo las agrupaciones y sus capacidades resultantes. Estas capacidades son la interfaz a través de la cual la capa de decisión usa los sistemas hardware del robot. La capa funcional presenta las siguientes características:

Orientación a objetos. El uso del diseño orientado a objetos es deseable por varias razones. La primera, éste puede ser estructurado y agrupado creando una correspondencia con el hardware del robot. Segunda, en todos los niveles de anidamiento, la funcionalidad y la información sobre los sistemas de componentes pueden ser codificados y agrupados en un mismo lugar lógico. Tercera, una apropiada estructuración del software puede usar las propiedades de herencia para manejar la complejidad del desarrollo del

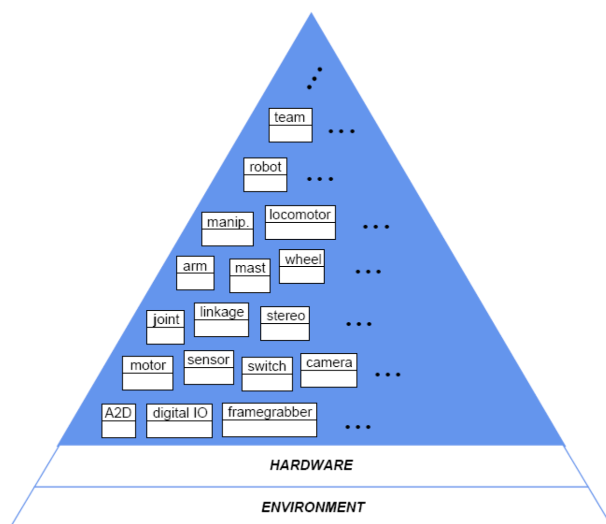

Figura 4.16: Capa funcional en CLARAty [8]

software. Finalmente, esta estructura puede ser gráficamente diseñada y documentada usando el estándar UML.

En la figura 4.17, se muestra una descripción simplificada de la jerarquía encontrada en la capa funcional. En este diagrama una cuarta dimensión es agregada, que ha sido usada para representar el atributo de herencia de clases en la capa funcional. En la parte superior un objeto *Rover* es representado por la composición de objetos de la clase *arm* y la clase *locomotor*.

Una ventaja de esta estructura es que permiten la exención del sistema de una manera más fácil. Primero, múltiples copias de los objetos pueden ser instanciados. Segundo, dos subclases pueden heredar todas las propiedades de la clase *Appendage* (accesorio).

Encapsulamiento del funcionamiento interno. Todos los objetos contienen una funcionalidad básica, accesible en la capa funcional. El propósito es ocultar los detalles de la implementación y proveer una interfaz genérica.

Registro del estado. El estado de los componentes del sistema es registrado y puede ser recuperado por medio de métodos.

Planificadores locales. A pesar de existir un planificador global en la capa de decisión, éste puede ser utilizado como una parte de los subsistemas de la capa funcional.

Predicción de uso de los recursos. De igual manera que los planificadores locales, la predicción de uso de recursos puede localizarse en los objetos que usan estos recursos. La consultas a estas predicciones son hechas por la capa de decisión durante la planificación y la programación de tareas, y pueden ser hechas con varios niveles de fidelidad.

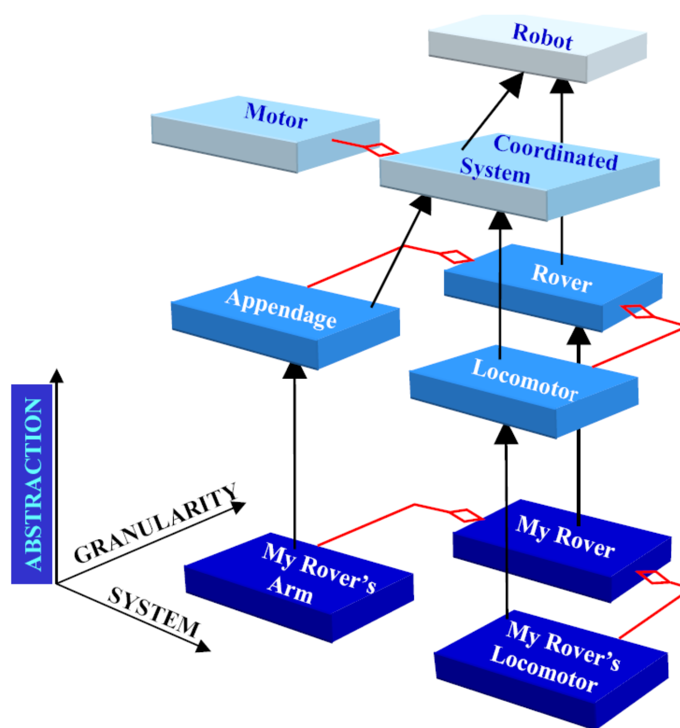

Figura 4.17: Jerarquía de Clases en CLARAty [8]

Simulación. En la forma mas simple, la simulación del sistema puede ser llevada a cabo por medio de la simulación del funcionamiento de los objetos de nivel mas cercano al hardware.

Pruebas y depuración. Desde el desarrollo inicial, todos los objetos deben contener interfaces para la depuración y pruebas.

4.6.2. Capa de decisión.

La capa de decisión descompone los objetivos de alto nivel en pequeños objetivos. Éstos son ordenados en el tiempo debido a restricciones, al estado del sistema y a accesos a la capacidades ofrecidas por la capa Funcional. La figura 4.18 muestra una representación simplificada de la capa de decisión.

Red de objetivos. La red de objetivos es la descomposición conceptual de los objetivos de alto nivel. Estos contienen la representación declarativa de los objetivos durante el proceso de planificación, las restricciones temporales de la red resultantes de la programación, y el posible árbol de tareas, obtenido de la descomposición procedimental, a ser usado durante la ejecución.

Objetivos. Los objetivos son especificados como restricciones de estado sobre el tiempo. De esa manera se especifica como *lo que **no** debe ser hecho*. Por

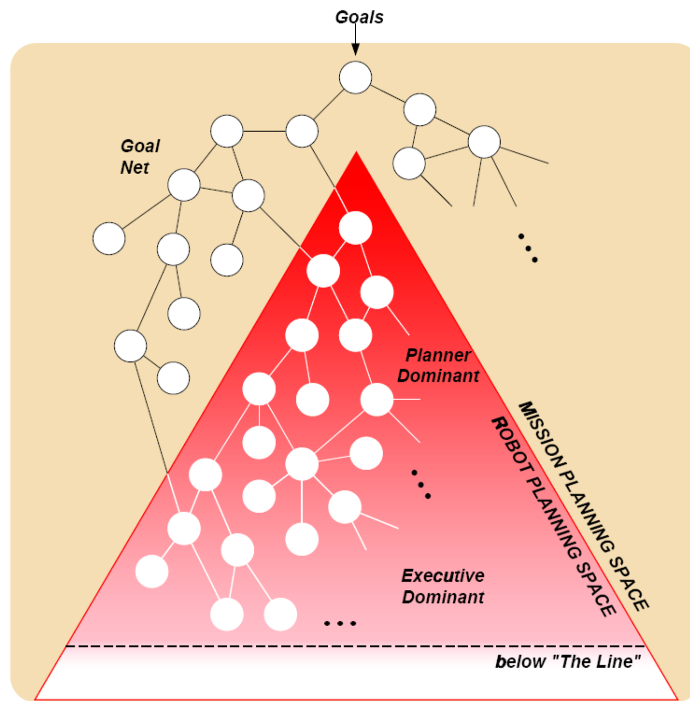

Figura 4.18: Capa de decisión en CLARAty [8]

ejemplo, el ángulo en una unión no debe ser mayor de 30º ni menor de 20º.

Tareas. Las tareas son actividades que explícitamente pueden ser paralelas o secuenciales y que están fuertemente enlazadas. Las tareas son el resultado de las descomposición procedimental de un objetivo en una secuencia, la cual es posiblemente una condición. En contraste con los objetivos, las tareas especifican exactamente *lo que debe ser hecho.*

Comandos. Los comandos son especificaciones unidireccionales de actividad en el sistema. Típicamente ello provee la interfaz entre el borde de la red de objetivos y las capacidades de la capa funcional. El cierre del lazo de control con la capa de decisión es mantenido por la observación del estado del sistema y los comandos ejecutados.

El borde (the line). El borde es el borde conceptual entre la toma de decisiones y la ejecución funcional. Este existe en el borde de la red de objetivos resultante, y puede moverse de acuerdo a la granularidad actual. Cuando es proyectada sobre la capa funcional, ésta describe el borde inferior en el cual el sistema es una caja negra para la capa de decisión.

Estado. El estado de la capa funcional es obtenido por medio de una consulta. El estado de la capa de decisión, es esencialmente su plan, la elaboración activa, y la historia de ejecución. Ésta puede ser almacenada, o recargada, como un todo o como una parte.

4.6.3. Implementación.

Los siguientes herramientas y estándares son usadas por CLARAty en su desarrollo:

- El lenguaje para la unificación de modelos (UML).
- Lenguaje de programación C++
- Sistemas operativos soportados. VxWorks, Linux, Solaris.
- Biblioteca de plantillas estándar (STL).
- La mayoría del código ha sido desarrollado usando editores de texto. Aunque se ha aconsejado el uso de herramientas como Rhapsody y Visio, pero no es obligatorio.

En [62] se describe a CLARAty como una arquitectura software para la robótica reusable, esta nueva característica es presentada varios años después del lanzamiento inicial del proyecto. Los mecanismos usados para alcanzar un mayor nivel de reusabilidad del código se basa en el uso intensivo de: programación orientada a objetos, programación genérica, patrones de funciones, especialización de clases y funciones y flujos de datos en tiempo de ejecución.

4.6.4. Observaciones.

CLARAty presenta una arquitectura sólida, bastante detallada y completa. El apoyo del JPL Laboratory plantea un gran reto para su uso en los robots que la NASA envía al espacio en condiciones inhóspitas.

4.7. Skilligent.

Skilligent [63] es un sistema de control basado en comportamientos de robots. El objetivo es que el robot pueda aprender a partir de demostraciones y de interacciones, con el objetivo de crear una nueva generación de robots industriales y robots multiservicios.

Skilligent permite la creación de robots sociables que puede ser entrenados por sus usuarios, en vez de ser programados por un ingeniero de software. Esta característica es de gran importancia tanto para los usuarios finales, como para los constructores de productos de la industria robótica, ya que incrementaría el utilidad del robot.

El software de control de robots Skilligent permite ciertas capacidades para soportar la interacción social, el cual hace las tareas de entrenamiento y colaboración con robots de una aproximación más natural para los humanos. Un robot controlado por medio de Skilligent puede ser entrenado por una persona

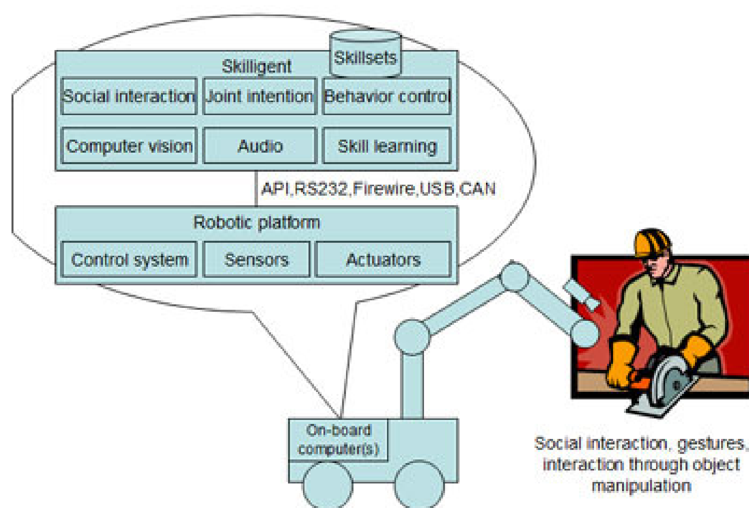

Figura 4.19: Diagrama de la arquitectura software Skilligent[9].

a pesar de que ésta no sea una experta en robótica, sin usar un teclado, un ratón o un joystick. Durante la sesión de entrenamiento, el robot interactúa con su entrenador humano a través de gestos, sonidos, manipulación de objetos y atención conjunta.

El software está basado en recientes avances en las áreas de conocimiento intrínseco[3], fijación de la percepción[4], aprendizaje automático, coordinación de comportamientos e interacción social humano-máquina.

4.7.1. Posibilidad de extensión.

Skilligent ha sido escrita en C++ y compilada y empaquetada en una biblioteca de enlace dinámico disponible para sistemas operativos Windows y Linux. Hay adaptaciones de la biblioteca de C# y Java.

Skilligent ha sido diseñado para permitir una integración transparente con lazos de control de robots controlados por medio de un ordenador personal.

La herramienta complementa los sistemas de control electrónicos por medio de proveer *componentes inteligentes* de alto nivel que aprenden nuevas habilidades por medio de la interacción y colaboración con el usuario.

4.7.2. Justificación.

Hoy en día la pequeñas y medianas industrias no usan robots de forma intensiva en su labores de producción. La razón fundamental radica en el costo de los

[3]traducción aproximada del termino *embodied cognition*.
[4]traducción aproximada del termino *grounded perception*.

robots, las dificultades de la programación y las posibles incompatibilidades de la integración de robots para compartir trabajo con los trabajadores humanos.

Los creadores de Skilligent pretenden abrir un mercado para ofrecer robots a las pequeñas y medianas empresas de manufactura, granjas y talleres.

La nueva generación de robots industriales y robots de servicios son construidos para ser usados en los medianos y pequeños negocios o como robots domésticos. Los robots son capaces de aprender habilidades de los humanos y trabajar con empleados de una forma segura y colaborativa con estos.

Muchos de los robots de hoy en día no están capacitados para aprender nuevas destrezas robóticas. Por el contrario, los diseñadores de Skilligent pretenden poner a disposición robots que puedan ser entrenados en nuevas tareas, de la misma manera que una mascota puede ser entrenada para hacer nuevos trucos.

4.7.3. Observaciones.

En [63] se anuncia la creación del establecimiento entre Skilligent y Microsoft Robotics Studio. También hay disponibles ficheros de vídeos con demostraciones de entrenamientos de un robot usando Skilligent.

Hasta ahora la forma de programar robots por medio de interacciones con el robot platea un nuevo paradigma orientado a herramientas de Inteligencia Artificial y de Visión. Es posible que en los próximos años estas técnicas se consoliden y presente una alternativa para cualquier robot.

4.8. URBI.

La empresa GOSTAI son los creadores de URBI[10], que es presentado como un producto inovador en el área de la programación. Este producto se basa en un lenguaje interpretado que dice permitir la programación de tareas paralelas, programación basadas en eventos y una plataforma de objetos distribuidos propia.

También es posible usar URBI por medio de la biblioteca liburbi desde lenguajes de programación C++, Java, Matlab Python, Ruby. La biblioteca liburbi es distribuida por medio de una licencia GPL[64].

Las personas que conforman GOSTAI piensan que la industria necesita una plataforma software robótica, que permita enfrentar los retos de la Inteligencia Artificial y a la vez la programación de robots autónomos.

En GOSTAI se piensa que los requerimientos de una plataforma universal son:

Flexibilidad. Debe operar en cualquier plataforma hardware, sistema operativo o lenguaje.

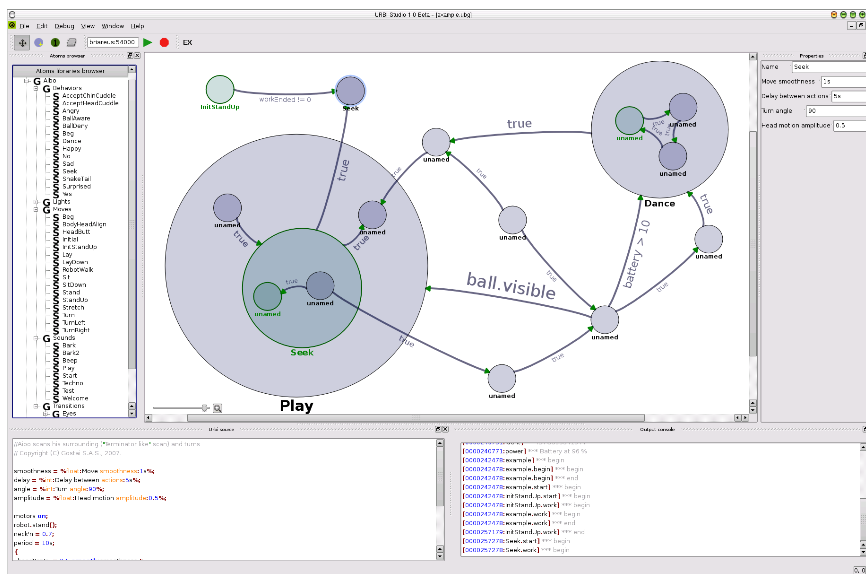

Figura 4.20: Interfaz de usuario del URBI Studio[10].

Modularidad. Debe estar integrada en una arquitectura de componentes modulares, con componentes ejecutándose de forma transparente en un robot o en un ordenador remoto.

Amplitud. La robótica es un dominio complejo, donde es necesario una gran abstracción para trabajar con paralelismos y programación basada en eventos, donde las soluciones deben estar a la altura de investigadores en el área de la inteligencia artificial.

Simplicidad. Ninguno de los requerimientos anteriores debe impedir su facilidad de uso, o que estos requieran cientos de hojas de documentación. Las personas jóvenes y los aficionados al área deben ser capaz de usar el sistema. Esto último estimulará la creatividad y será beneficioso para la robótica en general.

4.8.1. Innovaciones en URBI.

URBI es un plataforma que aspira a revolucionar la programación en la robótica. Por ahora se plantea atraer a la comunidad aficionada a la robótica y a la comunidad científica en base a sus innovaciones. A continuación se describe las innovaciones de URBI.

La arquitectura UObject.

Se pueden importar objetos escritos en C++ y conectarlo con URBI para usarlo como objetos del lenguaje. Estos son llamados UObjects.

Se puede también usar un UObject como un objeto remoto, como un programa autónomo en Windows, Linux, Mac OSX. No es necesario cambiar el código para conmutar el modo enlazado con el modo remoto.

Paralelismo y eventos.

La robótica necesita el uso herramientas de paralelismos, y requieren ir mas allá de la hebras y aplicaciones paralelas. URBI integra paralelismos y programación basada en eventos en el centro de su semántica.

Comandos tales como *whenever* o *at* el cual ejecuta un código cuando un evento ocurra. El símbolo & puede ser usado para sustituir los tradicionales *;* : *A&B* significa que A y B se ejecutan de forma paralela, así de simple. A continuación se muestra un ejemplo:

```
whenever (ball.visible)
{
    headPan.val += camera.xfov * ball.x
    &
    headTilt.val += camera.yfov * ball.y;
}
```

Características avanzadas del lenguaje.

Por ser un lenguaje paralelo real, URBI ofrece abstracciones que no son de fácil acceso en otros lenguajes interpretados.

Es posible asignar una variable para que alcance dicho valor en un tiempo determinado, a determinada velocidad o asignar una oscilación sinusoidal en esta. Todo esto puede ser ejecutado en forma paralela. A continuación se muestra un ejemplo:

```
neck = 10 time:450ms
& leg = -45 speed:7.5
& tail = 14 sin:4s ampli:45;
```

Cada variable tiene un *blend mode* el cual especifica como deben ser tratadas las asignaciones conflictivas: es una extensión del concepto *mutextes*. A continuación se muestra un ejemplo:

```
x->blend = add;
x = 1 & x = 3;
// el valor final de x en este caso es 4
```

Una parte del código puede ser usado por medio de una etiqueta. Por medio de esta etiqueta es posible detener, suspender y activar el código. A continuación se muestra un ejemplo:

```
mytag: { porción de código }
stop mytag;
freeze/unfreeze mytag;
```

4.8.2. Observaciones.

Como se ve claramente URBI presenta un nuevo lenguaje orientado a la programación por eventos con interesantes facilidades para la programación de paralelismos. Entre las plataformas robóticas para las cuales se han portado URBI se encuentra los robots: LEGO MINDSTORM NXT , Sony Aibo, iRobot create, Surveyor y ePuck. Entre los desarrollos actuales de urbi se encuentra el URBI Studio.

4.9. iRobots AWARE.

iRobot Aware[65] es la apuesta de la empresa iRobot por recuperar el mercado de los investigadores y desarrolladores en el área de robots móviles. Fue presentado en el verano de 2005 y está orientado al desarrollo de aplicaciones de la familia de iRobots. El producto es presentado junto al RDK (Robot Developer's Kit) y entre sus funcionalidades anuncian:

- Una arquitectura extensible, basada en componentes diseñada para el desarrollo fácil de aplicaciones sobre los robots vendidos por iRobot.

- Portabilidad entre diferentes sistemas operativos, aunque no se indica cuales.

- Un conjunto de herramientas de desarrollo y soporte para poder usar tecnologías de código abierto.

4.9.1. Observaciones.

A pesar de ser una de las empresas pioneras en el suministro de software y plataformas en el campo de los robots móviles, hoy en día se encuentran pocas referencias del uso de Aware fuera de iRobot. Apenas ha pasado un año del lanzamiento de la versión 2.0 es posible que existan grupos probando el software y pronto se publiquen algunos resultados. El grupo de la marina norteamericana *Space and Naval Warfare Systems Command (SPAWAR)* y otros patrocinantes se han ofrecido para probar Aware y el RDK. Ellos esperan poder integrar sus desarrollos a Aware en sus aviones militares.

4.10. Proyecto Open JAUS

Open JAUS[11] es una implementación de código abierto de la *arquitectura unificada para sistemas no tripulados*[5]. La arquitectura tiene su origen de la *Joint Architecture for Unmanned Ground System (JAUGS)*, formada en 1995 por la *Unmanned Ground Vehicles/Systems Joint Project Office* del gobierno de los Estados Unidos de América. En el año de 1998 JAUS fue adoptada por el Departamento de Defensa de los Estados Unidos. Objetivos de la Arquitectura JAUS:

- Soportar todas las clases de vehículos no tripulados.
- Inserción rápida de nuevas tecnologías.
- Permitir componentes militares interoperables e intercambiables.
- Permitir Sistemas no tripulados interoperables.

JAUS está basado en componentes, y en el pase de mensajes entre estos componentes. Los objetivos del proyecto son OPEN JAUS:

- Facilitar la adopción del estándar JAUS entre partes interesadas.
- Proveer una implementación de software libre a la comunidad de usuarios de JAUS, haciendo énfasis en la correctitud de la programación, portabilidad y fácil lectura del código el cual pueda ser usado como herramienta educacional.
- Soportar la interoperabilidad por medio de aplicaciones comunes.
- Auspiciar la colaboración de los usuarios de código abierto en aplicaciones bajo JAUS y proveer un soporte para los proyectos de código abierto.

4.10.1. Implementación.

Los principales usuarios de la arquitectura JAUS es el departamento de defensa de los Estados Unidos. Para los militares OpenJaus es estable, seguro y probado en sistemas militares no tripulados. Con JAUS convertido en una especificación libre, cualquier implementación de JAUS puede se usada para ser tan segura según lo indique la especificación.

Open JAUS también permite a los sistemas militares no tripulados poseer una interfaz bien documentada que puede ser implementada por un fabricante externo al DOD, permitiendo la integración de componentes de diferentes fabricantes en un solo sistema.

[5]Joint Architectura for Unmanned Systems

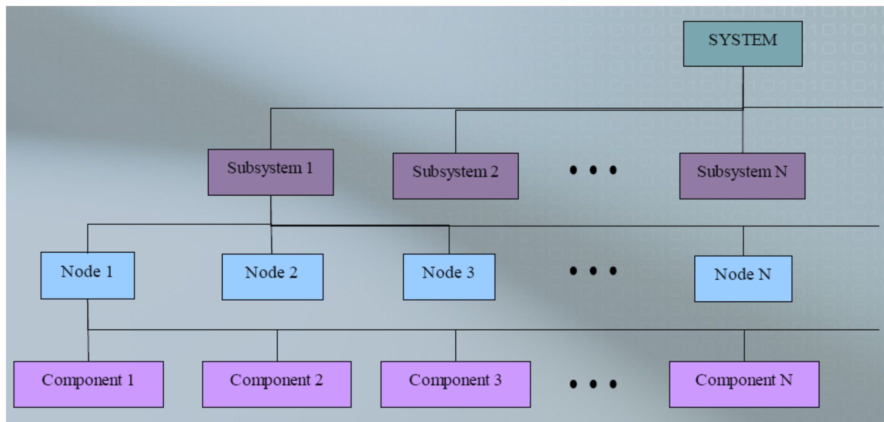

Figura 4.21: Topología del Sistema en JAUS [11]

Otra de las ventajas de Open JAUS para aplicaciones militares no tripuladas radica en que varias implementaciones pueden ser llevadas en menos tiempo, disminuyendo los costos de integración y permitiendo la participación de múltiples proveedores.

El proyecto Open JAUS inicialmente constaba de la biblioteca y aplicaciones creadas pro el *Centro de Robótica y Máquinas Inteligentes (CIMAR)* de la Universidad de Florida.

4.10.2. Características.

- Licencia que sigue las líneas de OSI para permitir el uso libre, modificación, y distribución de software construido con Open JAUS.
- Soporta las plataformas hardware x86, PowerPC y ARM.
- Puede compilarse y ejecutarse en entornos linux, incluyendo Microsoft Windows (bajo Cygwin).
- Código base estable, desarrollado con éxito en el *2005 DARPA Grand Challenge*.
- Documentación completa en formato HTML de las interfaces y todos los componentes.

4.10.3. Especificaciones de JAUS.

La especificación de la arquitectura detalla todas las funciones y los mensajes empleados por los componentes, y además define las reglas que regulan el intercambio de mensajes.

La topología JAUS, mostrada en la figura 4.21, crea agrupaciones de uno o más subsistemas. Típicamente los grupos tiene una ventaja cooperativa entre los componentes constituyentes del sistema.

Con esta topología es posible agrupar una o más unidades de control, una o más instalaciones de sensores, y uno o más vehículos para trabajar de forma conjunta para alcanzar un objetivo común.

Los componentes es el nivel más bajo en jerarquía de la arquitectura. La cohesión en el software permite funciones bien definidas o grupos de estas. En JAUS, un componente es una tarea o un proceso ejecutable.

Un nodo está compuesto de uno o más componentes.

4.10.4. Observaciones.

Open JAUS es un apuesta del DOD para captar aportes de la comunidad que desarrolla software libre. A cambio se ofrece un marco de trabajo con reglas bien definidas, que deben ser seguidas para lograr desarrollos que cumplan con los estándares de calidad que exige la arquitectura JAUS. Una implementación de Open Jaus esta hecha en el robot BEAR[66], que es un robot asistencial para extraer soldados heridos en el campo de batalla.

4.11. Webots.

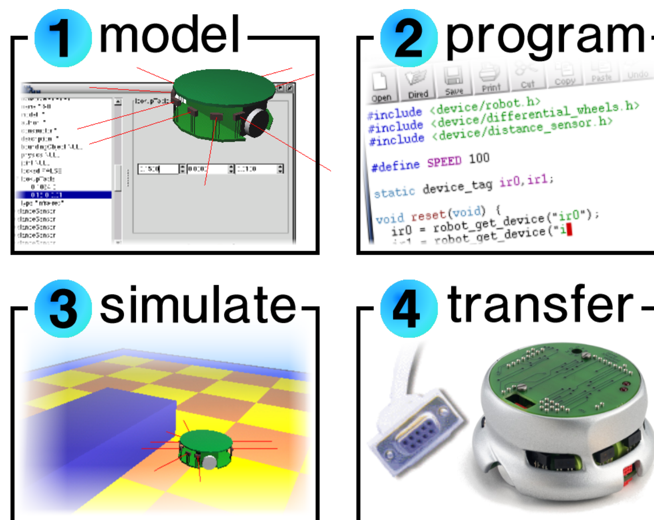

Figura 4.22: Etapas en la programación de un robot usando Webots [12]

La empresa Cyberbotics son los desarrolladores de Webots[67], presentado en [12], como un software de simulación que provee un entorno para la rápida creación de prototipos, con el objetivo de modelar, programar y simular robots

móviles. La biblioteca de robots incluidas en Webots, permite implementar programas de control en varios robots reales comercialmente disponibles. Webots también permite definir y modificar parámetros de configuración de robots, así como compartir un entorno de simulación por varios robots. Para cada objeto de la simulación, se pueden definir varias propiedades, entre éstas se encuentran forma, color, textura, masa, fricción, etc. Se puede equipar cada robot con varios sensores y actuadores disponibles en la biblioteca. También es posible programar a los robots usando varios entornos de desarrollo, simularlos y opcionalmente transferirlos a un robot real, ver figura 4.22.

4.11.1. Características.

Webots tiene un grupo de funcionalidades para permitir la realización de simulaciones y programas de robots:

- Capacidad de simular la gran mayoría de robots móviles, incluyendo robots con ruedas, con piernas y con capacidad de vuelo.
- Una biblioteca con una gran variedad de sensores y actuadores.
- Permite programar los robots en C, C++ y Java, o software de terceros que permitan la capacidad por medio del protocolo TCP/IP.
- Transferencia de los programas de control construidos para robots móviles reales, incluyendo AIBO, Lego Mindstorm, Khefera, Koala y Hemisson.
- Usa la biblioteca ODE (Open Dynamics Engine) para garantizar la fidelidad de las simulaciones.
- Crea vídeos de las simulaciones en formato AVI o MPEG.
- Incluye varios ejemplos de códigos fuentes de controladores y modelos de robots reales disponibles.
- Permite la simulación de sistemas multiagente, con facilidades para la comunicación local y global.

4.11.2. Observaciones.

Webots se distribuye por medio de licencias que se deben comprar con una periodicidad anuales y existen versiones para sistemas operativos Linux o Windows. Esta plataforma es usada actualmente en clases de pregrado de la Universidad Carlos III de Madrid. El uso académico de Webots para cursos introductorios a la robótica, permite al estudiante entrar en contacto con los conceptos básicos del área. Por medio de entornos virtuales es posible lograr simular las principales plataformas disponibles en el mercado, agregar piezas y observar su comportamiento. Webots es una herramienta en pleno desarrollo. Las diferentes versiones han dado problemas de compatibilidad, pero aun así son detalles en comparación con la funcionalidad de la herramienta.

Figura 4.23: Simulación multiagente usando Webots [12]

4.12. Comparación de las alternativas vistas.

Las alternativas revisadas en este capítulo son las principales en uso en robots móviles al momento de escribir el presente trabajo. En la tabla 4.1 se muestra un resumen de algunas de las principales características, éstas pueden ser tal vez la más significativas para el presente proyecto. Éstas son:

Integración con otros productos. Esta característica indica si la aplicación posee bibliotecas de funciones, que permitan de alguna manera usar servicios de otras herramientas. Esto permitiría agregar implementaciones de algoritmos útiles, como lo son algoritmos de análisis de imágenes, construcción de mapas, interconexión con interfaces gráficas, etc.

Mecanismo de comunicación. En el proceso de integración con otras herramientas, es posible que éstas se encuentren en ordenadores diferentes. Para ello es necesario conocer el mecanismo de comunicación usado. Si los mecanismos de comunicación son compatibles la integración es más fácil de realizar. En cambio si son incompatibles es posible que sea necesario construir una adaptación por software del canal de comunicación. Esto es equivalente a construir una pasarela entre los protocolos de comunicación involucrados.

Capacidad de tiempo real En algunas aplicaciones el tener respuestas temporales deterministas es fundamental. En esos casos es importante saber si la plataforma a usar cumple con tales funcionalidades.

	Cualidades						
	I	**C**	**TR**	**L**	**E**	**SO**	**SIM**
OROCOS	Si	CORBA	Si	GPL	Si	Linux	No
Player & Stage	Si	BSD Sockets	No	GPL	Si	Linux	Si
CARMEN	Si	IPC propio	No	GPL	Si	Linux	Si
MSRS	Si	DCOM	No	Privada	Si	Windows	Si
ERSP	Si	API propia	No	Privada	Si	Linux, Windows	No
CLARAty	Si	API propia	No	GPL*	Si	Linux, Solaris, VxWorks	Si
Skilligent	-	Api Propia	No	Privada	Si	Linux, Windows	No
URBI	Si	Leng. propio	No	Privada	Si	Linux, Windows	No
iRobots Aware	Si	CORBA	No	Privada	Si	Linux, Windows.	-
Open JAUS	Si	Pase de Mensajes	Si	Libre	Si	Linux	No
Webots	Si	API propia	No	Privada	Si	Linux, Windows	Si

Integración con otros productos. Mecanismo de **C**omunicación.**TR** Capacidad de Tiempo real. Tipo de **L**icencia de Distribución. Posibilidad de **E**xtensión. **SO** Sistemas operativos soportados. **SIM** Entornos de simulación disponible.

Tabla 4.1: Resumen comparativo entre las plataformas software descritas

Tipo de licencia de distribución. Algunas aplicaciones tienen funcionalidades atractivas, pero los derechos de uso son privados y se debe pagar el derecho de uso. Otras herramientas se distribuyen por medio de licencias GPL[64], que permiten el uso, pero las modificaciones y extensiones deben estar bajo licencia GPL y no se puede obtener beneficios económicos sobre dichas extensiones. Es importante destacar que la mayoría de herramientas distribuidas con licencias privadas ofrecen descuentos para aplicaciones de docencia e investigación sin fines de lucro.

Posibilidad de extensión. Un factor interesante de una plataforma es la capacidad de extensión. Ésta puede ser para poder soportar un hardware nuevo, agregar nuevas funcionalidades, etc.

Sistemas operativos soportados. Antes de hacer la selección es importante conocer sobre cual sistema operativo se puede ejecutar la herramienta. Eso puede implicar la necesidad de usar o no algún grupo de aplicaciones extras, como lo son entornos de desarrollos, lenguajes, bibliotecas de funciones, etc.

Entorno de simulación disponible. El uso de un buen sistema de simulación puede acelerar el diseño de estrategias de control. A pesar de que un simulador de un robot nunca podrá sustituir un robot real, puede servir de instrumento de formación en el uso de robot o entorno de pruebas, cuando puede existir alto riesgo tanto para el robot en si como para los investigadores.

4.12.1. Criterios para la selección.

A pesar de haber descrito algunos de los atributos de cada una de las herramientas especificadas, indicar cual es la mejor de ellas es una tarea bastante difícil. La respuesta debe encontrarse dependiendo de varios factores particulares a la investigación a realizar. Entre estos factores se tiene:

Robot a utilizar. La mayoría de aplicaciones en el área de robótica tiene un fin específico y éste solo requiere un robot para comenzar a realizar experimentos. En este caso se requiere una herramienta que cumpla con unos requisitos básicos y sea compatible con un robot disponible en el mercado. En otros casos el robot es la razón de la investigación, por lo que se requiere una herramienta que pueda ser adecuada al diseño del robot.

Tipo de aplicación a realizar. Este factor es aparentemente el más influyente en el momento de seleccionar una herramienta y su peso debe considerarse. La campañas de promoción de algunos productos pueden crear falsas expectativas. En algunos casos las correcta funcionalidad de una herramienta depende del robot, terceros productos y niveles de destreza que no son claramente definidos en los boletines de descripción de la herramienta. Por lo tanto es importante la validación de cada una de las funcionalidades antes de ratificar la bondad o no de un producto.

La destreza de los desarrolladores. En el momento de selección de una herramienta se debe tener claro el perfil de las personas que deben usarla. En algunos casos pueden requerir un nivel entre medio y alto de conocimientos en el área de sistemas distribuidos, diseño orientado a objetos, programación a bajo nivel, diferentes lenguajes como C, C++, Java, Python, Ruby, C#, tecnología .NET, etc.

Presupuesto. Algunas de las herramientas pueden requerir costosas licencias que deben ser renovadas con una periodicidad anual o semestral.

Tiempo máximo de entrega. La mayoría de los proyectos en ambientes académicos tienen financiamientos por un tiempo limitado. Estos a su vez incluyen periodos para la planificación , diseño, desarrollo, pruebas, y elaboración de informes. Algunos son a corto plazo, otros en cambio son a mediano plazo, por ejemplo dos años, y otros a largo plazo como son los proyectos a cuatro años.

4.12.2. Observaciones finales.

Tomando en cuenta la descripción de las herramientas se puede establecer algunas conclusiones sobre las mismas. A continuación se describe las observaciones sobre las herramientas de desarrollo vistas.

Herramientas con éxito probado son las que han sido desarrolladas y usadas en los proyectos del Laboratorio JPL de la NASA, específicamente la arquitectura CLARAty usada en los robots móviles para la exploración espacial. Otra platafoma con éxito probada es JAUS, usada intensivamente por robots en el área militar. A pesar del éxito de estas plataformas, las aplicaciones han sido orientadas a aplicaciones específicas como lo es el control de vehículos no tripulados en entornos hostiles.

La herramienta ERSP de Evolution Robotics, presenta una amplia biblioteca de funcionalidades y el uso de éstas pueden implicar un ahorro significativo en tiempo de desarrollo por medio de la reutilización de los componentes suministrados. Si el objetivo es tener un lenguaje especializado en aplicaciones distribuidas, URBI presenta su propio lenguaje con estructuras predefinidas para el manejo de concurrencias y parelismo.

El entorno de desarrollo y el soporte de documentación en línea de MS Robotics Studio es tal vez el ideal para el desarrollo de aplicaciones. El problema fundamental de esta herramienta es el uso obligatorio de herramientas de la familia de Microsoft, dejando por fuera los demás sistemas operativos. Webots incluye el uso en sistemas Linux y Windows, una biblioteca para una gran variedad de robots comerciales y un entorno de simulación de alta fidelidad con entornos reales, pero al igual que MSRS son plataformas de desarrollo comerciales.

Como un aporte fundamental y un nuevo paradigma de desarrollo se sitúa a Skilligent como la herramienta al menos conceptualmente ideal. En ésta se entrena directamente al robot en el comportamiento a seguir. Este entrenamiento

puede ser hecho por personal sin titulaciones en el área informática, ya que no es necesario construir programas en el sentido literal de la palabra. Es posible que Skilligent se encuentre en pleno desarrollo, pero si se consigue una mayor madurez de esta plataforma y se logran los objetivos planteados se convertiría en un estándar de facto en el desarrollo de aplicaciones robóticas.

Por el lado de código abierto hay tres principales protagonistas: CARMEN, OROCOS y Player. El primero es fundamentalmente orientado a aplicaciones en el área de navegación. Tal vez su principal defecto es la poca documentación del proyecto y la baja participación en los foros de desarrolladores. OROCOS hoy en día es independiente de EURON. Desde el punto de vista técnico el Proyecto OROCOS permite el uso de operaciones con duración definida de forma determinista. El precio que se paga es la necesidad de que los desarrolladores tengan sólidos conocimientos en el uso de CORBA en tiempo real. Tal vez, el sistema general de desarrollo con mayor popularidad es Player & Stage. La cantidad de seguidores de player se sitúa en la posibilidad de usar sistemas linux como sistema operativo y sockets BSD como mecanismo de comunicación. El entusiasmo de los seguidores de Player se observa al emitir cualquier duda en los foros de desarrolladores de Player, ya que éstas son respondidas por otros usuarios o por los mismos desarrolladores de la herramienta.

El proyecto actual posee particularidades de los robots personales móviles, con un gran énfasis en la interacción entre humanos y robots, donde la construcción y diseño del robot tiene igual peso en la investigación, que el peso que tiene el sistema de control que coordina el comportamiento de cada componente físico o lógico del robot.

5

Alternativas para la integración

En este capítulo se estudian las principales herramientas para integrar las diferentes partes software de control del robot Maggie. Para el desarrollo de este proyecto se analizó dos visiones de desarrollo: una implica la creación de una aplicación monolítica compleja, con una alta multifuncionalidad, desarrollada como un todo; y una segunda alternativa es la de desarrollar partes de forma independiente, que posean una única función bien definida, y que luego puedan combinarse funcionalmente unas con otras.

La alternativa monolítica se ajusta a un desarrollo sobre una única plataforma homogénea. La plataforma de experimentación, en el robot Maggie, está construida con componentes hardware de diferentes fabricantes, varios ordenadores y posiblemente sistemas operativos diferentes. Además es posible que algunas partes del hardware o el sistema operativo cambie en el transcurso del desarrollo, como en efecto ha ocurrido hasta ahora.

El sistema ideal de desarrollo se debe basar en partes de software con objetivos bien definidos. Estas luego deben interactuar con otras partes, integrándose para operar de forma cooperativa. En este contexto a este tipo de desarrollo y trabajo cooperativo se entenderá como *integración de procesos*.

Para poder integrar procesos es indispensable tener mecanismos de comunicación bien definidos, que permitan modificaciones internas sin requerir modificaciones en las otras partes. La aplicación se crea a partir de la unión de partes cuyo principales mecanismos de cohesión son los mecanismos de integración.

Existe una técnica que se ajusta a las necesidades de este trabajo, ésta se basa en componentes software [68]. Éstos están orientados al diseño de sistemas a partir de elementos de aplicaciones que son construidos independientemente por diferentes desarrolladores, usando diferentes lenguajes, herramientas y diferentes plataformas de cómputo. El objetivo es lograr un desarrollo similar al de los disponibles en el campo de la electrónica, donde existen componentes

electrónicos que pueden ser ensamblados y luego ser reemplazados.

El objetivo será entonces construir partes de software, que pueden probarse e integrarse a otras, con el fin de construir sistemas mas complejos totalmente integradas e integrables.

En este capítulo se revisaran los principales mecanismos de integración de software. Se inicia con la integración de software en ejecución (procesos) sobre un mismo ordenador y luego entre ordenadores interconectados por una red.

5.1. Integración de procesos en un ordenador

Hoy en día los sistemas operativos ejecutan procesos de forma concurrente, la ejecución puede ser de forma independientemente o de forma cooperante. Un proceso se ejecuta de forma independiente si éste no afecta o es afectado por otros procesos ejecutándose en forma concurrentemente. Cualquier proceso que no comparta datos con ningún proceso es independiente. Un proceso es cooperante si este es afectado o puede afectar a otros procesos en ejecución en el sistema. Por lo tanto, cualquier proceso que comparte datos con otro proceso es un proceso cooperante [69].

La razones para permitir la cooperación entre procesos son:

Compartir información. Algunas partes del software pueden necesitar usar la misma información.

Aumento de la velocidad de cómputo. Si se desea que una tarea se ejecute más rápido, es posible dividirla en subtareas que se ejecutarán paralelamente en varios procesadores.

Modularidad. Es posible construir el sistema de forma modular, dividiendo las funciones del sistema en procesos separados o hebras.

Conveniencia. Es posible que varias tareas del sistema pueden estarse ejecutándose de forma concurrente o paralela. Por ejemplo, los sistemas de visión trabajando en paralelo con el sistema de navegación.

El principal entorno de desarrollo es el sistema operativo Linux, por lo que las herramientas revisadas en esta sección son para dicho sistema. A continuación se describen algunos mecanismos de comunicación entre procesos.

5.1.1. Ficheros regulares con bloqueo.

La primera técnica de intercambio de datos entre procesos es por medio de ficheros regulares [1]. El problema fundamental es lograr la coordinación entre las

[1]Ficheros regulares: son aquellos que contienen datos representados como un flujo continuo de bytes

lecturas y escrituras por varios procesos de forma concurrente. Es preciso disponer de un mecanismo que permita inhabilitar las lecturas y escrituras mientras un proceso se encuentra modificando los datos en el fichero. El bloqueo de ficheros es uno de los mecanismos disponibles en la mayoría de sistemas operativos lograr la coordinación en la actualización de ficheros.

Existen dos tipo de bloqueos:

- Ficheros de bloqueo.

- Regiones bloqueadas.

La primera técnica exige que se cree y se abra un fichero de bloqueo antes de escribir el fichero de datos. Si se falla en la creación del fichero, se puede suspender el proceso de actualización por un tiempo, y luego intentarlo de nuevo. El contenido del fichero de bloqueo no importa, y puede ser un fichero vacío. Esta técnica es válida, si y solo si, los procesos involucrados cooperan y obedecen esta técnica.

La segunda técnica consiste en bloquear una región en un fichero. El mecanismo está implementado en los sistemas Linux por medio de llamadas a primitivas del núcleo del sistema operativo. Ejemplo de estas primitivas son *lockf* de System V [70] o *fcntl* de POSIX [71, 72, 73]. Una región consiste uno o más bytes consecutivos a partir de una dirección inicial dentro del fichero.

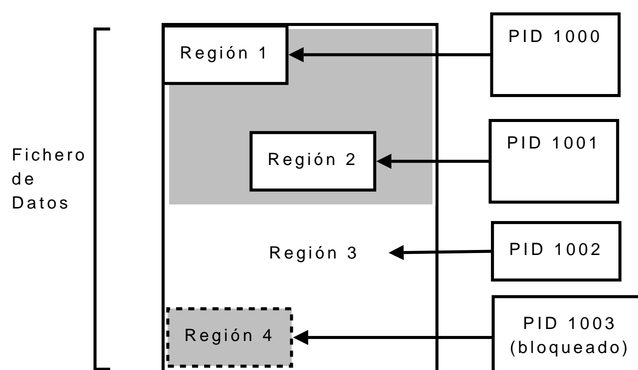

Figura 5.1: Coordinación entre procesos por bloqueo de región

En la figura 5.1 se muestra el funcionamiento del bloqueo de región. Los procesos 1000, 1001 y 1002 poseen regiones bloquedas en el fichero de datos. El núcleo del sistema operativo garantiza estos bloqueos ya que las regiones no se solapan. Estos tres procesos pueden actualizar sus regiones de forma concurrente. Si el proceso 1003 desea bloquear la región 4, esto plantea un conflicto ya que la región solicitada está solapada con la región 3. Por lo que el proceso 1003 es suspendido mientras se mantenga el bloqueo en la región 3.

5.1.2. Señales.

La ejecución de un programa usualmente se hace de forma síncrona, donde cada instrucción se ejecuta a continuación de la que la precede. Algunas acciones deben ejecutarse inmediatamente, interrumpiendo el flujo de la ejecución. Esto puede ser por una solicitud de terminar el programa o porque se debe procesar una nueva instrucción. La mayoria de los sistemas operativos proveen esta capacidad por medio de las *señales*.

Una señal es una interrupción asíncrona de software. La naturaleza asíncrona de la señal avisa al proceso que debe ejecutar una acción alterna. Consecuentemente, una acción debe registrarse antes de que la señal sea recibida.

Una señal suspende la ejecución de un programa. El procesamiento de la señal invoca la ejecución de acciones previamente registradas. La función que es llamada para procesar la señal es conocida como el *manejador de la señal*. Algunas señales pueden bloquearse para evitar las interrupciones y procesarlas luego.

