Arquitectura Genérica para Sistemas de Robótica Móvil

Juan Fernando Florez Marulanda

Monografía presentada como requisito parcial para optar al título de
Magíster en Electrónica y Telecomunicaciones

Director:
Mg. Jaime Diaz

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Grupo en Automática Industrial
Línea de Investigación: Control y Robótica
Popayán, 2009

Resumen

El tema de trabajo de esta tesis es la propuesta y diseño de una arquitectura de control genérica, denominada AGC-MAS, para robots móviles basada en sistemas multi agentes, enfoncándose en particular en la semántica de los diagramas de "cajas y flechas" empleados en los diagramas de diseño conceptual de las arquitecturas de control.

En el presente trabajo se realiza una actualizada exploración del estado actual del estado de arte en lenguajes, plataformas, herramientas, esfuerzos de comparación y estandarización realizados en robótica móvil. Igualmente se realiza un análisis de la evolución de las arquitecturas de control empleadas en robotica móvil al igual de las principales plataformas de trabajo *software* en robótica que actualmente están disponibles para uso libre. Se realiza la propuesta de una nueva arquitectura de control denominada Arquitectura Multiagente.

Con el fin de ilustrar la potencialidad de la arquitectura AGC-MAS propuesta en el presente trabajo de tesis se realizan ocho diseños incrementales de arquitecturas de control para un robot móvil, iniciando desde un robot móvil teleoperado y llegando hasta un robot móvil totalmente reactivo, pasando por un robot móvil deliberativo y uno híbrido. Finalmente se recomiendan arquitecturas complementarias y plataformas *software* de trabajo robótico que pueden ser empleadas en conjunto con AGC-MAS.

Palabras clave: Robótica móvil, Arquitecturas de control, Sistemas multi agente, Plataformas de trabajo *software* robotico

Tabla de Contenido

Lista de Figuras

Lista de Tablas

1. Introducción

El tema de trabajo de esta tesis es una propuesta de arquitectura genérica para el sistema de control de un robot móvil, enfocándose en particular en los diferentes componentes y el lenguaje de comunicación entre los mismos.

En este capítulo se presenta una revisión histórica de los paradigmas de control, de las plataformas de trabajo en robótica móvil empleadas y de los estudios comparativos y esfuerzos de estandarización en los que se enmarcan los aportes del trabajo realizado.

1.1 Qué es un robot móvil?

Básicamente un robot móvil es una máquina integrada por tres sistemas: un sistema mecánico, un sistema electrónico y un sistema de procesamiento cognitivo, ver figura 1.1, los cuales en conjunto le brindan capacidad de locomoción, interacción y autonomía [1]. La anterior definición técnica no establece la utilidad de la máquina por lo que la *International Federation of Robotics* (IFR) define un robot móvil desde la perspectiva de su utilidad clasificándolo como un robot de servicio, siendo éste aquel robot móvil que opera total o semi autónomamente para realizar servicios útiles bien sea a seres vivos o máquinas, excluyendo operaciones de manufactura [2].

La locomoción se puede realizar tanto con extremidades inferiores antropomórficas o con ruedas para desplazamiento en tierra firme o bien con hélices o turbinas para el desplazamiento en aire o agua, siendo la locomoción con ruedas la más común en la mayor parte de los robots móviles. En la actualidad grupos de investigación a lo ancho del mundo diseñan y construyen nuevos robots móviles terrestres, aéreos y acuáticos en forma independiente, ver figura 1.2, para los cuales se requiere un sistema de control de características similares [3].

Sistema Cognitivo

Sistema Electrónico

Sistema Mecánico

Figura 1.1 Diagrama en sistemas constitutivos de un robot móvil organizados por niveles

a) [4]

b) [5]

c) [6]

d) [7]

e) [8]

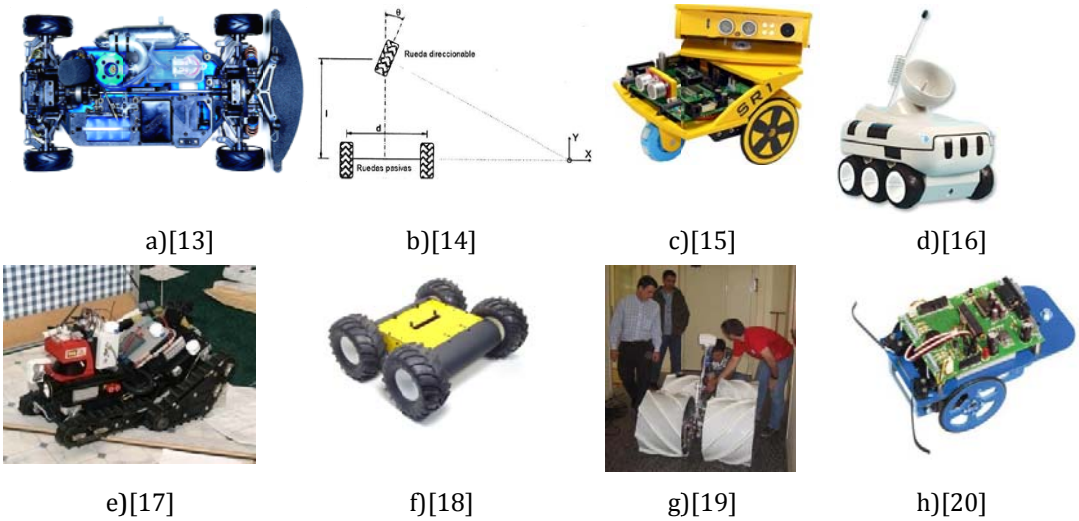
f) [9]

g) [10]

h) [11]