5.1.3. IPC System V

En proyectos complejos de desarrollo de software a menudo se usan procesos separados para manejar la complejidad. Algunas veces, procesos separados presentan mejores desempeños en los sistemas con varios procesadores, como los disponibles hoy en día. Algunos problemas a resolver se adaptan de forma natural al modelo Cliente/Servidor. Sin embargo, una vez que se toma la decisión de usar procesos separados, crece la necesidad de compartir datos. Las herramientas estudiadas en esta subsección son las llamadas mecanismos para la intercomunicación de procesos IPC[2] System V[74, Cap 22, 23].

Los mecanismos IPC a describir son:

- Colas de mensajes.
- Memoria compartida.
- Semáforos.

Estos mecanismos de intercambio de información establecen un nuevo grupo de facilidades, éstos se caracterizan por usar manejadores particulares diferentes.

Colas de mensajes

Las colas de mensajes en System V implementan una cola de mensajes basadas en prioridades. El mensaje es un bloque de memoria que mantiene una

[2]IPC acrónimo de interprocess communications.

estructura de mensaje definido por cada aplicación. Cuando un mensaje es colocado en la cola, este se mantiene en el espacio de memoria del núcleo del sistema operativo. Este mensaje puede ser extraído de la cola por cualquier otro proceso.

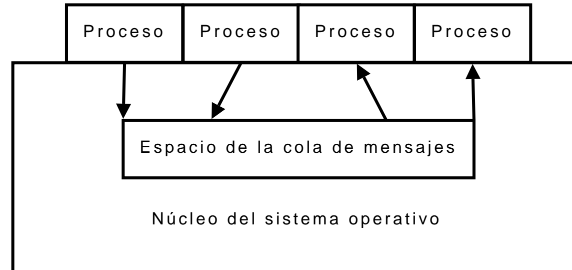

Figura 5.2: Cola en el núcleo del sistema operativo

La figura 5.2 muestra dos procesos colocando mensajes y dos procesos tomado mensajes. Hay una cola en general, pero es posibles que varios procesos coloquen mensajes y otro grupo los tome de la cola.

Cada mensaje tiene una prioridad. En la documentación se llama a ésta como el tipo de mensaje. Este tipo de mensaje determina la prioridad del mensaje cuando es colocado en la cola. La figura 5.3 muestra una serie de mensajes desde la letra *A* hasta la *J* entrando en la cola. El número que precede cada letra del mensaje indica la prioridad del mensaje. Por ejemplo, *3C* indica que el mensaje *C* fue colocado en la cola de prioridad *3*.

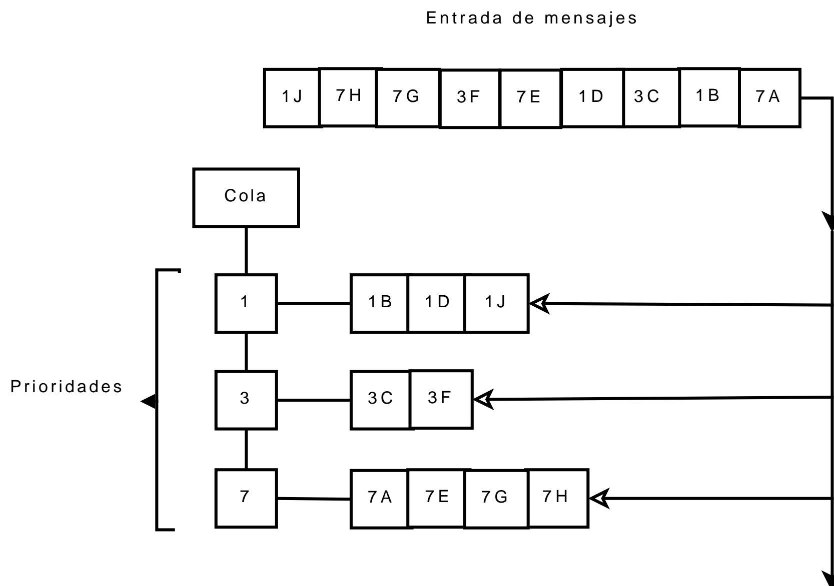

Figura 5.3: Cola de mensajes con prioridad

El núcleo del sistema operativo coloca cada mensaje en la subcola de prioridad a la que corresponde el mensaje. Los números más bajos indican la mayor prioridad en la cola de mensajes.

Cuando el proceso receptor toma un mensaje tiene varias opciones:

- Recibir un mensaje de prioridad x, o no recibir ningún mensaje si no existe alguno con prioridad x.
- Recibir el mensaje de mayor prioridad (menor número de tipo) con una prioridad mayor o igual que x.
- Recibir el primer mensaje que entró en la cola, siguiendo el modelo FIFO, sin tomar en cuenta la prioridad.

Mientras que la figura 5.3 muestra a todos los mensajes en subcolas de prioridad, el núcleo del sistema operativo mantiene aparte una lista enlazada con los mensajes ordenados siguiendo un modelo de cola FIFO. De esta manera, un proceso puede escoger si ignora la prioridad de mensaje simplemente eligiendo el mensaje que primero haya entrado en la cola.

Debido a que un mensaje puede ser recuperado en base a su prioridad, es posible usar la prioridad del mensaje (tipo del mensaje) para direccionar algún mensaje con un tipo en particular.

Memoria compartida

Cuando múltiples procesos cooperan, éstos a menudo necesitan compartir tablas de datos. La mayoría de los sistemas operativos proveen esta facilidad por medio de servicios de memoria compartida.

Aunque el concepto de memoria compartida es simple, ciertas complicaciones pueden ocurrir. Por ejemplo, en la figura 5.4, la región compartida puede ser enlazada al espacio de memoria de tres procesos en diferentes direcciones de memoria. Esto significa que si una dirección de memoria es usada para direccionar la región compartida, ésta no podrá ser usada por los procesos. Como alternativa debe ser usada una dirección de memoria offsets[3]. Ésta es la razón por la cual las bibliotecas compartidas deben ser compiladas usando direcciones independientes de la ubicación de almacenamiento.

Otra complicación es el problema de sincronización entre los tres procesos. Si múltiples procesos están cambiando datos en la región compartida, éstos pueden perder validez o consistencia. Es posible utilizar las colas de mensajes para sincronizar a los tres procesos, pero son los semáforos los mecanismos indicados para este propósito.

Semáforos

Un semáforo, en este contexto, es un mecanismo que mantiene un seguimiento a un contador y notifica los cambios a los procesos que estén interesados en

[3]dirección de memoria offsets: Valor inicial más valor adicional para producir una segunda dirección

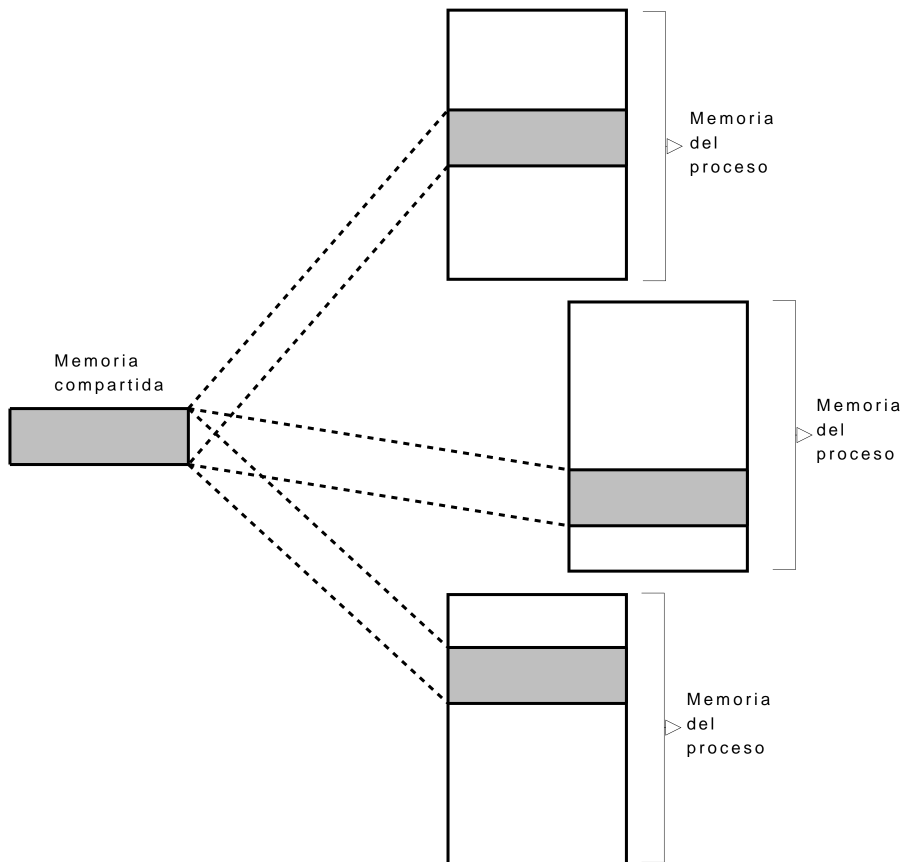

Figura 5.4: Región de memoria compartida por tres procesos

esos cambios. El semáforo más simple es un semáforo binario. el cual sólo permite mantener en el contador los valores de cero y uno. Un *mutex* es una forma particular de semáforo simple para permitir exclusiones mutuas entre hebras.

Otros semáforos permiten hacer un seguimiento a n instancias de un recurso en particular. Por ejemplo, si se tienen tres servidores de transacciones para servir a procesos clientes, se debería inicializar el contador del semáforo en tres. Al comenzar a ser atendido un cliente de debería decrementar el contador. Cuando la cuenta llegue a cero, el semáforo indica que no existen recursos disponibles en ese momento. Más tarde, cuando un cliente finalice con un servidor, se incrementa el contador del semáforo. Cuando todos los clientes liberen los servidores, el contador debería tener el valor 3. De esta manera, el semáforo mantiene la cuenta del número de recursos disponibles.

El acto de decrementar el contador de un semáforo es conocido como *espera en el semáforo*. Esta espera tiene sentido cuando el contador tiene el valor de cero, y debiendo esperar el proceso solicitante a que un recurso esté disponible.

El acto de incrementar el contador de un semáforo es conocido como *notificación al semáforo*. El incremento del contador causa que los procesos en espera

por un recurso sean notificados de la existencia de un recurso libre.

Los semáforos individuales trabajan bien controlando recursos individuales. Sin embargo, obtener varios recursos a la vez es a menudo requerido. Imagine el caso de una bolera, que tiene 50 pares de zapatos para bowling, 30 bolas de bowling y 6 pistas disponibles. Para jugar bowling, un jugador necesita de un par de zapatos de bowling, una bola de bowling y una pista. Sin embargo, un jugador no puede jugar si alguno de los recursos (zapatos, bolas, o pista) no está disponible. Un conjunto de semáforos permite hacer una solicitud a todos los recursos a la vez. De esta manera, no hay posibles bloqueos por recursos, debido a que la solicitud es exitosa o falla (queda en espera) en su totalidad.

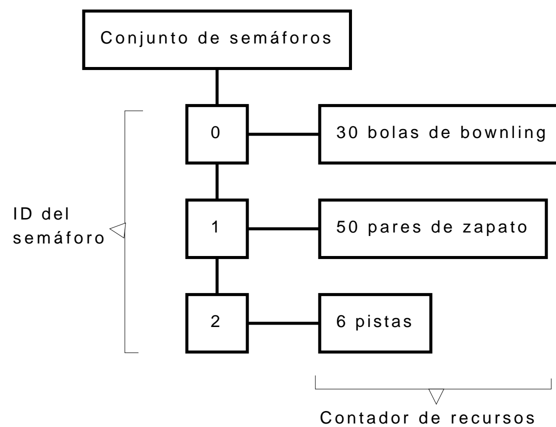

Figura 5.5: Conjunto de semáforos

La figura 5.5 muestra un conjunto de semáforos. En el conjunto, el semáforo cero *0* el recurso *bolas de bowling*, el semáforo *1* controla los pares de *zapatos de bowling* y el semáforo 3 controla las pistas.

La ventaja de agrupar estos recursos en conjuntos es que el solicitante puede obtener todos los recursos necesarios en una sola llamada al sistema, sin caer en situaciones de bloqueos activos(deadlock) si uno o más de los recursos no está disponible. La solicitud simplemente se bloquea hasta que todos los recursos solicitados estén disponibles.

5.2. Integración de procesos en varios ordenadores

Muchos de los desarrollos tradicionales de software se realizan sobre ordenadores independientes, en donde un solo ordenador contiene todos los componentes de software. En las aplicaciones independientes (*stand alone*), el flujo de control reside únicamente en el ordenador donde se inicia la ejecución.

En contraste, las aplicaciones distribuidas en varios ordenadores son divididas en servicios que pueden ser compartidos y reusados por múltiples apli-

caciones en diferentes ordenadores. Los servicios de red comunes proveen a los clientes en dichos entornos servicios de: directorios de nombres, servicio de ficheros en red, gestión de tablas de encaminamiento de paquetes, bitácoras y acceso remoto, entre otros.

En tiempo de ejecución, el flujo de control en una aplicación distribuida en varios ordenadores reside en uno o más ordenadores. Todos los componentes de la aplicación se comunican de forma cooperada, transfiriendo datos y consignas de control entre estos. La interoperabilidad entre componentes separados puede ser alcanzada, siempre y cuando, se usen protocolos compatibles. Esto ocurre a pesar de que el hardware, las capas subyacentes de redes, sistemas operativos y lenguajes de programación son heterogéneos [75]. Esta delegación de responsabilidades de servicios entre múltiples plataformas tiene los siguientes beneficios:

Aumenta la conectividad y la colaboración. La Información es diseminada rápidamente a más potenciales usuarios. Esta conectividad evita la necesidad de transferir manualmente y duplicar un dato.

Potencia el desempeño y la escalabilidad. Permite que las configuraciones de los sistemas sean cambiadas rápidamente y de forma robusta para configurar los recursos computacionales con las actuales predicciones de demandas de recursos.

Reduce costos. Permite a las aplicaciones compartir software y periféricos costosos.

En las siguientes subsecciones se estudiarán las principales alternativas para desarrollar e integrar aplicaciones distribuidas en diferentes ordenadores.

5.2.1. Sockets BSD

Las aplicaciones sobre redes de ordenadores requieren mecanismos para la comunicación entre procesos (IPC) y habilitar a los clientes y servidores el intercambio de información. El mecanismo de IPC disponible en los sistemas operativos puede ser clasificado en dos categorías generales:

- Los locales, son ciertos mecanismos de IPC, como lo es la memoria compartida, los cauces entre procesos (pipes), los sockets en el dominio de direcciones UNIX, o las señales (signals) que permiten la comunicación entre programas que se ejecutan en un mismo ordenador [76].
- Los remotos, otros mecanismos de IPC, tales como los sockets sobre el dominio de direcciones de Internet [55], circuitos X.25, y causes nombrados sobre Windows de 32 bits(Win32 Named Pipes), que soportan la comunicación entre programas que se ejecutan en ordenadores interconectados por medio de una red.

La API Socket fue desarrollada en BSD UNIX [77] para permitir una interfaz a nivel de capa de aplicación a los servicios ofrecidos por la familia de protocolos

TCP/IP. Esta API ha sido portada a la mayoría de los sistemas operativos y se ha convertido en el estándar por defecto para la programación entre procesos sobre TCP/IP.

Las aplicaciones pueden usar las funciones de C que forman la biblioteca de primitivas del API Socket para crear y gestionar los puntos finales de comunicación. Estos puntos finales son conocidos como sockets. Cada socket es usado por medio de un manejador, conocido en los entornos UNIX como *descriptor*. Un descriptor de un socket identifica un punto final de comunicación mantenido por el Sistema Operativo. Aunque a nivel superior se mantiene una interfaz similar, a bajo nivel del sistema operativo existen diferencias:

- En UNIX, el descriptor del socket y otros descriptores de entrada y salida pueden ser utilizados por varias operaciones de entrada y salida, como lo es write, read o close.
- En Microsoft Windows, el descriptor del socket no puede ser utilizado por otras operaciones de entrada y salida, aunque sean usados por operaciones similares.

Cada socket puede ser enlazado a una dirección local y recibir o enviar datos a través de éste, y en el caso de ser un socket orientado a conexión, puede ser conectado a una dirección remota. Estas asociaciones definen la asociación entre dos procesos que se comunican por medio del socket.

La API socket tiene algunas limitaciones: es propensa a errores, tiene una complejidad alta y no es portable ni uniforme.

5.2.2. Llamadas a procedimientos remotos

Este mecanismo fue introducido a principios de 1980 por los investigadores Andrew Birrel y Bruce Nelson como parte de su trabajo en el entorno de programación CEDAR [78]. La idea de las *llamadas a procedimientos remotos* (usualmente conocido como RPC, por sus iniciales en Inglés) es bastante simple. Ésta es basada en que las llamadas a los procedimientos y funciones locales desde un programa en C, Fortran, Pascal, etc, son bastantes conocidas y también es bien conocido el mecanismo para transferir el control y los datos dentro de un programa en ejecución en un ordenador, ver figura 5.6.

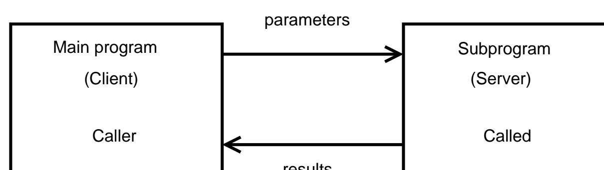

Figura 5.6: Modelo de una llamada a un procedimiento local

El hecho de que este mecanismo sea conocido y usado ampliamente entre la mayoría de programadores, ha sido la razón por la cual se eligió las llamadas a funciones en C para ser extendido con el objetivo de permitir la transferencia de control y datos a través de una red de comunicación. Cuando un procedimiento remoto es invocado, el contexto donde se realiza la llamada es suspendido. Prosteriormente, los parámetros son pasados a través de la red al contexto donde el procedimiento será ejecutado, ver figura 5.8. Cuando el procedimiento ha finalizado su ejecución y ha producido su resultado, el resultado es devuelto al contexto donde fue realizada la llamada. En ese momento se reactiva de nuevo el contexto suspendido de la misma manera que si se estuviera retornando de una llamada en el ordenador local, ver figura 5.7.

Mientras el entorno que hizo la llamada es suspendido, otros procesos en este ordenador pueden continuar ejecutándose (dependiendo de los detalles de paralelismo del contexto y de la realización del RPC).

El principal objetivo de la creación de RPC fue hacer la programación de sistemas distribuidos de una manera más fácil, ya que hasta la aparición de RPC la construcción de sistemas distribuidos era una tarea difícil, entendida solamente por un selecto número de expertos en comunicaciones. Esta idea llevó a la necesidad de mantener la semántica de una llamada a un procedimiento remoto lo más parecido posible a una llamada a un procedimiento local. Este principio se vería bastante atractivo para asegurar que las funcionalidades de RPC fuesen fáciles de usar, ya que es un mecanismo con el que los programadores están familiarizados en el uso de funciones y paquetes de software.

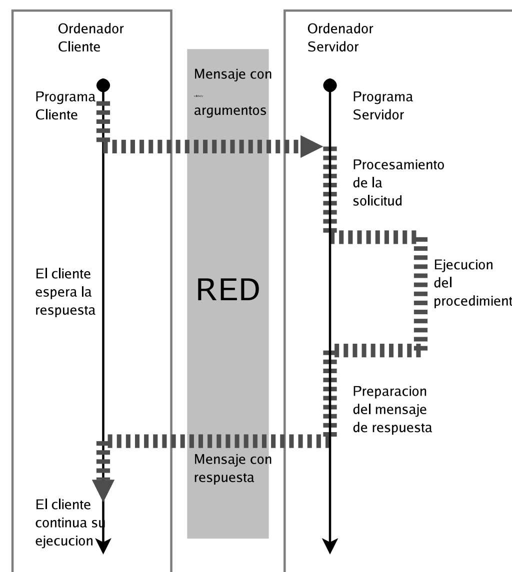

Figura 5.7: Diagrama temporal de llamada a un procedimiento remoto.

Desarrollo de una aplicación RPC

Aunque existen diferentes realizaciones de RPC, el desarrollo de una aplicación distribuida con RPC está basada en una metodología bien definida. En [79] se encuentra la principal referencia para comenzar a utilizar RPC, específicamente la implementación conocida como ONC-RPC, cuya especificación de la versión 2 esta registrada bajo el RFC 1831 y se puede obtener en [80]. Éste es un software comúnmente incorporado en las distribuciones Linux más populares [4], del que también existen versiones privadas para sistemas basados en los productos Windows de Microsoft, HPUX, VAX, Solaris.

El primer paso en la construcción de una aplicación bajo RPC es definir la interfaz para los procedimientos. Esto se realiza usando un *lenguaje de definición de interfaz (IDL)* que provee una representación abstracta de los procedimientos en términos de parámetros de entrada, parámetros de salida (respuestas) y nombres de los procedimientos. Esta descripción en IDL es considerada como la especificación de los servicios prestados por el servidor. Algunos autores como [79], llaman a esta descripción como la definición del protocolo de la aplicación. Teniendo la descripción en IDL, es posible pasar al desarrollo del servidor y del cliente.

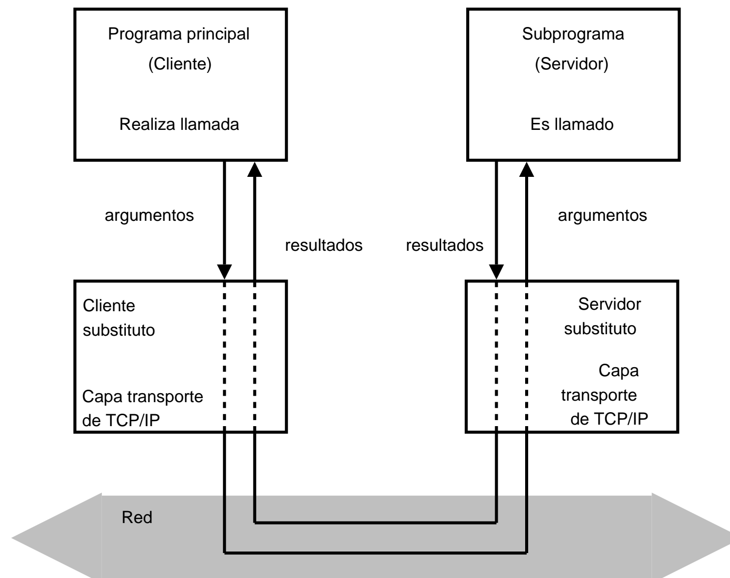

Figura 5.8: Modelo de una llamada a un procedimiento remoto

El segundo paso es compilar la descripción en IDL. La mayoría de implementaciones de RPC provee un compilador de interfaz [5] que produce:

[4]Debian, Mandrake, RedHat, Fedora
[5]En el caso de ONC-RPC el más común es *rpcgen*

- Los clientes suplentes (Client stub). Cada procedimiento descrito en el fichero que contiene la descripción en IDL tiene un cliente suplente. El suplente es una pieza de código para ser compilado y enlazado con el cliente. Cuando el cliente invoca a un procedimiento remoto, la llamada es realizada como una llamada a un procedimiento local definido en el cliente suplente. El cliente conoce o detecta la localización del servidor, da formato a los datos de manera apropiada [6] y los envía al servidor. A continuación espera el mensaje de respuesta, y al llegar éste lo reenvía como el parámetro de respuesta del procedimiento invocado por el cliente. En otras palabras, el cliente suplente sirve como interfaz o *proxy* para el actual procedimiento implementado en el servidor. El cliente suplente hace que el procedimiento remoto sea tratado como un procedimiento normal local. El cliente suplente, sin embargo, no define el procedimiento, solo implementa los mecanismos necesarios para interactuar con el servidor remoto y solicitar la ejecución del procedimiento remoto.

- Servidor suplente (Server stub). Éste es de naturaleza similar al cliente suplente excepto que implementa el lado del servidor en la llamada. Esto es, contiene el código para recibir la llamada desde el cliente suplente, colocando los datos en un formato apropiado, invocando el procedimiento implementado en el servidor y devolviendo los resultados obtenidos del procedimiento al cliente suplente. Al igual que el cliente suplente, éste debe ser compilado y enlazado al código del servidor.

- Plantillas de código y referencias. En muchos lenguajes de programación es necesario definir en tiempo de compilación los procedimientos que serán usados. El compilador de IDL ayuda en esta tarea produciendo los archivos auxiliares necesarios para el desarrollo. De hecho, las primeras versiones de RPC fueron desarrolladas para el lenguaje de programación C [16]. Adicionalmente al cliente y servidor suplente también son creados los archivos de declaraciones (los archivos **.h*) necesarios para la compilación. Los modernos compiladores de IDL ofrecen la posibilidad de generar plantillas de código para el servidor y el cliente.

Los suplentes en RPC pueden gestionar todos los detalles de programación de la interfaz hacia la red de datos, incluyendo los tiempos de espera y las retransmisiones si se ha usado un protocolo no confiable como UDP. Si es necesario un poco más de control, RPC provee diferentes funcionalidades, desde las más simples a las más complicadas.

5.2.3. Gestores de objetos

"El paradigma de orientación por objetos define un objeto como una entidad encapsulada, con una identidad distinta e inmutable, a cuyos servicios se puede acceder únicamente a través de interfaces bien definidas. Los clientes usan los servicios de un objeto para emitir peticiones al mismo. Los detalles de implementación del objeto y su localización están ocultos para los clientes"[75].

[6] En la mayoría de los casos usando el protocolo de representación de datos externos XDR

Los gestores de objetos[7], o corredores de objetos, extienden el modelo RPC al paradigma de orientación por objetos y provee un conjunto de servicios que simplifican el desarrollo de aplicaciones distribuidas orientadas a objetos. Se ha definido en [81, pag. 53] que los gestores de objetos son las infraestructuras de middleware que soportan la interoperabilidad entre objetos.

Los gestores de objetos aparecieron a principios de los años 90. El propósito de los gestores de objetos, especialmente en el inicio, fue exactamente el mismo que en RPC: ocultar mucha de la complejidad detrás de la invocación remota de procedimientos por medio de la similitud de las llamadas a los procedimientos locales. La diferencia es que en vez de invocar un procedimiento se invoca un método de un objeto. Debido a que la orientación por objetos incluyen las nociones de herencia y polimorfismo, la función desempeñada por el servidor depende de la clase a la cual pertenece el objeto en el servidor, y mas aún, diferentes objetos pueden responder de formas diferentes a la invocación del mismo método. Esto significa que el middleware debe crear correspondencia entre objetos clientes con objetos específicos ejecutándose en el servidor, gestionando la interacción entre ambos. Ésta era una de las principales funciones de un gestor de objetos. Con el tiempo, los gestores de objetos han extendido su funcionalidades, por ejemplo, incluyendo mecanismos de localización, nuevas técnicas de enlace dinámicos y gestión del ciclo de vida del objeto y persistencia.

Java Beans RMI

RMI [82] es un middleware RPC sobre Java que provee un modelo simple y directo para la creación de aplicaciones distribuidas con objetos Java. Está disponible únicamente para aplicaciones Java. RMI usa un concepto similar al ORB de CORBA (ver en pag 101) para permitir la localización de un objeto remoto. Debido a que es una implementación pura de Java RMI tiene la ventaja de que objetos enteros (en vez de sólo sus datos) pueden moverse entre clientes y servidores, permitiendo un polimorfismo totalmente orientado a objetos.

El la Figura 5.9 se observa el modelo de funcionamiento de una aplicación bajo RMI. El servidor crea varios objetos disponibles a otras aplicaciones y luego registra los objetos por parte del servidor. Una vez registrados los objetos, el servidor se queda a la espera de una solicitud sobre cualquiera de los métodos de los objetos registrados. Si un cliente desea obtener un acceso a un objeto, busca el objeto en la lista de registros y con la referencia obtenida invoca un método en el objeto remoto.

RMI usa un mecanismo estándar para la comunicación con objetos remotos: objetos suplentes (stubs) y servidores suplentes (skeletons). Un stub sirve de representante (proxy) del objeto remoto ante el cliente. El cliente invoca un método sobre el objeto suplente, el cual es responsable de llevar a cabo la llamada del método sobre el objeto remoto. En RMI, un objeto suplente de un objeto remoto implementa el mismo conjunto de interfaces remotas que el objeto remoto.

[7]Traducción aproximada del término original en inglés *Objetc Brokers*

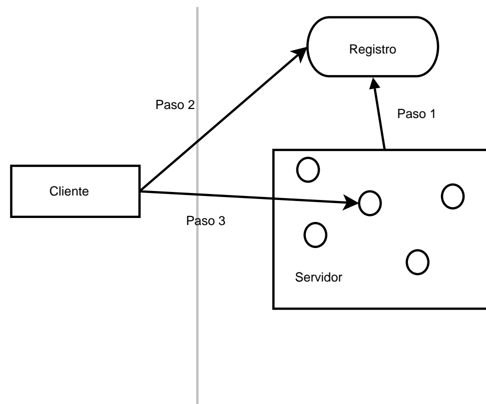

Figura 5.9: Modelo de una llamada a un método de un objeto remoto usando RMI

Cuando un método del objeto suplente es invocado ejecuta las siguientes tareas:

- se inicia una conexión con la *máquina virtual java (JVM)* remota que contiene al objeto remoto,
- se serializa y se envían (marshals) los parámetros a la JVM remota,
- se lee (unmarshals) el valor devuelto como resultado, y
- se devuelve el resultado al cliente.

El objeto suplente oculta la serialización de los parámetros, así como la comunicación a nivel de red con el fin de presentar al cliente un mecanismo simple de invocación de métodos.

En la JVM remota, cada objeto remoto tiene un objeto suplente (skeleton)[8], que es responsable de recibir y gestionar la solicitud del cliente. Cuando se recibe una invocación el skeleton:

- lee (unmarshals) los parámetros para el método remoto,
- invoca el método sobre la implementación del método remoto, y
- escribe y transmite (marshals) el resultado al stub.

CORBA

CORBA (Common Object Request Broker Architecture) es un estándar abierto para la construcción de aplicaciones distribuidas bajo el paradigma de

[8] en entornos con la plataforma Java 2, los skeleton no son necesarios.

orientación a objetos. Este estándar fue definido por el Object Management Group (OMG) [83]. CORBA es un *bus de objetos* que permite a los clientes invocar métodos de objetos remotos en un servidor independiente del lenguaje en el cual los objetos han sido escritos y, posiblemente, en un ordenador diferente. La interacción entre el cliente y el servidor es mediada por un **gestor de peticiones a objetos** (Object Request Brokers ORB). En ambos lados del cliente y el servidor, la comunicación es estandarizada por medio del protocolo IIOP (Internet Inter-ORB Protocol). Usando el protocolo estándar IIOP, un programa basado en CORBA en casi cualquier ordenador, sistema operativo, lenguaje de programación y red, puede comunicarse con otro programa basado en CORBA en casi cualquier otro ordenador, sistema operativo, lenguaje de programación, y red.

CORBA proporciona interfaces de programación independientes de la plataforma y modelos para aplicaciones portables basadas en objetos distribuidos. Su independencia del lenguaje de programación, de la plataforma del ordenador, y de protocolo de red, lo hace muy adecuado para el desarrollo de aplicaciones nuevas y para su integración en sistemas distribuidos ya existentes.

Como todas las tecnologías, CORBA tiene una terminología propia. Aunque algunos de estos conceptos y términos se han tomado prestados de otras tecnologías similares, a continuación se muestra una lista con algunos de los términos más importantes:

- Un *objeto CORBA* es una entidad "virtual"que puede ser localizada por un ORB y que puede aceptar peticiones de clientes. Es virtual en el sentido de que no existe realmente, a menos que se concrete mediante una implementación escrita en un determinado lenguaje de programación. La realización de un objeto CORBA mediante las construcciones de un lenguaje de programación es análoga a la forma en que la memoria virtual no existe en un sistema operativo, pero se simula usando memoria física.

- Un *objeto destino*, en el contexto de una petición CORBA, es el objeto al que se le hace la petición. El modelo de objetos CORBA es un modelo de *recepción única* en el cual el objeto destino de una petición se determina únicamente por la referencia usada para hacer la petición.

- Un *cliente* es una entidad que hace una petición sobre un objeto CORBA. Un cliente puede existir en un espacio de direcciones completamente separado del objeto CORBA o puede coexistir con el objeto CORBA dentro de una misma aplicación. El término cliente es significativo únicamente en el contexto de una petición particular, porque la aplicación que actúa de cliente en una petición puede tratar de ser servidor para otra petición.

- Un *servidor* es una aplicación en la que residen uno o más objetos CORBA. Como ocurre con los clientes, este término es significativo únicamente en el contexto de una petición.

- Una *petición* es una invocación de una operación [9] de un objeto CORBA realizada por un cliente. Las peticiones fluyen desde un cliente a un objeto

[9] realización de un método de un objeto

destino en el servidor. El objeto destino devuelve los resultados en su respuesta, si la petición los necesita.

- Una *referencia a un objeto CORBA* es un manejador que se usa para identificar, localizar y dar la dirección de un objeto CORBA. Las referencias a objetos son entidades opacas para los clientes. Los clientes usan referencias a objetos para dirigir las peticiones a los objetos, pero no pueden crear referencias a otros objetos a partir de sus constituyentes, ni pueden acceder o modificar el contenido de una referencia. Una referencia a un objeto está relacionada siempre con un único objeto CORBA.
- Un *sirviente* es una entidad de un lenguaje de programación que implementa uno o más objetos CORBA. Se dice que los sirvientes *encarnan* objetos CORBA porque proporcionan el cuerpo, o la implementación, de esos objetos. Los sirvientes existen en el contexto de una aplicación servidor. En C++ [17], los sirvientes son instancias de objetos de una clase particular.

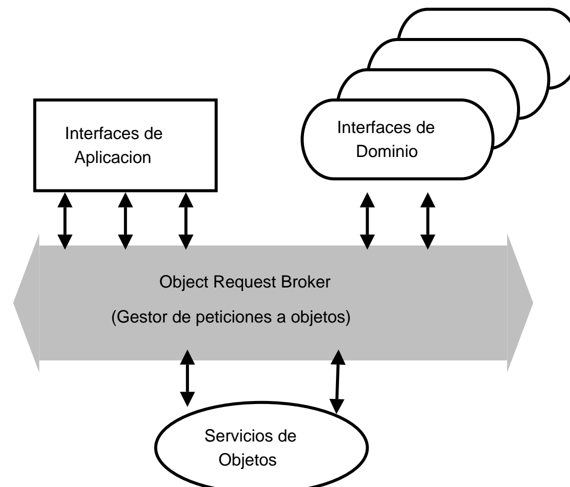

Figura 5.10: Categorías de interfaces en CORBA

La labor de ofrecer una comunicación transparente entre clientes de objetos es responsabilidad del *Object Request Broker (ORB)* que además enlaza conceptualmente todas las categorías de interfaz y activa los objetos inactivos cuando se emiten peticiones para los mismos. El ORB también proporciona una interfaz que puede ser usada directamente por los clientes, así como por otros objetos.

La figura 5.10 muestra las categorías de interfaz que usan las facilidades de comunicación y activación del ORB:

- Los *Servicios de Objetos* son interfaces dependientes del dominio, usadas por muchas otras aplicaciones con objetos distribuidos. Por ejemplo, todas las aplicaciones deben tener referencias a los objetos que quieren usar. El Servicio de Nombres y el Servicio de Trading [84] son servicios que permiten a las aplicaciones buscar y descubrir referencias a otros objetos.

- Las *Interfaces de Dominio*, que juegan un papel similar a aquellas existentes en las categorías de Servicios a Objetos, excepto porque estas interfaces son específicas de dominios. Por ejemplo hay interfaces que se usan en aplicaciones para la salud que son específicas para este tipo de industria, como el servicio de Identificación de Persona (*Person Identification Service*) [85]. Otras interfaces son específicas para las finanzas, la fabricación, las telecomunicaciones u otros dominios. El catálogo de especificaciones de interfaces de dominio hasta la fecha puede obtenerse en [86].
- Las *Interfaces de Aplicación*, desarrolladas específicamente para una determinada aplicación. No han sido estandarizadas por la OMG. Sin embargo, si ciertas interfaces de aplicación comienzan a aparecer en varias aplicaciones distintas, se convierten en candidatas para la estandarización en una de las categorías de interfaz existentes.

5.2.4. Web Services

El término *Web Services* es usado muy a menudo hoy en día, aunque no siempre con el mismo significado. Sin embargo, los conceptos y las tecnologías subyacentes son independientes de cómo pueden ser interpretados.

Existen rangos de definiciones que van desde conceptos muy genéricos y amplios hasta conceptos muy específicos y restrictivos. A menudo un Web service es visto como "*una aplicación en un servidor Web accesible a otras aplicaciones en la red*"(por ejemplo, ver [87]). Ésta es una definición muy amplia, que indica que *cualquier cosa* que tenga un URL (Uniform Resource Locator)[10] es un Web service. Este puede incluir, por ejemplo, a las aplicaciones sobre CGI [88]. Esta definición puede también hacer referencia a un programa accesible sobre un servidor Web con una interfaz software de aplicación (API) estable, con información descriptiva extensa y de dominio público en algún servicio de directorios.

Una definición más precisa es la definida por el Consorcio UDDI [11], el cual caracteriza un Web service como "*una aplicación comercial, modular y autónoma, que posee una interfaz basada en estándares, orientada a Internet y abierta*"[89, pag. 1]. Esta definición es más detallada, y hace énfasis en la necesidad de cumplir con los estándares de Internet. Adicionalmente, requiere que el servicio sea *abierto*, lo que significa esencialmente que posea una interfaz pública y disponible en Internet. A pesar de esta aclaración, la definición no es lo suficientemente precisa. Por ejemplo, no es claro lo que significa una aplicación modular, autónoma y comercial.

Un avance que ayuda a refinar la definición de Web Service es la dada por el Consorcio World Wide Web (W3C)[12], específicamente el grupo involucrado en las actividades sobre Web services: "*un software de aplicación identificado por*

[10] Un URL es una dirección global de documentos y otros recursos en la Web
[11] http://www.uddi.org
[12] http://www.w3c.org

un URI (Uniform Resource Identifier)[13]*, cuyas interfaces y enlaces son capaces de ser definidas, descritas, y halladas por entidades XML*[14]*. Un Web Service soporta interacciones directas de otros agentes software usando mensajes en el formato XML por medio de protocolos orientados a Internet*"[90].

La definición de la W3C es muy precisa y también sugiere como deberían trabajar los Web services. La definición impone que los Web services deben ser capaces de ser *definidos, descritos, y hallados* como consecuencia del significado de *accesible* y haciendo más concreta la noción *interfaces orientadas a Internet, basadas en estándares*. Esto también establece que los Web services deben ser *servicios* de manera similar a los encontrados en el Middleware convencional, aunque no solamente deben estar *disponibles y en ejecución*, sino que adicionalmente deben ser descritos y anunciados para que sea posible escribir clientes que puedan enlazarse e interactuar con éstos. En otras palabras los Web services son *componentes* que pueden ser integrados en aplicaciones distribuidas complejas.

5.2.5. Justificación de la selección

Las aplicaciones distribuidas son a menudo más difíciles de diseñar, implementar, depurar y optimizar que las aplicaciones residentes en un solo ordenador y construidas como un todo [91]. En [92] se describen técnicas para resolver las complejidades inherentes y accidentales asociadas con el desarrollo de aplicaciones distribuidas. Las complejidades inherentes surgen a partir de las características propias del dominio de la aplicación entre éstas se incluyen:

- Selección del mecanismo de comunicación y diseño del protocolo para su uso efectivo.
- Diseño de los servicios de red que permitan usar de una manera eficiente los recursos.
- Uso correcto de la concurrencia para alcanzar un alto desempeño y confiabilidad del sistema a desarrollar.

En este caso no se ha decidido un solo mecanismo de comunicación. El objetivo es tratar de adecuarse a cada problema específico y usar la herramienta que mejor se ajuste. Las líneas para la toma de decisión que se siguen son:

- En los casos donde la comunicación sea solo entre procesos de un mismo ordenador, se usará el sistema de colas de mensajes con prioridad System V. En casos de requerir el envío de estructuras de datos complejas, se serializaran (*marshelling*) en flujos de bytes para luego transmitirlos por

[13]URI es el término genérico para todos los tipos de nombres y las direcciones que se aplican a los objetos en la Web, un URL es una clase de URI

[14]La abreviatura para Extensible Markup Language, una especificación desarrollada por el W3C. XML es un especialización de SGML, diseñado especialmente para documentos en la Web. Habilitando a diseñadores a crear etiquetas hechas a la medida, y permitiendo la definición, la transmisión, la validación, y la interpretación de datos entre aplicaciones y entre organizaciones.

medio de sockets con direcciones de la familia *AF_ UNIX*. Todo esto con el fin de obtener el mejor rendimiento de velocidad de envío.

- Como herramienta principal para la comunicación entre procesos en diferentes ordenadores, se usa la herramienta ONC-RPC para la creación de componentes que se ajusten al modelo cliente-servidor y en aquellos casos donde los tipos de datos a transmitir sean de alta complejidad. En los casos donde se requiera transmitir datos simples se usará sockets BSD, con direcciones pertenecientes a la familia *AF_ INET*.
- Para la construcción de las interfaces a los dispositivos físicos se usará el lenguaje C.
- Las habilidades de la capa reactiva y algunas de la capa deliberativa deben construirse usando el lenguaje de programación C++.
- Las habilidades que requieran un nivel de abstracción, así como el uso de herramientas externas como manejadores de base de datos, herramientas de inteligencia artificial, interfaz gráfica con el usuario o acceso a contenido web pueden construirse con lenguajes de alto nivel como Java, Python, o similares, siempre y cuando pueda integrarse al resto de la plataforma desarrollada.
- Se debe dejar abierta la integración a sistemas como Web Service.

A pesar de existir varios mecanismos para la construcción de servicios, los componentes creados deben ocultar el tipo de mecanismo usado, y se deben garantizar que cumplan con:

Mínima complejidad. Cada componente debe tener una función única y bien definida.

Fácil mantenimiento. En el proyecto participan un grupo de desarrolladores y se debe pensar que los componentes serán revisados por personas diferentes al dessarrollador inicial.

Mínimo acoplamiento. Esto involucra interfaces de mínimo número de parámetros.

Extensibilidad. El sistema debe ser capaz de extender su funcionalidad sin afectarse la estructura subyacente.

Reusabilidad. Los componentes diseñados debe poder ser reusados en nuevas habilidades, y éstas a su vez conformar una biblioteca de habilidades, con el fin de construir nuevas habilidades complejas.

Portabilidad. Significa que el sistema debe poderse usar en las plataformas actuales y poder ser migrado a nuevas plataformas.

Compatible hacia atrás. Los nuevos componentes y las tareas de mantenimiento deben garantizar la reusabilidad de los elementos en la biblioteca de habilidades y componentes.

En este capítulo se han descrito las herramientas a usar en el diseño y construcción de la plataforma de control. En el siguiente capitulo se mostrará los componentes diseñados y construidos.

6

Diseño de componentes

En este capítulo se describen los componentes creados para construir una realización de la arquitectura Automática Deliberativa. Hasta ahora se han descrito las bases teóricas de las arquitectura de control de robots móviles y las posibles herramientas disponibles para su implantación en el Robot.

Se inicia el capítulo con el marco de diseño y construcción usado. Luego se entra en detalle de cada componente construido. Es importante destacar que gran parte del sistema construido tiene más de dos años en funcionamiento, otros en cambio varios meses. Algunos componentes no han sufrido muchas modificaciones desde sus versiones originales, mientras que otras en cambio han sido modificados para ajustarse a cambios en el hardware, por el surgimiento de nuevas aplicaciones o por razones de mantenimiento.

6.1. Metodología usada

Un **componente** (componente software) es un objeto que es definido por una interfaz y por una especificación. Un **objeto** es una entidad con un nombre (identidad) y tres tipos de responsabilidades: responsabilidad de recordar valores (atributos) que definen su estado; responsabilidad de llevar a cabo acciones (operaciones/métodos); y las responsabilidades de hacer cumplir reglas referentes sus atributos y operaciones (restricciones). Una **interfaz** es una lista de servicios que un objeto ofrece. Un `tipo` es una lista de reglas que debe cumplir un objeto (su especificación). Una **clase** es una interfaz con una implementación. Una clase tiene instancias y un componente tiene implementaciones.

En esta realización de la Arquitectura AD las habilidades se implementan por medio de componentes y estos a su vez se implementan por medio de clases de objetos. Por lo anterior, el elemento básico de construcción (constructo) de la Arquitectura AD son las Habilidades y la realización de éstas se basan en componentes software. Cada componente nuevo fue probado y luego pasó a ser usado directamente por los investigadores del grupo. Al usarlo crecían nuevas

expectativas sobre el mismo y surgían aplicaciones paralelas o colaborativas, que al final se han ido agregando como nuevas habilidades.

El desarrollo ha sido incremental [93, Capítulo 4]. Se trató de tener una plataforma inicial constituida por los componentes básicos y sobre estos se ha ido construyendo el resto de la arquitectura de control. Este tipo de desarrollo ha permitido el uso del hardware básico del robot desde las etapas iniciales del proyecto. La guía de cuáles componentes a desarrollar ha sido la misma arquitectura AD, ver 2.2 y la arquitectura AD es por lo tanto el *dominio* de la aplicación. Los desarrollos se inician con la realización de los actuadores y sensores virtuales, una vez que estos se construyen y prueban se valida el marco de trabajo y se inicia la etapa de los componentes que permitirán construir los diferentes elementos de la arquitectura.

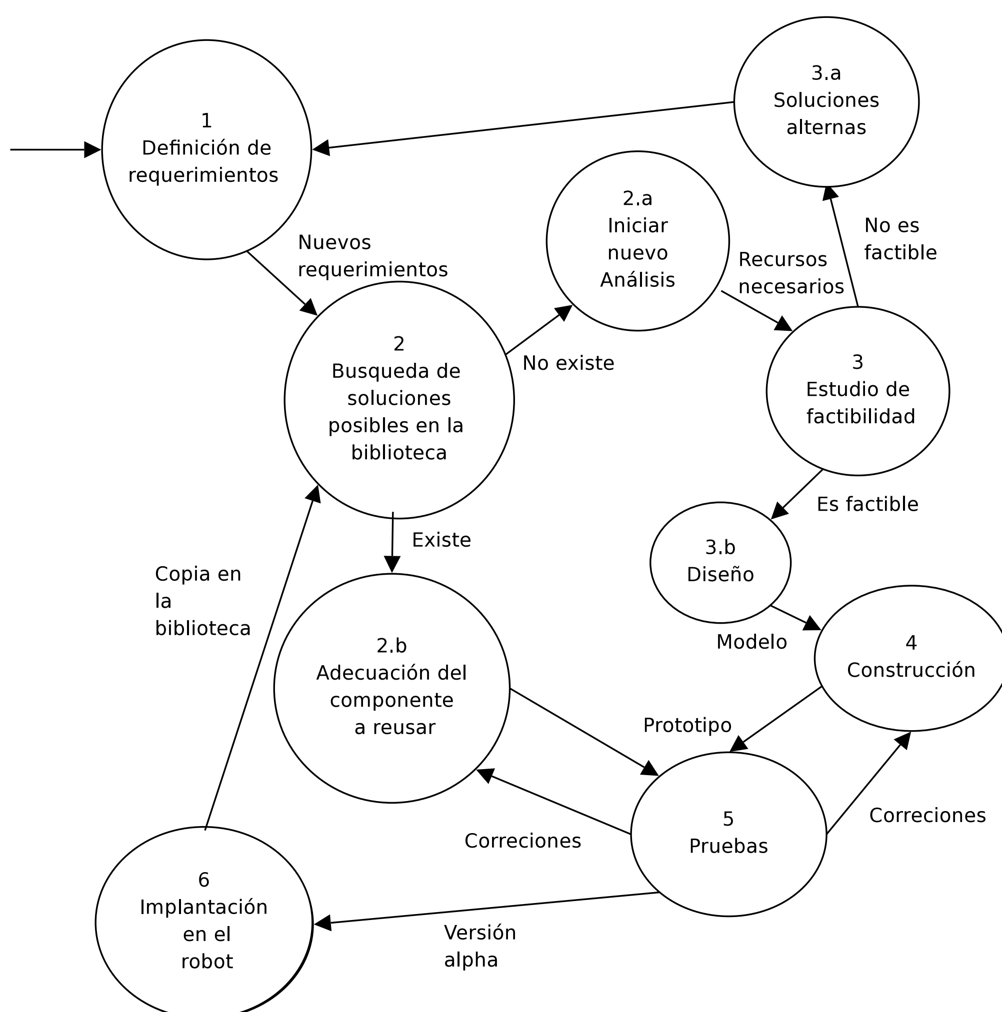

Figura 6.1: Fases del desarrollo de cada componente

En la figura 6.1 se muestra la metodología usada para la construcción de los elementos de la arquitectura AD. Cada elemento puede ser un componente o

parte de uno. Uno o varios componentes pueden combinarse para formar una habilidad simple. Esta metodología se basa en la *metodología de desarrollo de componentes* [94, pag. 28,473-475] y desarrollo incremental [94, pag. 23-24], en el dominio de la arquitectura AD. A continuación se describen cada una de las etapas:

Definición de requerimientos. En esta fase se evalúa el desarrollo del proyecto y los componentes a desarrollar. Una vez que se elige el componente a desarrollar se definen los requerimientos que debe satisfacer.

Búsqueda de posibles soluciones en la biblioteca. La biblioteca de componentes del proyecto está soportado por medio del sistema manejador de versiones *subversion* [95]. Este sistema mantiene las diferentes versiones de los componentes creados y en desarrollos. La documentación del mismo es llevada por la página Wiki del proyecto [96]. Si existe un componente que pueda ser adecuado para cubrir los requerimientos se pasa a la fase de Adecuación, en caso contrario se inicia un nuevo análisis de desarrollo.

Adecuación del componente a usar. El nuevo componente puede ser un componente ya creado o un componente creado partir de la especialización y/o composición de otros componentes en la biblioteca. En este caso se realiza la operación necesaria para la obtención del nuevo prototipo.

Iniciar un nuevo análisis. En el caso de que no exista un componente en la biblioteca del proyecto que pueda ser usado, se inicia el análisis para el diseño y la construcción del nuevo componente. Se estima también el hardware y el software necesario, la disponibilidad de tiempo del desarrollador, estaciones de trabajo, etc. Al final se obtiene la lista de recursos necesarios.

Estudio de factibilidad. En esta fase se analiza la lista de recursos contra las disponibilidades actuales. Si es necesario adquirir nuevo hardware o software se realiza la solicitud de cotización para la compra en caso de existir recursos económicos. En caso contrario se decide si se suspende por un tiempo el desarrollo, en espera de los recursos. También se deciden que personas se involucrarán en el desarrollo, su disponibilidad de tiempo y el análisis de tiempos esperados de desarrollo vs fecha máxima en el cual el componente debe estar listo. En el caso de existir disponibilidad de recursos se pasa a la fase de Diseño, en caso contrario se pasa al estudio de Alternativas Alternas.

Soluciones alternas. En esta etapa se trata de buscar una salida alterna a la falta de algún recurso. Esto puede ser utilizar hardware en calidad de préstamo, o usar software alterno para el desarrollo. En este caso se estudia el impacto de la nueva alternativa sobre la satisfacción de requerimientos. Si es posible plantear nuevos requerimientos se pasa de nuevo a la redefinición de los mismos. En caso contrario se suspende el desarrollo del componente por tiempo indefinido.

Diseño. En esta etapa se define en que parte de las capas de arquitectura AD se ubica el componente. También se hace una descomposición en subsistemas cuyo trabajo integrado satisface los requerimientos de la creación

del elemento. En cada subsistema se especifican las clases a usar o a construir, las relaciones entre estas, así como los mecanismos de comunicación. También se especifican los orígenes y destinos de los datos involucrados en el uso del componente.

Construcción. En la fase inicial se selecciona el lenguaje de programación, y los sistemas operativos donde se ejecutará el componente software a usar. Una vez definida esta parte, se procede a la codificación de los diseños obtenidos en la fase anterior. Se hacen las primeras pruebas de caja negra y pruebas de caja blanca. A medida que se va codificando el componente se agrega la información necesaria para que el sistema de documentación [1] genere la misma de forma automática.

Pruebas. Cuando el componente pasa a esta etapa, se realizan las primeras pruebas de integración al resto de la plataforma, usándose datos reales para pruebas de caja negra y caja blanca. Se realizan pruebas para medir el tiempo de ejecución y de desempeño general al estar integrado en la arquitectura.

Implantación en el robot. Una vez que el componente ha sido creado, pasa a estar disponible al resto de usuarios del robot. La documentación generada de forma automática se almacena en la Wiki del proyecto y el componente se almacena el la biblioteca de componentes del proyecto [2].

El marco de trabajo se establece por medio de patrones de diseño y la selección de este último depende exclusivamente de la naturaleza del componente en sí. Los componentes básicos han sido diseñados como una interfaz entre el hardware y la arquitectura AD. En este caso se ha utilizado el modelo *cliente-servidor* como principal patrón de diseño.

El software intermedio, llamado *Bus de Datos* de la Arquitectura AD, es el encargado de permitir la comunicación entre habilidades. Este está conformado por el sistema de gestión de eventos y el sistema de memoria compartida. El sistema de gestión de eventos usa el patrón de diseño Publisher/Subscriber. El sistema memoria compartida usa un sistema de mapas distribuidos por medio de una jerarquía de servidores auxiliares.

Las plantillas de habilidades usa la técnica de programación parametrizada para la derivación de cada una de las habilidades a partir de un patrón inicial. El secuenciamiento de habilidades se describe al final del capítulo y se basa en una realización de modelos basados en Sistemas Discretos Manejados por Eventos.

6.2. Primeras aproximaciones

Unos de los primeros pasos a abordar fue la abstracción de un robot móvil simple que tiene un sistema electro-mecánico que permite trasladar al robot sobre una superficie plana. El objetivo principal de esta primera abstracción

[1] El sistema actualmente usado para la documentación es Doxygen
[2] La biblioteca de componentes es gestionada actualmente por el sistema Subversion

fue generar un modelo que ocultase todos los detalles internos referentes a los componentes que permiten la traslación del robot. La solución a este problema fue la creación de un tipo de dato abstracto, que posee atributos que representan el estado del robot y todas operaciones necesarias para controlar el movimiento. El uso de un tipo abstracto de datos permite aislar solo aquellos aspectos que son importantes, según el diseñador, para el control del robot, así como el conjunto finito de operaciones con interfaces bien definidas.

Una vez definido el tipo abstracto de datos se procede a su implementación en un lenguaje de alto nivel, en este caso se ha elegido al lenguaje C++. Por ser este un lenguaje compilado y orientado a objetos permite un buen desempeño y un nivel aceptable de abstracción.

A continuación se presenta el código original de la declaración del tipo abstracto de dato Robot implementada como una clase en C++:

```
#ifndef CROBOT
#define CROBOT
/************************************************************
 * Clase RobotBasico v0.1
 * Rafael Rivas Estrada
 * UC3M-ULA rafael.rivas@uc3m.es rafael@ula.ve
 *
 * Esta clase solo presenta las operaciones Básicas con fines de
 * demostración del B21 Robot System
 * **********************************************************/

#include <termios.h>
#include <stdio.h>

class CRobot {
private:
        int dispositivo;
        int bitacora;
        struct termios parametrosComunicacionAnteriores;
        struct termios parametrosComunicacionActuales;

        CRobot( const CRobot&);

public:
        CRobot();
        ~CRobot();

        int EncontrarEje();

        int DetenerRotacion();
        int RotarDerecha(unsigned int);
        int RotarIzquierda(unsigned int);

        int DetenerTraslacion();
        int TrasladarAdelante(unsigned int);
```

```
        int TrasladarAtras(unsigned int);

        int ConsultaRotacion();
        int ConsultaTraslacion();
};

#endif
```

La definición de las operaciones y los detalles del control de los motores están ocultos al usuario final. Estos detalles no dejan de ser importantes para la operación y correcto control del movimiento del robot, pero si estos son ya conocidos no es necesario reescribirlos cada vez que se requiere usar el robot. Basta con hacerlo una vez, compilarlo y colocarlo en una biblioteca, de forma que esté disponible en el momento de uso.

Lo importante es que por medio de esta clase es posible comenzar a usar el robot, al menos en movimientos a lazo cerrado. A continuación se muestra un programa completo que avanza el robot 1000 mm hacia adelante y luego se le indica que retroceda la misma distancia:

```
#include <robot.h>

int main(){
        CRobot zape;

        zape.TrasladarAdelante(1000);
        zape.TrasladarAtras(1000);

        return(0);
}
```

En la primera línea se incluyen las declaraciones de la implementación del tipo abstracto de datos Robot, implementado como la clase CRobot. En la línea

```
        CRobot zape;
```

se crea un objeto de la clase CRobot, y de forma implícita se invoca el constructor de esta clase. Este ejecuta todas las operaciones de inicialización de las comunicación con el hardware encargado del control de los motores y la activación de las operaciones de inicialización para realizar la traslación del robot. En las siguientes líneas

```
        zape.TrasladarAdelante(1000);
        zape.TrasladarAtras(1000);
```

Se le indica al robot que avance 1000 mm y luego retroceda la misma distancia. De nuevo todos los detalles del movimiento quedan ocultos al usuario del robot.

```
        return(0);
```

En la línea de código mostrada anteriormente se indica el fin del programa, pero implícitamente se activa el destructor de los objetos de la clase CRobot.

Este método ejecuta las operaciones de frenado del robot, desactivación del robot y cierra la comunicación con los manejadores de los motores del robot.

Esta abstracción que se hace sobre los mecanismos de traslación, para su fácil manipulación, fue hecha también a los dispositivos de teledetección inicialmente al sistema de sonars y posteriormente al sistema de Laser. Siguiendo siempre los mismos pasos: primero el diseño del tipo abstracto de datos y luego implementación de ese tipo abstracto de datos como una clase en el lenguaje C++, y todo sobre un sistema con una distribución Linux.

Al final de esta etapa se posee una biblioteca de clases que permiten el uso de parte del hardware del robot por medio de una aplicación hecha en lenguaje C++. La ventaja es muy clara: un fácil acceso a la programación del robot mediante un lenguaje de alto nivel. El inconveniente principal de esta aproximación es que se opera al robot en modo real[3], por medio de un lenguaje de alto nivel que de por si no posee las herramientas para la multiprogramación, la comunicación y sincronización de procesos[97, Capítulo 6].

Al hacer referencia a la operación del robot en modo real implica que no existe una arquitectura de control, y si existe es mínima, que gestione los servicios y recursos del robot. Otra de las desventajas es que por limitaciones del lenguaje las aplicaciones deben coexistir en un ordenador con conexión directa al hardware del robot. Todo esto conlleva a un diseño que permita la realización de la arquitectura AD por medio de un sistema que admita la existencia de varias habilidades activas, interactuando entre si por medio de eventos y un sistema de memoria a corto plazo, posiblemente esparcidas en varios ordenadores dentro o fuera del robot.

Como una plataforma para la implementación se plantea un nivel de abstracción que oculte los detalles del hardware del robot en sí, a esta primera capa de software en contacto directo con el hardware se llama Componentes Básicos y será descrita en la siguiente sección.

6.3. Implementación de sensores y actuadores virtuales

Los componentes básicos forman el subsistema que sirve de interfaz entre el hardware del robot y su sistema de control programado. Dentro de la Arquitectura AD este subsistema implementa los actuadores y sensores virtuales. Los requisitos de diseño de los componentes básicos son:

- Poseer la funcionalidad atómica básica de cualquier herramienta de programación de software de control de robots móviles.
- Ser escrito en una lenguaje "bien conocido"que permita reducir el tiempo de aprendizaje.

[3]El termino de operación en modo real es usado como una extrapolación de la operación en modo real de los sistemas operativos.

- Que presente un nivel alto de abstracción de los detalles de funciones de servicio de las redes de comunicaciones.
- Una interfaz clara, que permita el uso independientemente de la plataforma hardware subyacente.
- La capacidad de extensión de los componentes y la reutilización de componentes creados y probados.
- Rendimiento en ejecución que permita cumplir con los compromisos de tiempo requeridos por las aplicaciones.
- Al ser componentes de funcionalidad simple debe permitir la composición en componentes más complejos.

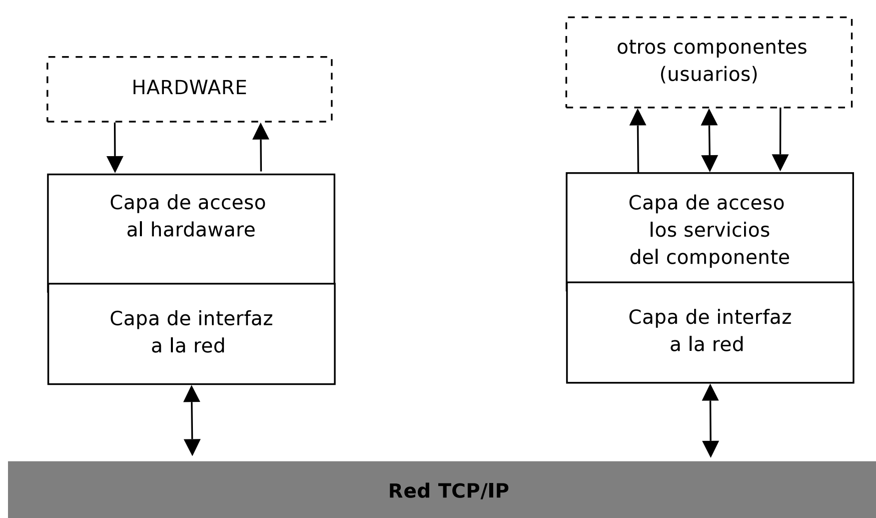

Figura 6.2: Arquitectura de un componente básico

El primer componente básico que se desarrolló fue el componente CMoveBase, que presenta la arquitectura de diseño de todos los componentes básicos que consiste en un sistema de tres capas. La figura 6.2 muestra la arquitectura de los componentes básicos en general. Los componentes básicos generalmente están formados por dos partes: una única instancia que se ejecuta en el ordenador que tiene acceso al hardware a controlar y una o mas instancias que pueden ejecutarse en uno o varios ordenadores diferentes.

Por razones de desempeño se hace esta segmentación, ya que originalmente hay ordenadores especializados para usar un hardware, en particular como lo son los dispositivos de vídeo.

Esta división también ha permitido hacer cambios en el hardware sin afectar la interfaz del componente. De hecho el componente que se encarga de implementar la interfaz permanece, o debe permanecer, constante ante cambios del hardware.

Otro factor importante radica en el hecho que la comunicación entre los elementos internos del componente es encapsulada, por lo que el uso no implica solicitud de conexiones, ni secuenciación de mensajes. Toda la comunicación es oculta al usuario, en este caso otro componente.

Por el hecho de ser autocontenido es posible el uso de los servicios del componente por aplicaciones fuera de la arquitectura AD. Este hecho ha permitido poder usar y probar algunos desarrollos en las etapas iniciales del proyecto, sin ser necesario tener que esperar que todo el proyecto esté terminado.

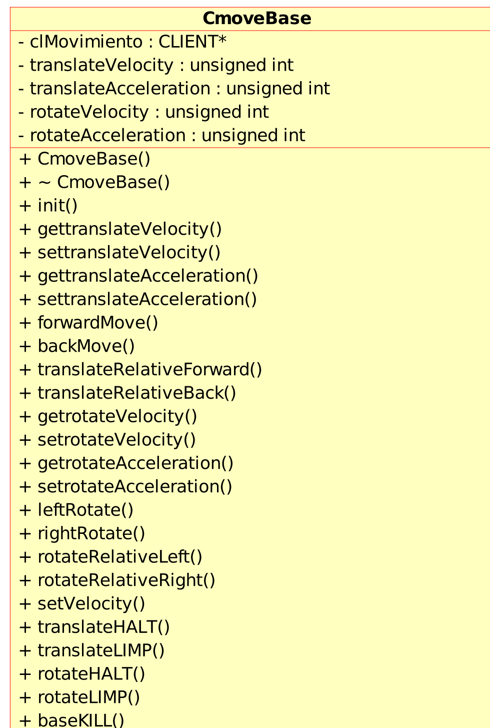

Figura 6.3: Diagrama de la clase CmoveBase

En la figura 6.3 se presenta la clase CMoveBase, los objetos de esta clase implementan la interfaz del componente MoveBase. La clase se encarga de especificar los servicios ofrecidos por los componentes. Estos servicios se han definido independientemente del hardware de movimiento con el fin de permitir una mayor independencia con el hardware usado.

En el lado más próximo al hardware el componente tiene dos elementos implementados por hebras, específicamente POSIX Threads. Una hebra periódicamente toma los datos de odometría de los motores y los coloca en memoria compartida local del componente. De esta región de memoria toma las consignas o punto de operación, como lo es velocidad lineal, velocidad angular, aceleración o movimientos a lazo cerrado. La otra hebra se encarga de atender peticiones de datos o consignas de control provenientes de la interfaz vía Internet.

Una vez probado el componente moveBase, se diseñó y construyó los componentes encargados de ofrecer los servicios de: Interfaz a los dispositivos SICK Laser, modelos MLS y PLS; y la Interfaz a los sensores SONAR.

Los primeros componentes han servido de modelos, o marco de desarrollo, para que el resto de los investigadores del grupo del robot Maggie propusiesen y desarrollasen posteriormente los siguientes componentes:

- Interfaz a los sensores de tacto.
- Interfaz a los sensores de temperatura.
- Interfaz a los motores de brazos.
- Interfaz a las cámaras de vídeo.
- Interfaz a los reconocedores de voz.
- Interfaz a los sintetizadores de voz.
- Interfaz a los detectores de tarjetas RFID.

Los componentes básicos son la base para el desarrollo de la implementación de la Arquitectura AD, pero estos también pueden ser usados en otros desarrollos como el proyecto IVLM International Virtual Laboratory on Mechatronics [4], en el cual usando herramientas de Web Services se pudo usar los servicios del Robot desde sitios remotos usando Internet como mecanismo de comunicación.

6.3.1. Ejemplo de uso de los componentes básicos

Los componentes básicos son parte de software con capacidades de multiprogramación y comunicación entre procesos. Como se ha descrito anteriormente éstos han sido diseñados siguiendo el paradigma cliente-servidor. Una vez que el servidor se encuentra disponible en un ordenador anfitrión con acceso al hardware dentro del robot, lo clientes pueden comenzar a enviar solicitudes de ejecución de operaciones. A continuación se muestra un programa completo que usa los servicios de los componentes básicos, específicamente el actuador virtual que controla los motores, y el sensor virtual del láser.

```
/**
  @file giraCercano.cpp
  @brief Programa de prueba de la clase CLaserBase y CMoveBase.

  Requiere:
    laserBase.h, moveBase.h  libLaserBase.a libMovebase.a

  Modo de compilacion:
    g++ giraCercano.cpp -o giraCercano.exe -lLaserBase \
    -lMoveBase -I/usr/local/AD/include -L/usr/local/AD/lib
```

[4]Más información en http://roboticslab.uc3m.es/roboticslab/proyecto.php?id_proy26

```
  Modo de ejecucion en vismaggie:
  ./giraCercano.exe localhost

  @author Rafael Rivas
  @code
*/

#include <iostream>
#include <string>
#include <math.h>
#include <signal.h>
// Archivos de declaraciones necesarios
#include <laserBase.h>
#include <moveBase.h>

int fin;

void finMovimiento(int e){
        fin = 0;
        return;
}

int main(int argc, char** argv){
  int i;
  std::string host;
// Declaraciones requeridas
  CLaserBase laser;
  CLaserBase::t_datosLaser* v;
  CmoveBase ruedas;

  float dmenor, tmenor;

  if( argc == 2){
    host = argv[1];
  } else {
    std::cout << "Ordenador donde se encuentra conectado el laser\
                 y los motores: ";
    std::cin >> host;
  }

  fin = 1;

  signal(SIGINT, finMovimiento);

  if ( laser.init(host.c_str())){
    std::cerr << "Error: Servidor laser desconocido\n";
    return(1);
  }
  if ( ruedas.init(host.c_str())){
```

```
    std::cerr << "Error: Servidor de motores
desconocido\n";
    return(2);
  }


  while ( fin ) {
// Peticion de datos
    v = laser.get();

//Busqueda del objeto mas cercano
    dmenor = 20000;
    for(i = 0; i < laser.length(); i++){
      if ( dmenor > v[i].r) {
        tmenor = v[i].theta*180.0/M_PI;
        dmenor = v[i].r;
      }
    }
  if ( tmenor >= 95 ) {
    std::cout << "a mi derecha\n";
    ruedas.setVelocity(0, -8);
  }else
    if ( tmenor <= 85 ) {
      std::cout << "a mi derecha\n";
      ruedas.setVelocity(0, 8);
    } else {
      std::cout << "al frente\n";
    }
  }
// Colocar velocidades en cero.
  ruedas.setVelocity(0, 0);
  return(0);
}
```

Este es un ejemplo simple del uso combinado del sensor virtual del láser junto al actuador virtual de los motores. En este caso sensor virtual del láser continuamente explora los alrededores del robot, la aplicación utiliza los datos de la última exploración en búsqueda del obstáculo más cercano. Una vez obtenido éste, se envía las consignas de giro al actuador virtual de los motores para lograr un ángulo mínimo entre el frente del robot y el obstáculo. Estas operaciones se hacen en forma continua hasta que el operador envía la señal SIGINT que detiene al robot. Al finalizar de forma correcta la aplicación, los destructores de los objetos creados cierran los enlaces entre la aplicación y las implementaciones de los actuadores y sensores virtuales.

Como se indicó anteriormente, este programa utiliza los servicios ofrecidos por los componentes básicos, por lo que se hace necesario que las implementaciones de los actuadores y sensores virtuales estén activos, se encuentren en un ordenador cuya dirección sea bien conocida. El programa descrito debe ejecutarse en un equipo anfitrión con conexión a los ordenadores donde residen los

actuadores y sensores virtuales, ya que cada componente básico puede estar en ordenadores diferentes.

La función definida en:

```
void finMovimiento(int e){
  fin = 0;
  return;
}
```

da servicio a la señal SIGINT o señal de interrupción, enviada al presionar las teclas Control y C al mismo tiempo. Esta señal debe capturarse con el fin de no permitir que el programa sea interrumpido sin antes enviar la orden de frenado en las ruedas del robot.

```
CLaserBase laser;
CmoveBase ruedas;
```

Las declaraciones anteriores crean instancias de los objetos pertenecientes a las clases que implementan los enlaces a los actuadores y sensores virtuales. Los constructores de estos enlaces preparan el sistema de comunicación para el intercambio de órdenes y datos.

Las llamadas a los métodos laser.init(host.c_str()) y ruedas.init(host.c_str()) enlazan la aplicación con los sensores y actuadores virtuales. En este nivel de abstracción se debe indicar exactamente la ubicación de los anfitriones a los componentes. Si las operaciones ejecutadas por lo métodos son exitosas, se crea un canal de comunicación y el sistema está listo para el intercambio de consignas de control y datos obtenidos de los actuadores y sensores virtuales.

Una vez verificados los enlaces se pueden obtener los datos del láser por medio del método laser.get(). Estos datos son procesados y se calcula el giro necesario para colocar el robot frente al objeto más cercano. Una vez calculado el giro se envía la orden de giro por medio de ruedas.setVelocity(0, -8) o ruedas.setVelocity(0, +8), según sea la posición del objeto detectado.

Al capturar la señal SIGINT definida en signal.h se termina el lazo de operación y se detiene el movimiento del robot con ruedas.setVelocity(0, 0).

En este punto del documento, la primeras aproximaciones han servido como bases para la implementación de los actuadores y sensores virtuales, a este nivel se ha llamado componentes básicos. Éstos tienen la abstracción de los componentes físicos, pero también poseen la capacidad de recibir consignas y/o enviar datos a procesos en ordenadores interconectados a los ordenadores donde residen los componentes básicos.

Para avanzar en la realización de la arquitectura AD, hace falta crear los mecanismos para el envío de eventos y la implementación de la memoria a corto plazo. En el siguiente capítulo se detalla en la funcionalidad de estos elementos de la arquitectura AD.

7

Eventos y Memoria a Corto Plazo

Hasta ahora se han visto los mecanismos de abstracción usados para ocultar los detalles de la manipulación del hardware en la realización de la arquitectura AD. También se ha descrito como la concurrencia y/o paralelismo[1] son implementadas. En este capítulo se describen la implementación del sistema de eventos, que provee los mecanismos para la propagación de eventos a través de los elementos que requieren esta funcionalidad dentro de la arquitectura AD. En la segunda parte del capítulo se describe el sistema de memoria a corto plazo, implementado con el fin de permitir el intercambio de datos cuya naturaleza no requiere el almacenamiento en memoria secundaria.

7.1. Sistema de gestión de eventos

En el año de 1987 Ramadge y Wonham[98] describen una extensión a la teoría de control de sistemas discretos. En esta propuesta se incorpora los lenguajes formales y Autómatas como herramientas en el diseño de controladores. El método usado fue la emisión de eventos con el fin de lograr inhibición/activación de transiciones de estado para evitar alcanzar ciertos estados. En la arquitectura AD las habilidades se comportan de manera cooperativa, y en algunos casos una o varias habilidades están subordinadas a otras habilidades. El mecanismo ideal para la coordinación del comportamiento de habilidades en el paradigma AD se realiza por medio la emisión y captura de eventos. Para poder lograr una realización es necesario disponer de un mecanismo de gestión de eventos, pero esta gestión debe realizarse pensando en que las habilidades son componentes básicos, en la mayoría de los casos reusables en diferentes aplicaciones.

El sistema de eventos usado en Maggie es débilmente acoplado, lo que impli-

[1] Se obtiene un paralelismo real al asignar partes del software construido a diferentes ordenadores o a ordenadores con varios procesadores

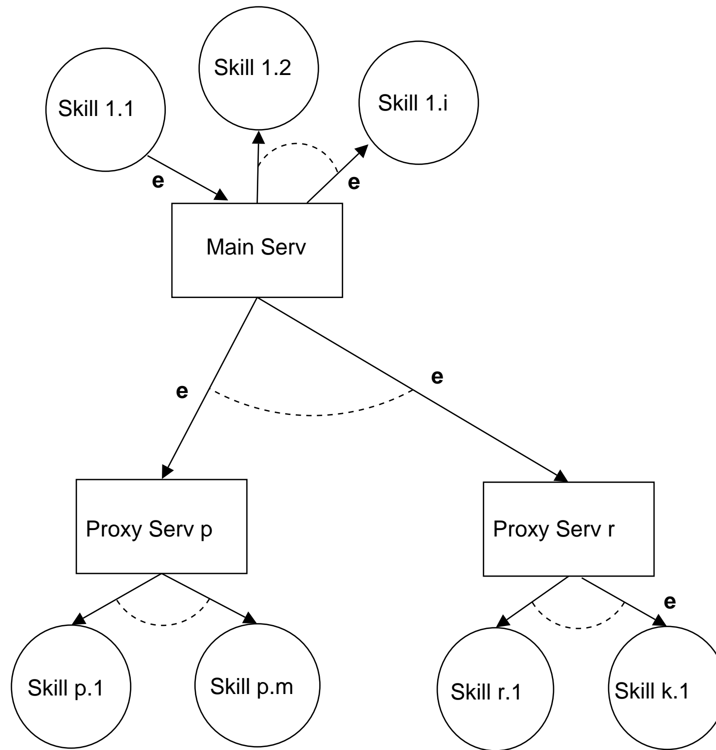

Figura 7.1: Propagación de un evento

ca que principalmente se captura el evento en sí, relegando la información sobre la emisión y captura del mismo. Una habilidad que produce eventos no conoce de antemano si existen o no otras habilidades que están dispuestas a recibir el evento emitido. Esto permite tener habilidades dedicadas a una tarea especifica, ocultando los mecanismos de comunicación como lo es el direccionamiento de los subscriptores, la transmisión del evento a sus posibles destinatarios y la confirmación de la aceptación del evento. Por ejemplo, una habilidad diseñada para detectar obstáculos al encontrar uno de éstos emite el evento de *OBSTACULO ON*, que puede tener parámetros donde se especifique la naturaleza y ubicación del obstáculo. Lo que no sabe la habilidad es si el evento emitido fue capturado y procesado por la o las habilidades encargadas de capturar dicho evento. La habilidad al emitir el evento, lo entrega al gestor de eventos y este se encarga de llevar el evento a los subscriptores, que pueden encontrarse en diferentes ordenadores.

El diseño del sistema gestor de eventos se ha realizado por medio de la implementación del patrón de diseño Publisher/Subscriber descrito en [99].

Para permitir la distribución de eventos entre habilidades activas en diferentes ordenadores, se creó un servidor principal de eventos y la posibilidad de instanciar servidores secundarios. El funcionamiento de los servidores del sistema de gestión de eventos puede verse el figura 7.1. Un evento es producido por una habilidad y entregado al gestor de eventos por medio del método *emit*. El

gestor local de eventos lo coloca en los buzones de cada una de las habilidades subscritas a dicho evento. El componente distribuido del gestor de eventos en el servidor lo propaga hacia los servidores secundarios (proxies), y estos a su vez los colocan en lo buzones de las habilidades subscritas a dicho eventos.

7.1.1. Emisión y subscripción de eventos

Si una habilidad requiere el uso de eventos, ésta debe crear una instancia, u obtener una referencia, de un objeto de la clase *CEventManager* en su espacio memoria local. Los principales métodos de dicha clase son *subscribe* y *emit*. El primero crea una subscripción al evento. La segunda emite un evento.

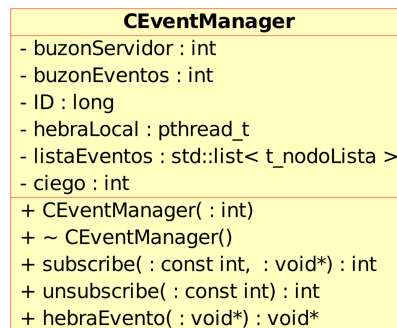

Figura 7.2: Diagrama de la clase CEventManager

En la figura 7.2 se muestra el diagrama de la clase CEventManager. Un objeto activo (en ejecución) de la clase CEventManager posee dos hebras en ejecución.

La primera hebra, es la hebra esclava del servidor local. Esta hebra se crea de manera implícita en el contenedor del objeto. En el momento de la instanciación se conecta al sistema de mensajes del núcleo IPC System V, ver sub-sección 5.1.3, por medio de una clave de acceso *bien conocida*. Esta conexión es registrada en el servidor y por medio de ésta recibirá los eventos que lleguen al sistema. Esta hebra posee un conjunto de funciones de servicio asociadas a los eventos por medio del método *CEventManager::subscribe()*.

Para realizar la subscripción se debe anexar los siguientes parámetros:

Evento. Identificador del evento al que se debe reaccionar.

Función de servicio. Dirección de una Función de la forma *void (*ptr_Funcion)(void *, int);*. El primer parámetro formal es una dirección de una región de memoria para cualquier tipo de datos y será la función de servicio la encargada de reinterpretar los datos almacenados en dicha región.

El segundo parámetro es proporcionado por el gestor de eventos y a través de este parámetro formal se transfiere a la función información opcional del evento recibido. Esta información opcional es suministrada por la entidad

que emite el evento. La función de servicio es ejecutada en el momento de recibir el evento.

Para garantizar la captura de los eventos la duración de la ejecución de la función de servicio debe ser menor que el tiempo de duración del lazo de control de la habilidad a servir.

Parámetro de la función de servicio. Este parámetro de la forma *void ** permite pasar la dirección de memoria con los parámetros actuales de la función de servicio.

La segunda hebra es la hebra principal del contenedor del evento, la función principal es realizar las funciones subscripción y la emisión de eventos. El método *CEventManager::emit()* es usado para la emisión del evento y es posible enviar adicionalmente un parámetro al evento. Si el parámetro existe debe ser de tipo entero. El mecanismo de emisión de eventos se basa también en el uso de una cola de mensajes IPC System V. En el instante de emitir un evento se encapsula en un mensaje del protocolo del gestor de eventos y es colocado en el buzón de mensajes del servidor, donde el identificador de la prioridad de la cola de mensajes es usado para indicar el tipo de operación. Los tipos establecidos en el protocolo son: EVENTO, SUBSCRIBIR, DESUBSCRIBIR e INVALIDO.

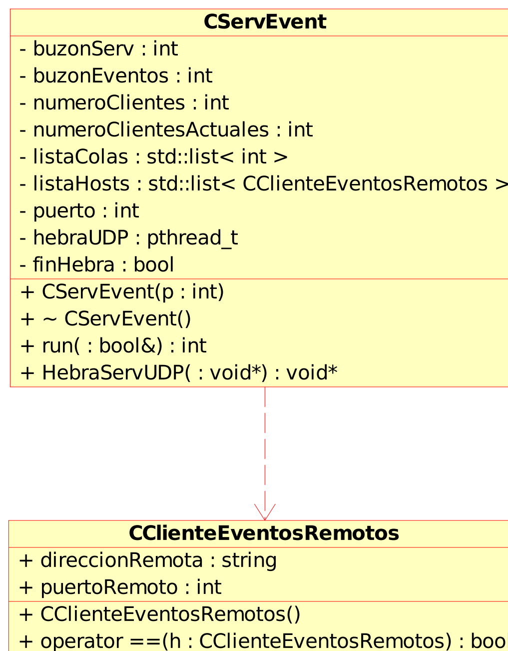

Figura 7.3: Diagrama de la clase CServEvent

7.1.2. Servidores para la gestión de eventos

La versión actual del gestor de eventos utiliza componentes encargados de gestionar la propagación de eventos. El funcionamiento se hace por medio del

servidor principal del manejo de eventos y los servidores auxiliares de eventos. Ambos servidores cursan eventos desde y hacia procesos locales. El servidor principal instancia un objeto de la clase *CServEvent*, cuyo diagrama puede verse en la figura 7.3. Este servidor además de propagar y recibir eventos de procesos locales, tiene las funcionalidades necesarias para propagar y recibir eventos entre diferentes ordenadores.

El servidor principal posee dos mecanismos de comunicación, uno para la comunicación local y un otro para la comunicación entre procesos residentes en ordenadores interconectados. El mecanismo local usa colas de mensajes IPC System V. Por otro lado, para la comunicación entre ordenadores usa un *Socket BSD* enlazado a una dirección *bien conocida* del dominio *AF_INET*, usando en la capa de transporte el protocolo UDP.

Para la comunicación local el servidor asume el papel de emisor del paradigma Publisher/Subscriber. Todos los clientes se subscriben al crearse. El servidor posee dos directorios de subscriptores, éstos son mantenidos por sendas listas de la biblioteca Standard Templates Library. En el momento de arribar un evento se toman las direcciones de los subscriptores y se les envía una copia del evento a cada uno de los subscriptores.

Los servidores auxiliares se encargan de permitir la distribución de eventos entre procesos en ejecución en ordenadores interconectados. Estos servidores crean instancias de un objeto de la clase *CServEventRmt*, cuyo diagrama es mostrado en la figura 7.4. Las funcionalidades del componente remoto son dos: gestionar los servicios para los subscriptores locales y usar los servicios del servidor principal para informar y recibir eventos. La gestión de subscriptores es similar a la del servidor principal. Los servidores auxiliares llevan una lista con los identificadores de cada uno sus subscriptores, que se hace sin notificar al servidor principal. En cambio, los eventos recibidos de los clientes locales no son propagados, son reenviados directamente al servidor principal y este los propaga a sus subscriptores. En otras palabras, los servidores auxiliares sólo propagan a sus subscriptores los eventos que son enviados por parte del servidor principal.

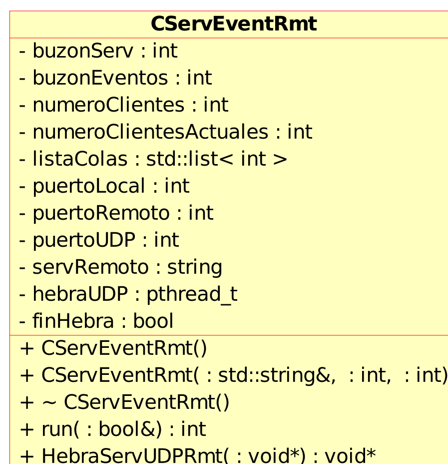

Figura 7.4: Diagrama de la clase CServEventRmt

Se puede resumir que para la gestión de eventos es necesario un servidor principal y cero o más servidores auxiliares. Esta organización permite crear jerarquías en la gestión de eventos.

7.1.3. Ejemplo de envío y recepción de eventos

A continuación se presenta segmentos de código en lenguaje C++ que usa los servicios del sistema de gestión de eventos. Para poder ejecutar se debe garantizar que el servidor de eventos principal, y los auxiliares de ser necesario, se encuentren activos.

```
#include<iostream>
#include<stdlib>
#include <cevent.h>

#define WANDERING 102
#define OBSTACLE_ON 103

int main(){
   int numero;
   CEventManager emisor;

   srand((unsigned)(time(NULL));
   do {
      r = rand () % 5;
      if ( r == 4 )
        emisor.emit(OBSTACLE_ON,0);
      else
        emisor.emit(WANDERING,0);
      sleep(1);
   }while(1);

   return(0);
}
```

Para poder usar el sistema de eventos se debe crear una instancia de la clase CEventManager, esto se hace en la línea:

```
   CEventManager emisor;
```

El constructor de la clase busca en un puerto bien conocido al servidor local del sistema de gestión de eventos. En este caso, no hace falta indicar donde se encuentra el servidor, el constructor se encarga de hacer la localización de forma transparente. El objetivo de esta primera aplicación es enviar un evento cada segundo. Para ejecutar esta operación se invoca el método CEventManager::emit(), en este caso hay dos posibles eventos a enviar OBSTACLE_ON y WANDERING. A pesar que es posible enviar un parámetro adicional al evento, en este ejemplo esa opción no es usada. Una de cada 5 veces se enviará el evento OBSTACLE_ON, el resto de las veces se enviará el evento WANDERING.

El siguiente programa muestra un ejemplo de la subscripción a un evento y muestra la forma de definir la función de servicio en el caso de la captura de un evento al cual se está subscrito.

```
#include <iostream>
#include <cevent.h>

#define WANDERING 102
#define OBSTACLE_ON 103

void funcion1(void* aux, int n){
    // ACTIVAR DEAMBULAR
    return;
}

void funcion2(void* aux, int n){
    // DESACTIVAR EXPLORACION
    // ACTIVAR  EVITAR COLISIÓN
    return;
}

int main(){
    int e;
    CEventManager sensor;

    sensor.subscribe(WANDERING, funcion1, (void *) 0);
    sensor.subscribe(OBSTACLE_ON, funcion2, (void *) 0);

    std::cout << "Un numero para finalizar: ";
    std::cin >> e;

    return(0);
}
```

Al crear una instancia de la clase CEventManager el constructor localiza el servidor de eventos mas próximo. En este momento, ese objeto se puede usar para emitir un evento o para subscribirse a uno o más eventos, adicionalmente crea un proceso liviano (thread) para que supervise la llegada o no de un evento en el caso de realizar una subscripción. Las instrucciones siguientes realizan la subscripción:

```
    sensor.subscribe(WANDERING, funcion1, (void *) 0);
    sensor.subscribe(OBSTACLE_ON, funcion2, (void *) 0);
```

El primer parámetro es el evento al cual se está realizando la subscripción. El segundo es un apuntador a la función de servicio, que debe haber sido declarada anteriormente. El tercer parámetro es la dirección de una variable, estructura de datos u objeto activo, que permite cambiar modificar el valor de un dato o ejecutar un método de un objeto, en el caso de que se haya pasado la dirección de un objeto como parámetro.

Después de la subscripción la aplicación es bloqueada a la espera de un valor pasado desde la consola. Es importante destacar, que a pesar que el hilo principal de la aplicación está bloqueado por la consola, el proceso liviano creado por el gestor de eventos sigue bloqueado, pero por los eventos a los que se está subscrito. En caso de la ocurrencia de cualquiera de los dos eventos el proceso liviano ejecutará la función predefinida en el momento de la subscripción.

Cuando una sistema evoluciona según la ocurrencia de eventos el proceso crucial es la captura del eventos. En este caso no se ha creado un proceso liviano por cada evento al cual se ha subscrito. Por lo tanto la función de servicio debe ejecutarse en el menor tiempo posible, y nunca permitir que existan instrucciones que pueden bloquear al proceso liviano, ya que si esto ocurre la aplicación puede perder la captura de eventos mientras el proceso liviano se encuentre bloqueado.

7.2. Sistema de memoria a corto plazo

El compartir datos entre habilidades es una necesidad fundamental en el paradigma AD. Desde la información procesada por los sensores hasta la información utilizada en la capa deliberativa debe estar disponible para actuales o futuras habilidades. En algunos casos, como en el de los sensores físicos, existen habilidades específicas que requieren procesar los datos obtenidos y a partir de éstos extraer información que luego debe ser distribuida entre las habilidades.

En la realización de la arquitectura AD los flujos de datos obtenidos de sensores físicos se extraen de manera síncrona con periodos de transmisión que van desde los 50 mseg en los sonars hasta los 120 mseg en el lector láser. Pero la información obtenida al procesar estos datos se produce de forma asincrónica. Por ejemplo, si se trata de descubrir una puerta por medio del láser, se procede a tomar medidas del láser de forma síncrona y a procesarlas. En el momento en que se tenga la certeza de haber detectado una puerta se coloca las coordenadas de las mismas a disposición de otras habilidades y se emite un evento. La información a distribuir es usualmente un resumen de datos obtenidos de los sensores, y en algunos casos puede ser transmitida por medio del sistema de gestión de eventos como parámetros del mismo.

No sólo se transmiten datos desde los sensores físicos hasta habilidades en las capas reactivas, sino que también se envían consignas de control desde la capa deliberativa a la capa reactiva y entre habilidades de una misma capa. Eso implica que la naturaleza de los datos es variable, y depende fundamentalmente de la justificación de la habilidad en sí.

Para satisfacer los requerimientos de distribución de información y datos se creó un sistema de memoria compartida distribuida basada en las siguientes premisas:

- El sistema debe ser capaz de distribuir diferentes tipos de datos.
- Adicionalmente al dato, se debe almacenar la fecha de la captura del mismo.

- Los datos deben estar disponibles a todas las habilidades de la arquitectura de control.

El sistema construido tiene dos componentes: un componente simple basado únicamente en herramientas IPC System V para la distribución local y un componente distribuido gestionado a través de servidores dobles construidos con la herramienta ONC RPC (ver sub-sección 5.2.2). El sistema construido se caracteriza por:

- Los datos son registrados y catalogados como tipos semánticos, aunque sintácticamente tengan la misma estructura, y se identifican por medio de un código único.
- Los datos son almacenados como flujos de bytes en una jerarquía de servidores similares a los usados en el sistema de gestión de eventos.
- Se almacenan el valor actual y el valor anterior.
- Las lecturas no son destructivas y las diferentes versiones de la información sobrescribe la versión anterior.
- Si una habilidad requiere el uso de la memoria a corto plazo debe instanciar en su espacio de memoria local un objeto de la clase *CmemCortoPlazo*, cuyos métodos principales son *RegistrarDato*, *ColocarDato* y *ObtenerActual*.

7.2.1. Clases requeridas para usar la memoria a corto plazo

La unidad lógica de almacenamiento de datos es implementada por medio de la clase CItem, ver figura 7.5. Esta clase posee cuatro atributos que se describen a continuación:

Código. identificador único del dato a compartir. Dos datos pueden tener el mismo contenido pero son iguales, si y solo si, sus códigos de identificación son iguales.

Fecha. Información opcional que asocia una fecha al dato tomado. Este dato se representa en milisegundos, tomando como el cero estándar en el Sistema Operativo.

Buffer. Dirección de área de memoria donde se encuentra la información del CItem. Ésta es representada como un flujo de Bytes no estructurados.

Tamaño. Tamaño en Bytes de los datos señalados por *Buf*.

Los objetos de la clase CItem sólo se encargan de mantener la información a compartir. La naturaleza de los datos y la forma como están estructurados es ajena a los objetos de CItem. La ventaja fundamental detrás es que se puede

representar en teoría cualquier dato que pueda ser serializable en un flujo de Bytes. La limitación del uso de CItem es el tamaño de los datos a contener, pero esto es una limitación en el contenedor de objetos CItem.

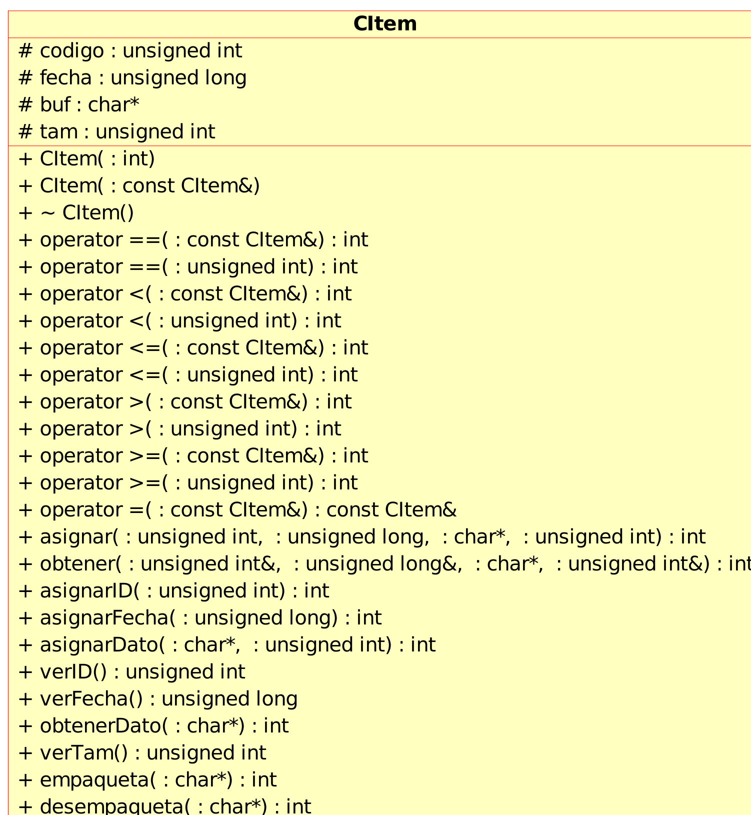

Figura 7.5: Diagrama de la clase CItem

Para dar soporte al transporte y manipulación de un objeto de la clase CItem, se han definido un conjunto de métodos y operaciones útiles, como lo son leyes de equivalencia y leyes de orden. También se han definido métodos para la serialización y desserialización de la información a almacenar/extraer en un objeto CItem.

Para poder usar el sistema de memoria a corto plazo se debe crear una instancia o poseer una referencia a un objeto de la clase *CmemCortoPlazo*, ver figura 7.6. Los objetos de esta clase ofrecen una interfaz al servicio. El único atributo de la clase es *nombreServidor*, que es una estructura que contiene la dirección *bien conocida* de un servidor local del gestor de memoria compartida.

Los métodos de la clase *CmemCortoPlazo* permiten el acceso a todos los servicios del sistema. Estos son:

RegistrarDato. Este método permite registrar un dato en el servidor. El registro se hace de forma indirecta ya que para tal fin se usa una referencia a un objeto de la clase CItem. Ésta debe tener un código y un tamaño no

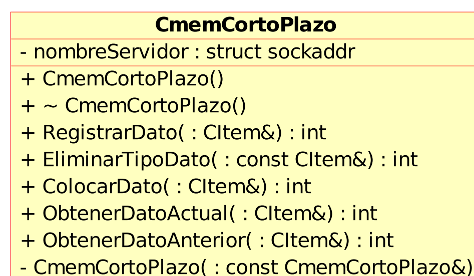

Figura 7.6: Diagrama de la clase CmemCortoPlazo

nulo.

EliminarTipoDato. Permite eliminar un tipo de dato del sistema de memoria a corto plazo. Se usa una referencia a un objeto CItem cuyo código sea no nulo.

ColocarDatoActual. Permite actualizar un dato en el sistema de memoria a corto plazo. El dato actual antes de la actualización no se pierde, éste pasa a ser la versión anterior de ese tipo de dato.

ObtenerActual. Obtiene una copia de un dato previamente en el sistema de memoria a corto plazo.

ObtenerAnterior. Obtiene la versión anterior de un dato previamente registrado.

Todos estos métodos usan una referencia a un objeto CItem, ya que el sistema sólo reconoce este tipo de datos. Es decir, en vez de construir un sistema que almacene enteros, puntos flotantes, cadenas de caracteres y otros tipos predefinidos de datos, en este sistema sólo se almacenan objetos de la clase CItem, y éstos a su vez contienen cualquier tipo de datos que pueda ser serializable en una secuencia de bytes.

7.2.2. Servidores requeridos

La interfaz a los servicios del sistema de memoria a corto plazo se hace por medio de los objetos de la clase *CmemCortoPlazo*. El canal de comunicación se establece por medio de un socket enlazado a una dirección de la familia *AF_UNIX*, esto significa que la dirección es sobre dominio del sistema de ficheros del ordenador donde se ejecuta el servidor. El canal de comunicación es orientado a conexión, cuyo espacio de memoria es tomado del núcleo del sistema operativo.

Un servidor puede tener tres modos de operación:

Independiente. Con esta configuración los servicios prestados se reducen a los clientes en el ordenador local.

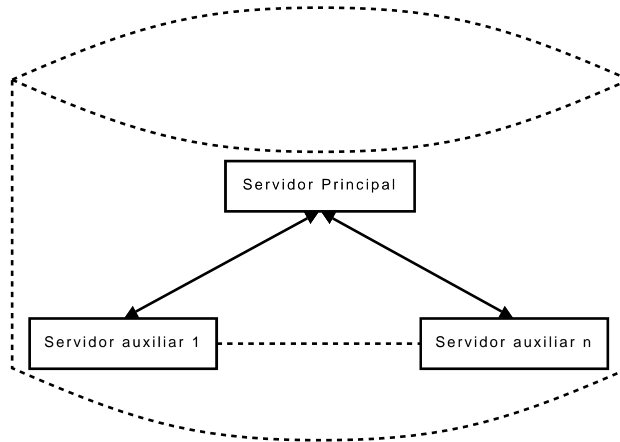

Figura 7.7: Servidores de memoria a corto plazo

Servidor principal. En este modo se activan los servicios de distribución de datos a servidores auxiliares y a los clientes locales.

Servidor auxiliar. Activa el servicio a clientes locales compartiendo datos obtenidos de un servidor principal.

Para activar el servicio es necesario poner en funcionamiento un servidor en modo independiente o en modo servidor principal y cero o más servidores en modo auxiliar. Un servidor independientemente de su modo de funcionamiento presta servicios a los procesos en el ordenador donde se ejecuta.

Los servicios locales se inician con la creación de un punto de acceso al servicio. Éste tiene una dirección sobre el sistema de ficheros local, no se deben usar direcciones sobre un sistema de archivos en red (NFS).

Una vez creada la dirección, la atención de las peticiones se hace siguiendo el modelo cliente-servidor: Se bloquea la hebra de servicio a la espera de una solicitud. Al llegar la solicitud se crea una nueva hebra para que la atienda y la hebra principal regresa inmediatamente a esperar nuevas solicitudes, ver figura 7.8.

Los datos son mantenidos en el servidor usando un objeto de la clase map de la biblioteca STL. La clase map implementa el tipo de dato abstracto *diccionario*, donde la clave de búsqueda es el atributo código del objeto CItem. Todos estos datos son almacenados en memoria dinámica y la persistencia es garantizada mientras el servidor se encuentre en ejecución.

En el modo de funcionamiento de servidor principal, se activa una hebra de escucha de peticiones. Ésta está asociada a una dirección del dominio de direcciones *AF_INET*, cuya finalidad es atender exclusivamente las peticiones de los servidores auxiliares. Las servicios solicitados por los servidores son: registro de tipos de datos y actualización de datos.

En el caso en que un servidor auxiliar recibe una solicitud de registro, pri-

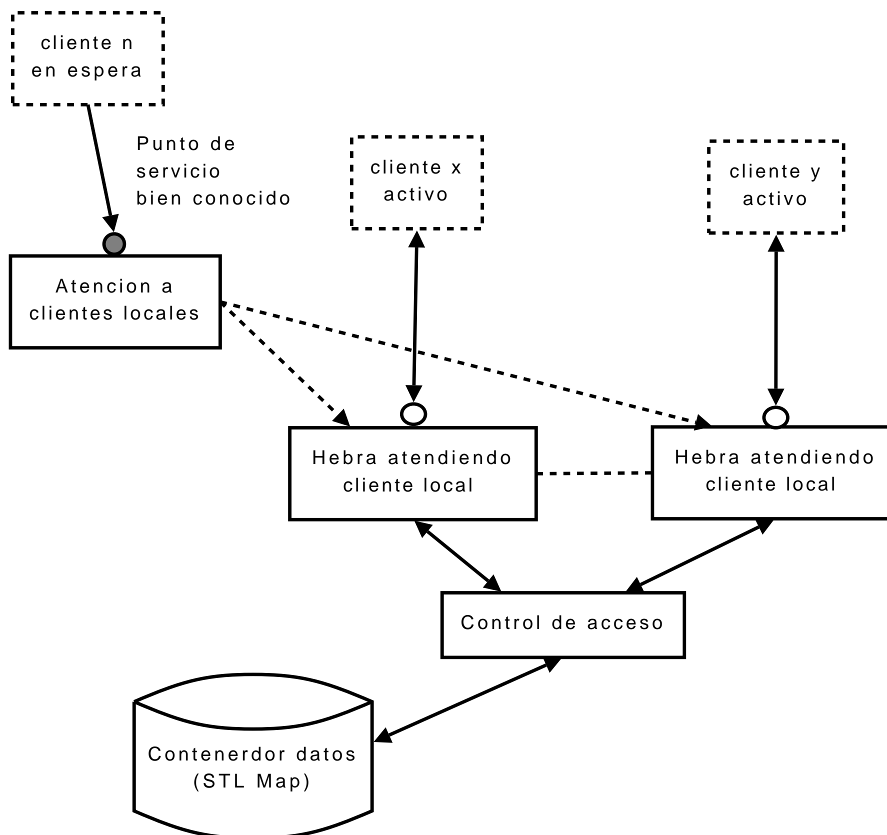

Figura 7.8: Arquitectura del funcionamiento local

mero se verifica si existe localmente. Una vez registrado localmente se envía una solicitud al servidor principal, en ésta se indica el código a registrar. Si no existe, éste es registrado en el servidor principal, y se pide que se registre el nuevo tipo en el resto de servidores auxiliares. En este caso la operación de registro se ha propagado por el resto de servidores.

Igual pasa en el caso de actualización de los datos, primero se actualiza localmente, luego en el servidor principal, y éste a su vez en los servidores auxiliares.

Cuando se solicita el valor del dato actual o anterior, sólo se busca en el valor almacenado en el diccionario de dato actual.

Los servidores auxiliares al inicio de la ejecución buscan al servidor principal y se registran ante estos. En el momento de registro toman todos los datos del servidor principal e inicializa con éstos su diccionario local. Una vez que ha cargado todos los datos activa una hebra de escucha de actualización enlazada al servidor principal. Y por último activa una hebra para atender a los clientes locales.

Fundamentalmente se ha diseñado y construido un sistema que permite com-

partir datos serializables en secuencias de Bytes. Estos datos son compartidos directamente entre procesos locales y a través de los servidores principales y auxiliares con otros procesos en otros ordenadores interconectados entre sí.

7.2.3. Ejemplo de uso de la memoria a corto plazo

Como se ha sido descrito anteriormente, el sistema de memoria a corto plazo es capaz de almacenar y compartir cualquier dato que pueda ser serializado, es decir que pueda ser representado como un flujo no estructurado de bytes. Debe quedar claro que el sistema de memoria a corto plazo no pretende sustituir al almacenamiento secundario, simplemente pretende registrar, almacenar y compartir pequeñas porciones de memoria, de forma eficiente y transparente.

El mover datos en el sistema de memoria a corto plazo es y será mas rápido en comparación con un disco duro, ya que en éste ultimo al almacenar y recuperar información requiere girar un plato magnético, y posicionar un cabezal para la lectura y escritura de los datos, en cambio en el sistema construido se mueve información entre la memoria principal[2].

El primer paso que se debe hacer es declarar los datos a almacenar, definir un código de identificación, así como el tamaño en bytes de los datos a manipular. Esto se puede hacer en un archivo de declaraciones del lenguaje C++, comúnmente llamado archivo de cabecera, como el que se muestra a continuación:

```
/**
 @file prueba1MCPdef.h
 @brief Archivo de descripcion de Tipos para la prueba \
        1 del manejador de memoria de corto plazo

 @code
*/
#ifndef PRUEBA1MCPDEFH
#define PRUEBA1MCPDEFH

// Tipos:
typedef struct sPuntoEntero{
  int x;
  int y;
} tPuntoEntero;

typedef struct sPuntoFloat{
  float x;
  float y;
  float z;
} tPuntoFloat;
```

[2]Indudablemente se está obviando, que por la gestión de memoria del sistema operativo, es posible que algunas páginas del proceso que controla la memoria a corto plazo puedan encontrarse en el área de intercambio, esto escapa al diseño del sistema

```
//Codigo Datos:
#define DATOTIPO1 102
#define DATOTIPO2 104


//Definiciones
#define TAMDATOTIPO1 sizeof(tPuntoEntero)
#define TAMDATOTIPO2 sizeof(tPuntoFloat)

//Eventos:
#define NVODATOTIPO1 1002
#define NVODATOTIPO2 1004

#endif
/**
 @endcode
*/
```

En este archivo se declaran dos tipos de estructuras. La primera es llamada tPuntoEntero, y contiene dos campos de tipo entero. El segundo es llamado tPuntoFloat , y contiene tres campos en punto flotante simple. Además se declaran dos códigos para cada tipo, en este caso por medio de dos macros, DATOTIPO1 es equivalente al código 102 y DATOTIPO2 es equivalente al código 104. Por último, y también por medio de macros, se asigna el tamaño del primer tipo a la macro TAMDATOTIPO1 y el tamaño del segundo tipo a la macro TAMDATOTIPO2.

Se debe tener presente que el código de los datos debe ser único en toda la aplicación y debe ser conocido por todas los componentes que requieran su uso. En este ejemplo se definen también dos eventos, para participar la colocación de datos en el sistema de memoria a corto plazo.

A continuación se muestra un productor de datos, éste utiliza un evento para avisar que un nuevo dato ha sido colocado.

```
/**
 @file productor1MCP.cpp

 @brief Este programa simula ser un procesador que \
    lee datos de un sensor y los coloca en la memoria \
    temporal, al colocar el dato con exito genera un \
    evento de nuevo dato.

 @author Rafael Rivas Estrada
  rafael.rivas@uc3m.es rafael@ula.ve

 @code

*/

#include <memCortoPlazo.h>
```

```
#include <cevent.h>
#include <prueba1MCPdef.h>
#include <stdlib.h>


int main() {

  CEventManager emisor;
  tPuntoEntero p;
  CmemCortoPlazo memCortoPlazo;
  CItem item(TAMDATOTIPO1);
  unsigned int n;
  unsigned int espera;

  srand((unsigned)time( NULL ));

  item.asignarID(DATOTIPO1);
  item.asignarFecha(0);

  if ( memCortoPlazo.RegistrarDato(item) ){
    std::cout << "Error al registrar el Tipo 1\n";
    return(1);
  }

  std::cout << "Cantidad de datos a generar: ";
  std::cin >> n;

  while ( n--) {
    p.x = (int)(((float)rand()/(float)RAND_MAX)*100 + 10);
    p.y = (int)(((float)rand()/(float)RAND_MAX)*500 + 1000);
    item.asignarDato((char*)&p, TAMDATOTIPO1);
    if (memCortoPlazo.ColocarDato(item)){
      std::cout << "Error al colocar datos\n";
    } else {
      emisor.emit(NVODATOTIPO1);
      std::cout << "Colocado " << p.x << ", " << p.y <<"\n";
    }
    espera = (unsigned int)(((float)rand()/(float)RAND_MAX)*10 + 5);
    std::cout << "Proximo dato en " << espera << " seg\n";
    sleep(espera);
  }

  memCortoPlazo.EliminarTipoDato(item);
  return(0);
}
/**
 @endcode
*/
```

Los archivos a incluir son memCortoPlazo.h y prueba1MCPdef.h. En el primero se encuentra las declaraciones necesarias para usar el sistemas de memoria a corto plazo y en el segundo se describen los datos a almacenar en el sistema. Al crear una instancia de un objeto de la clase CmemCortoPlazo el constructor de dicha clase se encarga de ubicar el servidor del sistema de memoria compartida y establece comunicación con el mismo.

El método CmemCortoPlazo::RegistrarDato() requiere un objeto de la clase CItem, antes de hacer el registro éste debe tener el código del nuevo dato a registrar, así como el tamaño que debe ser reservado. El registro se puede observar en las siguientes líneas:

```
if ( memCortoPlazo.RegistrarDato(item) ){
  std::cout << "Error al registrar el Tipo 1\n";
  return(1);
}
```

Si el registro es exitoso, se especifica el número de datos a generar y se entra en un lazo de repetición donde se generan números aleatorios en tiempos aleatorios. Un vez que se tiene un nuevo par de números, estos se asigna al objeto de la clase CItem, y se procede a colocar en el sistema a corto plazo, si esta operación es exitosa se envía un evento para notificar que un nuevo valor esta disponible. La instrucciones involucradas en esta operación se muestran a continuación:

```
if (memCortoPlazo.ColocarDato(item)){
  std::cout << "Error al colocar datos\n";
} else {
  emisor.emit(NVODATOTIPO1);
  std::cout << "Colocado " << p.x << ", " << p.y <<"\n";
}
```

Es importante destacar que en ningún momento el productor del dato se vincula con el proceso que usará el dato, y en este caso se avisa por medio de un evento la colocación de un nuevo valor, y sólo si existe un proceso subscrito al evento será notificado.

A continuación se muestra un programa que captura datos del sistema de memoria compartida. La notificación de la colocación de un nuevo dato es anunciada por medio de un evento, para evitar estar buscando datos continuamente, el programa se suscribe a un evento y define las operaciones necesarias para la captura del evento.

```
/**
 @file cliente1MCP.cpp

 @brief Este programa simula ser un procesador que espera \
  por datos  del tipo 1, el permance a la espera de que por \
  teclado se le pida el dato, cuando ocurre un evento del \
  tipo DATO1_DISPONIBLE, el busca el nuevo valor en el \
  servidor de memoria de corto plazo, lo procesa, imprime \
  y regresa a esperar por consola.
```

```
 @author Rafael Rivas Estrada
 rafael.rivas@uc3m.es rafael@ula.ve

 @code

*/

#include <memCortoPlazo.h>
#include <cevent.h>
#include <prueba1MCPdef.h>

void atiendeNuevoDato(void*, int);

int main() {

  tPuntoEntero p;
  CEventManager sensor;
  int n = 100;

  sensor.subscribe(NVODATOTIPO1, atiendeNuevoDato, (void*)&p);

  p.x = p.y = 0;

  char resp;
  int t = 3;

  std::cout << "Se comienza a mostrar los datos cada "
            << t << " segundos\n";

  do {
   std::cout << "Dato: " << p.x << ", " << p.y << '\n';
   sleep(t);
  } while (--n);

  return(0);
}

void atiendeNuevoDato(void* q, int e){
  CmemCortoPlazo lector;
  tPuntoEntero* p;
  CItem nvoP;

  p = (tPuntoEntero*)q;

  nvoP.asignarID(DATOTIPO1);
  nvoP.asignarDato((char *)p, TAMDATOTIPO1);

  if (! lector.ObtenerDatoActual(nvoP) ){
    nvoP.obtenerDato((char*)p);
```

```
    std::cout << "\n——————> Dato Actualizado "
              << p->x << ", " << p->y << std::endl;
  } else {
    std::cout << "\n——————> Error al actualizar"
              << std::endl;
  }

  return;
}

/**
 @endcode
*/
```

En el código mostrado, la función a activar ante la llegada de un evento es

```
void atiendeNuevoDato(void*, int);
```

En el momento de las subscripción, se indica que a dicha función se le pasará la dirección de una variable del tipo tPuntoEntero, con el fin de almacenar el dato copiado del sistema de memoria a corto plazo.

Una vez que el evento sea detectado, se activa la función de servicio, que se hace por medio de un proceso liviano (thread) gestionado por el gestor de eventos. En esta función se llama al método CmemCortoPlazo::ObtenerDatoActual() que es el encargado de copiar el dato más reciente del sistema de memoria a corto plazo en la referencia de un objeto de la clase CItem. Esta operación se muestra a continuación

```
if (! lector.ObtenerDatoActual(nvoP) ){
  nvoP.obtenerDato((char*)p);
  std::cout << "\n——————> Dato Actualizado " <<
  p->x << ", " << p->y << std::endl;
} else {
  std::cout << "\n——————> Error al actualizar" << std::endl;
}
```

Este ejemplo mostró como se coloca y como se obtiene una copia de un dato en el sistema de memoria a corto plazo. Es importante destacar que la operación de obtención del dato no es destructiva, es decir, que al obtener el dato, éste sigue en el sistema hasta que se actualice. La operación de obtener el dato pudo haber sido realizado en forma concurrente o paralela por otros procesos.

7.3. Observaciones finales del capítulo

En este capítulo se muestran dos importantes elementos en la arquitectura AD. Con estos dos elementos se permite compartir datos, así como el envío y recepción de eventos, entre aplicaciones en diferentes ordenadores interconectados. Todo eso se hace de forma transparente: no se observa ninguna primitiva

de conexión, solo existen invocación de un número reducido de métodos, con interfaces bien definidas y objetivos claramente definidos.

Uniendo estos dos sistemas junto a la implementación de los actuadores y sensores virtuales se logra establecer una base sólida para la realización de la arquitectura AD. Por ahora, solo falta describir el elemento básico de la arquitectura, este elemento es la habilidad, y su implementación es descrita en el siguiente capítulo.

8

Habilidades en la arquitectura AD

Para exponer la implementación de la arquitectura AD se ha mostrado, hasta ahora, la realización de los sensores y actuadores virtuales, el sistema gestor de eventos y el sistema de memoria a corto plazo. La justificación de esta forma de diseño incremental es debido a la gran cantidad de elementos involucrados en el diseño y construcción de un sistema de control para un robot. En este capítulo se aborda la implementación del concepto de habilidad en el contexto de la arquitectura AD y su características de funcionamiento. Además se presenta el secuenciador de habilidades, como un mecanismo de planificación de ejecución dinámica de habilidades. En el uso de las habilidades y su secuenciador se integran todos los desarrollos presentados. Por último se muestra como la arquitectura funciona como un conjunto de habilidades que cooperan entre sí, con el fin de alcanzar un objetivo específico.

8.1. Modelo de una habilidad

La habilidad (como se ha definido en 2.2) es el elemento básico del diseño de la arquitectura AD. Una habilidad debe ser capaz de activar y a su vez poder ser activada por otras habilidades (principio de subordinación). Una habilidad puede estar compuesta por una o más habilidades o ser simplemente una relación entre estas.

Hasta ahora se han definido algunas habilidades dentro de AD [100, 101], otras están en pleno desarrollo, y algunas están simplemente planificadas. Esto último conlleva a la necesidad de establecer un tipo abstracto de datos que sea capaz de adaptarse a las habilidades ya hechas, a las que están en curso, y a las que se harán en un futuro.

Este tipo de dato abstracto debe captar la esencia del concepto de habilidad, tal como se ha diseñado hasta ahora y sin inhibir ningún desarrollo futuro. El

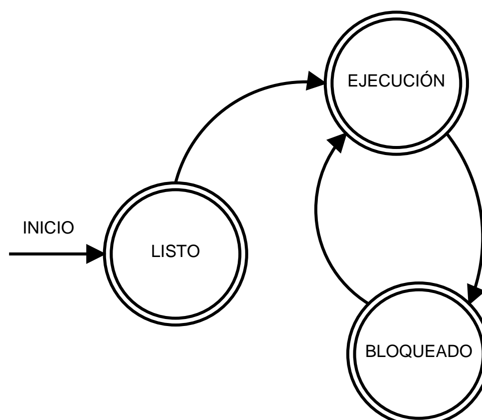

Figura 8.1: Estados de una Habilidad

primer paso consiste en definir los posibles **estados** en los que se puede encontrar una habilidad, ver figura 8.1. En el estado LISTO las operaciones de iniciación de la habilidad son llevadas a cabo y la habilidad se encuentra lista para pasar al estado de ejecución. En el estado EJECUCIÓN de una habilidad se activa la operación principal: también llamado lazo de control. Si el lazo de control de una habilidad debe ser detenido, se pasa al estado de BLOQUEADO. En este estado debe permanecer hasta que se ordene pasar al estado de EJECUCIÓN.

En el estado de EJECUCIÓN se debe activar la operación principal de la habilidad. La naturaleza de esta operación depende exclusivamente del objetivo de la habilidad, por lo tanto, no es posible definir una estructura de lo que debe hacer una habilidad. Entre las posibles operaciones que puede hacer una habilidad se encuentran: obtener un dato, activar o desactivar otra habilidad, tomar o colocar un dato en el sistema de memoria a corto plazo, reconocer ordenes de voz, acceder a una base de datos, sintetizar voz, analizar una imagen, deliberar, etcétera. Existe un gran número de operaciones que puede hacer una habilidad. Lo que si se puede inferir es la existencia de una tarea por hacer, y si ésta puede ser hecha una o más veces, o hacerla en forma continua.

Por lo anteriormente expuesto, la operación principal de una habilidad es desconocida hasta el momento del diseño mismo de la habilidad, por lo que en el modelo este atributo debe ser lo mas genérico posible, y es en el momento de la implementación donde se será definido completamente. A esta operación principal se llamará le **proceso** de la habilidad.

El proceso de la habilidad debe ejecutarse en un tiempo finito, y posee diferentes **modos de operación**: periódico, cíclico y activado una sola vez. En el modo de operación periódico, el proceso de la habilidad es ejecutado cada cierto tiempo predefinido. Indudablemente que en el diseño del proceso, se debe garantizar que el tiempo de ejecución del lazo de control sea menor el período de ejecución. De esta manera se evita el solapamiento en la ejecución de la operación.

En el modo de operación cíclico al finalizar una ejecución del lazo de control,

éste se activa nuevamente. Tanto en el modo de operación periódico y cíclico la ejecución se repite mientras la habilidad se encuentre en el estado de EJECUCIÓN. En el modo de operación activado una sola vez, el lazo de control se ejecuta y al finalizar la habilidad, pasa al estado BLOQUEADO.

Como se observa, una habilidad que ha pasado al estado de EJECUCIÓN no regresa al estado LISTO, ya que en ese estado se activan los procesos de inicialización de la habilidad.

Una vez que se han definidos los estados y los modos de operación de una habilidad, se pueden definir la operaciones de la misma. Entre las operaciones básicas se tienen las que permiten la observación del estado de la habilidad, así como algunas operaciones necesarias para su inicialización. Pero hay dos operaciones que son fundamentales si se desea obtener un modelo fiel al concepto de habilidad, estas operaciones son **activa** y **bloquea**. En la operación de activa el estado de la habilidad cambia de LISTO a EJECUCIÓN, o de BLOQUEADO a EJECUCIÓN, según sea el caso, y se inicia la ejecución del proceso de la habilidad. En el operación bloquea, se detiene la ejecución del proceso, y la habilidad cambia su estado de EJECUCIÓN a BLOQUEADO.

Al establecer las operaciones de una habilidad, y en unión con los atributos que definen el estado de la misma, es posible pasar al diseño e implementación de la habilidad. Continuando con la decisión en el diseño e implementación de la arquitectura AD se utiliza el lenguaje C++. Para poder dar una mayor flexibilidad con futuras habilidades aun no pensadas, y para poder representar el proceso de la habilidad como una entidad genérica, se presenta a continuación la clase base Habilidad. Ésta es una clase abstracta que no puede crear objetos directamente, ya que el método que implementa el proceso de la habilidad no es definido. Por lo tanto CHabilidad no es una clase simple, sino que sirve de clase generadora para las habilidades actuales y las que se harán en un futuro. Por la naturaleza del lenguaje usado, la clase acepta extensiones que permiten agregar nuevos atributos y nuevas operaciones. Sólo las clases derivadas de la clase habilidad que tengan definido el proceso de la habilidad están capacitadas para crear objetos de dicha clase.

8.2. Implementación de una Habilidad

En la arquitectura AD las habilidades son implementadas como clases derivadas de la clase abstracta habilidad. Como se describió en la sección anterior. La habilidad base debe ser abstracta, ya que el método proceso no se encuentra definido. Esta clase es una abstracción usada para describir los principales aspectos de una habilidad sin especificar los detalles de ninguna en particular.

Una instancia de una Habilidad se caracteriza por:

- Poseer tres estados: Lista, en ejecución y bloqueada.
- Tres modos de ejecución: continuo, periódico, por eventos.
- La dinámica puede ser definida:

- En el momento de la compilación(estática).
- Por medio de una documento XML interpretado por un secuenciador.

- La organización:
    - Cada habilidad en ejecución es al menos un proceso. La comunicación entre procesos es por medio de eventos, la memoria a corto plazo, y/o información para lograr un todo.
    - Una habilidad se representa por una o más tareas o por un agregado de habilidades.
    - Una mezcla de las dos anteriores.

- Todo el manejo de multiprogramación, comunicación entre procesos, composición, etc., debe ser transparente al usuario, o al menos simple de usar.

En la figura 8.2 se muestra el diagrama de la clase CHabilidad. Esta clase sólo sirve como clase base, ya que no es posible crear objetos de esta clase debido a que su definición es incompleta. Los atributos en la clase mantienen el estado de la habilidad, así como la hebra del proceso principal y los semáforos que controlan su correcta ejecución.

Los métodos y operaciones de la habilidad se pueden dividir en tres grupos de acuerdo a su funcionalidad: constructores, públicos para el control de la habilidad, y privados para la creación y el control interno de la habilidad.

Entre el grupo de los constructores permiten indicar el tiempo en milisegundos de espera entre la ejecución de un lazo de control. Es posible indicar el número de veces que se va a ejecutar repetidamente el lazo de control. También existe la posibilidad de tomar los datos de una habilidad creada con anterioridad o tomarlos a partir de una cadena de texto.

En el grupo privado se encuentran métodos que directamente crean y controlan la habilidad.

La habilidades pertenecientes al grupo público sirven de interfaz al control y supervisión de la habilidad en sí. Los métodos *CHabilidad::activa()* y *CHabilidad::bloquea()* activan y suspenden la ejecución de la hebra de control. También existen métodos para la modificación del tiempo de espera entre una ejecución y otra del lazo de control.

La clase CHabilidad se dice incompleta ya que el método *CHabilidad::proceso()* está declarado pero no definido. Esto permite saber que todas los objetos de la clase CHabilidad tienen un lazo de control propio y para invocar dicho lazo de control se hace por medio del método *CHabilidad::proceso()*, que no requiere de ningún parámetro y que no devuelve ningún valor. Por esa razón no se pueden crear objetos, pero se pueden crear nuevas clases de habilidades ya que el comportamiento común esta totalmente declarado y definido parcialmente en CHabilidad.

El componente creado posee los servicios de sincronización, creación de subprocesos y temporización. El usuario elige el modo de operación y la forma de

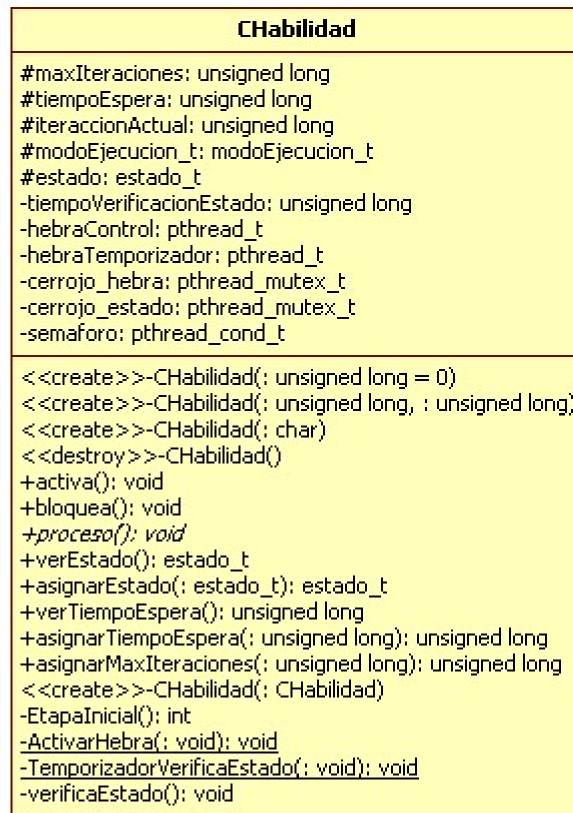

Figura 8.2: Diagrama de la clase abstracta CHabilidad

organización de las habilidades. Se debe definir el control, y para esto puede utilizar otras habilidades creadas y probadas separadamente.

A continuación se muestra el código que declara la clase abstracta Habilidad:

```
/**
 @file habilidad.h
 @brief Archivo en el cual se declara el espacio de nombre AD eventos basicos, tipo
*/
#ifndef CHABILIDAD
#define CHABILIDAD

#include <pthread.h>
#include <unistd.h>
#include <assert.h>
#include <signal.h>
#include <sys/types.h>

/**
```

```
 @namespace habAD
 @brief Espacio de nombres para agrupar constantes y estructuras \
 usados en el Sistema de Detector de Eventos

 @version 1.0
*/
namespace habAD{

  enum estado_i {
    LISTO = 0, // LISTO
    EJECUCION, // EJECUCION
    BLOQUEADO  // SUSPENDIDO
  };

  typedef enum estado_i estado_t;

  enum modoEjecucion_i{ //tipo de ejecucion
    CICLICA = 0,
    PERIODICA,
    EVENTOS,
    ARCHIVO_AUTOMATA, // Para aplicaciones futuras
    OBJETO_AUTOMATA   // Pasra aplicaciones fururas
  };

  typedef enum modoEjecucion_i modoEjecucion_t;

/**
  @class CHabilidad habilidad.h
  @brief CHabilidad es una clase abstracta que sirve \
         de patrón para definir nuevas habilidades dentro\
         de la arquitectura AD.

  @author Rafael Rivas Estrada
  rafael.rivas@uc3m.es rafael@ula.ve

   v 1.0
   @version 1.2 Noviembre, 2005
*/

 class CHabilidad {

  public:
    CHabilidad(unsigned long = 0);
    CHabilidad(unsigned long, unsigned long);
    CHabilidad(const char*);
    ~CHabilidad();

   void activa();
   void bloquea();
```

```
//
// METODO ABSTRACTO QUE CARACTERIZA UNA HABILIDAD EN PARTICULAR
//
    virtual void proceso() = 0;

    habAD::estado_t verEstado();
    habAD::estado_t asignarEstado(habAD::estado_t);
    unsigned long   verTiempoEspera() const;
    unsigned long   asignarTiempoEspera(unsigned long);
    unsigned long   asignarMaxIteraciones(unsigned long);

  protected:
    unsigned long maxIteraciones; // 0 Se ejecuta por tiempo indefinido
    unsigned long tiempoEspera;
    unsigned long iteraccionActual;
    habAD::modoEjecucion_t modoEjecucion_t;
    habAD::estado_t estado;

  private:
    CHabilidad(const CHabilidad&);
    int EtapaInicial();
    static void* ActivarHebra(void *);
    static void* TemporizadorVerificaEstado(void *);
    void verificaEstado();

    unsigned long tiempoVerificacionEstado;
    pthread_t hebraControl;
    pthread_t hebraTemporizador;
    pthread_mutex_t cerrojo_hebra;
    pthread_mutex_t cerrojo_estado;
    pthread_cond_t semaforo;
 };
}
#endif
```

El archivo de cabecera mostrado presenta inicialmente la declaración del espacio de nombres a ser utilizado en la implementación. La clase CHabilidad es parte de ese espacio de nombres.

La clase CHabilidad posee tres zonas de visibilidad. La primera es la parte pública, en ésta se declaran los métodos visibles a cualquier usuario. Entre éstos se encuentran los constructores, observadores, modificadores de estado y el proceso de la habilidad. La parte protegida será visible a aquellas clases que se deriven de la clase CHabilidad. Se finaliza la declaración con una parte no visible o privada, donde se pueden observar algunos de los mecanismos usados en la realización como lo son: procesos livianos Posix (pthreads) y semáforos de procesos livianos (pthreads_mutex).

Esta última parte de la declaración oculta al usuario, encargado de construir nuevas habilidades, todos los detalles del manejo de la multiprogramación,

comunicación y sincronización de procesos livianos, requeridos en la implementación de la habilidad.

8.3. Ejecución de secuencias de habilidades

El control de la ejecución de secuencias de habilidades es uno de los objetivos secundarios de este trabajo. La visión que impulsa su desarrollo es la necesidad de controlar la activar y desactivar secuencias de habilidades, es decir, la forma como una habilidad inicia o detiene su ejecución de una manera efectiva, desacoplada, transparente y dinámica.

Para ser efectiva, la habilidad debe haber sido diseñada y construida para permitir su activación y desactivación, siguiendo el patrón de diseño propuesto en este documento. Esto último conlleva a la subordinación de la habilidad a la arquitectura de control AD. Se logra haciendo que la habilidad responda correctamente a los eventos de ACTIVAR y DESACTIVAR, sin exceder en tiempo la duración de una iteración del lazo de control de la habilidad.

La secuenciación es desacoplada siempre y cuando el secuenciador mantenga una interfaz mínima con las habilidades que controla. Esto permite intercambiar la fuente de generación de consignas con el mínimo impacto sobre la aplicación. En uno de los prototipos se habilitó la capacidad de instanciar habilidades desde el secuenciador, pero esto creaba dependencias entre el secuenciador y las habilidades instanciadas por éste. En la versión más reciente, presentada en este documento, se asume que las habilidades se encuentran instanciadas en un contenedor de habilidades. El contenedor se debe encontrar en ejecución en un ordenador perteneciente al sistema de cómputo del robot.

El requisito de transparencia se ha seguido en todo el desarrollo de este trabajo, y al igual que en los casos anteriores, se refiere al ocultamiento de los mecanismos de intercomunicación de procesos, apertura de canales de comunicación, localización de servidores y serialización de la información.

El garantizar que el secuenciador opere de forma dinámica es satisfecha a partir de la versión 2.0. Esta característica implica la capacidad de realizar cambios en estructura de la secuencia de control en tiempo de ejecución, y ha requerido un esfuerzo adicional para proveer dicha capacidad. La descripción es realizada en la subsección referente a la versión 2.0 en el presente capítulo.

8.3.1. Requisitos de diseño

La idea fundamental es la de contar con un mecanismo que permita al usuario final indicarle al robot un modelo de comportamiento. Esto implica la definición de un esquema de cooperación, entre habilidades dentro del sistema de control del robot en un momento dado.

En la figura 8.3 se muestra un caso general de uso del secuenciador. En éste, el usuario provee la forma como deben ejecutarse las habilidades involucradas

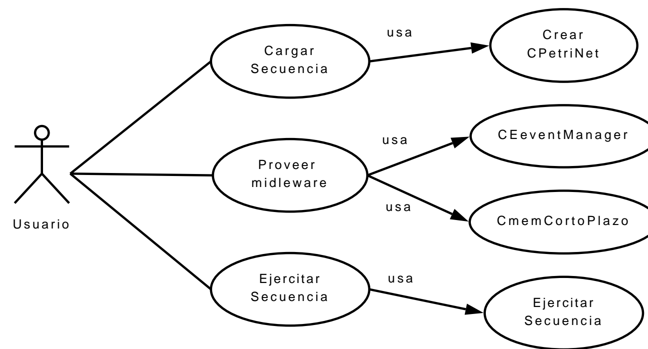

Figura 8.3: Caso de uso para el secuenciador

en la secuencia. Originalmente la forma como es suministrada la secuencia no está definida, ésta puede estar descrita en una estructura de datos externa, una estructura de datos interna, un archivo, o un paquete de datos de un protocolo TCP/UDP.

Una vez que es suministrado el modelo, se crea una imagen del modelo en el espacio de datos de la aplicación. El mecanismo de iteración en el secuenciador es por medio del bus de integración conformado por el sistema de gestión de eventos y el sistema de memoria distribuida, esta descripción se muestra en la figura 8.4. En el momento de activación del secuenciador, éste se suscribe a los eventos involucrados en la secuencia, activando los métodos de evolución del modelo, enviando los eventos programados en el instante de activarse una transición.

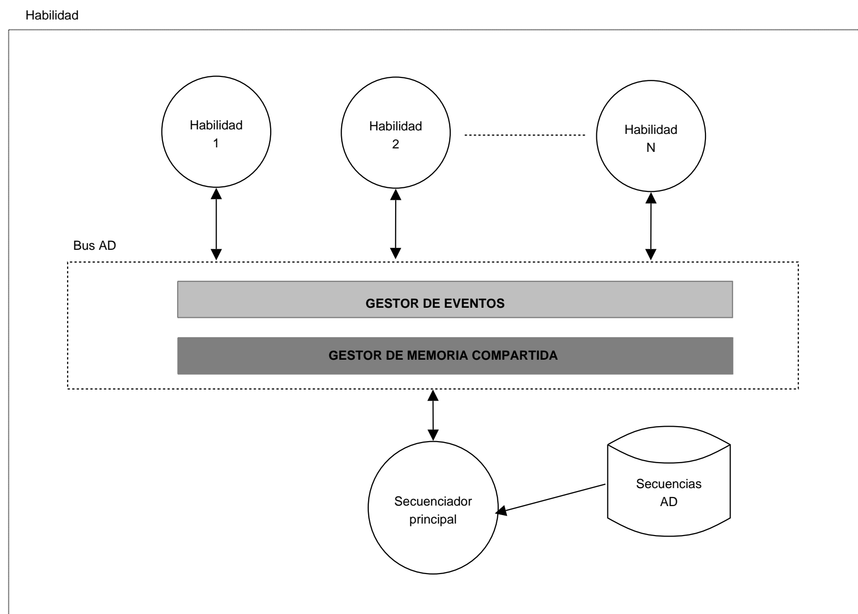

Figura 8.4: Secuenciador y habilidades

Es importante destacar, que por líneas de diseño, el secuenciador puede estar embebido en una habilidad, convirtiéndose en un elemento más de una habilidad del sistema. Esta nueva habilidad posiblemente se encuentra subordinada a otras habilidades. Lo anterior da paso a la creación de jerarquías de habilidades autónomas que pueden ser necesarias para la construcción de secuencias complejas, así como de la creación una biblioteca de habilidades de la arquitectura AD.

8.3.2. Versión 1.0

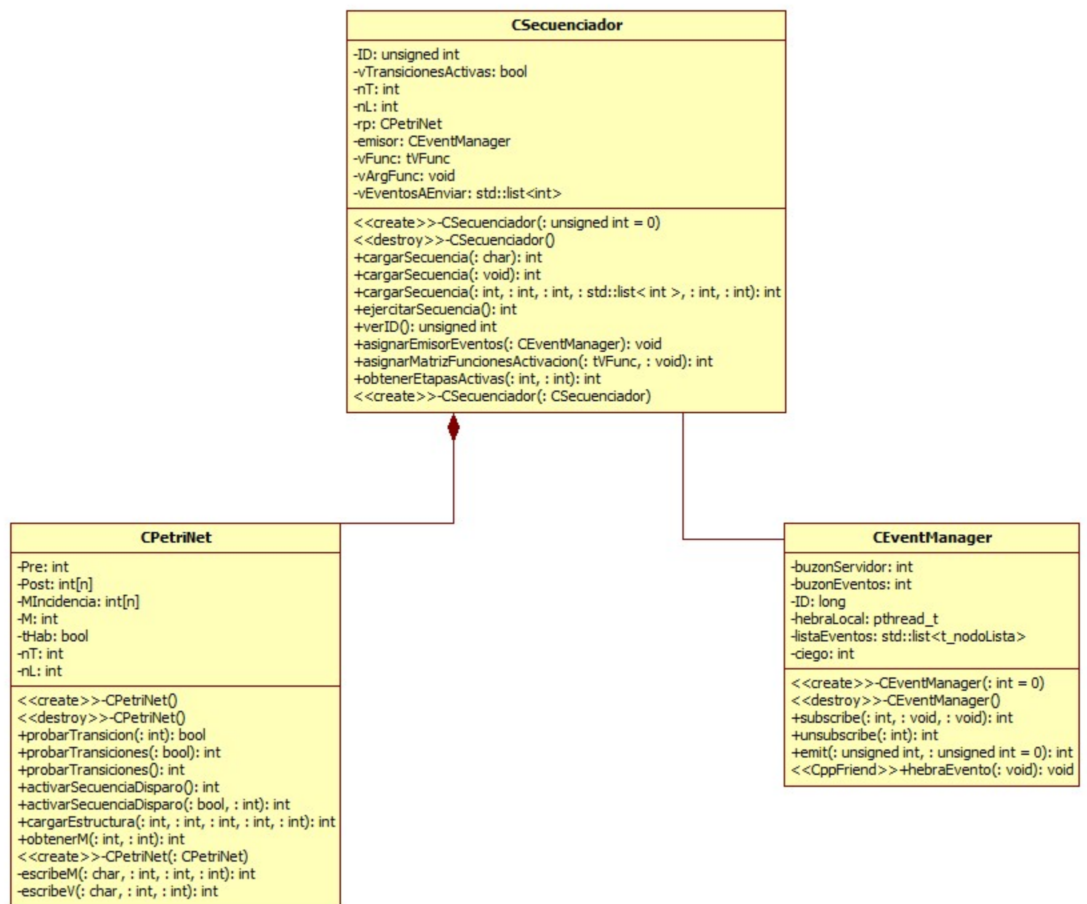

Figura 8.5: Diagrama de clases del Secuenciador v1.0

El requerimiento principal de la primera versión fue permitir la interoperabilidad entre habilidades. La coordinación se hizo principalmente por medio del envío de eventos. Entre las herramientas de descripción de comportamiento se establecieron las siguientes alternativas:

- Creación de un nuevo lenguaje.
- Uso de un lenguaje de programación puro.
- Uso de un modelo formal conocido representado por medio de un lenguaje de programación.

La creación de un nuevo lenguaje da la libertad de definir totalmente la semántica y la sintaxis. La ventaja de esta alternativa es que se puede hacer un diseño ajustado a los requerimientos del problema. El principal inconveniente es la necesidad de construir un analizador sintáctico, así como un compilador del lenguaje.

El uso de un lenguaje de programación conocido, como C, C++, Pascal, Java, SmallTalk, Lisp, Prolog, ADA. etc., permite poder usar el analizador sintáctico del lenguaje elegido de forma inmediata. La dificultad radica en la complejidad de la aplicación a construir. Las primeras pruebas del presente trabajo se realizaron usando el lenguaje C y C++. Las consignas de control se sintetizaban usando las estructuras de decisión propias del lenguaje. Una vez escritos los programas se compilaban, para luego ejecutarlos directamente sobre el robot. Como una aproximación es una solución válida, pero al tratar de realizar habilidades más complejas los programas son cada vez más complejos de construir.

La tercera alternativa plantea un esquema híbrido, ya que se utiliza un formalismo matemático realizado en un lenguaje de programación. El formalismo matemático su usa para modelar una aplicación compleja. Una vez obtenido el modelo se construye el sistema programado que realiza el modelo. La aplicación entre el modelo y el sistema programado puede ser manual, como se hizo en la versión 1.0.

Entre los formalismos usados en este trabajo se encuentran las máquinas de estado finito (ver sección A.2) y las Redes de Petri (ver subsección A.3). En un trabajo anterior [39] se usó la aproximación por medio de un lenguaje desarrollado a partir de un grafcet [102].

Como primera alternativa se usó una máquina de estado finito, que permite modelar el comportamiento del sistema en base a eventos y estados discretos. La complejidad para manejar la explosión de estados se maneja por medio de máquinas de estado finitos jerárquicas descritas en [103].

En la primera versión se logró la secuenciación de habilidades por medio de un modelo descrito en una red de Petri, en la figura 8.5 se observa las principales clases involucradas en esta versión. Las funciones de activación de las transiciones son almacenas en un vector de funciones. Estas funciones deben estar compiladas y enlazadas estáticamente a la aplicación antes de iniciar la ejecución de la secuencia. Los argumentos a la funciones de activación son de tipo abstracto (void *), que permite el pase de una dirección de memoria donde se encuentran los parámetros. El número y la naturaleza de cada grupo de parámetros depende exclusivamente de la función de activación, en otras palabras, la semántica es sólo conocida por esta función.

También se puede destacar la utilización de un vector de listas de eventos.

Se usa un vector porque en el momento de compilación se conoce el número de transiciones, y se usa una lista de eventos por cada transición ya que es posible que en una transición se envíen cero o más eventos. El uso de una referencia de gestor de eventos radica en que esto evita múltiples instancias del gestor. En este caso la función del gestor es una: enviar la lista de eventos asociadas a un transición en el momento en que ésta se activa.

Los pasos para la construcción de la aplicación son:

1. Modelar el comportamiento de la aplicación como un todo por medio de un autómata de estado finito.
2. Descomponer el proceso en componentes distribuidos y construir la Red de Petri que modela la aplicación distribuida.
3. Listar todos los eventos de activación y desactivación de las habilidades involucradas en la aplicación.
4. Escribir las funciones de activación de las transiciones.
5. Construir el contenedor de habilidades.

Esta versión inicial se caracterizó por la necesidad de compilar el modelo antes de la ejecución, lo que permite una mayor velocidad en el procesamiento. El control de las habilidades lo garantizaba la programación de las reacciones ante los eventos emitidos por el secuenciador.

8.3.3. Versión 2.0

Una de las primeras modificaciones de la versión inicial fue la de permitir que las funciones de activación se pudiesen modificar en tiempo de ejecución. Para lograr este nuevo objetivo se necesitó el poder interpretar código de un subprograma en tiempo de ejecución. En vista de que el núcleo central del sistema estaba construido en C/C++ no se deseaba migrar hacia un lenguaje interpretado. El objetivo era seguir usando C/C++ y desde programas en estos lenguajes interpretar subprogramas en tiempo de ejecución.

Entre las alternativas estaban usar archivos de guiones de comandos (shell-scripts), subprogramas en Perl, subprogramas en Python, subprogramas en Ruby, como decisión se optó por el lenguaje de programación Python [14]. Una vez embebido el interprete de Python al sistema construido es capaz de interpretar subprogramas en tiempo de ejecución. La razón fundamental de la selección es debido a las herramientas de integración que existe entre C/C++ y Python. Este último ha sido considerado como una evolución de C/C++ para programas interpretados, teniendo un rendimiento superior a la integración usando CSh-Scripts o Perl. La alternativa de usar el lenguaje Ruby no fue explorada.

La experiencia obtenida al usar Python puede solucionar problemas al integrar la plataforma de control a herramientas de alto nivel como los son acceso a Sistemas Manejadores de Bases de Datos, acceso a información Internet desde

el robot, etc. Estos avances en el sistema abren las posibilidad de modificar la estructura del secuenciador y dio origen a la versión 2.0.

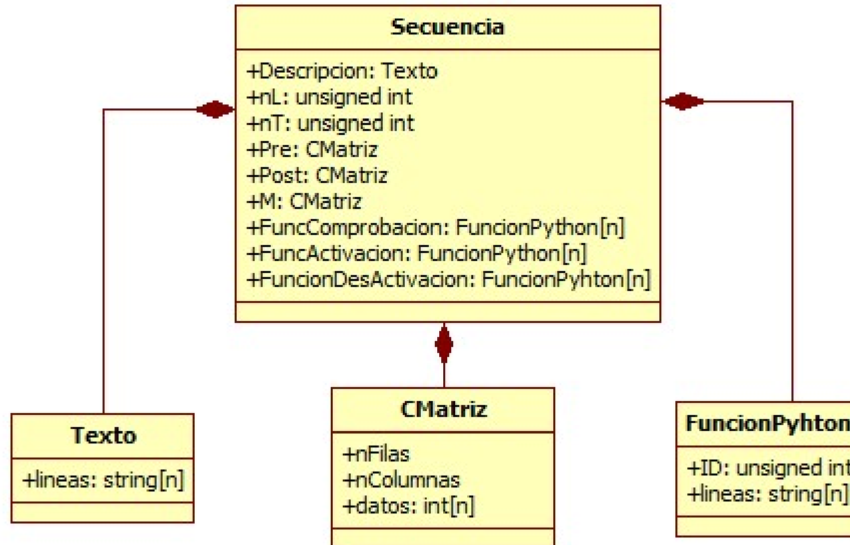

Figura 8.6: Diagrama UML de una Secuencia descrita en XML

Entre las modificaciones más significativas se encuentra la posibilidad de suministrar el modelo de comportamiento en forma dinámica, y sobre todo modificarlo una vez que el sistema se encuentre en ejecución. Para ser esto posible toda la estructura de almacenamiento de datos del modelo debe ser dinámica. Esto permite crecer o reducir en el número de lugares, transiciones y eventos involucrados en el modelo, y por supuesto dichas modificaciones deben ser capaz de realizarse en tiempo de ejecución liberando y obteniendo el espacio requerido.

La información que describe una secuencia en AD se puede observar en el diagrama de clases mostrado en la figura 8.6.

Para la suministración del modelo se pensó en archivos de texto plano, sin estructura. Esta alternativa es simple pero requiere un analizador sintáctico que sea capaz de obtener la información representada en el archivo y almacenarla en las estructuras de datos diseñadas para tal fin.

Otra alternativa es definir una estructura en base a la posición de la información del archivo. Esto requiere un analizador sintáctico menos complejo, ya que una información específica debería estar en una fila y una columna específica del archivo. El principal inconveniente de esta alternativa es que se debe predefinir el tamaño máximo o al menos dejar el suficiente espacio para almacenar toda la información que se crea que es necesaria, en el momento de guardar la información del archivo se debe dejar espacios en blanco para la información no usada. Esto puede conllevar a errores comunes por escribir la información en columnas o filas equivocadas.

La alternativa elegida se basa en el uso de un archivo estructurado, donde la información es almacenada usando marcas. Aprovechando el auge que hoy en día tiene el intercambio de información por medio de Internet se decidió por el uso del formato XML [104, 105], sigla en inglés de eXtensible Markup Language. XML es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium [106]. Es una simplificación y adaptación del SGML [107] y permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML

Para normalizar los datos de la secuencia a representar en XML se ha definido la estructura y la información a representar por medio de un esquema bajo la norma XML Schema [108]. XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción.

Secuencias de habilidades en XML

El XML Schema que define una secuencia AD en XML es:

```
 <?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <xsd:annotation>
      <xsd:documentation xml:lang="es">
         <br>Esquema XML de una secuencia en la arquitectura AD.</br>
         Cada secuencia debe tener:
         <ul>
         <li>una Descripción, </li>
         <li>un Número nL de etapas,</li>
         <li>un Número nT de transiciones,</li>
         <li>una Matriz de Pre-Condiciones,</li>
         <li>una Matriz de Post-Condiciones,</li>
         <li>un Vector de marcas inicial,</li>
         <li>nT Funciones Pyhton para comprobar la condición de disparo de cada transición.</li>
         <li>nT Funciones Pyhton a ejecutar en el momento en que se active un lugar.</li>
         <li>nT Funciones Pyhton a ejecutar en el momento en que se desactive un lugar.</li>
         </ul>

         <br>Rafael Rivas E.</br>
         <br>rafael@ula.ve</br>

         <br>Julio, 2007</br>
         <br>Version 2.0</br>

         <br>Copyrigth 2007, Roboticslab UC3M, All rights reserved</br>
      </xsd:documentation>
   </xsd:annotation>

   <!-- Declaración de secuencia -->
   <xsd:element name="Secuencia" ref="tSecuenciaAD"/>

   <!-- Definicion de vector Eventos -->
   <xsd:complexType name="tVectorEventos">
      <xsd:secuence>
         <xsd:element name="id" type="xsd:nonNegativeInteger" minOccurs="1" maxOccurs="1" />
         <xsd:element name="e" type="xsd:positiveInteger" minOccurs="0"/>
      </xsd:secuence>
```

```
</xsd:complexType>


<!-- Declaración de una matriz de enteros positivos -->
 <xsd:complexType name="tVectorEnterosPositivos">
   <xsd:element name="valor" type="xsd:unsignedInt" minOccurs="1" />
</xsd:complexType>

<xsd:complexType name="tMatrizEnterosPositivos">
   <xsd:element name="fila" ref="xsd:tVectorEnterosPositivos>" minOccurs="1" />
</xsd:complexType>

<!-- Tipo Funcion Python a interpretar -->
<xsd:complexType name="tFuncionPython">
  <xsd:sequence>
      <xsd:element name="id" type="xsd:nonNegativeInteger" minOccurs="1" maxOccurs="1" />
      <xsd:element name="linea" type="xsd:string" minOccurs="1" />
   </xsd:sequence>
</xsd:omplexType>

<!-- Definición del tipo SecuenciaAD -->
<xsd:complexType name="tSecuenciaAD">
   <xsd:sequence>
      <xsd:element name="Descripcion" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="nEtapas" type="xsd:positiveInteger" minOccurs="1" maxOccurs="1" />
      <xsd:element name="nTransiciones" type="xsd:positiveInteger" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="mPreCondiciones" ref="tMatrizEnterosPositivos" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="mPostCondiciones" ref="tMatrizEnterosPositivos" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="vMarcas" ref="tVectorEnterosPositivos" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="FuncionEvaluacion" ref="tFuncionPython" minOccurs="1"/>
      <xsd:element name="FuncionActivacion" ref="tFuncionPython" minOccurs="1"/>
      <xsd:element name="FuncionDesActivacion" ref="tFuncionPython" minOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

El esquema se inicia con una cabecera donde indica la versión de XML usada, así como el tipo de codificación usada.

La parte siguiente (annotation) es la descripción del contenido del archivo. Se indica las partes que debe contener una secuencia XML, así como la versión del documento.

A continuación se describe cada uno de tipos definidos por el usuario. Estos son extensiones de tipos básicos de datos en XML, así como las veces que puede aparecer cada componente en el documento.

```
<FuncionEvaluacion>
  <id>1</id>
  <linea>import datosMCAD</linea>
  <linea>import MC</linea>
  <linea>(evento,p) = MC.ObtenerDatoActual(datosMCAD.id, datosMCAD.tam, datosMCAD.formato)</linea>
  <linea>if (evento == datosMCAD.OBSTACLE_OUT):</linea>
  <linea>\ returnAD = 1</linea>
  <linea>else:</linea>
  <linea>\ returnAD = 0</linea>
</FuncionEvaluacion>
```

Figura 8.7: Ejemplo de función de evaluación

En la parte final se especifica que una secuencia en AD debe contener:

- Un descripción que indique el objetivo de la secuencia, versión y autor.

- El número de lugares de la red de Petri que modela la secuencia. Se indica el tipo (entero positivo) y el numero máximo de ocurrencias de este dato en el documento.

- El número de transiciones de la red de Petri que modela la secuencia, y al igual que el número de lugares se indica el tipo y el número máximo de ocurrencias.

- La matriz de precondiciones para la habilitación de las transiciones de la red de Petri. Ésta se define como una instancia de tipo tMatrizEnterosPositivos, definida en el mismo documento, con un numero de filas igual al número de lugares y un número de columnas igual al numero de tranciociones de la red de Petri que modela la secuencia. También se indica que sólo puede existir una matriz de precondiciones en el documento.

- La matriz de postcondiciones indicando el cambio en las marcas de la red de Petri después de activarse las transiciones habilitadas. Al igual que la matriz de precondiciones se define como una instancia del tipo tMatrizEnterosPositivos, con las mimas dimensiones, así como el número máximo y mínimo de ocurrencias.

- Vector de marcas, con la marcación inicial de la red Petri antes de iniciar la activación de las transiciones, el tipo y la ocurrencia en el documento.

- Funciones de evaluación en Pyhton, donde cada transición tiene una función de activación asociada. En el momento que una transición esté habilitada se ejecuta la función de activación y si el resultado de esta función entrega el valor *verdadero* (diferente a cero) la transición se activa y se dispara. En la figura 8.7 se muestra un ejemplo de una función de evaluación. También se puede observar la instanciación de un objeto del servidor de memoria compartida migrado a Python. El el archivo XML de la secuencia deben existir tantas funciones de evaluación como transiciones haya en la red de Petri que modela la secuencia.

- Funciones de activación: cuando un lugar en la red de Petri pasa de cero a más de cero marcas se invoca la función de activación asociada a ese lugar. La función debe estar escrita en el lenguaje de programación Python. En la figura 8.8 se muestra un ejemplo de una función de activación. También puede observarse la instanciación de un objeto del gestor de eventos migrado a Python. En el archivo XML de la secuencia deben existir tantas funciones de activación como lugares haya en la red de Petri que modela la secuencia.

- Funciones de desactivación: cuando un lugar en la red de Petri pasa a cero marcas luego de una transición se invoca la función de desactivación asociada a ese lugar. La función debe estar escrita en el lenguaje de programación Python. En el archivo XML de la secuencia deben existir tantas funciones de desactivación como lugares haya en la red de Petri que modela la secuencia.

```
<FuncionActivacion>
    <id>0</id>
    <linea>import datosMCAD</linea>
    <linea>import eventosAD</linea>
    <linea>eventosAD.emite(datosMCAD.WANDERING_ON)</linea>
    <linea>return 0</linea>
</FuncionActivacion>
```

Figura 8.8: Ejemplo de función de activación

Adaptación del lenguaje Python para el manejo de secuencias

Python es un lenguaje de programación creado por Guido van Rossum en el año 1990. Es comparado habitualmente con TCL, Perl, Scheme, Java y Ruby. En la actualidad Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. La última versión estable del lenguaje es actualmente la 2.5.1 (18 de abril de 2007).

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Posee una gran colección de módulos estándar que se pueden utilizar como base de los programas. También hay módulos incluidos que proporcionan entrada/salida de ficheros, llamadas al sistema, sockets e interfaces a GUI (interfaz gráfica con el usuario) como Tk, GTK, Qt entre otros.

Python es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa.

Como se dijo en la subsección anterior, la selección de Python en el desarrollo de este sistema se basa en la capacidad de integrar el interprete Python en un programa de C/C++. Esto permite invocar dinámicamente un programa en Python desde una habilidad y ejecutarlo. La principal desventaja de Python en comparación con programas en C/C++ es la velocidad de ejecución. El hecho de activar el interprete Python y ejecutar un programa tiene una penalización de al menos 250 mseg en las actuales versiones y plataformas usadas en este trabajo. Por esta última razón se usa Python solo en los casos donde las penalizaciones de tiempo no comprometen el desempeño global de la aplicación.

Indudablemente el uso de Python da una gran flexibilidad para el cambio de contexto en el control de ejecución de las habilidades, permitiendo conmutar modelos de comportamientos completos con solo cambiar el modelo almacenado en un documento XML.

Python usa como herramienta de agrupación de sentencias el sangrado de líneas. Otros lenguajes de programación como el Pascal[109, 110] usan las palabras reservadas *begin* - *end*, otros lenguajes como el C/C++, Java reservan los caracteres *{ }*. Al almacenar un programa de Python en un documento XML se debe garantizar el sangrado de líneas, por esta razón, cada línea de código se representa en el documento XML como una instancia del tipo línea y los espacios en blanco son representados con una barra invertida seguida de un espacio

en blanco.

Adaptación del gestor de eventos para ser usado desde Python

En este desarrollo la capacidad de activar o desactivar habilidades se basa en la reacción de éstas ante eventos específicos. Por lo tanto, si se desea activar o desactivar una habilidad de forma automática, se debe disponer de los mecanismos necesarios para el envío de eventos. No se justificaría la incorporación de programas hechos en lenguaje Python al desarrollo, a no ser que se pueda activar o desactivar habilidades por medio de eventos desde ese lenguaje.

La exportación de los mecanismos de emisión de eventos ha requerido hacer extensiones al lenguaje Python. Éstas se han construido siguiendo las pautas descritas en [111]. En este caso se ha creado un componente que contiene un objeto de la clase *CEventManager*. Este componente no activa la hebra de escucha de eventos, por lo que sólo puede emitir eventos. El componente presenta un solo servicio llamado *emite*, este acepta dos parámetros: el identificador del evento a emitir y un parámetro al mismo. Este componente se ha compilado como una biblioteca de enlace dinámico de Python.

Para usar el gestor de eventos desde un programa escrito en Python se debe importar la biblioteca *eventosAD*. Para el correcto enlace de la biblioteca en tiempo de ejecución, ésta debe estar en al camino de búsqueda de los componentes dinámicos definidos en las variables de entorno de la consola de ejecución.

En la figura 8.8 se muestra un ejemplo del envío de un evento desde un programa escrito en Python para la arquitectura AD.

Adaptación del manejador de memoria compartida para ser usado desde Python

El uso de memoria compartida en las secuencias de la arquitectura AD es fundamental. Esto es debido a la necesidad de implementar las funciones de evaluación de las transiciones habilitadas. Muchas veces la activación de una transición se realiza al cambiar el valor de una variable, que puede ser la distancia a un punto, nivel de carga de batería, etc. El compartir datos entre componentes de AD es fundamental independientemente del lenguaje que se use. Al igual que en el caso de la exportación de la interfaz al sistema de gestión de eventos para la memoria compartida también se sigue las pautas descritas en [111].

Para la extensión se ha creado un componente que contiene un objeto de la clase *CmemCortoPlazo*. Este componente tiene dos servicios: *ColocarDatoActual* y *ObtenerDatoActual*. Con estos métodos es posible obtener y colocar datos en la memoria compartida.

Para usar el sistema de memoria compartida desde un programa escrito en Python se debe importar la biblioteca *memCompartida*. Y al igual que en el caso del gestor de eventos la biblioteca debe encontrarse en un subdirectorio en el camino de búsqueda de la consola de aplicación.

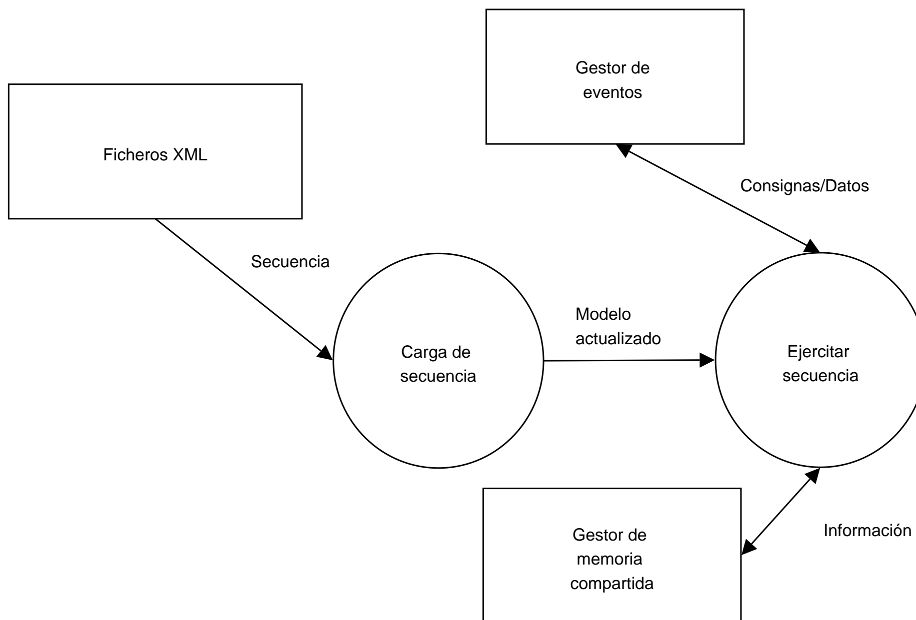

Figura 8.9: Diagrama Cero del flujo de datos en el secuenciador

En la figura 8.7 se muestra un ejemplo del uso de la memoria compartida desde un programa escrito en Python.

Funcionamiento de la secuenciación automática de habilidades.

El flujo de datos en la ejecución en forma automática de habilidades es visualizada en la figura 8.9. En este caso la secuencia es almacenada en un documento XML. Este documento contiene toda la información necesaria para la correcta evolución en el comportamiento de la aplicación. Este documento puede ser introducido de forma explícita o puede ser tomado de un Manejador de Bases de Datos, un servidor Web o un directorio en un sistema de ficheros.

Al realizar la carga del documento el sistema crea la estructura de datos, almacena las matrices de pre y post condiciones y genera la matriz de coincidencia de la red de Petri. Una vez almacenado el modelo en la memoria de la aplicación, se registran las marcas en la red de Petri según el vector de marcas almacenado en el documento.

Para el procesamiento de la secuencia se usa el sistema manejador de memoria compartida para lograr el intercambio de información entre las diferentes habilidades. Algunos de estos datos son usados por las funciones de activación en la toma de decisiones para el disparo de transiciones. Estos datos también pueden ser modificados por las funciones de activación y desactivación para permitir el trabajo colaborativo entre habilidades.

El uso del gestor de eventos permite activar la actuación del secuenciador,

que reacciona ante cualquier evento emitido por cualquier habilidad. Una vez procesadas y disparadas las transacciones, las funciones de activación y desactivación, asociadas a lugares en la red de Petri, pueden emitir eventos y de esta forma bloquear o activar habilidades.

Una vez cargada la información todos los datos se encuentran en la memoria principal del secuenciador

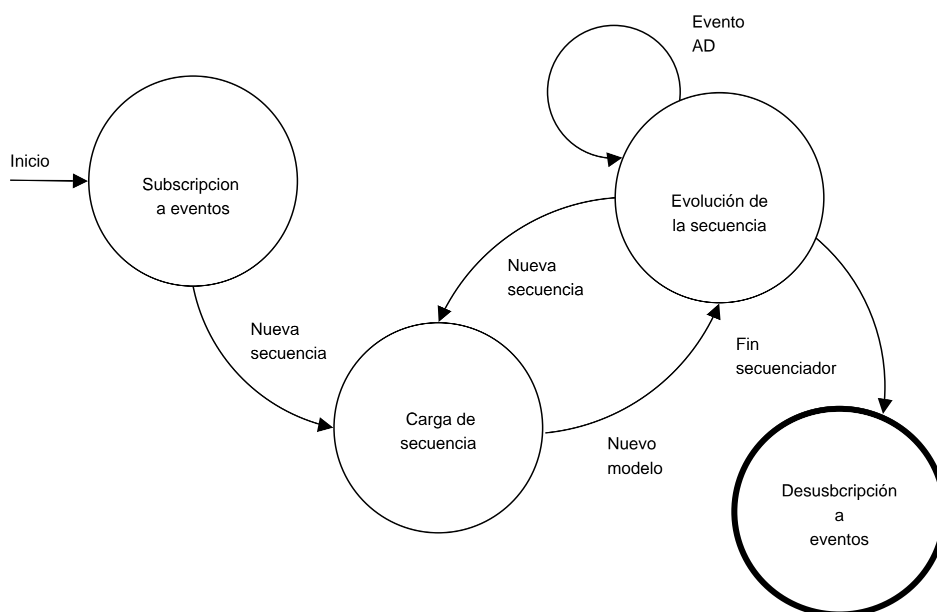

Figura 8.10: Autómata del secuenciador

El comportamiento de la secuenciación automática es resumido en la figura 8.10. Cada secuenciador requiere poseer una referencia al gestor de eventos. En el momento de la inicialización se procede a la subscripción de eventos. En este caso se subscribe a todos los eventos excepto a los suyos. En el momento que una habilidad o un sensor virtual emita un evento, éste es capturado por el secuenciador.

Al capturar un evento el secuenciador crea una lista de las transiciones habilitadas para activarse. Para cada una de la transiciones activas ejecuta la función de activación asociada a ésta. Al final de la evaluación se recupera el resultado de la función y si es diferente de cero, se marca la transición como activa.

Una vez que se tienen todas las transiciones activas, se procede a la evolución de la red de Petri y se actualiza la nueva posición de las marcas.

Los lugares cuyas marcas pasen de cero (0) a un valor mayor o igual a uno (1) se consideran que cambiaron a estado de activas, por lo tanto, la función de activación asociada a ese lugar se ejecuta, emitiendo los eventos preprogramados o modificando valores en la memoria compartida.

De igual manera al caso anterior, si un lugar en la red de Petri pasa de un

número estricto positivo a cero marcas, se considera que el lugar ha entrado en estado de desactivación. La función asociada al lugar recién desactivado se ejecuta, modificando valores en la memoria compartida o emitiendo eventos preprogramados.

El secuenciador como una clase

En el diagrama de clases (figura 8.11) muestra las clases involucradas en la versión 2.0 del secuenciador. Ésta se caracteriza porque en sus componentes aparecen los contenedores para las funciones de evaluación, activación y desactivación. Estos son contenedores implementados mediante vectores parametrizados de la STL de C++. El tipo del objeto contenidos son cadenas de tamaño dinámico también de la clase STL.

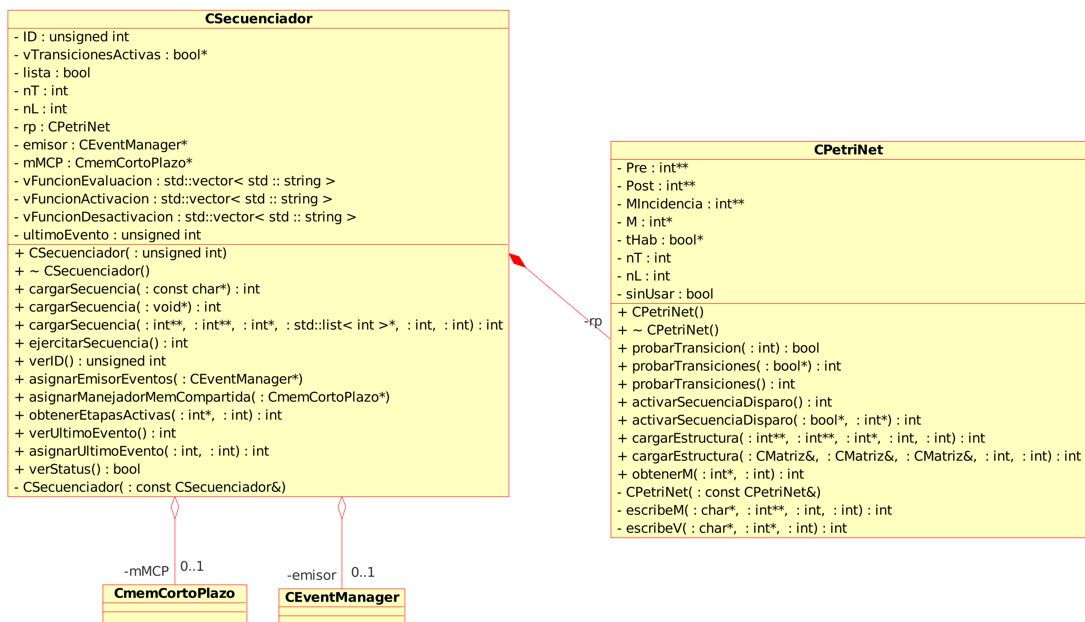

Figura 8.11: Diagrama de clases de la clase CSecuenciador versión 2.0

Entre los principales cambios en los métodos se encuentra el nuevo método CSecuenciador::cargarSecuencia(const char *). Este método es el encargado de cargar una secuencia a partir de un documento XML. El parámetro de entrada sería la ubicación en un sistema de archivos del fichero que contiene la secuencia.

El método CSecuenciador::ejercitarSecuencia() ha sido reescrito en su totalidad para contener el interprete del lenguaje Python. Éste se carga en la primera instanciación de un objeto de la clase CSecuenciador. Esta tarea consume alrededor de más de 300 mseg y las siguientes llamadas consumen alrededor de 80 mseg. Esto se debe a que el interprete se mantiene en la memoria *cache* del sistema operativo.

El acceso al manejador de memoria compartida y del gestor de eventos no se

realiza por medio de la instanciación de un objeto de dichas clases, se hace por medio de referencias. La justificación de esto es para evitar la sobre carga del sistema, ya que cada instancia de estas clases implanta una parte del Middleware de AD. Por ser una clase no es un programa completo, es decir, es un componente que requiere de una aplicación para poder activarse. Esta aplicación puede ser un contenedor o un programa independiente.

La clase CSecuenciador puede estar instaciada en un programa compilado. Este programa sería un secuenciador y se ejecutaría como un programa independiente. Por haber sido diseñada como una clase y no como un programa, un objeto o una referencia de la clase secuenciador puede también ser parte de una habilidad. En este caso esta habilidad tendría su propio secuenciador con un modelo de secuenciación propio. El entorno de desarrollo de la aplicación es Linux, este sistema se caracteriza porque sus manejadores se usan de una manera similar a la que se acceden los archivos. Por lo tanto, el cargador de secuencias por medio de archivos documentos XML no es muy diferente a la carga por medio se BSD Sockets, Sistemas Manejadores de Base de Datos, Sistemas de Archivos Distribuidos, y Servidores Web.

La clase CPetriNet ha mantenido su estructura inicial, ya que su funcionalidad es la misma en las versiones realizadas: calcular las transiciones habilitadas y el nuevo marcaje en caso de activarse una o varias transiciones. De hecho por ser un componente reusable, la clase CPetriNet puede ser usada en otras aplicaciones que no sea el secuenciador.

Otra ventaja del diseño propuesto es la capacidad de extensión de las funcionalidades hechas. Esto se haría usando la capacidad de heredar código, especializando las clases diseñadas en nuevas versiones del secuenciador. Entre las posibles extensiones se puede pensar en secuenciadores no deterministicos, o secuenciadores con capacidades de aplicar prioridades entre habilidades o penderación para la selección de una habilidad u otra.

El hecho de tener la funcionalidad de secuenciamiento en una habilidad, da la capacidad de tener una jerarquía de habilidades con diferentes niveles de secuenciación.

8.4. Editor y generador de secuencias

La edición de secuencias, tal como se ha presentado, es una tarea que requiere de conocer la creación de secuencias de habilidades, y su representación en XML. En la representación de una secuencia en XML se debe tener cuidado y respetar el formato reconocido por el secuenciador de habiliades. En [112] se presenta un editor gráfico y generador de secuencias en XML. Este trabajo se realizó como trabajo posterior a la construcción del secuenciador de habilidades. El objetivo principal del editor de secuencias fue la creación de un entorno gráfico donde los usuarios puedan crear y editar secuencias de control que permitan activar y desactivar comportamientos en un robot. La interfaz del editor de secuencias es mostrada en la figura 8.12.

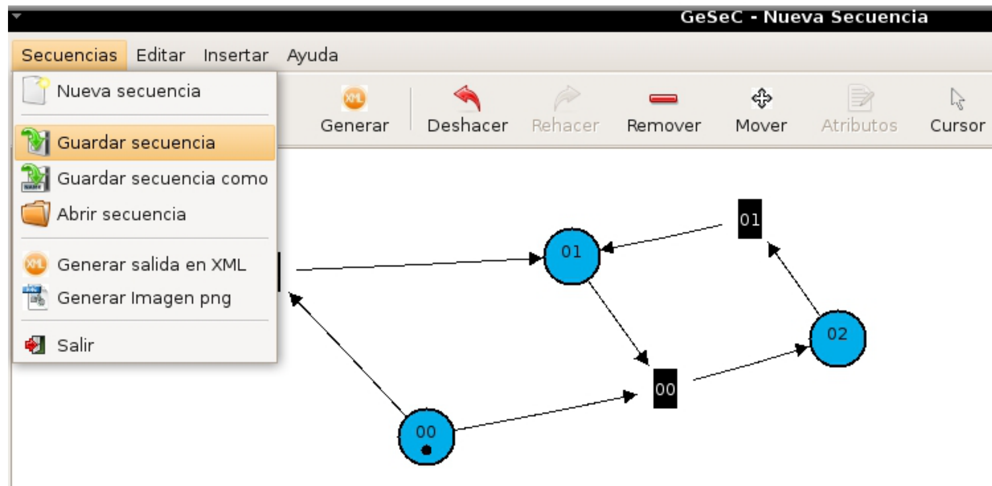

Figura 8.12: Interfaz del editor de secuencias

Entre las funcionalidades adicionales se encuentra la capacidad para permitir la edición de programas cortos escritos en lenguaje Python, adaptados para las secuencias de control de habilidades en la arquitectura AD. Con estos programas se describen las funciones asociadas a los Lugares y la Transiciones del modelo.

En el editor de secuencias el diseñador puede dibujar la red de Petri que representa la secuencia de habilidades, permitiendo:

- Definir las funciones de activación y desactivación en los lugares de la red de Petri. Ver figura 8.13.
- Definir la función de evaluación en las transiciones de la red.

Un secuencia puede ser almacenada, a pesar que no este terminada, y luego ser cargada de nuevo para continuar su edición. Una de la funcionalidades mas atractivas es que al finalizar la red de Petri que modela una secuencia, el editor puede generar de forma automática el fichero en formato XML que contiene la secuencia de control.

8.5. Observaciones finales del capítulo

En este capítulo ha sido presentado la implementación de habilidades para la arquitectura AD. Estas permiten describir cualquier habilidad, y puede extenderse para satisfacer nuevo requerimientos de diseño.

El secuenciador permite ejecutar secuencias de control de habilidades, el lenguaje de representación es XML, un lenguaje que se ha convertido en un estándar para la representación de documentos de cualquier tipo.

Figura 8.13: Edición de los atributos un Lugar

El incorporar funciones de activación, desactivación y evaluación en lenguaje Python, que son interpretadas por el secuenciador, permite la flexibilidad de ejecutar funciones que solo son conocidas en el momento de la ejecución de la secuencia.

Por último, el editor y generador de secuencias, permite un entorno de diseño y construcción de secuencias por medio de un entorno gráfico.

9

Datos de desempeño de los principales componentes

En este capítulo se presentan algunos resultados experimentales del sistema diseñado y construido. La arquitectura software para aplicaciones robóticas se basa en tres componentes principales como es el sistema de memoria a corto plazo, el sistema distribuido de gestión de eventos y la plantilla abstracta de clases de habilidades.

Existen diferentes escenarios de operación, cada uno de éstos posee un comportamiento particular que depende del mecanismo de comunicación subyacente. La palabra *par*, en este capítulo, hace referencia a un par de habilidades o a un par de componentes software que usan los servicios de la arquitectura desarrollada. Los escenarios son:

- Pares en un mismo ordenador.
- Pares residentes en ordenadores diferentes dentro del robot.
- Pares que se encuentran distribuidos entre un ordenador fuera del robot y un ordenador instalado dentro del robot.

En el primer escenario los mecanismos de comunicación subyacentes se implementan sobre primitivas del núcleo del sistema operativo. En este caso dicho mecanismo usa servicios contenidos en la biblioteca POSIX. Las funciones de esta biblioteca se caracterizan por tiempos de ejecución cortos y un espacio de memoria limitado.

En el segundo escenario los servicios de comunicación subyacentes se basan en la red de datos interna del robot. Ésta es una red Ethernet [113] [114] que sigue el estándar IEEE 802.3. Este tipo de red se caracteriza por tener una política de acceso al medio por contienda entre los participantes, por lo tanto, el tiempo de acceso al medio es *no determinístico*. En segmentos Ethernet donde el número de ordenadores conectados es alto, es posible que alguno de éstos no

pueda acceder al medio en un tiempo finito [115, Chapter 4]. Los ordenadores internos en el robot Maggie son usualmente tres ordenadores, y se considera el acceso el dispositivo de acceso a la red WIFI como un cuarto ordenador conectado. La velocidad de señalamiento al medio del bus Ethernet instalado es de 100 Mbps. El uso de este medio implica un coste de tiempo por la señalización al medio, codificación de los datos, envío, recepción, decodificación y entrega al proceso destinatario.

El tercer escenario se presenta cuando uno de los miembros del par se encuentra en un ordenador dentro del robot, y el otro miembro se encuentra fuera del robot. Este caso es similar al presentado en el segundo escenario, con la diferencia que en este caso el mensaje o dato enviado debe pasar a través de, al menos, tres segmentos de red: la red interna del robot, la red inalámbrica que comunica al robot con el exterior y, por último, la red donde se encuentra el ordenador donde reside el miembro del par.

Las pruebas serán presentadas en dos grupos: pruebas al sistema de memoria a corto plazo y pruebas al sistema de gestión de eventos.

Las gráficas de los datos obtenidos se han hecho usando MATLAB.

9.1. Sistema de memoria a corto plazo

El sistema de memoria a corto plazo permite compartir datos entre diferentes procesos en ordenadores interconectados a una red TCP/IP. Este sistema detecta los cambios en los datos registrados y propaga los cambios a través de los diferentes servidores auxiliares. Los tiempos de propagación dependen de la ubicación de los servidores auxiliares y estos se pueden encontrar en el segmento de red interno del robot o en un segmento de red externo al robot.

Cada servidor posee una copia del valor actual del datos, es decir, si recibe una solicitud de un dato especifico éste lo entrega de su memoria local. Si el sistema recibe la solicitud de actualizar un dato, éste lo registra quedando a disposición de los clientes locales del sistema. La actualización en el resto de servidores se realiza por procesos que residen en hebras de procesos independientes.

A continuación se muestran los datos obtenidos en pruebas de lectura y actualización de datos desde y hacia el sistema distribuido de memoria compartida.

9.1.1. Lectura desde la memoria a corto plazo

La pruebas de lecturas se hicieron tomando datos de 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K 2K, 4K, 8K, 16K, 32K y 64K Bytes. Por cada tamaño se tomaron 10000 muestras. Para medir el tiempo se tomó el tiempo del sistema usando la primitiva gettimeofday().

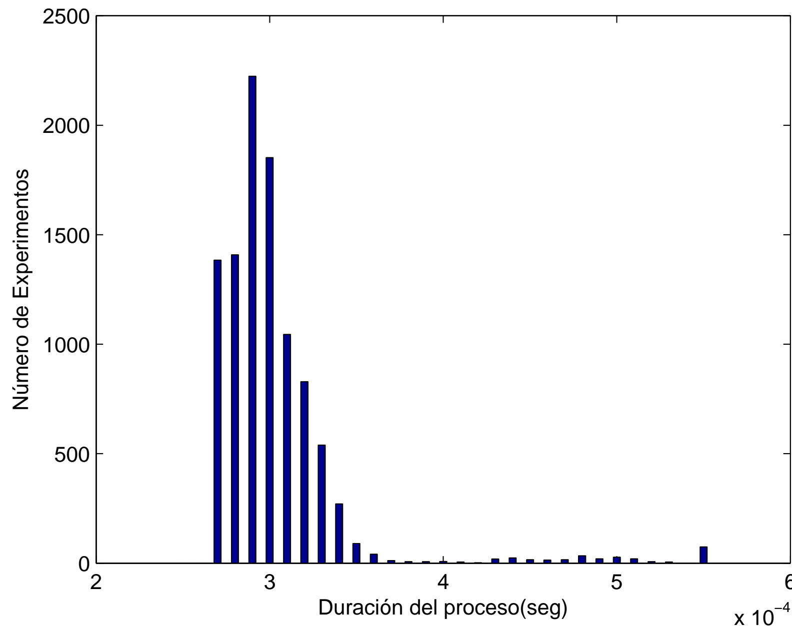

Figura 9.1: Duración del proceso de lectura de datos de tamaño 2 Bytes

En este informe se muestran algunas de la gráficas obtenidas. Las gráficas de las figuras 9.1 muestran el comportamiento de la muestra para lo que se consideran valores individuales que pueden ser representados por un entero o por número en coma flotante con tamaño de 2 bytes. En este caso los tiempos de duración de lectura están concentrados entorno a los 0.3 mseg.

Las secuencias de datos se pueden representar en 512 bytes: en este rango se puede tener un vector de 256 enteros, 256 flotantes o 128 datos en punto flotante de doble precisión. Como se observa en la figura 9.2 la duración del proceso de lectura se concentra alrededor de los 0.35 mseg.

Para tipos de datos de más tamaño, o datos almacenados en estructuras mas complejas, se muestran las gráficas de los experimentos con datos de 16KB y 64KB, mostradas en las figuras 9.3 y 9.10 respectivamente.

En el caso de datos de tamaño de 16 KB el tiempo de duración de la lectura se concentra en los alrededores de los 0.9 mseg, y en el caso de los datos de 64 KB los datos se concentran entre los 3.6 y 3.7 seg.

Los tiempos de duración promedio de cada uno de las 17 experimentos, de 10000 muestras cada uno, son mostrados en la figura 9.5. Los tamaños de los datos en este caso son mostrados en una escala logarítmica, donde los tamaños van desde 2^0 hasta 2^{16} Bytes. La información en esta gráfica es complementada por medio de la gráfica de la desviación estándar de cada una de las 17 muestras, ver figura 9.6.

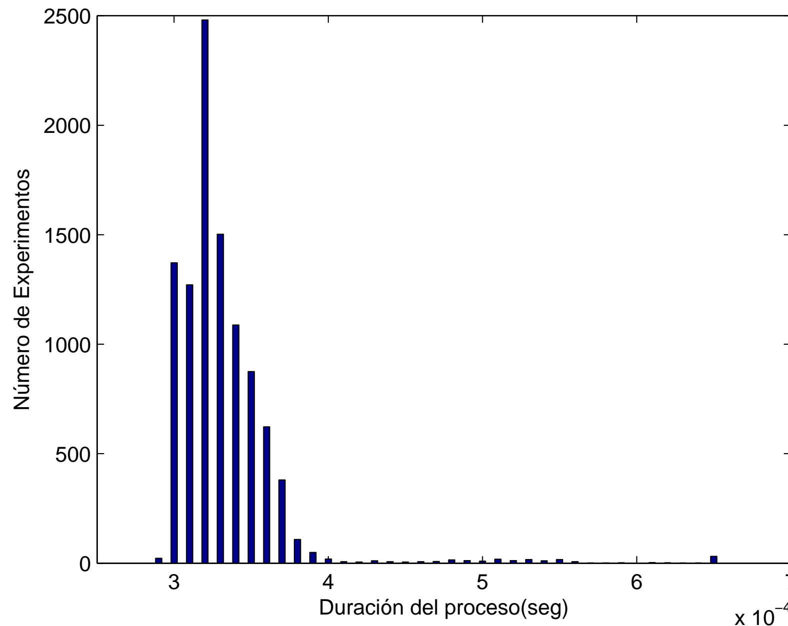

Figura 9.2: Duración del proceso de lectura de datos de tamaño 512 Bytes

En la tabla 9.1 se muestran los datos de las gráficas de promedio y desviación estándar de los 17 experimentos. Es importante destacar que la desviación estándar está representada en mseg, lo que indica que los valores de duración están fuertemente concentrados alrededor de los valores promedio de cada muestra. El núcleo del sistema operativo no opera en modo de tiempo real, ya que en ese caso las desviaciones estándar promedio serían nulas.

El coeficiente de correlación lineal de los datos obtenidos es de 0.996. Este coeficiente se ha calculado usando como variable independiente el tamaño del dato a leer, y como variable dependiente el tiempo empleado en la lectura. El coeficiente de determinación en este caso es del 0.992. Por lo que se puede asegurar que existe una relación lineal entre el tamaño de los datos y los tiempos obtenidos. Específicamente el modelo que mejor se ajusta es una recta con coeficiente igual a 5.02E-8 seg/bytes y constante igual a 0 seg.

En la tabla 9.2 se muestran los valores mínimos y máximos de duración en cada proceso de lectura del sistema distribuido de memoria compartida.

En esta subsección se han mostrado los resultados de las pruebas del sistema de memoria a corto plazo. Todos los datos se han tomado en uno de los ordenadores del robot, específicamente en el ordenador nombrado como Vismaggie, que es un ordenador con un procesador Centrino DUO y 1 GB de memoria RAM. Como se indicó en el párrafo anterior el núcleo del sistema operativo no opera en tiempo real, ya que es un núcleo Linux de la familia 2.6.21-SMP.

Observando la desviación estándar promedio de los tiempos registrados se

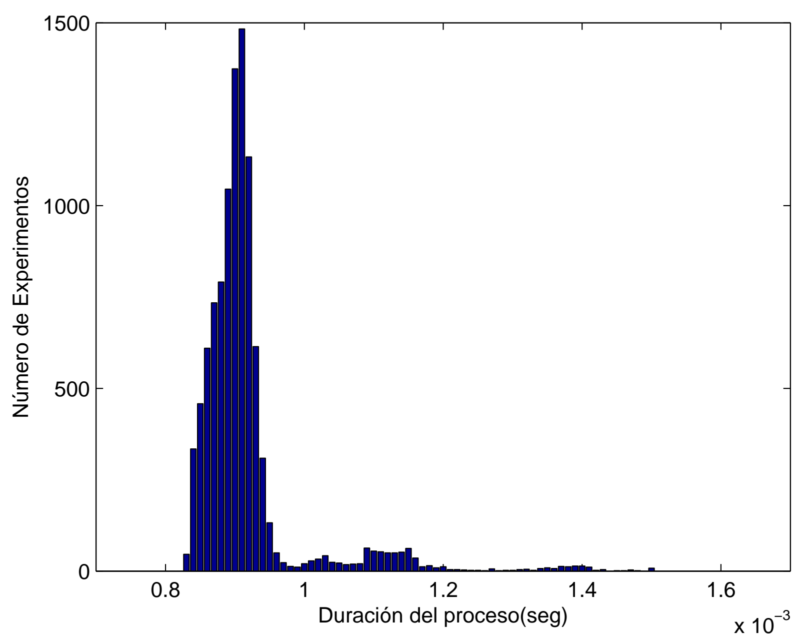

Figura 9.3: Duración del proceso de lectura de datos de tamaño 16 KByte

observa que estos se encuentran por debajo de 1 mseg. Esto indica que los tiempos de duración están altamente concentrados en los entornos de los valores promedio. El hecho de no operar en tiempo real indica que los tiempos de duración no son determinísticos, por lo que es posible que puedan aparecer medidas fuera de banda, como se indica en la tabla 9.2. La cantidad de datos *lejanos* del valor promedio pueden estimarse. El hecho de que se tenga un número grande de experimentos, en este caso 10000 muestras, permite aplicar el Teorema del Límite Central, descrito en [116]. Por medio de este teorema es posible aproximar la distribución de datos a una distribución Normal, lo que implica que la probabilidad de que se tenga una dato fuera de banda se reduce al 5 %.

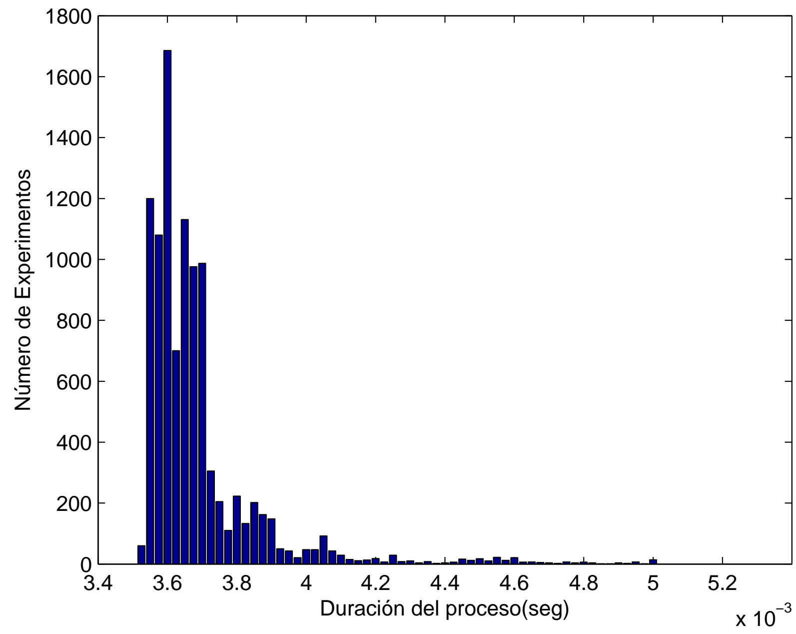

Figura 9.4: Duración del proceso de lectura de datos de tamaño 64 KByte

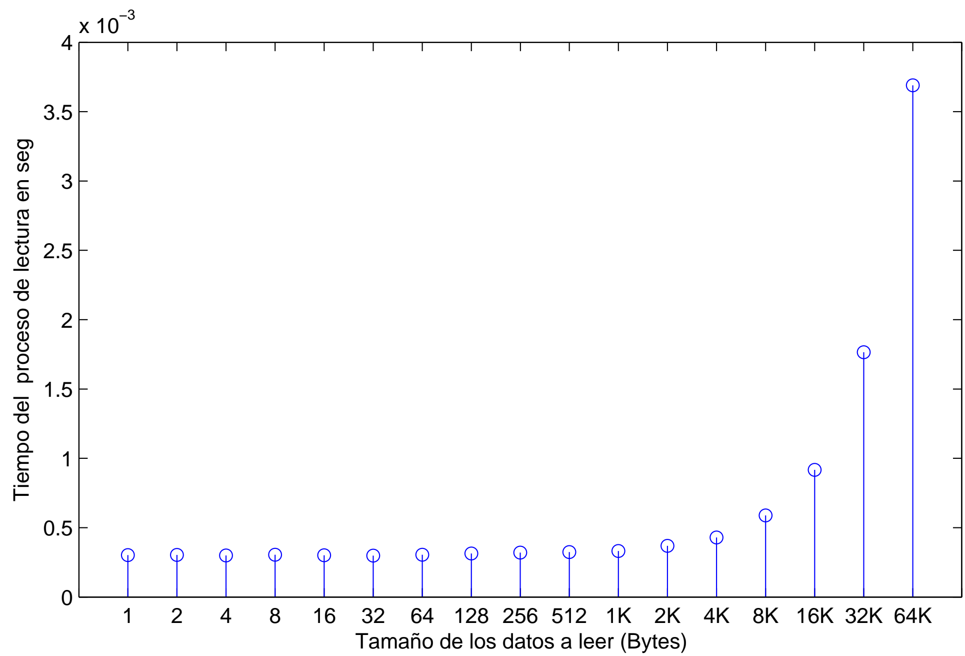

Figura 9.5: Duración promedio del proceso de lectura

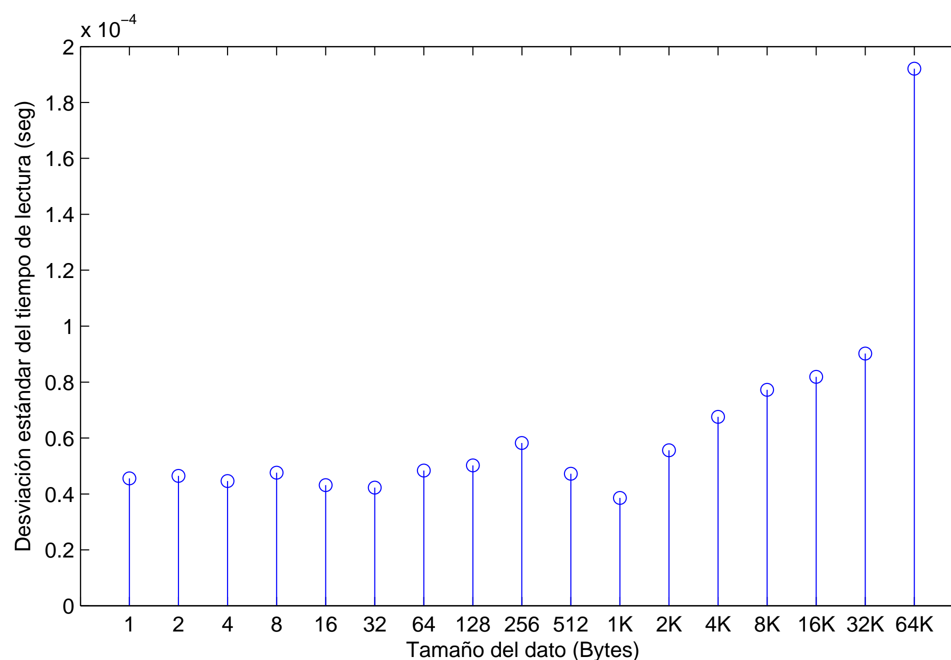

Figura 9.6: Desviación promedio del proceso de lectura

Tamaño del dato a recuperar (Bytes)	**Tiempo medio de lectura (seg)**	**Desviación Estándar de lectura (mseg)**
1	0.0003	0.0456
2	0.0003	0.0464
4	0.0003	0.0446
8	0.0003	0.0476
16	0.0003	0.0431
32	0.0003	0.0422
64	0.0003	0.0484
128	0.0003	0.0502
256	0.0003	0.0582
512	0.0003	0.0472
1K	0.0003	0.0386
2K	0.0004	0.0556
4K	0.0004	0.0675
8K	0.0006	0.0772
16K	0.0009	0.0819
32K	0.0018	0.0902
64K	0.0037	0.1921

Tabla 9.1: Resultados de las pruebas para las lecturas en memoria a corto plazo

Tamaño del dato a recuperar (Bytes)	**Tiempo mínimo registrado (seg)**	**Tiempo máximo registrado (seg)**
1	2.6000000e-004	9.3000000e-004
2	2.7000000e-004	1.0900000e-003
4	2.6000000e-004	1.5700000e-003
8	2.6000000e-004	8.7000000e-004
16	2.6000000e-004	9.2000000e-004
32	2.6000000e-004	8.6000000e-004
64	2.7000000e-004	1.3100000e-003
128	2.7000000e-004	1.6200000e-003
256	2.8000000e-004	1.2600000e-003
512	2.8000000e-004	9.5000000e-004
1K	2.9000000e-004	8.8000000e-004
2K	3.1000000e-004	1.1000000e-003
4K	3.7000000e-004	1.2600000e-003
8K	5.0000000e-004	1.6100000e-003
16K	8.3000000e-004	1.7300000e-003
32K	1.6700000e-003	3.2600000e-003
64K	3.5200000e-003	6.7300000e-003

Tabla 9.2: Tiempos mínimos y máximos registrados en la lectura de datos de memoria compartida

9.1.2. Escritura en la memoria a corto plazo

La pruebas de colocación de datos en el sistema de memoria compartida se hicieron con datos de 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K 2K, 4K, 8K, 16K, 32K y 64K Bytes. Por cada tamaño se tomaron 10000 muestras, y al igual que el caso de la lectura de datos, se usó para medir el tiempo la primitiva gettimeofday().

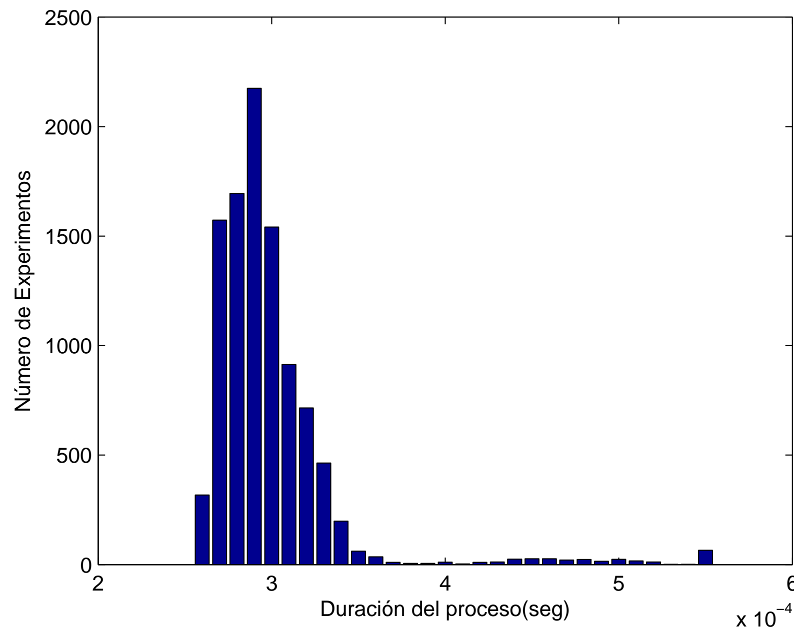

Figura 9.7: Duración del proceso de escritura de datos de tamaño 2 Bytes

El comportamiento de los tiempos para la colocación de información en el sistema de memoria compartida es similar al comportamiento de los tiempos en el proceso de lectura de datos. En las figuras 9.7, 9.8, 9.9 y 9.10 se observan la distribución de los tiempos empleados para colocar datos.

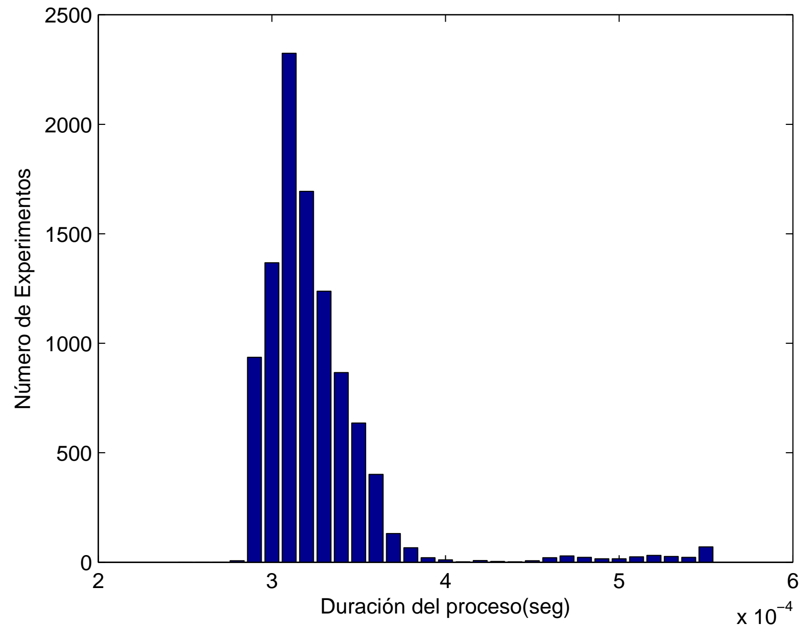

Figura 9.8: Duración del proceso de escritura de datos de tamaño 512 Bytes

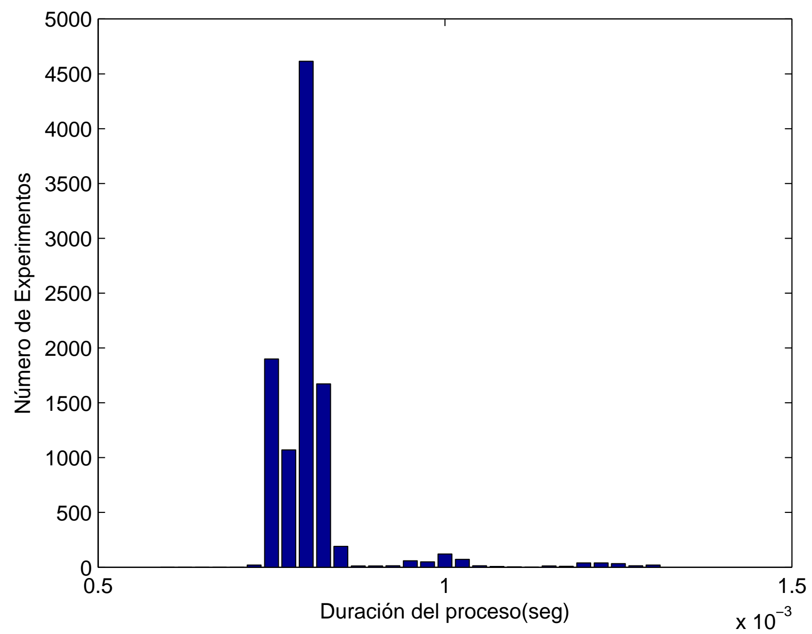

Figura 9.9: Duración del proceso de escritura de datos de tamaño 16 KByte

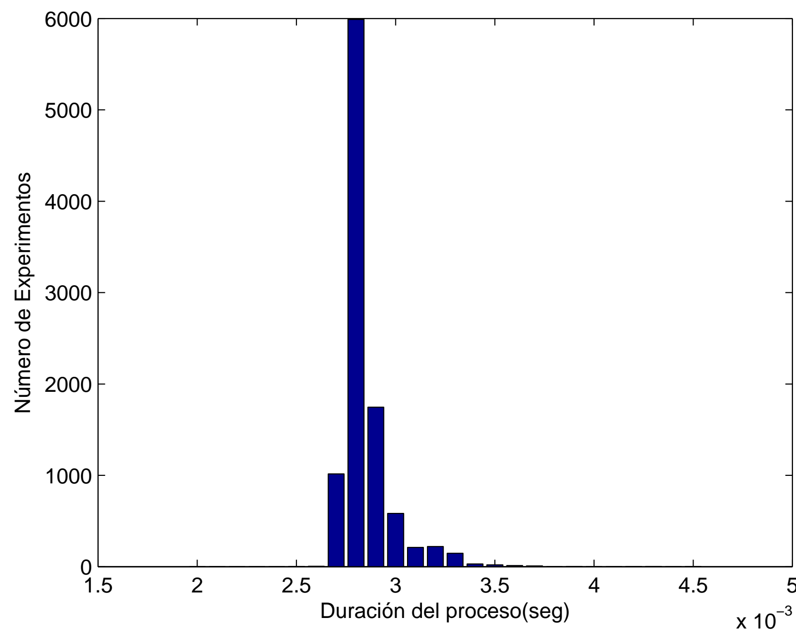

Figura 9.10: Duración del proceso de lectura de datos de tamaño 64 KByte

En la figura 9.11 se observa los tiempos promedio calculados en la muestra. El tamaño de la muestra global para la escritura de datos es de 170 mil pruebas. En este gráfico el eje que representa los diferentes tamaños. Los tamaños mostrados corresponden a 2, 512, 16K y 64K Bytes respectivamente. Estos tamaños han sido mostrados en una escala logarítmica, los tamaños van desde 2^0 hasta 2^{16} Bytes. La información en está gráfica es complementada por medio de la gráfica de la desviación estándar de cada una de las 17 muestras, ver figura 9.12.

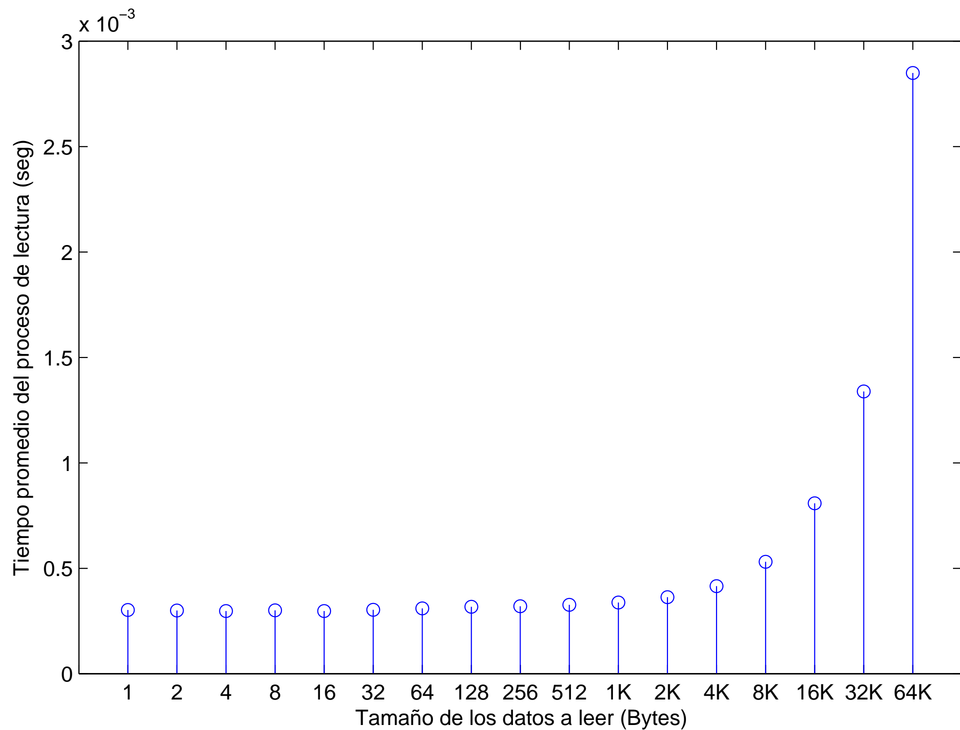

Figura 9.11: Duración promedio del proceso de escritura

La tabla 9.3 muestra el tiempo promedio y la desviación estándar en los experimentos de colocación de información en el sistema de memoria compartida. En esta tabla se observa que para tamaños de datos inferiores a los 4096 Bytes los tiempos medios requeridos para la operación de escritura se encuentran alrededor de los 0.3 mseg, con una desviación estándar por debajo de los 0.1 mseg.

También es importante mostrar los tamaños mínimos y máximos de los datos obtenidos en los experimentos, que son mostrados en la tabla 9.4. En el proceso de escritura de datos por debajo de los 4096 Bytes se observan tiempos máximos en la operación por debajo de los 2 mseg. Se hace énfasis en los tiempos máximos, ya que estos son los que afectan directamente en los fallos por retrasos en la actualización de la información compartida.

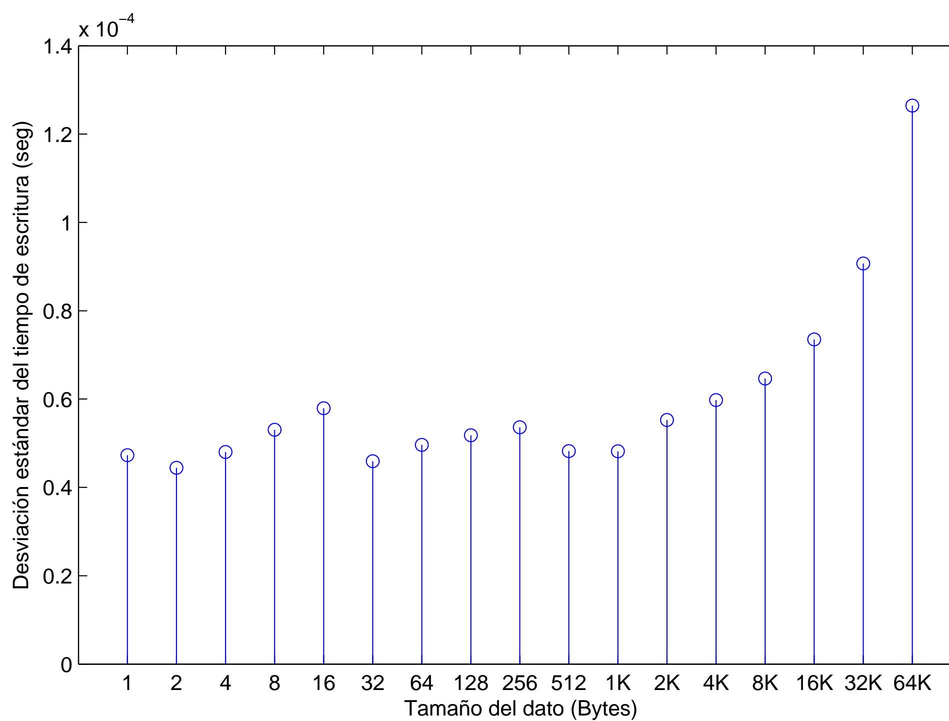

Figura 9.12: Desviación promedio del proceso de escritura

Tamaño del dato a recuperar (Bytes)	**Tiempo medio de escritura (seg)**	**Desviación Estándar de escritura (mseg)**
1	0.0003	0.0473
2	0.0003	0.0444
4	0.0003	0.0480
8	0.0003	0.0531
16	0.0003	0.0579
32	0.0003	0.0459
64	0.0003	0.0496
128	0.0003	0.0518
256	0.0003	0.0536
512	0.0003	0.0482
1024	0.0003	0.0482
2048	0.0004	0.0553
4096	0.0004	0.0598
8192	0.0005	0.0647
16384	0.0008	0.0735
32768	0.0013	0.0907
65536	0.0028	0.1265

Tabla 9.3: Resultados de las pruebas para las escrituras en la memoria a corto plazo

Tamaño del dato a recuperar (Bytes)	**Tiempo mínimo registrado (seg)**	**Tiempo máximo registrado (seg)**
1	2.6000e-004	1.2600e-003
2	2.6000e-004	8.5000e-004
4	2.6000e-004	1.0900e-003
8	2.6000e-004	2.7500e-003
16	2.6000e-004	3.2900e-003
32	2.6000e-004	1.2000e-003
64	2.7000e-004	1.5000e-003
128	2.8000e-004	1.3500e-003
256	2.8000e-004	9.9000e-004
512	2.8000e-004	1.6100e-003
1K	2.9000e-004	1.4100e-003
2K	3.1000e-004	1.4200e-003
4K	3.6000e-004	1.5200e-003
8K	4.7000e-004	1.4000e-003
16K	7.3000e-004	1.6300e-003
34K	1.2600e-003	3.2500e-003
64K	2.5000e-003	5.3500e-003

Tabla 9.4: Tiempos mínimos y máximos registrados en la colocación de datos de memoria compartida

9.2. Sistema de gestión de eventos

El sistema de gestión de eventos es parte del núcleo de la arquitectura implementada. Este es el encargado de garantizar que la ocurrencia de un evento sea notificada a una habilidad encargada de reaccionar ante éste. El diseño del gestor de eventos se basa en el modelo publisher/subscriber [99], lo que permite a una habilidad específica recibir a un evento mientras realiza una o más tareas de forma concurrente.

El sistema desarrollado se comporta de forma asíncrona y totalmente desacoplado. Esto implica que no existe una conexión permanente entre el emisor y el receptor del evento, mas aún, los receptores del evento desconocen directamente el emisor del evento. Los eventos son transmitidos siguiendo el modelos de multidifusión, es decir, un emisor puede enviar un evento a varios receptores que residan dentro del sistema distribuido que conforma la aplicación.

Los datos mostrados en las siguientes subsecciones han sido obtenidos de envíos de eventos diferentes entre un emisor y un receptor.

9.2.1. Transmisión entre pares en el mismo ordenador

El primer escenario de pruebas consiste en enviar eventos entre habilidades que están en ejecución sobre un mismo ordenador. Entre el envío de un evento y el siguiente se espera un tiempo constante por cada prueba, enviándose en cada prueba 1000 eventos. Los tiempos de espera en cada prueba fueron 10, 20, 30, 40, 50, 60, 70, 90 y 100 mseg, para un total de 100.000 eventos enviados en este escenario. A continuación se muestran algunos histogramas con la distribución en el retardo de la detección del evento. Es importante destacar que en las pruebas realizadas en este escenario no se detectó perdida en la detección de los 100000 eventos enviados.

El primer histograma obtenido se observa en la figura 9.13, donde se muestra que más del 80 % de los eventos fue detectado en un tiempo menor de 0,2 mseg.

El comportamiento descrito anteriormente se repite en las figuras 9.14, 9.15. 9.16, 9.17, donde se observa que los tiempos de respuesta para la detección de eventos se mantienen por debajo de los 2 mseg.

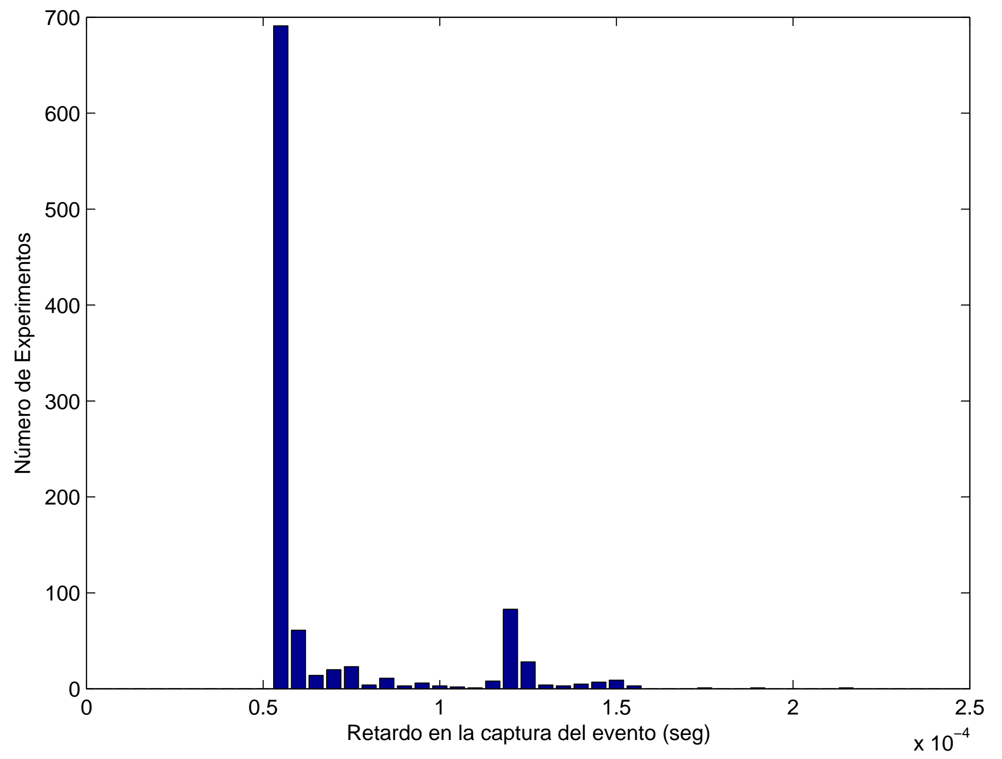

Figura 9.13: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 20 mseg.

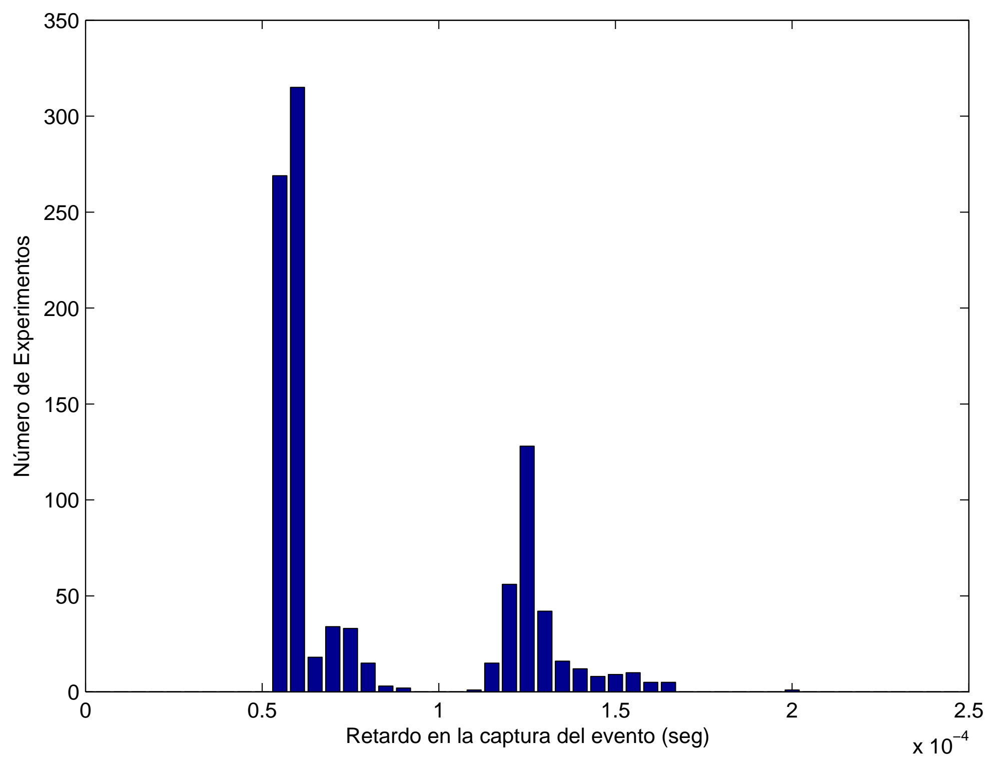

Figura 9.14: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 40 mseg

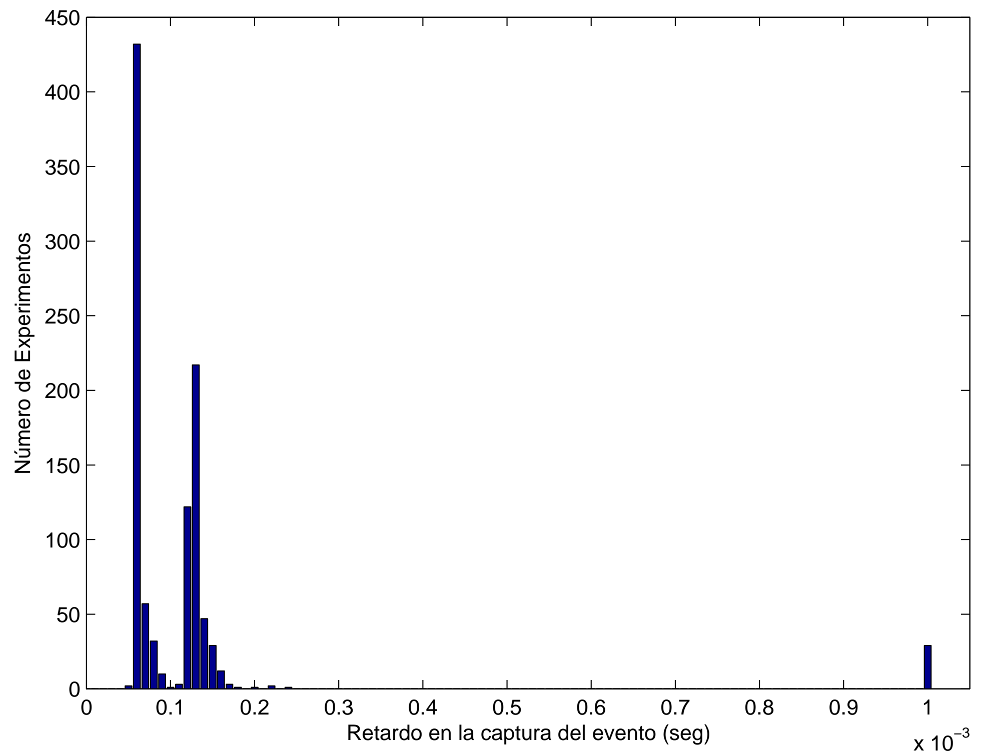

Figura 9.15: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 60 mseg.

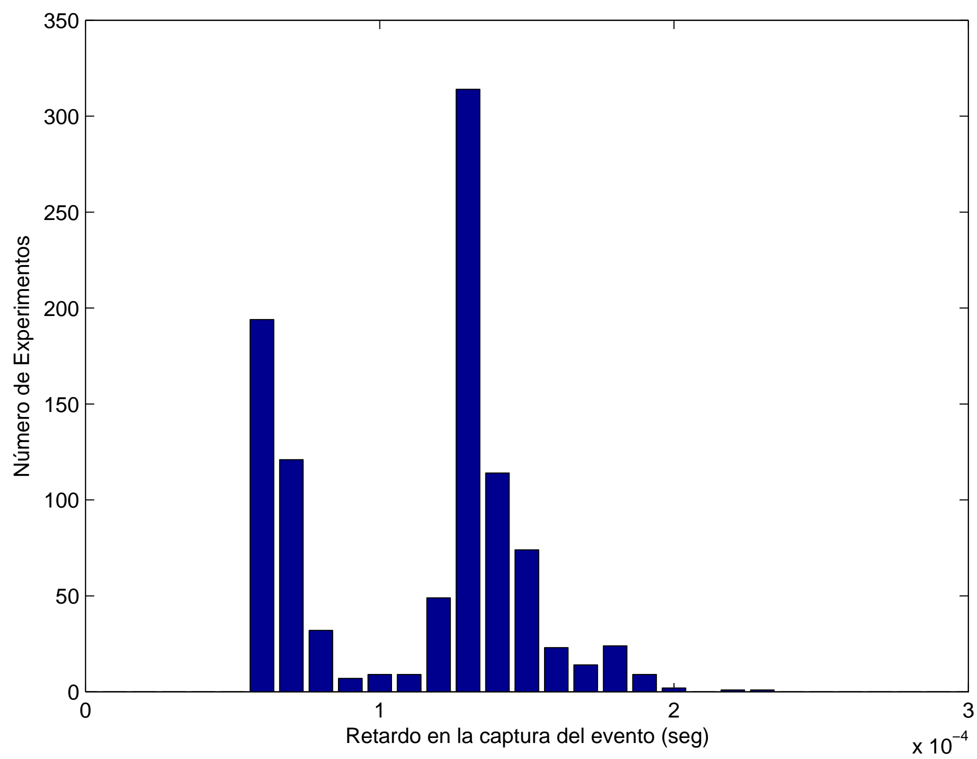

Figura 9.16: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 80 mseg

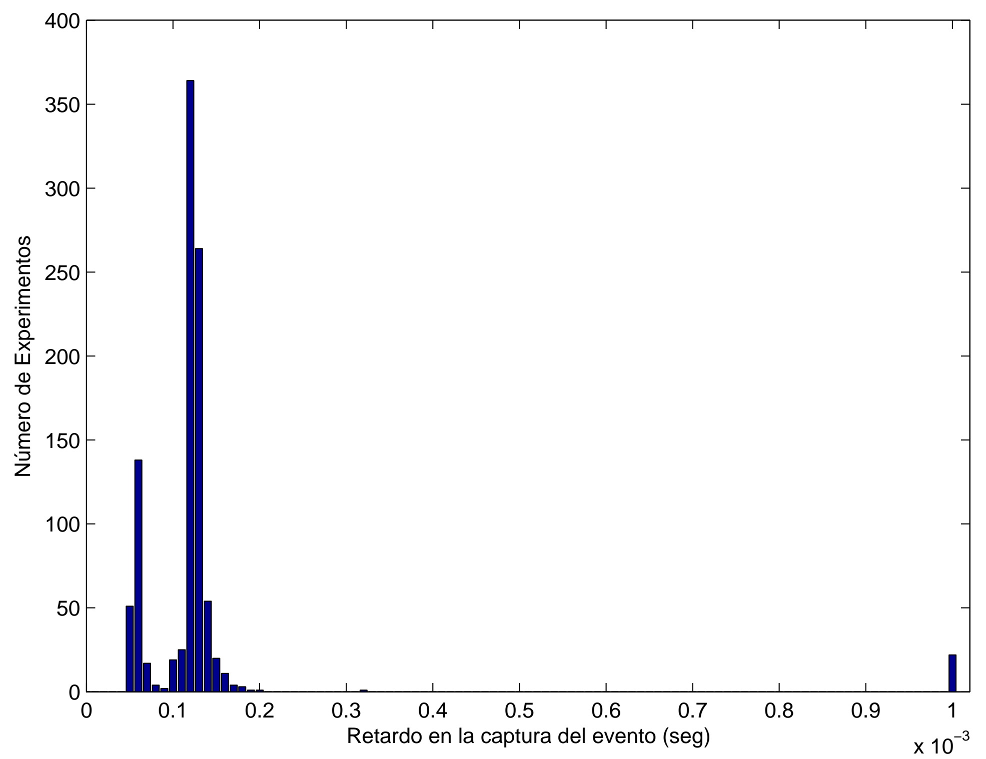

Figura 9.17: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 100 mseg

En la figura 9.18 se muestra el retardo promedio de la prueba. Este se caracteriza por estar por debajo de los 0,4 mseg. La desviación estándar de los promedios se muestra en la figura 9.19, que describe que la distancia de los valores de retardo está por debajo de los 1,5 mseg.

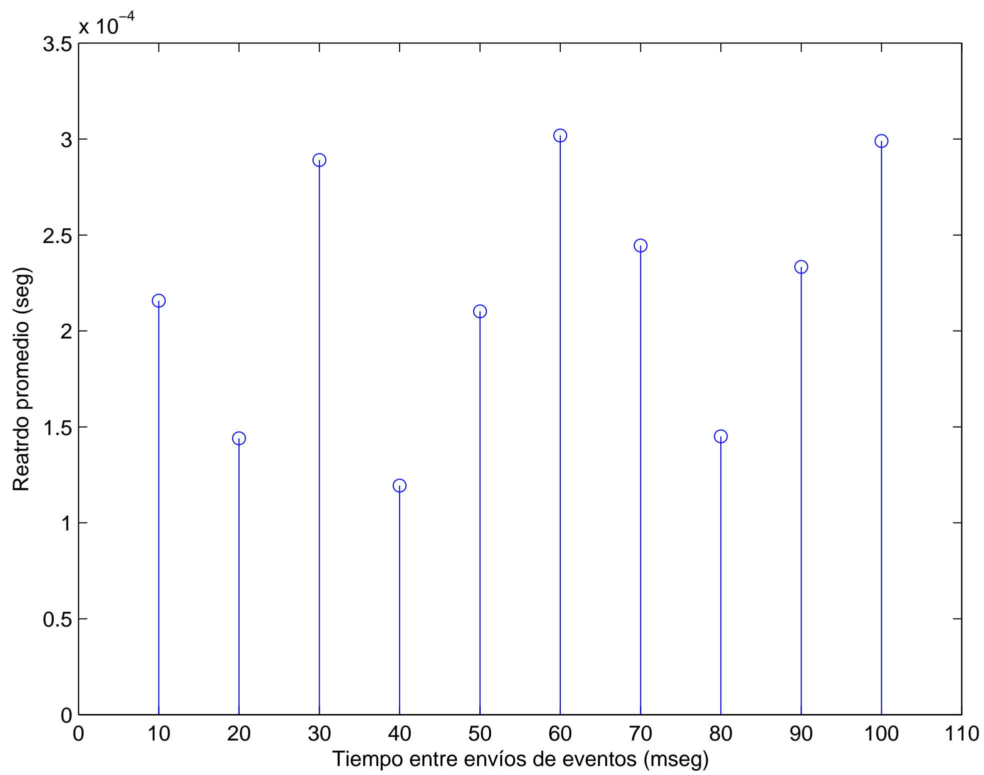

Figura 9.18: Comportamiento del retardo promedio en la detección de eventos entre pares en el mismo ordenador

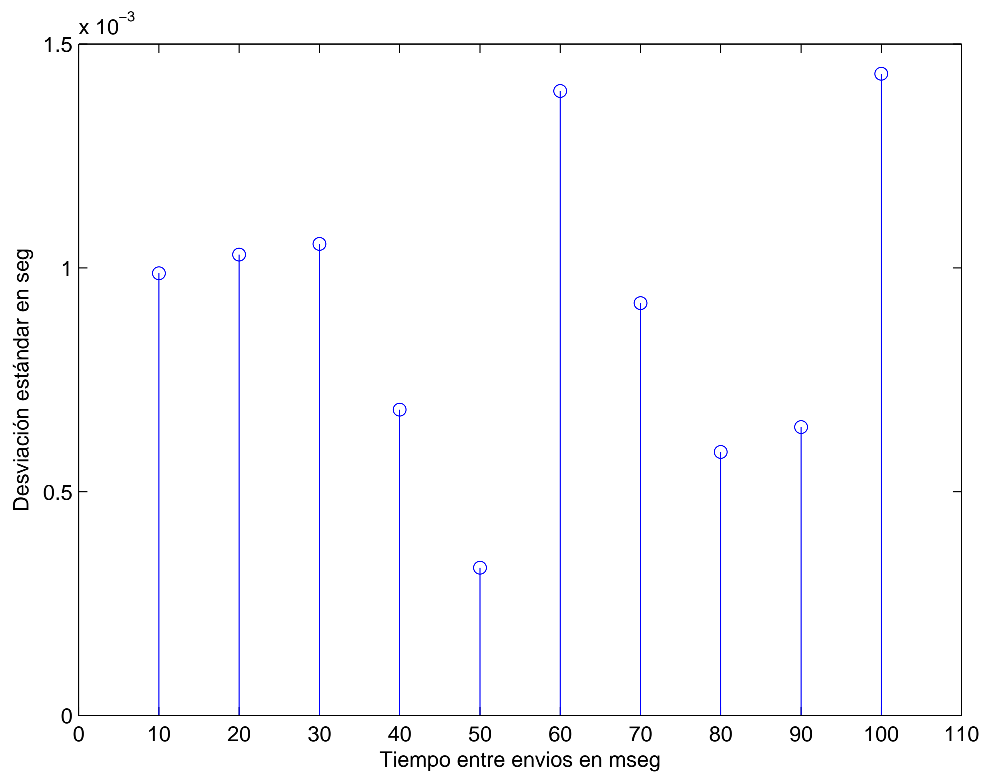

Figura 9.19: Desviación estándar del retardo promedio en detección de eventos entre pares en el mismo ordenador

En la tabla 9.5 se muestra el resumen de los datos obtenidos de las muestras en este escenario de detección local. Donde se muestra en tiempo promedio y su desviación estándar con los valores obtenidos, destacando que los tiempos promedios se encontraron siempre por debajo de los 2 mseg, concentrados alrededor de los 0,3 mseg.

Tiempo entre envíos (mseg)	**Tiempo medio de captura (seg)**	**Desviación Estándar de captura (seg)**
10	2.1578e-004	9.8807e-004
20	1.4409e-004	1.0295e-003
30	2.8903e-004	1.0533e-003
40	1.1937e-004	6.8338e-004
50	2.1018e-004	3.3021e-004
60	3.0189e-004	1.3950e-003
70	2.4453e-004	9.2129e-004
80	1.4502e-004	5.8861e-004
90	2.3340e-004	6.4449e-004
100	2.9894e-004	1.4333e-003

Tabla 9.5: Resultados de las pruebas para el sistema de eventos detectados entre pares en un mismo ordenador

La tabla 9.6 muestra los datos de los valores mínimos y máximos obtenidos en las 100000 muestras. Estos valores son importantes de mostrar, ya que son valores que aparecen extrañamente pero permiten tomar medidas para casos de retardos no habituales. El retraso máximo registrado estuvo por debajo de los 18 mseg.

Tiempo entre envíos (mseg)	**Tiempo mínimo registrado (seg)**	**Tiempo máximo registrado (seg)**
10	4.6000e-005	9.1530e-003
20	5.3000e-005	1.7255e-002
30	4.9000e-005	9.0790e-003
40	5.5000e-005	1.7954e-002
50	5.7000e-005	1.7003e-002
60	5.5000e-005	1.7180e-002
70	5.6000e-005	9.0470e-003
80	6.0000e-005	1.5525e-002
90	5.1000e-005	1.6936e-002
100	5.2000e-005	1.6942e-002

Tabla 9.6: Tiempos mínimos y máximos registrados en la detección de eventos entre pares en un mismo ordenador

9.2.2. Entre pares en ordenadores interconectados por la red interna del robot

Como se indicó al inicio de este capítulo, la transmisión en este escenario se hace usando los servicios provistos por una red Ethernet (IEEE 802.3) ubicada dentro del robot. Cada ordenador dentro del robot usa un dispositivo Ethernet integrado, que se conectan por medio de un cable de par trenzado de categoría 5[117] a un conmutador Ethernet a una velocidad de 100 Mbps. Hasta ahora el numero máximo de dispositivos Ethernet en el robot ha sido de 3 ordenadores y un enlace WIFI (IEEE 802.11). La comunicación se hace entre los pares que intercambian eventos entre procesos residentes en los ordenadores internos del robot. A continuación se muestran varias gráficas que resumen el comportamiento de las diferentes pruebas realizadas.

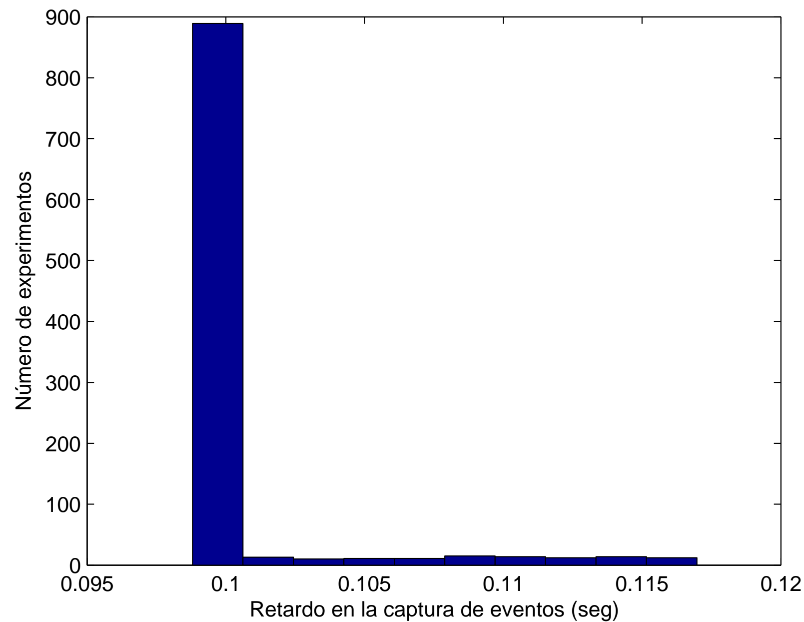

Figura 9.20: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 200 mseg

En la gráfica mostrada en la figura 9.20 se muestra el comportamiento de 1000 eventos emitidos a intervalos de 0.2 seg. Se observa como el 85 % de los eventos emitidos son detectados en 0.1 seg, y el mayor retardo es menor que 0.120 seg. Este comportamiento se repite en las gráficas 9.21, 9.22, 9.23.

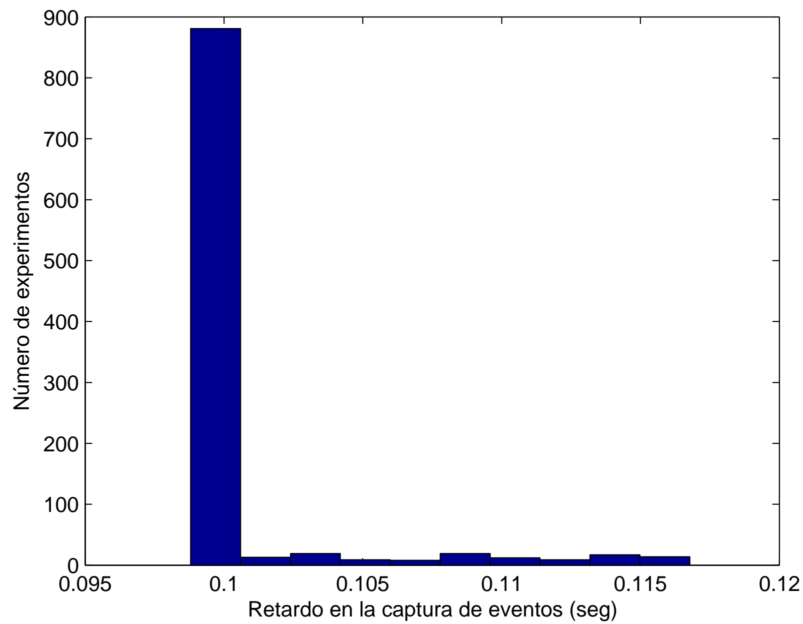

Figura 9.21: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 250 mseg

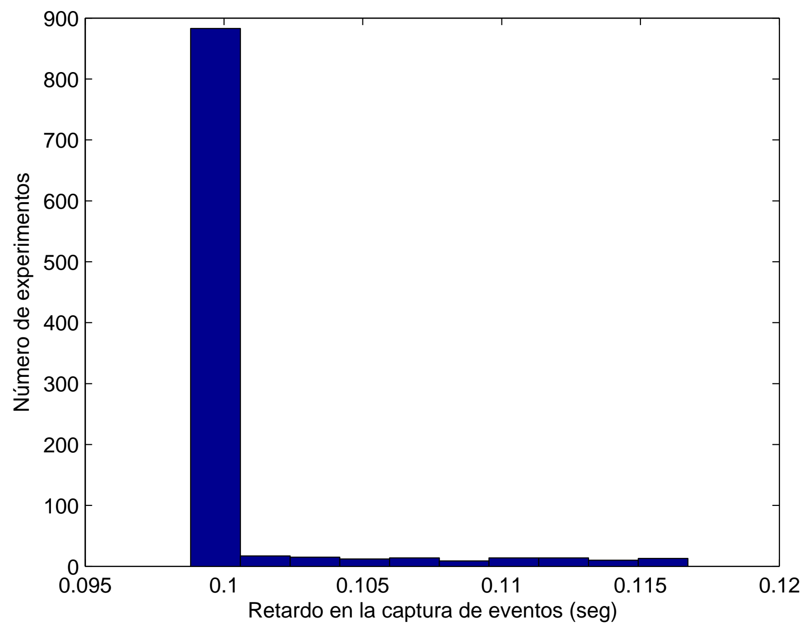

Figura 9.22: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 350 mseg

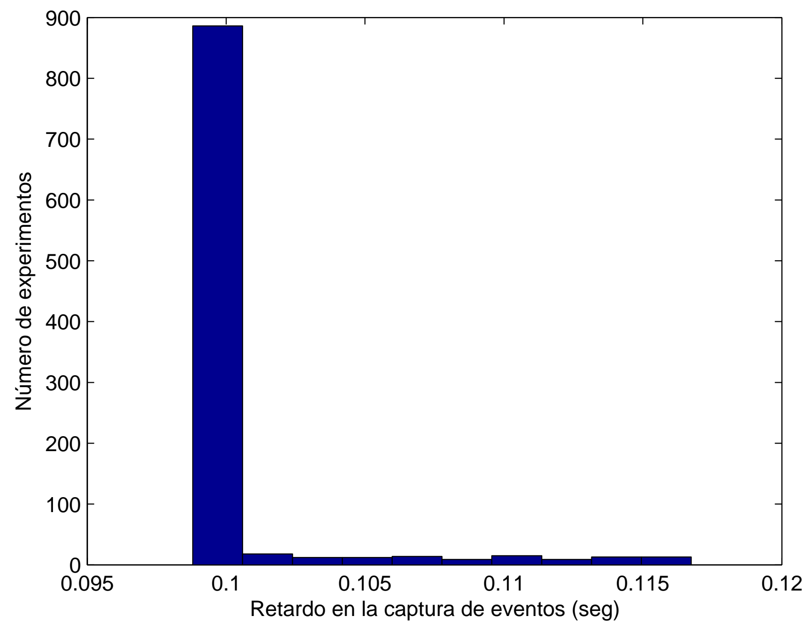

Figura 9.23: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 450 mseg

En la gráfica 9.24 se muestra el comportamiento del retardo promedio en la detección de eventos en el experimento realizado. Como se indicó previamente, cada experimento constó de la emisión de 1000 eventos. El proceso emisor está ubicado en un ordenador diferente al proceso encargado de la detección, aunque dentro de la LAN interna del Robot.

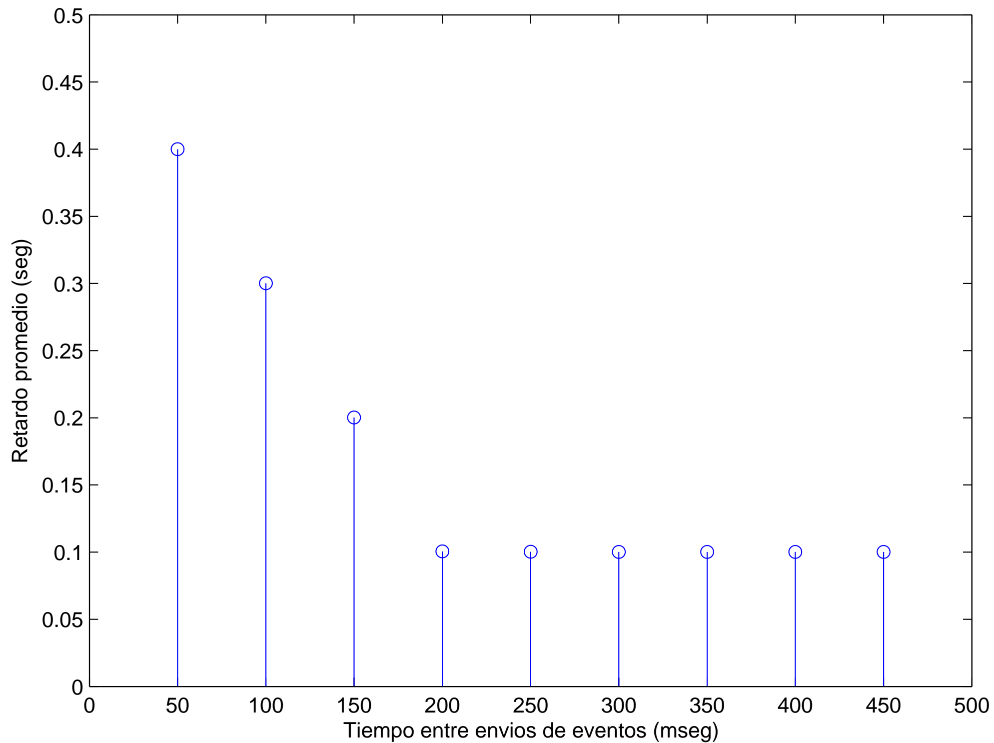

Figura 9.24: Comportamiento del retardo promedio en la detección de eventos entre pares en ordenadores conectados a la misma LAN

En la grafica 9.25 se observa la desviación estándar de retardo. En este caso los valores obtenidos se mantienen por debajo de los 3.5 mseg lo que indica que la concentración de los datos en el entorno de los promedios es alta.

La tabla 9.8 muestra los valores máximos y mínimos obtenidos en el experimento, que constó de 9000 eventos emitidos. Se observa claramente como en intervalos de emisión inferior a los 200 mseg los tiempos máximos de retardo aumentan. En cambio en periodos de emisión superior a los 200 mseg los retardos permanecen en un entorno cercano a los 110 mseg.

La transmisión de eventos entre ordenadores está a cargo de los servidores principal y secundario del sistema de emisión de eventos distribuidos, ya que los gestores en los procesos internos o habilidades sólo interactúan entre su servidor local, usando servicios basados en IPC provisto por el núcleo del sistema operativo. Al tener que enviar los eventos entre ordenadores, se usan servicios

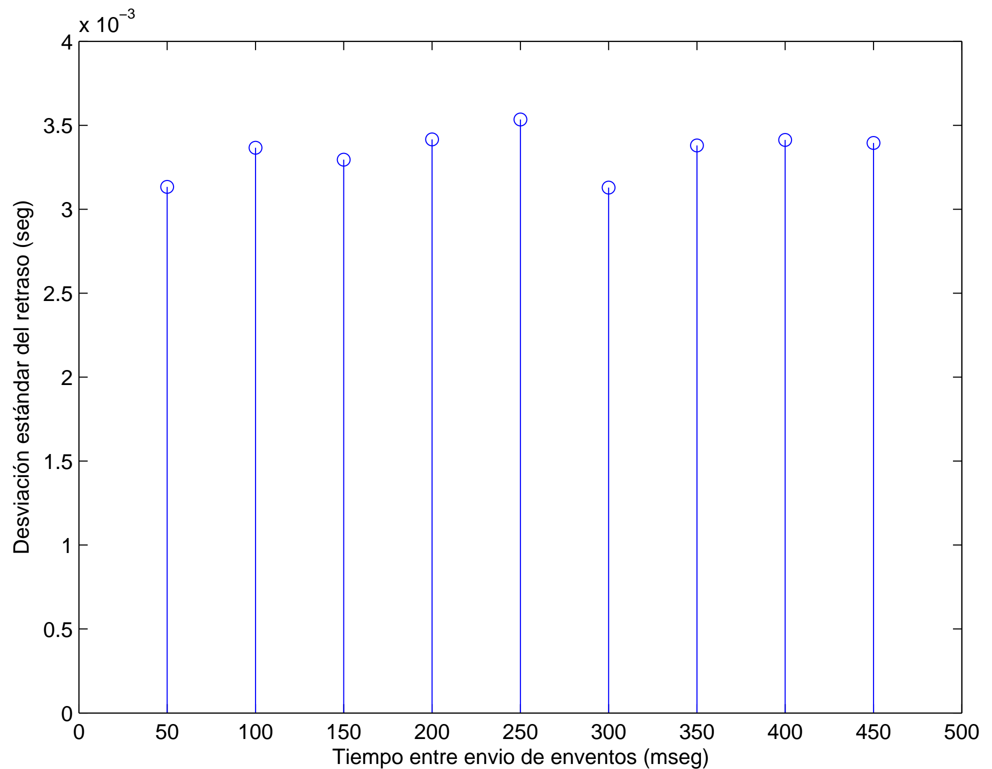

Figura 9.25: Desviación estándar del retardo promedio en detección de eventos entre pares en ordenadores conectados a la misma LAN

,

UDP del protocolo, luego IP y por último a los servicios provistos por Ethernet. Se indica esto para hacer énfasis en la diferencia con el caso anterior, en que sólo se hacia movimiento de datos dentro de un solo ordenador, sin necesidad de agregar tiempo al retardo requerido por señalización al medio y transmisión del dato a través de la LAN.

En vista de los datos obtenidos, se puede afirmar que en los casos donde los intervalos de emisión de eventos son superiores a los 200 mseg permite que eventos sean detectados por el sistema. En cambio, al emitir eventos por debajo de los 200 mseg para ser detectados entre procesos en diferentes ordenadores los objetos pueden presentar retardos de hasta los 450 mseg. Es importante destacar que esta prueba fue realizada dentro de ordenadores colocados en la LAN interna del robot, y sobre operaciones normales del robot. Indudablemente que el tráfico en la red Ethernet interna del robot tiene características particulares que no tienen el resto de ordenadores de RoboticsLab, como es el tráfico por transmisión de correo electrónico, navegadores Web, Sistema de Archivos en Red (NFS) y Sistema de información de páginas amarillas (NIS), entre otros.

Tiempo entre envíos (seg)	**Tiempo medio de captura (seg)**	**Desviación Estándar de captura (seg)**
50	3.9999e-001	3.1330e-003
100	3.0015e-001	3.3664e-003
150	2.0017e-001	3.2947e-003
200	1.0055e-001	3.4155e-003
250	1.0030e-001	3.5343e-003
300	1.0014e-001	3.1290e-003
350	1.0019e-001	3.3799e-003
400	1.0017e-001	3.4123e-003
450	1.0017e-001	3.3952e-003

Tabla 9.7: Resultados de las pruebas para el sistema de eventos entre pares en ordenadores en la misma LAN

Tiempo entre envíos (mseg)	**Tiempo mínimo registrado (seg)**	**Tiempo máximo registrado (seg)**
50	3.9880e-001	4.1477e-001
100	2.9880e-001	3.1666e-001
150	1.9880e-001	2.1651e-001
200	9.8800e-002	1.1698e-001
250	9.8800e-002	1.1677e-001
300	9.8800e-002	1.1523e-001
350	9.8800e-002	1.1671e-001
400	9.8800e-002	1.1670e-001
450	9.8800e-002	1.1674e-001

Tabla 9.8: Tiempos mínimos y máximos registrados en la detección de eventos entre pares en la misma LAN

9.2.3. Entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WIFI

En este tercer escenario se muestran los resultados de los experimentos que se caracterizan por la detección de eventos entre un proceso/habilidad, que reside en un ordenador dentro del robot, y un proceso/habilidad que reside en un ordenador fuera del robot. La red local interna del robot posee una interfaz para la conexión a redes externas y de allí a Internet. El mecanismo de comunicación en la capa de Enlace y Física de Maggie para conexión al exterior es realizado por medio del protocolo IEEE 802.11 [118], específicamente, el robot cuenta con un puente que implementa el estándar WiFi G. Éste utiliza la banda de 2.4 Ghz (al igual que el estándar 802.11b), pero opera a una velocidad teórica máxima de 54 Mbps, que en promedio es de 22.0 Mbps de velocidad real de transferencia, similar a la del estándar 802.11a. Es compatible con el estándar b y utiliza las mismas frecuencias. Con WiFi se pueden crear redes de área local inalámbricas de alta velocidad siempre y cuando el equipo que se vaya a conectar no esté muy alejado del punto de acceso. En la práctica, WiFi admite ordenadores portátiles, equipos de escritorio, asistentes digitales personales (PDA) o cualquier otro tipo de dispositivo de alta velocidad con propiedades de conexión también de alta velocidad (11 Mbps o superior) dentro de un radio de varias decenas de metros en entornos cerrados (de 20 a 50 metros en general) o dentro de un radio de cientos de metros al aire libre.

En la figura 9.26 se muestra un histograma con los retardos obtenidos al emitir eventos cada 300 mseg. En este caso la detección de los eventos tiene un retardo donde de los 1000 eventos cerca de 850 se encuentran entre los 122 mseg y los 125 mseg. Este comportamiento se mantiene al aumentar el tiempo entre la emisión de eventos, tal y como puede observarse en las figuras 9.27, 9.28, 9.29 y 9.30.

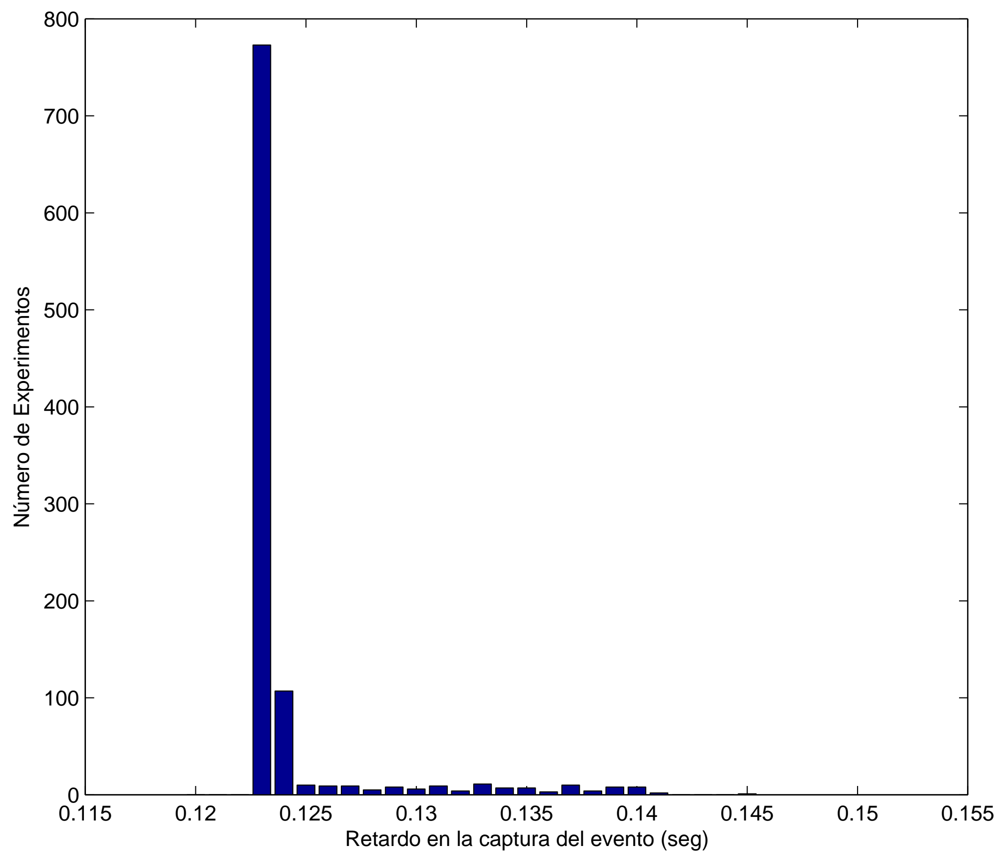

Figura 9.26: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 300 mseg

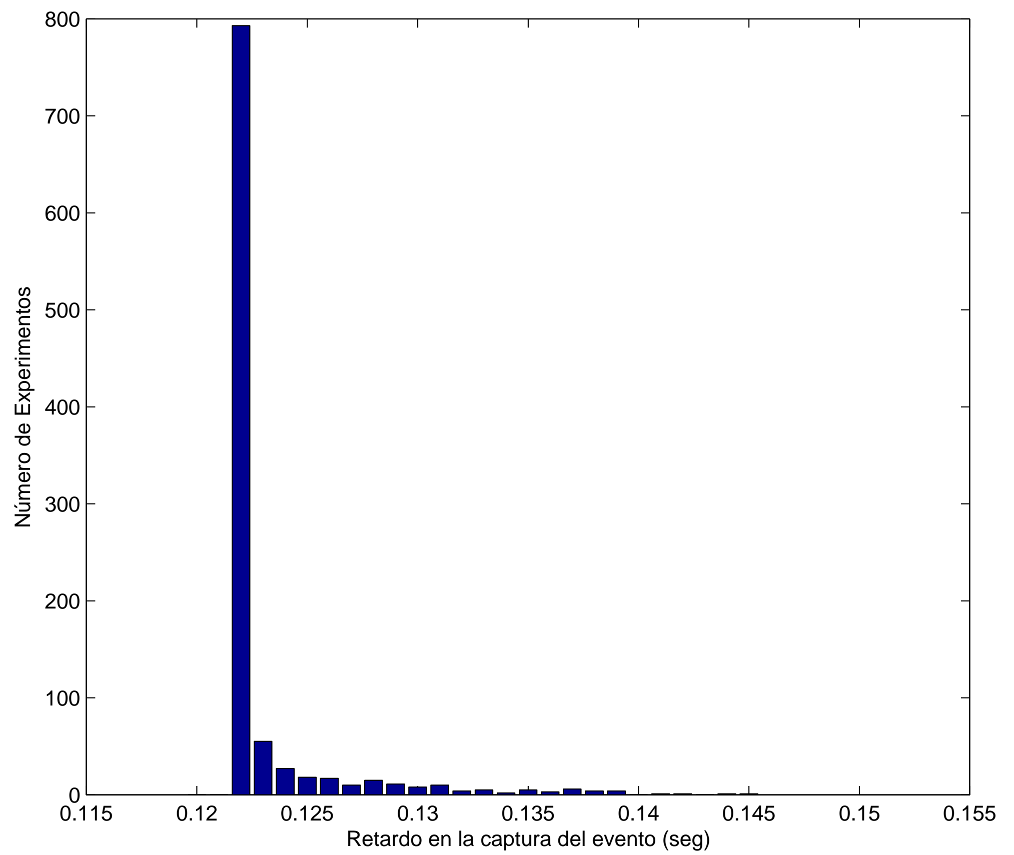

Figura 9.27: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 350 mseg

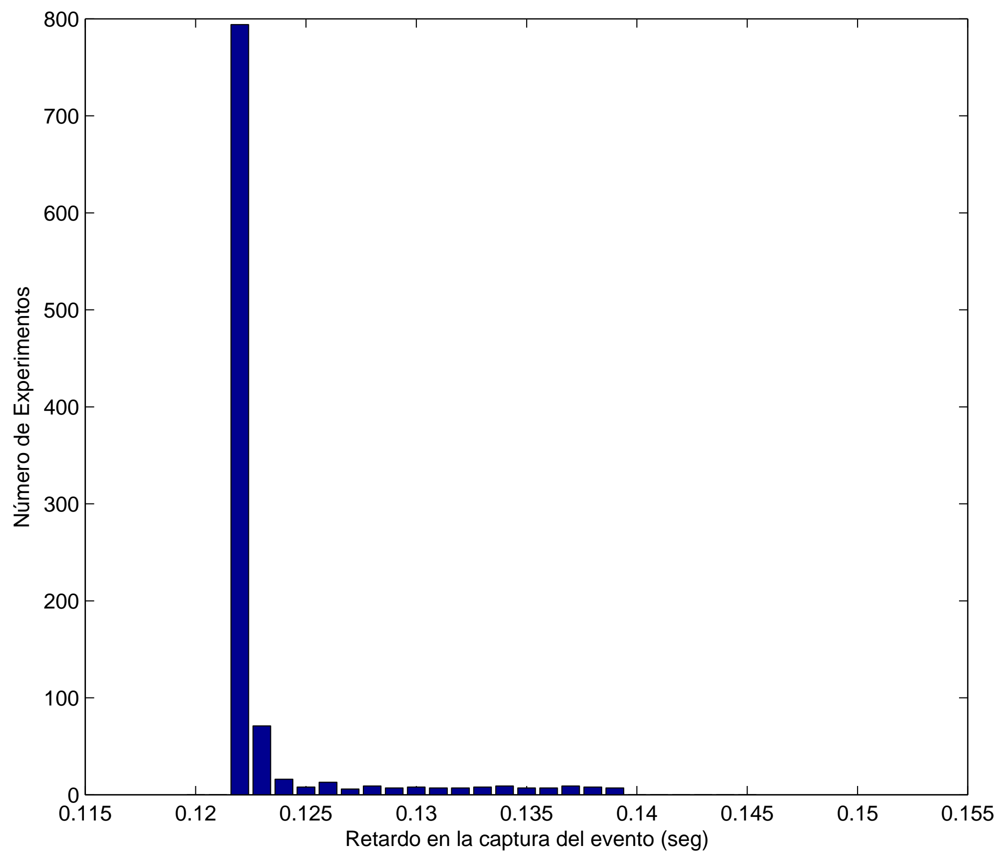

Figura 9.28: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 400 mseg

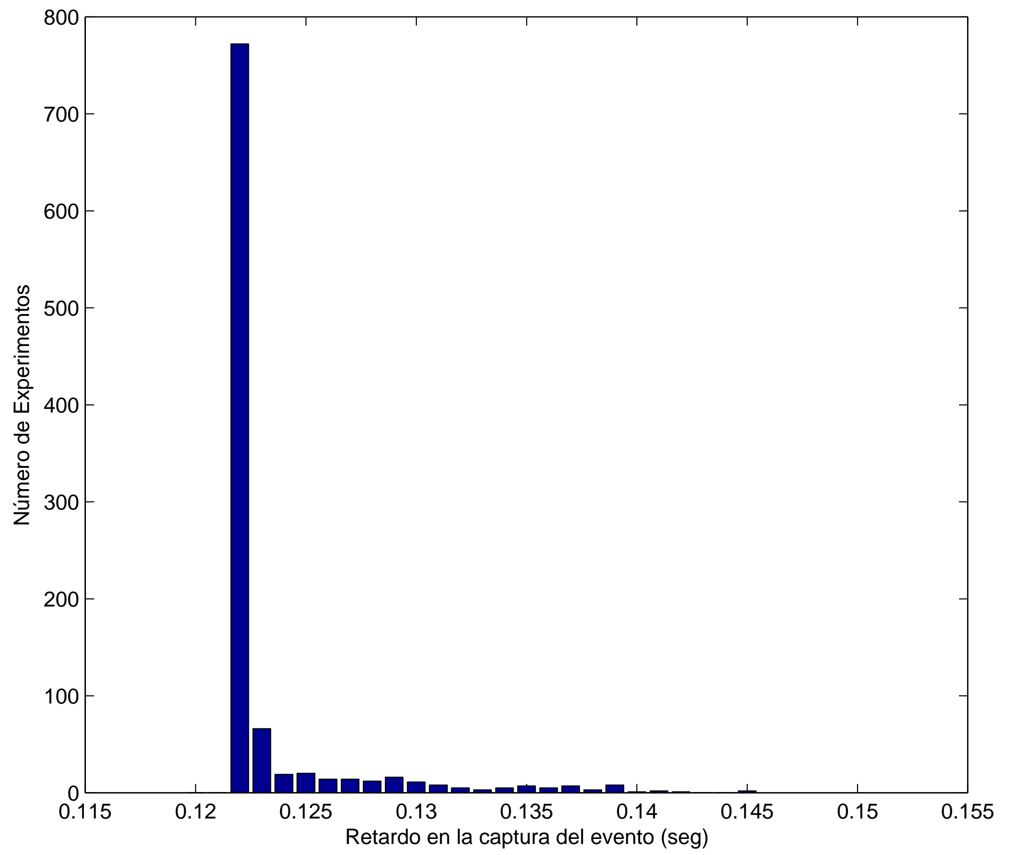

Figura 9.29: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 450 mseg

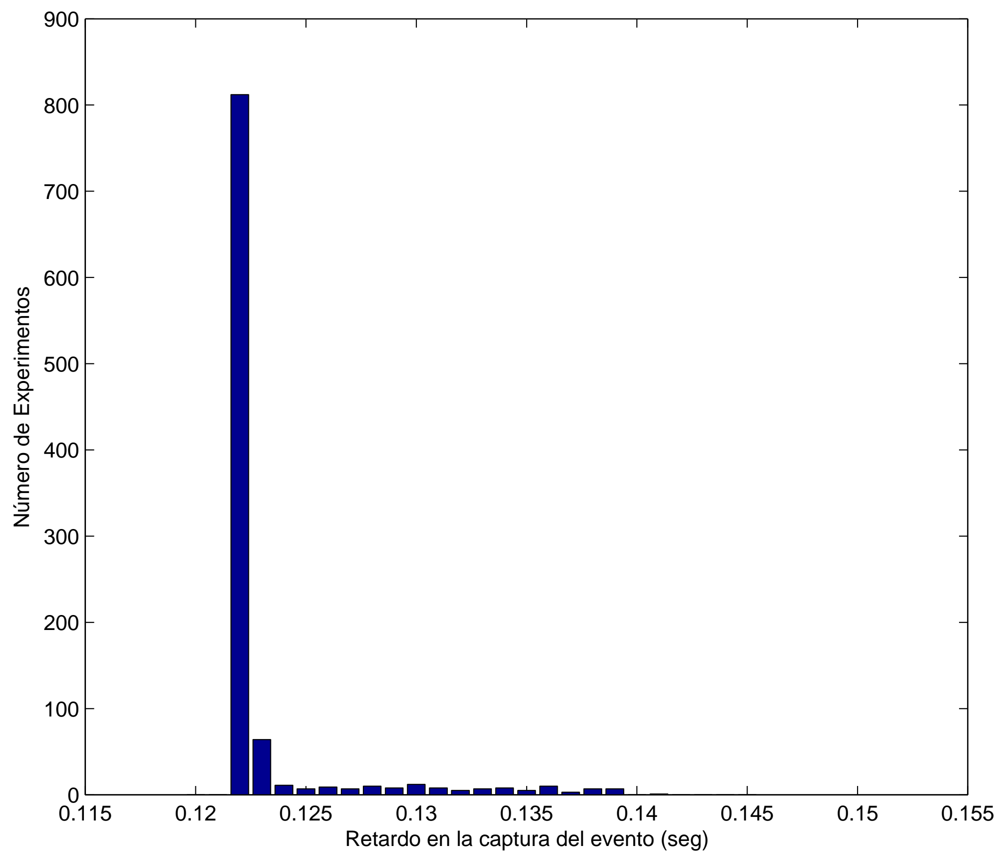

Figura 9.30: Histograma de la distribución del retardo en la detección de eventos enviados con un periodo de 500 mseg

En la figura 9.31 se muestra el retardo promedio de las muestras obtenidas en este escenario. Como se puede observar el retardo promedio en la detección de eventos se mantienen cercano a los 125 mseg al aumentar el tiempo entre emisión de eventos con un valor superior a los 300 mseg. En la figura 9.32 se observa la desviación estándar del retardo con respecto a su promedio, ésta se mantiene cerca de los 3.5 mseg lo que resume la fuerte concentración de los valores de retardos cerca de los puntos promedio de cada muestra.

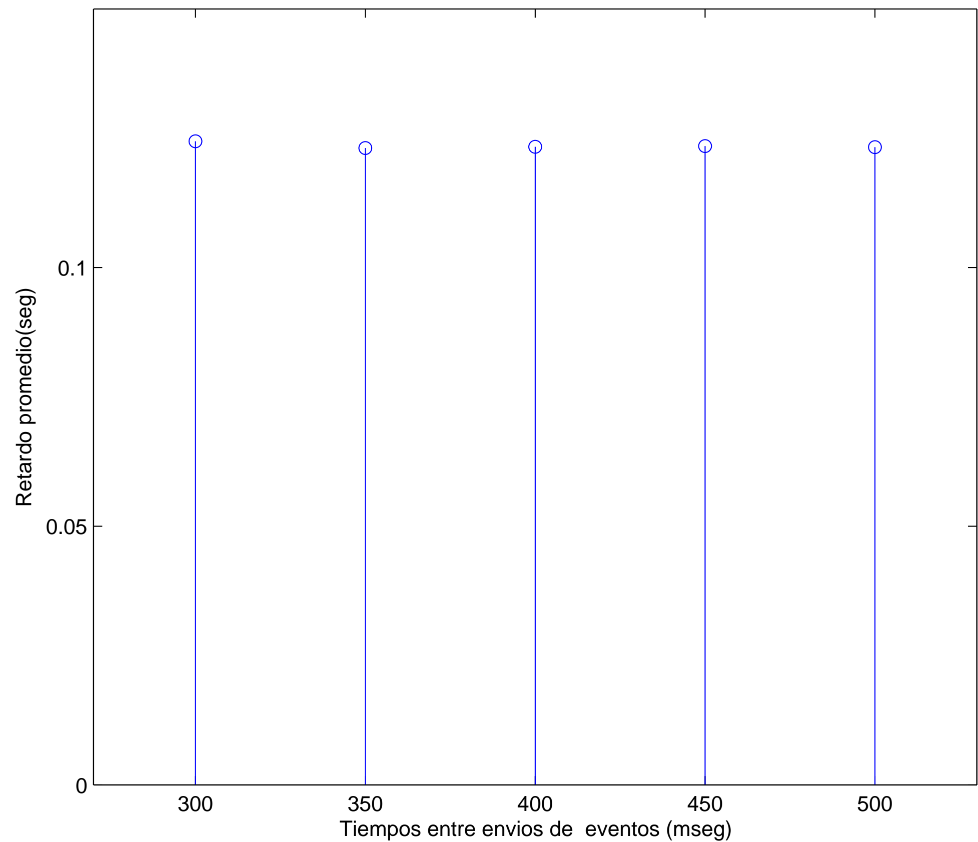

Figura 9.31: Retardo promedio en la detección de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WIFI

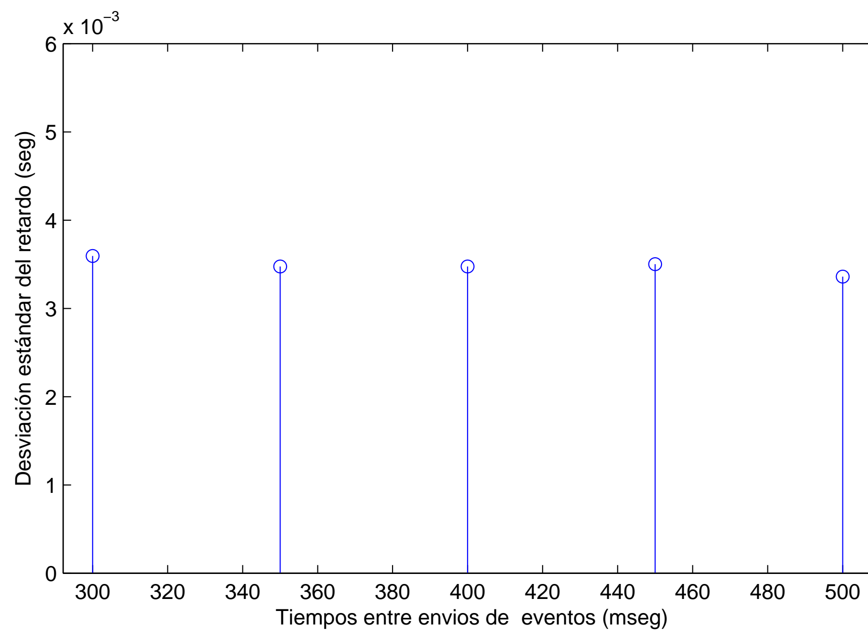

Figura 9.32: Desviación estándar en el retardo de la detección de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WIFI

Tiempo entre envíos (seg)	**Tiempo medio de captura (seg)**	**Desviación Estándar de captura (seg)**
300	0.12442	3.5935e-003
350	0.12308	3.4747e-003
400	0.12334	3.4741e-003
450	0.12349	3.5015e-003
500	0.12326	3.3600e-003

Tabla 9.9: Resultados de las pruebas para el sistema de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WiFi

Tiempo entre envíos (mseg)	**Tiempo mínimo registrado (seg)**	**Tiempo máximo registrado (seg)**
300	9.8988e-002	1.3836e-001
350	9.9850e-002	1.4958e-001
400	9.8914e-002	1.1618e-001
450	9.8942e-002	1.2217e-001
500	9.9010e-002	1.1887e-001

Tabla 9.10: Tiempos mínimos y máximos registrados en la detección de eventos entre un proceso en un ordenador dentro del robot y un proceso en un ordenador externo conectado a través de la red WiFi

En la tabla 9.9 se muestran los valores exactos de los promedios y la desviciación estándar de los retardos obtenidos en los experimentos, en este caso se tomaron 5000 emisiones de eventos a intervalos de tiempos fijados con anterioridad.

En la tabla 9.10 se muestran los retardos mínimos y máximos obtenidos en los experimentos. Es importante destacar que la desviación estándar mide la distancia promedio de los retardos, a pesar que ésta se mantiene en promedio cerca de los 3.5 mseg, no es cero, por lo que es común que aparezcan datos fuera de banda, como los presentados en esta tabla. En este caso los valores de retardos mínimos están por encima de los 90 mseg, en otras palabras no es posible con los equipos en el robot superar esta barrera, ya que en la transmisión inalámbrica requiere un tiempo para establecer la potencia de la señal, y el tiempo de señalización del medio es considerablemente mayor que el requerido en transmisión sobre el cable de par trenzado, como el usado en el segundo escenario de este experimento.

9.2.4. Comparación entre los tres escenarios de prueba

En esta sección se analizan los tres escenario que se usó para la realización de los experimentos. Se puede resumir que el objetivo de las prueba era observar el retardo en la detección de eventos, para esto se emitieron grupos de 1000 eventos a diferentes intervalos de tiempo. En el primer escenario el mecanismo de comunicación se baso en IPC puro, gestionado a partir de los servicios de núcleo del sistema operativo y bibliotecas de funciones POSIX. En el segundo escenario tuvo lugar en la LAN interna del robot, que es implementada por medio de una red Ethernet a 100 Mbps. El tercer escenario se hace en uno de los entornos de mayor crecimiento en el área de las telecomunicaciones como son las redes WiFi.

En la figura 9.33 se observa los promedios globales para cada uno de los escenarios. Se observa claramente cómo el retardo es mínimo cuando la emisión y detección de eventos se realiza dentro de un ordenador en el robot. La razón fundamental de esta situación radica en que en este escenario el intercambio de datos se realizan sobre una misma placa base de un ordenador. La diferencia en el promedio entre la transmisión usando la red Ethertnet 802.3 y la red WiFi 802.11, se encuentra por el orden de los 5 mseg, siendo el promedio de la red cableada 100.27 mseg, como se observa en la tabla 9.11. Se debe aclarar que estos son promedios calculados a partir de los promedios de los experimentos realizados. En la figura 9.34 se muestra la desviación estándar entre los tres escenarios de pruebas, manteniéndose siempre por debajo de los 3.5 mseg.

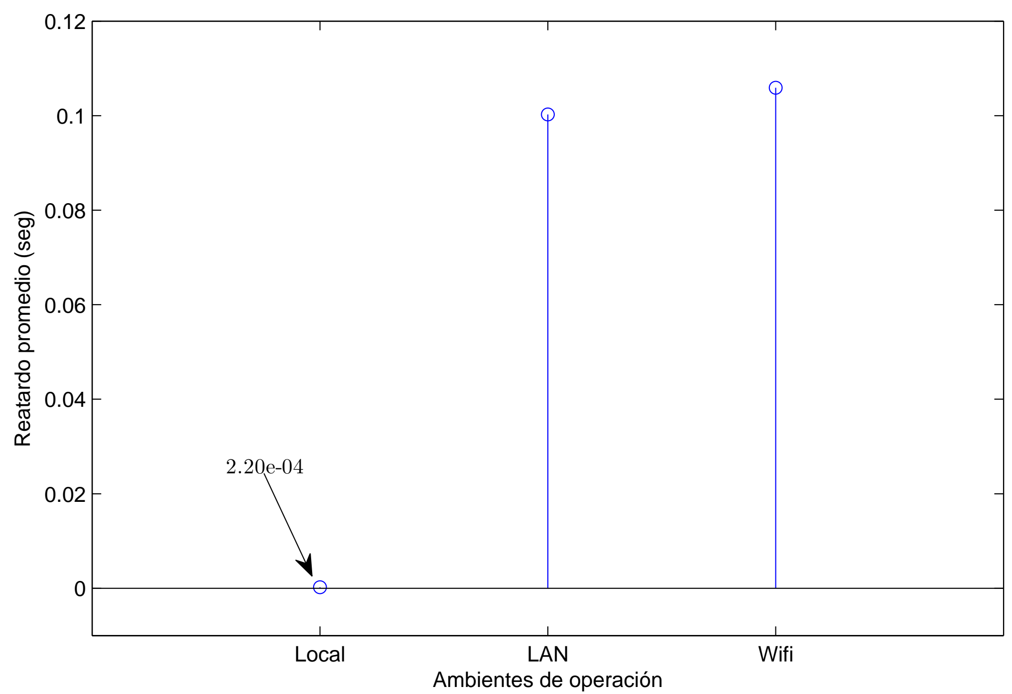

Figura 9.33: Retardo promedio en la detección de eventos en los tres escenarios de prueba

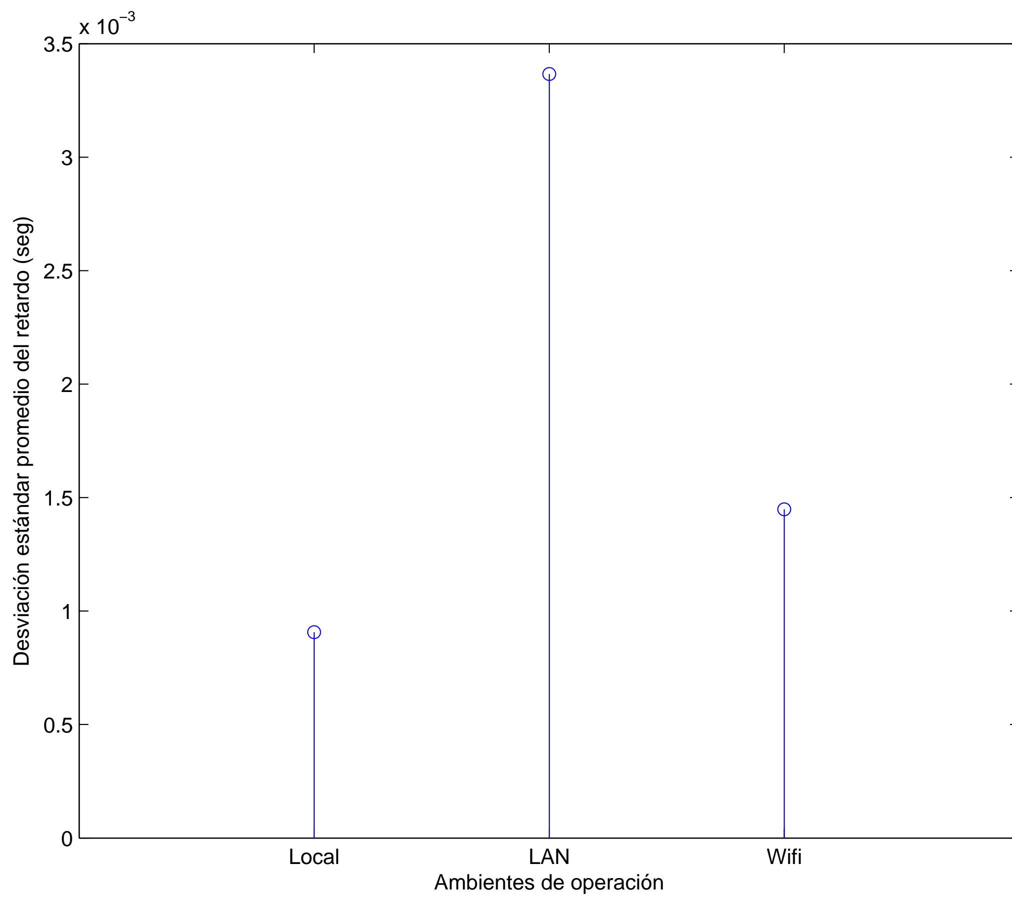

Figura 9.34: Desviación estándar entre los tres escenarios de prueba

Escenarios de operación	**Tiempo medio de captura (seg)**	**Desviación Estándar de captura (seg)**
Procesos en el mismo ordenador	2.2022e-004	9.0673e-004
Procesos en ordenadores en el robot	0.10027	3.3666e-003
Entre un proceso en un ordenador en el robot y un proceso en un ordenador fuera del robot	0.10589	1.4486e-003

Tabla 9.11: Resultados de las pruebas para los tres tipos de escenarios de operación

Para finalizar este capítulo, después de analizar más de 100000 datos tomados de igual número de pruebas, se puede decir que la detección de eventos se garantiza en emisiones superiores a los 50 mseg entre habilidades en un mismo ordenador. Esto implica, que si se trabaja usando emisiones con esa cadencia se deben optimizar las rutinas de servicios de los eventos, con el fin de garantizar un tamaño de cola mínimo, preferiblemente igual a cero, en los buzones de eventos. Si esto último no se cumple, se puede llegar a un crecimiento en la cola de eventos a ser atendidos, perdiendo la validez de los eventos no atendidos pertinentemente.

En la detección de eventos sobre ordenadores en la red local del robot, se recomienda la emisión de eventos con un intervalo de envío no menor a los 200 mseg entre eventos. Los subprogramas de servicios deben garantizar que todos los eventos sean atendidos, pero estas rutinas pueden tener una duración mayor que las del primer escenario.

En el escenario tres, donde el medio de comunicación usa la red WiFi, el tiempo entre emisión de eventos debe ser mayor de los 300 mseg., ya que el tiempo requerido para mover la información por la red inalámbrica y luego por la red Ethernet es mayor que en los dos escenarios anteriores.

En resumen, en el escenario interno de un ordenador se puede emitir eventos a una cadencia cercana a los 50 mseg., pero debe existir el compromiso para que las rutinas de servicio de cada evento sean lo suficientemente eficientes para garantizar el tratamiento del siguiente evento. En el escenario que involucra la red LAN interna del robot los eventos deben ser emitidos a intervalos superiores a los 200 mseg. En este caso el compromiso del tiempo de ejecución debe estar por debajo de los 200 mseg. En el tercer escenario se debe emitir eventos sobre los 300 mseg y los compromisos de tiempo de los servicios a dichos eventos deben ser inferiores a los 300 mseg.

La solución a estos tipos de escenarios conlleva a la necesidad de estudiar un modelo jerárquico de eventos, donde se identifiquen cuáles eventos deben transcender a los diferentes escenarios de trabajo. Este último punto debe ser considerado en trabajos futuros.

10

Conclusiones

El ser humano ha dedicado esfuerzos en diseñar y construir dispositivos que ayuden a realizar tareas de diversas índole. Originalmente se basaba en conocimientos de carpintería y/o mecánica para tal fin. Con el nacimiento de la electrónica, y como consecuencia de ésta, el ordenador, la humanidad llevó sus diseños a nueva escala de logros. Hoy en día, se dispone de dispositivos físicos y lógicos que permiten avanzar en la creación de esa entidad que esté en capacidad de ayudar o colaborar en el quehacer de la humanidad.

En este momento, se puede decir que el robot representa esa entidad que el ser humano ha estado buscando para que colabore en diferentes tareas, y es la robótica la ciencia que agrupa los conocimientos necesarios para el diseño y la construcción de los robots de hoy y del futuro próximo.

Las arquitecturas de control de robots permiten gestionar y coordinar los recursos disponibles en una plataforma robótica. Este documento ha mostrado una realización de la arquitectura AD, basándose en un desarrollo incremental, donde se pueden diferenciar las siguientes etapas:

1. Abstracción de los actuadores y sensores en un robot, que ocultan la necesidad de manipular y controlar directamente los dispositivos. Ofreciendo un conjunto de modelos y abstracciones, que una vez implementados presentan interfaces bien definidas y de fácil aprendizaje.

2. Una vez que se cuenta con la base que oculta los detalles propios del hardware, se procede a crear las funcionalidades necesarias, que permiten la operación de los dispositivos desde diferentes puntos de conexión, con el fin de lograr una implementación de los actuadores y sensores virtuales. Con esta etapa cumplida, los diferentes actuadores y sensores son gestionados por componentes software que pueden ser usados por una o más aplicaciones de forma concurrente.

3. Diseño y creación de mecanismo para la captura de eventos. En la arquitectura AD el principal elemento de señalización son los eventos, debido a esto la emisión y captura de los mismos debe ser una tarea simple y de

fácil aprendizaje por parte del investigador. De nuevo la ocultación de los detalles requeridos para la comunicación de procesos se debe alcanzar, con el fin de reducir la complejidad en la tarea de gestionar eventos.

4. Diseño y construcción de un mecanismo para el intercambio de información. En la arquitectura AD la implementación de la memoria a corto plazo permite el intercambio de información entre los diferentes actores que la conforman. Se debe permitir que en el intercambio de información se puedan usar diferentes tipos de datos, y que estos estén disponibles en tiempos que permitan la pertinencia de la información.

5. Una arquitectura de control de robots debe tener una entidad principal, y ésta debe permitir la representación de la funcionalidad del robot. En la arquitectura AD esa entidad es la habilidad, que es el elemento básico del diseño y construcción en el sistema de control del robot.

6. Un mecanismo que controle y supervise la operación de un robot. En la arquitectura usada el elemento básico de construcción es la habilidad, y es el secuenciador la herramienta que puede interpretar secuencias de control para activar o inhabilitar una o un grupo de habilidades.

10.1. Aportes

Un trabajo de investigación con una duración de varios años debe presentar aportes tangibles y otros no tan tangibles. Fundamentalmente este trabajo presenta una experiencia exitosa de como abordar el diseño de una arquitectura de control de un robot. Se crea la experticia del trabajo interdiciplinario entre todos los miembros del grupo de investigación, así como de investigadores visitantes y ayudantes de investigación. Se buscan y se usan herramientas para el trabajo colaborativo, documentación, gestores de versiones y mecanismos de divulgación de logros. El uso exitoso de todas estas herramientas es parte de los aportes no tangibles al grupo, lo que no implica que carezcan de valor.

Entre los aportes tangibles, se puede decir, que el mecanismo de desarrollo propuesto en este trabajo se basa, fundamentalmente, en la construcción de un sistema de control basado en componentes distribuidos y altamente desacoplados. Esto permite al investigador disponer de un conjunto de herramientas que cumplen con un único fin especifico:

- **Actuadores y sensores virtuales** ofrecen puntos de acceso a las funcionalidades de los actuadores y sensores físicos y lógicos del robot. El diseño de estas herramientas se basa en el uso bibliotecas que implementan el paradigma de llamadas a procedimientos remotos, en este caso RPC ONC. Estos desarrollos permiten la atención de solicitudes, y por medio de la multiprogramación son atendidos de forma concurrente, y algunas veces, permitiendo un paralelismos entre solicitudes y operaciones del dispositivo, como ocurre en el caso del sensor virtual del láser. Un usuario de los actuadores y sensores remotos solo debe conocer las operaciones implementadas en cada dispositivo virtual. La naturaleza de cada dispositivo

determina el tipo de operaciones. Por ejemplo, en el actuador de un brazo del robot, las operaciones tienen que ver con la velocidad y el giro que debe efectuar el motor encargado de realizar el movimiento. En cambio, en el sensor virtual de una sonaras, las operaciones pueden reducirse a la activación y solicitud de los datos capturados.

- **El gestor de eventos** permite la transmisión de eventos desde procesos en ejecución en el mismo ordenador o desde ordenadores interconectados por medio del protocolo TCP/IP. Es posible agregar un parámetro al evento enviado, lo que permite la implementación de máquinas de estado extendidas con un nivel de abstracción mayor. Una habilidad que use el gestor de eventos puede suscribirse a un evento y definir un comportamiento particular en el momento que éste ocurra. Todos los detalles de establecimiento de conexión, envío/recepción y cierre de conexiones, así como la programación de procesos livianos (threads) están ocultos al usuario.

- **El sistema de memoria a corto plazo** permite la creación de zonas de memoria común entre procesos. Los procesos pueden estar ubicados en diferentes ordenadores y los datos almacenados deben ser serializables. Esto último, permite que la naturaleza de los datos a registrar pueden estar representados en tipos predefinidos por el lenguaje, o en tipos particulares de acuerdo a la necesidad de la aplicación. Las herramientas para la implementación de la memoria a corto plazo son: para la comunicación sockets en el dominio de direcciones UNIX y, para la gestión de los datos, se usó contenedores de la biblioteca estándar STL. Los mecanismos de interconexión de procesos, gestión de memoria y serialización de datos están ocultos al usuario.

- **La clase abstracta habilidad** describe el comportamiento global de una habilidad y define un método vacío llamado "proceso". En este método se describe el lazo de control de la habilidad. El constructor de la clase está diseñado para activar una o más veces dicho lazo control. Además, se puede indicar si el proceso se repite de una manera continua o manteniendo un retardo estático o dinámico. También posee un método para activar o suspender el lazo del control desde fuera de la clase. Toda los detalles de multiprocesamiento están ocultos al usuario.

- **El secuenciador** integra todos los aportes descritos hasta ahora. Cada habilidad por si sola puede controlar o estar subordinada a otras habilidades. Una habilidad debe ser capaz de hacer una tarea especifica. La naturaleza de ésta depende de los requerimientos de diseño de la misma. Si una habilidad está activa se asume que ésta se encuentra realizando la tarea para la cual fue diseñada e implementada. El mecanismo usado para activar o desactivar una habilidad es el envío y captura de un evento específico.

  El funcionamiento del secuenciador se basa en estas ideas. Para soportar el funcionamiento, la operación coordinada de un grupo de habilidades es modelado por una red de Petri. En cada nodo de tipo Lugar de la red, un grupo de habilidades deben estar activas y el resto inactivas. Cada vez

que se alcanza un nuevo Lugar en la red, se envían los eventos para activar/desactivar cada uno de los grupos de habilidades. Al salir del Lugar, se pueden hacer las mismas tareas con el mismo grupo inicial o con un nuevo grupo de habilidades.

Los nodos de tipo Transición poseen funciones de evaluación, que junto a los requisitos de activación, permiten la transición de cambios de marcas de un lugar a otro.

El modelo de la estructura de la red de petri, el marcado, las funciones de activación/desactivación y las funciones de evaluación son descritos en un documento XML, con una estructura predefinida para el secuenciador. Cada secuencia de control de habilidades se describe en un fichero XML, por lo tanto, se puede disponer de varios ficheros de secuencias, donde cada uno de éstos represente un comportamiento, que puede se cargado en el secuenciador de forma dinámica.

Estos componentes han sido totalmente diseñados, desarrollados y probados, y la integración de éstos permiten la construcción de la Arquitectura Automática Deliberativa de una manera modular, fácil de integrar y de depurar. Todos los mecanismos propios del uso de sockets BSD, de herramientas IPC y gestión de procesos livianos (threads) son transparentes al usuario. El desarrollo se hizo usando herramientas POSIX y primitivas de TCP/IP.

Lo descrito en el parágrafo anterior satisface el objetivo principal del trabajo de investigación. Actualmente el sistema desarrollado es el sistema de control del robot Maggie.

La plataforma diseñada y desarrollada puede ser implementada en otro robot, o en un sistema que requiera un software intermedio con gestión de eventos y memoria compartida distribuida entre diferentes ordenadores.

La integración del software y el hardware de Maggie ha sido total, y sirve como plataforma de desarrollo de investigación de proyectos a nivel de Doctorado y Maestría en la Universidad Carlos III de Madrid, así como soporte para proyectos de pregrado en la Universidad Carlos III de Madrid y la Universidad de Los Andes en Mérida, Venezuela.

Desde verano de 2006, esta plataforma ha estado operativa. El robot ha cambiado varias veces sus componentes físicos y el sistema se ha adecuado correctamente a esos cambios.

Hasta este momento el robot Maggie y su sistema de control ha servido de plataforma de desarrollo e investigación, cuyos resultados han permitido más de trece publicaciones, tesis a nivel de Maestría y DEA, proyectos de fin de carrera a nivel de pregrado, reportajes en prensa, así como apariciones en programas de televisión nacional e internacional.

10.2. Trabajos futuros

El desarrollo de la arquitectura AD continua. Actualmente varias tesis a nivel de Doctorado y Maestría usan los desarrollos aquí expuestos, para lograr la implementación de nuevas habilidades en el área de navegación, interacción con personas, reconocimientos de imágenes y sistemas de control basados en emociones. Pero existen varias líneas de desarrollo que extienden y mejoran el funcionamiento actual del trabajo aquí presentado. Entre éstas tenemos:

- Diseño y construcción de nuevas versiones de servidores, con funcionalidades que permitan su recuperación ante fallos, permitiendo el reinicio automático y la restauración del estado del servidor antes del fallo. Para lograr este objetivo, se puede orientar el diseño basado en transacciones, de manera que si ocurre un fallo antes que una solicitud sea cumplida, el sistema sea capaz de continuar la solicitud una vez que el sistema se ha restaurado.
- Agregar servicios de tiempo real para el sistema de actuadores y sensores virtuales. Con esta extensión se permitirá usar la arquitectura AD en aplicaciones que posean restricciones en los tiempos de ejecución.
- Permitir múltiples parámetros en el envío de eventos.
- Adecuación del sistema de memoria a corto plazo, para que éste sea capaz de manejar grandes volúmenes de datos sin crear retardos en el sistema. Esto permitirá la manipulación de datos que contengan información multimedia, que pueda ser requerida por los niveles superiores de la arquitectura AD.
- Como la plataforma permite la comunicación entre habilidades que pueden estar en diferentes ordenadores, una línea interesante a desarrollar es la comunicación e interacción entre sistemas de control residentes en robots diferentes.
- Se puede pensar en establecer jerarquías de secuenciadores, que aíslen por capas de abstracción diferentes comportamientos.
- Extender las habilidades para que incorporen entre sus funcionalidades sistemas de decisiones, permitiendo una nueva generación de habilidades inteligentes.
- Derivar nuevas habilidades que permitan comportamientos basados en creencias.
- Modificar la arquitectura de control para permitir la implementación de enjambres de robots personales.

Apéndice A

Modelos de sistemas de eventos discretos

En este apéndice se muestra un breve resumen de los formalismos usados en este trabajo para modelar sistemas a eventos discretos, estos son, los autómatas de estados finitos y las redes de Petri.

A.1. Sistemas de eventos discretos

El concepto de *sistema*, es uno de los conceptos básicos cuyo significado puede ser dejado a la intuición, ya que existen diferentes definiciones aparentemente válidas, sin embargo se presentan 3 de éstas:

- Conjunto de cosas que relacionadas entre sí ordenadamente contribuyen a determinado objeto [*Diccionario de la lengua española*]
- Una agregación o ensamblaje de cosas combinadas de forma natural o artificial para formar un todo integral o complejo.[*Encyclopedia Americana*]
- Una combinación de componentes que actúan juntos para ejecutar una función que no sería posible por cualquiera de las partes de forma individual.[*IEEE Standar Dictionary of Electrical and Electronic Terms*]

Hay dos principales características que se destacan en estas definiciones. La primera, un sistema involucra la interacción de componentes y, la segunda, un sistema está asociado con una función. Es también importante señalar que un sistema no siempre está asociado a objetos físicos o leyes naturales. Por ejemplo, un sistema de teorías que provee un marco para describir mecanismos económicos o modelar el comportamiento humano y los cambios en la población.

La mayoría de los científicos e ingenieros, inicialmente se enfocaban en el análisis cuantitativo de sistemas, desarrollos de técnicas de diseño, control y

de la medición explícita del desempeño basados en criterios bien definidos. Sin embargo, el uso sólo de definiciones cuantitativas a menudo son inadecuadas. En cambio, con los desarrollos en el área de ingeniería de control, es posible crear un *modelo* matemático del sistema, que se comporte de manera similar al sistema mismo, y sobre éste realizar análisis de comportamiento.

La construcción de un modelo, se inicia con la definición de un conjunto de variables medibles asociadas con el sistema a modelar. Por ejemplo, velocidad de giro, velocidad lineal, posición y voltajes en un circuito, los cuales son números reales. No todas las variables en un sistema son números reales, algunas de estas pueden ser números enteros o conjunto finitos de valores. El *estado* de un sistema dinámico es el conjunto más pequeño de variables, denominadas variables de estado, de modo que el conocimiento de estas variables en $t = t_0$, junto con el conocimiento de entrada para $t \geq t_0$, determina por completo el comportamiento del sistema para cualquier $t \geq t_0$ [119].

Es posible clasificar un sistema en base a la naturaleza de las variables de estado. En los modelos de *estado continuo*, el espacio de estados X consiste de n-dimensional vectores de números continuos, o algunas veces números complejos. Normalmente, X es de dimensiones finitas, aunque pueden existir excepciones. En los modelos de *estados discretos* el espacio de estados es un conjunto discreto.

El comportamiento dinámico de un sistema de estados discretos es a menudo simple de visualizar. Esto se debe a que el mecanismo de transición entre estados es normalmente basado en una sentencia lógica simple de la forma “si algo específico pasa y el estado actual es x, entonces el estado cambia a x'”. Sin embargo, las herramientas matemáticas para expresar formalmente las ecuaciones del espacio de estados y su solución pueden ser consideradas complejas. Por otra parte, los modelos de estados continuos se reducen a el análisis de ecuaciones diferenciales, para las cuales existen técnicas y herramientas matemáticas para su solución [120].

Otra forma de clasificar los sistemas es en base a como se representa el tiempo. En el mundo físico el tiempo transcurre en forma continua, lo cual permite diseñar modelos basadas en ecuaciones diferenciales, los cuales son atractivos desde un punto de vista matemático. Pero es posible definir el valor de las variables de estado solamente en instantes discretos de tiempo. A este tipo de modelos se les conoce *sistemas en tiempo discreto* en contraste a los *sistemas en tiempo continuo*. Hay razones para considerar usar modelos de sistemas en tiempo discreto, entre éstas están:

- Los ordenadores digitales usan componentes electrónicos que operan en modo de tiempo discreto.De hecho éstos están equipados con un reloj digital interno, donde los valores de las variables son calculados en instantes indicados por dicho reloj digital del ordenador.
- Muchas ecuaciones diferenciales de interés en los sistemas modelados en tiempo continuo pueden ser resueltas solamente por medio de algoritmos de análisis numérico implementados en un ordenador digital.
- Las técnicas de control digital, en las cuales se basan los modelos de sistemas en tiempo discreto, ofrecen una alta fidelidad, velocidad y bajo coste.

- Algunos sistemas son por naturaleza en tiempo discreto, como los sistemas económicos, protocolos de redes de ordenadores, sistemas en robótica, etc.

En los sistemas modelados en tiempo discreto, la línea del tiempo se asume como un una secuencia de intervalos definidos por una secuencia de puntos $t_0 < t_1 < \cdots < t_k < \cdots$. Se asume que todos los intervalos son de la misma longitud T, por ejemplo, $t_{k+1} - t_k = T \ \forall k = 0, 1, 2, \cdots$. La constante T es a menudo llamada *intervalo de muestreo*.

Cuando el espacio de estados de un sistema es naturalmente descrito por un conjunto de valores discretos como $\{0, 1, 2, \cdots\}$, y las transiciones de estado son solamente observados en puntos discretos de tiempo, se asocia estas transiciones de estado con *eventos* y se dice que el *sistema es a eventos discretos* [121].

Al igual que el termino *sistema*, el término *evento* es también un concepto intuitivo. Lo importante es destacar que un evento ocurre de forma instantánea y causa una transición en el valor del estado.

Un evento puede ser identificado como una acción específica, como por ejemplo presionar un botón. Este puede ser visto como algo espontáneo, o puede ser el resultado de algunas condiciones las cuales son repentinamente cumplidas.

En los sistemas a tiempo continuo el estado generalmente cambia a medida que el tiempo cambia. Esto es particularmente evidente en sistemas a tiempo discreto. En estos casos la variable tiempo es una variable independiente.

Los sistemas a eventos discretos satisfacen dos propiedades:

1. El espacio de estados es un número discreto.

2. El mecanismo de transición de estados depende de la ocurrencia de eventos.

La primera propiedad permite construir modelos con posibles soluciones computacionales, aunque existen técnicas para la reducción de la complejidad en sistemas con muchos estados. Entre estas propuestas se encuentran las de Harel en [103].

La segunda propiedad indica que la transición entre estados *depende exclusivamente de la ocurrencia de eventos* aceptados por el sistema.

Muchos sistemas, particularmente los tecnológicos, son de hecho sistemas a eventos discretos. Y aun en muchas aplicaciones de interés, una visión como sistemas a eventos discretos de un sistema complejo puede ser necesaria.

A continuación se describen los formalismos usados en este trabajo para la manipulación de sistemas de eventos discretos.

A.2. Máquinas de estados finitos

Uno de los mecanismos formales para el estudio del comportamiento lógico de los sistemas de eventos discretos está basado en la teoría de lenguajes y autómatas. El punto de partida se basa en el hecho de que los sistemas de eventos discretos tiene un conjunto E de eventos asociados a éste. Al conjunto E se le considera el *alfabeto* de un lenguaje. Las secuencias de eventos son consideradas como la *palabras* pertenecientes a dicho lenguaje. Una mayor descripción de la teoría de lenguajes puede ser encontrada en [122].

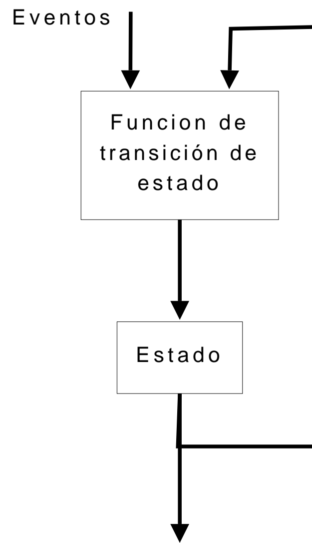

Figura A.1: Funcionamiento de una máquina de estados finitos.

Un máquina de estados finitos[1] es un mecanismo que es capaz de representar un lenguaje siguiendo reglas bien definidas. La forma más simple de representar una máquina de estados finitos es por medio de un grafo dirigido, o por medio de un diagrama de transición de estados. El funcionamiento de una máquina de estados puede verse en la figura A.1.

Formalmente una máquina de estados finitos se define: una 5-tupla denotada por

$G = (X, E, f, x_0, X_m)$

donde:

- X es un conjunto finito de estados.
- E es un conjunto finito de eventos asociados con las transiciones en G.

[1]La palabra Autómata es también usada en muchos entornos como un sinónimo de máquina de estados finitos.

- $f : X \times E \to X$ es la función de transición $f(x, e) = y$ significa que hay una transición etiquetada con el evento e desde el estado x al estado y.

- x_0 es el estado inicial.

- $X_m \subseteq X$ es el conjunto de estados marcados o estados finales.

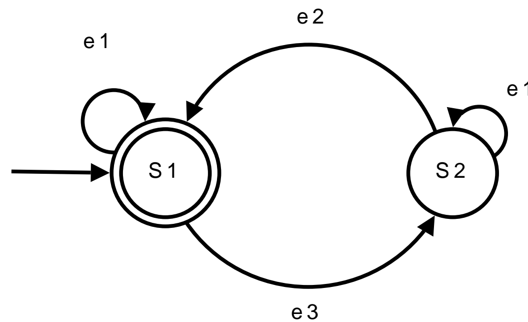

Figura A.2: Ejemplo de una máquina de estados finitos.

En la figura A.2 se muestra un diagrama de una máquina de estados finitos. El conjunto de estados $X = \{S1, S2\}$ presenta los estados posibles del sistema a eventos discretos. $E = \{e1, e2, e3\}$ representa el conjunto de eventos aceptados por la máquina de estados finitos. La función de transición $f$se describe en la siguiente tabla:

	$e1$	$e2$	$e3$
$S1$	$S1$	-	$S2$
$S2$	$S2$	$S1$	-

El estado inicial x_0 es el estado $S1$ y el conjunto de estados finales es $X_m = \{S1\}$.

Las máquinas de estados finitos modelan sistemas a eventos discretos como un todo. La representación de paralelismos y concurrencias no es directa, para tales casos es preferible el uso de otros mecanismos.

A.3. Redes de Petri

Una alternativa a las máquinas de estados finitos son las redes de Petri. Estos modelos fueron desarrollados por C. A. Petri en los años 60. Las redes de Petri son similares a las máquinas de estados, ya que representan explícitamente la función de transición de un sistema a eventos discretos. Al igual que una máquina de estado, una red de Petri es un dispositivo para manejar eventos

siguiendo reglas. Una de las características es que una red de Petri incluye explícitamente condiciones sobre las cuales un evento puede ser habilitado. Esto permite una representación más general de los sistemas de eventos discretos cuya operación depende de sistemas de control complejos. Esta representación es descrita gráficamente de una forma conveniente, al menos para los sistemas pequeños. Una máquina de estados finitos puede ser representada por una red de Petri, pero no todas las redes de Petri pueden ser representadas por máquinas de estados finitos.

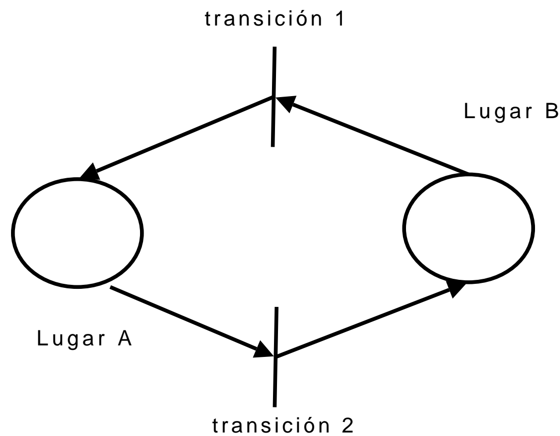

Figura A.3: Ejemplo de la estructura de una red de Petri .

La definición de una red de Petri se hace en dos pasos. En el primer paso se define la estructura de la red de Petri gráficamente. Esto se hace por medio de un diagrama similar al diagrama de una máquina de estado, ver figura A.3. En el segundo paso se indican el estado inicial, esto se hace por medio de fichas que son colocadas en al menos uno de los lugares.

En las redes de Petri los eventos están asociados con transiciones. Para permitir que una transición se ejecute, se deben satisfacer algunas condiciones. La información asociada a estas condiciones es almacenada en los lugares. Algunos de los lugares son vistos como las entradas a transiciones, y estos se asocian a las condiciones para que una transición se ejecute. Otros lugares son observados como salidas de las transiciones, éstos son vistos como las consecuencias de la que se ejecute una transición. Una red de Petri está formada por transiciones, lugares y las relaciones entre estos. Los lugares y las transiciones son los tipos de nodos en una red de Petri. Estos son conectados por medio de un arco dirigido. Un arco solo puede interconectar dos nodos de tipo diferente. En [123] se muestra a fondo el análisis, propiedades y aplicaciones de las redes de Petri.

Apéndice B

Ejemplo de una secuencia de habilidades

La secuencia presentada en este apéndice sirve de ejemplo en la construcción de habilidades. En este ejemplo están involucradas las habilidades **obstaclesavoidance** y **wandering**. La habilidad obstaclesavoidance ante la presencia de un obstáculo en su entorno realiza movimientos de rotación mientras existan obstáculos al frente del robot. La habilidad wandering traslada el robot de forma aleatoria. Esta habilidad es útil en tareas de exploración.

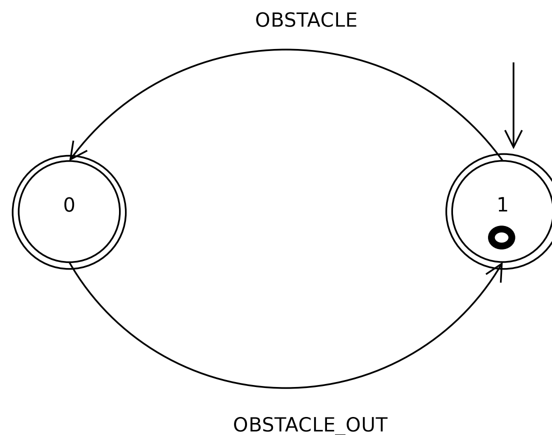

Figura B.1: Diagrama de estados de la secuencia obstaclesavoidance.

En este comportamiento la habilidad wandering está subordinada a la habilidad obstacles avoidance. El comportamiento de la secuencia se muestra en el diagrama de transición de estados mostrado en la figura B.1

Como se ve, la secuencia posee dos estados: en el estado 0 el robot explora el entorno, en el estado 1 el robot evita obstáculos. Los eventos en este diagrama son OBSTACLE y OBTACLE_OUT.

En la figura B.2 se muestra la red Petri de la secuencia. En este caso la transformación de la máquina de estados finitos a una red de Petri es trivial. El tener un primer modelo global por medio de una máquina de estado da una mejor apreciación de la estructura de la red Petri que describe la secuencia. En este caso la red de Petri posee dos lugares y dos transiciones. Inicialmente existe una marca en el lugar 1 y el lugar 0 no posee ninguna marca.

En este caso las habilidades obstaclesaovoidance y wandering pueden estar en ejecución en un mismo o en diferentes contenedores de habilidades, sobre el mismo ordenador o sobre ordenadores diferentes. El Middleware de la plataforma creada se encarga de integrar de manera transparente el secuenciador y las habilidades coordinadas por el mismo.

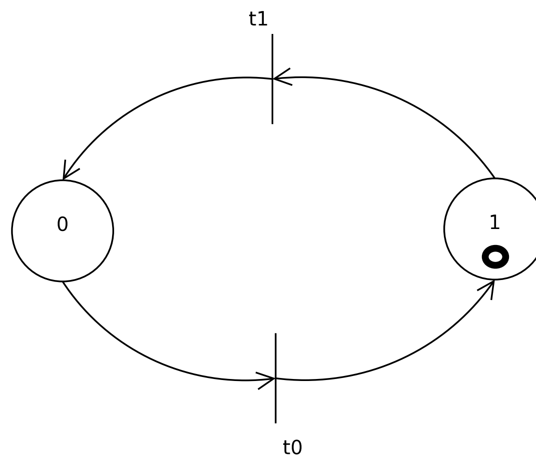

Figura B.2: Red de Petri de la secuencia obstaclesavoidance.

En el documento XML de la secuencia presentado a continuación, tal como se describió en 8.3.3, un encabezado donde indica el URL del documento, así como la descripción de la secuencia y los datos de la misma.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Secuencia  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://roboticslab.uc3m.es/AD/secuencia.xsd">
   <Descripcion> "Secuencia de ejemplo evitarobstaculo.
      El ejercicio posee:
      2 etapas,
      2 transiciones,
      la Matriz de precondiciones es:
      {{ 1, 0 }, { 0, 1}}
      La Matriz de postcondiones es:
      {{ 0, 1 }, { 1, 0}}
      El vector de marcas inicial es: { 0, 1}
      2 funciones Python de  evaluacion
      2 funciones de activacion
      2 funciones de desactivacion
   </Descripcion>

   <nEtapas> 2 </nEtapas>

   <nTransiciones> 2 </nTransiciones>

   <mPreCondiciones>
      <fila>
         <valor>1</valor> <valor>0</valor>
```

```
      </fila>
      <fila>
         <valor>0</valor> <valor>1</valor>
      </fila>
   </mPreCondiciones>

   <mPostCondiciones>
      <fila>
         <valor>0</valor> <valor>1</valor>
      </fila>
      <fila>
         <valor>1</valor> <valor>0</valor>
      </fila>
   </mPostCondiciones>

   <vMarcas>
      <fila>
         <valor>0</valor> <valor>1</valor>
      </fila>
   </vMarcas>

   <FuncionEvaluacion>
      <id> 0 </id>
      <linea>import datosMCAD</linea>
      <linea>import MC</linea>
      <linea>(evento,p) = MC.ObtenerDatoActual(datosMCAD.id, datosMCAD.tam, datosMCAD.formato)</linea>
      <linea>if (evento == datosMCAD.OBSTACLE):</linea>
      <linea>\ returnAD = 1</linea>
      <linea>else:</linea>
      <linea>\ returnAD = 0</linea>
   </FuncionEvaluacion>

   <FuncionEvaluacion>
      <id> 1 </id>
      <linea>import datosMCAD</linea>
      <linea>import MC</linea>
      <linea>(evento,p) = MC.ObtenerDatoActual(datosMCAD.id, datosMCAD.tam, datosMCAD.formato)</linea>
      <linea>if (evento == datosMCAD.OBSTACLEOUT):</linea>
      <linea>\ returnAD = 1</linea>
      <linea>else:</linea>
      <linea>\ returnAD = 0</linea>
   </FuncionEvaluacion>

   <FuncionActivacion>
      <id> 0 </id>
      <linea>import datosMCAD</linea>
      <linea>import eventosAD</linea>
      <linea>returnAD = eventosAD.emite(datosMCAD.WANDERINGON,datosMCAD.WANDERINGON )</linea>
   </FuncionActivacion>

   <FuncionActivacion>
      <id> 1 </id>
      <linea>import datosMCAD</linea>
      <linea>import eventosAD</linea>
      <linea>returnAD = eventosAD.emite(datosMCAD.OBSTACLESAVOIDANCEON,datosMCAD.OBSTACLESAVOIDANCEON )</linea>
   </FuncionActivacion>

    <FuncionDesActivacion>
      <id> 0 </id>
      <linea>import datosMCAD</linea>
      <linea>import eventosAD</linea>
      <linea>returnAD = eventosAD.emite(datosMCAD.WANDERINGOFF,datosMCAD.WANDERINGOFF )</linea>
   </FuncionDesActivacion>

   <FuncionDesActivacion>
      <id> 1 </id>
      <linea>import datosMCAD</linea>
      <linea>import eventosAD</linea>
      <linea>returnAD = eventosAD.emite(datosMCAD.OBSTACLESAVOIDANCEOFF,datosMCAD.OBSTACLESAVOIDANCEOFF )</linea>
   </FuncionDesActivacion>

</Secuencia>
```

Referencias

[1] The OROCOS Project, "What is Orocos?," 2007. http://www.orocos.org/orocos/whatis, Visited on Jun 2007.

[2] The Player Project, "Stage," 2007. http://playerstage.sourceforge.net/stage/stage.html, Visited on Jun 2007.

[3] The Player Project, "Gazebo," 2007. http://playerstage.sourceforge.net/gazebo/gazebo.html, Visited on Jun 2007.

[4] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack, "Experiences with an architecture for intelligent, reactive agents," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, pp. 237 – 256, April 1997.

[5] R. Simmons, "Inter process communication (ipc)," 2007. http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html.

[6] Microsoft, "Microsoft Robotics Studio," 2007. http://msdn2.microsoft.com/robotics/.

[7] Evolution Robotics, "ERSP 3.1: Robotic development platform," 2007. http://www.evolution.com/products/ersp/, Visited on Sep. 2007.

[8] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARAty architecture for robotic autonomy," in *Proceedings of the 2001 IEEE Aerospace Conference*, (Big Sky Montana), March 2001.

[9] Skilligent, "Criteria for selection of robotic manipulators for trainable robots guided. technical guidelines," tech. rep., 2007.

[10] Gostai, "URBI: The innovative & easy platform for robotics.," 2007. http://gostai.com/, Visited on Oct, 2007.

[11] CIMAR, "JAUS Architecture," 2007. Center for Intelligent Machines and Robotics. University of Florida. http://openjaus.com/, Visited on Sept, 2007.

[12] O. Michel, "Webots: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39 – 42, 2004. http://www.cyberbotics.com/publications/ars.pdf.

[13] M. A. Salichs and R. Barber, "A new human based architecture for intelligent autonomous robots," in *The Fourth IFAC Symposium on Intelligent Autonomous Vehicles*, (Sapporo, Japón), pp. 85– 90, Sep 2001.

[14] Python Software Foundation, "Python programming language," 2007. http://www.Python.org. Visited on Jun 2007.

[15] RoboticsLabs UC3M, "El robot maggie," 2007. http://roboticslab.uc3m.es/maggie, Visited on Jan., 2006.

[16] B. W. Kerninghan and D. M. Ritchie, *The C Programming Language*. Prentice Hall, 2nd ed., 1988.

[17] B. Stroustrup, *El Lenguaje de Programación C++, Edición Especial*. Addison Wesley, 1 ed., 2002.

[18] T. Fukuda, R. Micheline, V. Potkonjak, S. Tzafestas, K. Valavanis, and M. Vukobratovic, "How far away is "artificial man"," *IEEE Robotics & Automation Magazine*, pp. 67–72, March 2001.

[19] W. Jacobs, "Control systems in robots," in *ACM'72: Proceedings of the ACM annual conference*, (New York, NY, USA), pp. 110–117, ACM Press, 1972.

[20] J. H. Munson, "Robot planning, execution, and monitoring in an uncertain environment.," in *IJCAI*, pp. 338–349, 1971.

[21] M. J. Matarić, "Behavior-based systems: Main properties and implications," in *IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, (Nice, France), pp. 45–54, May 1992. http://www-robotics.usc.edu/~maja/publications.html.

[22] R. C. Arkin, *Intelligent robots and autonomous agents*. MIT Press, 1998.

[23] SRI Technology, "Shakey, the robot," 2006. http://www.sri.com/about/timeline/shakey.html, Visited on Jan., 2006.

[24] J. S. Albus, J. C. Fiala, A. J. Wavering, and R. Lumi, "NASREM – the NASA/NBS standard reference model for telerobot control system architecture," in *20th International Symposium on Industrial Robots*, (Tokio, Japan), pp. 4– 6, October 1989.

[25] U.S. Commerce Department's Technology Administration, "National Institute of Standars and Technology," 2006. http://www.nist.gov, Visited on Jan., 2006.

[26] J. Albus, "James Albus Homepage," 2006. http://www.jamesalbus.org, Visited on Jan., 2006.

[27] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Transaction on Robotics and Automation*, vol. 2, pp. 14– 23, Mar 1986.

[28] M. J. Matarić, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 304–312, Jun 1992.

[29] M. Matarić, *Interaction and Intelligent Behavior*. PhD thesis, MIT EECS, May 1994. `http://www-robotics.usc.edu/~maja/publications.html`.

[30] R. C. Arkin and T. R. Balch, "Aura: Principles and practice in review," *Journal of Experimental and Theoretical Artificial Intelligence(JETAI)*, vol. 9, pp. 175–188, April 1997.

[31] T. Furuta, T. Tawara, Y. Okumura, and M. Shimizu, "Design and construction of a series of compact humanoid robots and development of biped walk control strategies," *Robotics and Autonomous Systems*, vol. 37, pp. 81–100, Nov. 2001.

[32] P. Dario, E. Guglielmelli, and C. Laschi, "Humanoids and personal robots: design and experiments," *Robotic Systems*, vol. 18, pp. 673–690, Dec 2001.

[33] R. M. Shiffrin and W. Schneider, "Controlled and automatic human information processing: In perceptual learning, automatic attending and a general theory," *Psicological Review*, pp. 127–190, 1997.

[34] V. Egido, *Sistema de navegación topológica para robots móviles autónomos.* PhD thesis, Departamento de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, 2003.

[35] M. A. Salichs, R. Barber, and M. J. Boada, "Visual approach skill for a mobile robot using learning and fusion of simple skills," *Robotics and Autonomous Systems*, vol. 38, pp. 157– 170, Mar 2002.

[36] M. J. López, *Sistema de control para robots móviles autónomos basados en habilidades reactivas.* PhD thesis, Departamento de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, 2002.

[37] Real World Interface, Inc, *B21 Robot System Manual.* Real World Interface, Inc, 1996.

[38] R. Barber, *Desarrollo de una arquitectura para robots móviles autónomos. Aplicación a un sistema de navegación topológica.* PhD thesis, Departamento de Ingeniería Eléctrica, Electrónica y Automática, Universidad Carlos III de Madrid, 2000.

[39] D. Rivero, *Framework para el desarrollo de habilidades para robots móviles inteligentes basado en la arquitectura AD.* PhD thesis, Departamento de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, 2004.

[40] Real World Interface, Inc, *Mobility 1.1: Robot Integration Software*. Real World Interface, Inc, 1999.

[41] The Player Proyect, "rFlex driver," 2007. `http://playerstage.sourceforge.net/doc/Player-manual-1.5-html/node97.html`. Visited on Jun 2007.

[42] T. H. J. Collet, B. A. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Australasian Conference on Robotics and Automation*, (Sydney, Australia), Dec 2005.

[43] J. Fernández, "Documentación para el diseño de un robot autónomo," tech. rep., Roboticslab - UC3M, Dic 2004.

[44] L. Cobos, "Descripción de los componentes de hardware del robot maggie," tech. rep., Roboticslab - UC3M, Junio 2007.

[45] M. A. Salichs *et al.*, "Maggie: A robotic platform for human-robot social interaction," in *IEEE International Conference on Robotics, Automation and Mechatronics (RAM 2006)*, (Bangkok. Thailand), Jun 2006.

[46] A. Gokhale, B. Kumar, and A. Sahuguet, "Reinventing the wheel? corba vs. web services," in *International WWW Conference*, (Honolulu, Hawaii), July 2002.

[47] CORDIS, "IST project fact sheet," 2007. http://cordis.europa.eu/fetch?ACTION=D&CALLER=PROJ_IST&QM_EP_RCN_A=58069, Visited on Jun 2007.

[48] O. Robotics, "Orca: Components for robotics," 2010. http://orca-robotics.sourceforge.net/orca_doc_overview.html, Visited on Jan, 2010.

[49] SmartSoft, "Smartsoft: Components and toolchain for robotic," 2010. http://smart-robotics.sourceforge.net/, Visited on Jan, 2010.

[50] The OROCOS Project, "Applications," 2007. http://www.orocos.org/orocos/applications, Visited on Jun 2007.

[51] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos Project," in *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, (Taipei, Taiwan), pp. 797 – 802, September 2003.

[52] D. C. Schmidt, "TAO: Real time corba," 2006. http://www.cs.wustl.edu/~schmidt/TAO.html . Visited on Jan., 2006.

[53] Player Project, "Player & stage," 2006. http://playerstage.sourceforge.net, Visited on July 2006.

[54] B. P. Gerkey, R. T. Vaughan, K. Støy, A. Howard, G. S. Sukhatme, and M. J. Mataric, "Most valuable player: A robot device server for distributed control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, (Wailea, Hawaii), pp. 1226 – 1231, October 2001.

[55] W. R. Stevens, B. Fenner, and A. M. Rudoff, *UNIX Network Programing, Volume 1: Networking APIs: Sockets and XTI.* Prentice Hall, second ed., 1998.

[56] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, (Coimbra, Portugal), pp. 317 – 323, June 2003.

[57] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming : The Carnegie Mellon Navigation (CARMEN) Toolkit," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, (Las Vegas, NV), pp. 2436 – 2441, October 2003.

[58] The Carmen Project, "CARMEN: Robot navigation toolkit," 2007. `http://carmen.sourceforge.net/`, Visited on Jun 2007.

[59] K. Konolige, "A gradient method for realtime robot control," in *In Proc. of the IEEE/RSJ Interational Conference on Intelligent Robotic Systems (IROS 2000)*, (Takamatsu, Japan), pp. 639 – 646, October 2000.

[60] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, "A system for volumetric robotic mapping of abandoned mines," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, (Taipei, Taiwan), September 2003.

[61] S. Cherry, "Robots incorporated," *IEEE Spectrum*, vol. 44, pp. 24 – 29, Aug 2007.

[62] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim, "CLARAty: An architecture for reusable robotic software," in *SPIE Aerosense Conference, Orlando*, (Florida), April 2003.

[63] Skilligent LLC, "Skilligen," 2007. `http://www.skilligent.com`, Visited on Sept, 2007.

[64] I. Free Software Foundation, "GNU General Public License," 2007. `http://www.gnu.org/copyleft/gpl.html`, Visited on Oct, 2007.

[65] iRobot, "iRobot Aware," 2007. `http://www.irobot.com/sp.cfm?pageid=86&id=280`, Visited on Sept, 2007.

[66] Vecna Robotics, "The BEAR: Battlefield Extraction-Assist Robot," 2007. `http://vecnarobotics.com/robotics/product-services/bear-robot/index.shtml`. Visited on Sept, 2007.

[67] Cyberbotics, "Webots," 2007. `http://www.cyberbotics.com/publications/ars.pdf`, Visited on Sept, 2007.

[68] R. M. Adler, "Emerging standards for component software," *Computer*, vol. 28, pp. 68 – 77, Mar 1995.

[69] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, ch. 3. Seventh Edition. John Wiley & Sons, Inc., 2006.

[70] Wikipedia, "System v," 2007. `http://es.wikipedia.org/wiki/System_V`, Visited on Jun 2007.

[71] IEEE Computer Society Portable Application Standards Committee, "Open group posix," 2006. `http://http://www.pasc.org/plato/` . Visited on Jan., 2006.

[72] IEEE POSIX, "Posix," 2006. http://standards.ieee.org/regauth/posix/. Visited on Jan., 2006.

[73] Wikipedia, "Posix," 2006. http://es.wikipedia.org/wiki/POSIX/ . Visited on Jan., 2006.

[74] W. R. Stevens, *Advanced programming in the UNIX environment.* Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1992.

[75] M. Henning and S. Vinoski, *Programación Avanzada en CORBA con C++.* Addison Wesley, 2002.

[76] W. R. Stevens, *UNIX Network Programing, Volume 2: Interprocess Communication.* Prentice Hall, second ed., 1999.

[77] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman, *The desing and implementation of the 4.4 BSD Operating System.* Addison Wesley, 1996.

[78] A. D. Birrel and B. J. Nelson, "Implementing remote procedure calls," *ACM Transactions on Computer Systems*, vol. 2, pp. 39–59, February 1984.

[79] J. Bloomer, *Power programming with RPC.* O'Reilly & Associates, Inc, 1 ed., 1992.

[80] R. Srinivasan, "Rpc: Remote procedure call protocol specification version 2," 2005. http://www.ietf.org/rfc/rfc1831.txt, Visited on Sept. 2005.

[81] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services. Concepts, Architectures and Application.* Springer, 2004.

[82] Sun Microsystem, "Java remote method invocation," 2006. http://java.sun.com/products/jdk/rmi, Visited on July, 2006.

[83] Object Management Group, "The object management group," 2005. http://www.omg.org,Visited on July, 2005.

[84] Object Management Group, "Corba services: Common object services specification," 2005. ftp://ftp.omg.org/pub/docs/formal/98-07-05.pdf, Visited on Sept, 2005.

[85] Object Management Group, "Person identification services," 2005. ftp://ftp.omg.org/pub/docs/corbamed/97-11-01.pdf, Visited on Sept, 2005.

[86] Object Management Group, "Catalog of domain specifications," 2005. http://www.omg.org/technology/documents/domain_spec_catalog.htm, Visited on Sept, 2005.

[87] M. Fisher, "The java web services tutorial 1.0," Visited on Sept. 2005. Sun Developer Network, http://java.sun.com/webservices/docs/1.0/tutorial/.

[88] The National Center for Supercomputing Applications, "Common gateway interface," 2005. http://hoohoo.ncsa.uiuc.edu/cgi/intro.html, Visited on Sept. 2005.

[89] UDDI ORG., "Uddi executive white paper," Visited on Sept., 2005 2005. http://uddi.org/pubs/UDDI_Executive_White_Paper.pdf, Visited on Sept., 2005.

[90] World Wide Web Consortium, "Web services architecture requirements," 2006. http://www.w3.org/TR/wsa-reqs/, Visited on July, 2006.

[91] D. C. Schmidt and S. D. Huston, *C++ Network Programming: Mastering Complexity Using ACE and Patterns*, vol. 1. Addison-Wesley Longman, 2002.

[92] F. P. Brooks., "No silver bullet: Essence and accidents of software engineering," *IEEE Computer*, vol. 20, pp. 10 – 19, April 1987.

[93] I. Sommerville, *Ingeniería del Software*. Pearson Educación, S. A., séptima ed., 2005.

[94] R. S. Pressman, *Ingeniería del software, un enfoque práctico*. McGraw Hill, 2002.

[95] Tigris, "Subversion," 2005. http://subversion.tigris.org/, Visited on Sept 2005.

[96] Roboticslab, "Wiki de maggie," 2007. http://roboticslab.uc3m.es/dokuwiki/, Visited on Jun 2007.

[97] J. Carretero, F. García, P. Anasagasti, and F. Pérez, *Sistemas Operativos: Una visión aplicada*. Mc Graw Hill, segunda ed., 2007.

[98] R. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, pp. 206–230, Jan 1987.

[99] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Patrones de diseño. Elementos de Software*. Pearson Educación, S. A., 2003.

[100] J. F. Gorostiza *et al.*, "Multimodal human-robot interaction framework for a personal robot," in *The 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 06)*, (Hatfield, United Kingdom), Sept 2006.

[101] R. Rivas *et al.*, "Schab: Sistema mínimo de supervisión y control de habilidades en el robot maggie," in *Jornadas de Automática*, (Almeria, España), Sept 2006.

[102] Wikipedia, "Grafcet," 2007. http://es.wikipedia.org/wiki/GRAFCET, Visited on Jun 2007.

[103] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.

[104] M. Morrison, *XML al descubierto*. Prentice Hall, 2000.

[105] WC3, "Extensible markup language (xml)," 2007. http://www.w3.org/XML/, Visited on Jun 2007.

[106] WC3, "The world wide web consortium," 2007. http://www.w3.org , Visited on Jun 2007.

[107] Wikipedia, "Sgml," 2007. http://es.wikipedia.org/wiki/SGML , Visited on Jun 2007.

[108] WC3, "Xml schema," 2007. http://www.w3.org/TR/xmlschema-1/, Visited on Jun 2007.

[109] K. Jensen and N. Wirth, *Pascal user manual and report : ISO Pascal standard*. Springer, 1991.

[110] Wikipedia, "El lenguaje de programación pascal," 2007. http://es.wikipedia.org/wiki/Lenguaje_deprogramacion_Pascal, Visited on Jun 2007.

[111] G. van Rossum, *Extending and Embedding the Python Interpreter*. Python Software Foundation, Fred L. Drake, Jr., editor, release 2.5 ed., Sept 2006.

[112] C. Rodríguez, "Creación de un ambiente de desarrollo gráfico para la edición de secuencias de control en un robot móvil," tech. rep., Escuela de Ingeniería de Sistemas, Universidad de Los Andes, Venezuela, 2008.

[113] C. E. Spurgeon, *Ethernet: The Definitive Guide*. O'Reilly and Associates, 1 ed., Feb 2001.

[114] Cisco, "Ethernet technologies," 2007. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ethernet.htm, Visited on Dic, 2007.

[115] A. S. Tanenbaum, *Computers networks*. Prentice Hall, fourth ed., March 2003.

[116] H. P.G., P. S. C., and S. C. J., *Introduction to probability theory*. Houghton Mifflin company, 1971.

[117] Wikipedia, "Cable de par trenzado," 2008. http://es.wikipedia.org/wiki/Cable_de_par_trenzado, Visited on Dic, 2008.

[118] IEEE, "Draft standards for project ieee 802.11mb," 2008. http://grouper.ieee.org/groups/802/11/index.html, Visited on Dic, 2008.

[119] K. Ogata, *Ingeniería de control moderna*, ch. 3. Prentice-Hall, 1998.

[120] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Kluwer Academic Publishers, 1999.

[121] C. G. Cassandras, "Multimedia introduction to discrete event systems." Dept. of Manufacturing Engineering, Boston University, 2007. http://vita.bu.edu/CgC/mideds/, Visited on Sept 2007.

[122] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation.* Addison-Wesley, 2 ed., 2001.

[123] T. Murata, "Petri nets: properties, analysis, and applications," *Proceedings of the IEEE*, vol. 77, pp. 541 – 580, April 1989.