Figura 1.2 Ejemplos de robots móviles con diferentes tipos de locomoción: terrestre (a, b,c y d), aéreo(g y h) y acuático(e y f)

Un robot móvil con ruedas constituye la forma más simple y eficiente para conseguir una buena movilidad en terrenos suficientemente duros y libres de obstáculos, con velocidades relativamente altas, y en el mundo son los robots móviles que más prevalecen [12]. Existen diferentes tipos de arquitecturas mecánicas de locomoción con ruedas, ver figura 1.3, que les confieren características y propiedades diferentes respecto a la eficiencia energética, dimensiones, cargas útiles y maniobrabilidad: *ackerman*, triciclo, diferencial, *skid steer*, pistas de deslizamiento, síncronas y configuraciones alternativas [1].



a)[13] b)[14] c)[15] d)[16]

e)[17] f)[18] g)[19] h)[20]

Figura 1.3. Ejemplos de diferentes tipos de locomoción terrestre con ruedas: a) *ackerman*, b) triciclo, c) y h) diferencial, d) *skid steer* ruedas, e) *skid steer* pistas deslizamiento y f) y g) alternativos

El tipo de aplicación del robot determina las tareas que llevará a cabo y, éstas a su vez, determinan el tamaño, el tiempo de autonomía, el tipo de locomoción, la opción de disponer o no de un manipulador o efector final y las características del sistema sensorial. En la actualidad las aplicaciones que puede desarrollar un robot móvil con ruedas cubren desde: vigilancia en edificios, recolección y entrega de materias primas o elementos varios, limpieza, guías de museos, acompañantes o enfermeros, entretenimiento, plataformas de investigación, etc [3]. Estas aplicaciones normalmente se realizan en interiores (*indoor*) donde las condiciones son controladas. Sin embargo también existen aplicaciones en exteriores (*outdoor*), donde las condiciones son críticas y se requiere de una mayor exigencia tanto del sistema de locomoción como del sensorial. Tareas comunes en estas condiciones son: búsqueda y rescate, exploración, milicia, etc. Esta explosión de servicios robóticos son atendidos principalmente con robots móviles comerciales y una pequeña parte, investigación en la academia, es realizada en agencias (NASA) o institutos en centros de educación superior (LAAS – CNRS, CMU, etc) donde diseñan y construyen sus propios robots o bien emplean plataformas móviles comerciales como sucede en la gran mayoría de los centros de investigación del mundo [3]. Algunos de estos servicios robóticos (búsqueda y exploración), exige que los vehículos puedan interactuar de una forma más directa con su entorno, esto se puede realizar por medio de un simple efector final tipo pinza (*gripper*) o con un actuador más elaborado tipo brazo industrial, por lo que la

plataforma normalmente se diseña acoplada con un manipulador de peso ligero de uno o más grados de libertad, ver figura 1.4.

La configuración mecánica del sistema de locomoción del robot móvil determina el tipo y número de lazos de control de velocidad para las ruedas del vehículo. Básicamente los sistemas de locomoción o configuración de las ruedas son de los tipos: triciclo y diferencial, ver figura 1.3 b) y c), para los robots en interiores, mientras que para robots en exteriores el tipo de locomoción o configuración más común es *ackerman* y *skid steer* con llantas u orugas, ver figura 1.3 a), d) y e), aunque configuraciones alternativas son ampliamente investigadas [1], ver figuras 1.3 f), g) y h). El tipo de locomoción igualmente determina la complejidad de los algoritmos que implementan el sistema de control de posición y velocidad del vehículo.

a)[21] b)[22] c)[23]

Figura 1.4. Robots móviles con diferentes tipos de manipuladores: a)pinza simple y b) y c) pinza con manipulador avanzado

El requerimiento de disponer de un manipulador exige que se deba contar con un sistema de control para gobernar el accionamiento de actuadores de los enlaces del brazo y del efector final. Igualmente la presencia del manipulador hace más exigente las características que debe cumplir el sistema sensorial, ya que adicional al sensado requerido para permitir la navegación se debe sensar la ejecución de la tarea realizada por el manipulador.

Los dos anteriores conjuntos de determinaciones fijan las características del sistema electrónico del robot móvil, el cual se dividirá, en forma general, en tres sistemas: el sistema de control de velocidad del robot móvil, el sistema de control del manipulador y el sistema sensorial, ver figura 1.5.

El sistema de control de velocidad hace uso de motoreductores para las ruedas, codificadores incrementales u otro tipo de sensores de velocidad, controladores y

actuadores o manejadores de potencia para los motores DC. El sistema de control del manipulador hace uso de motoreductores para los enlaces, codificadores absolutos u otro tipo de sensor de posición angular, controladores y drivers de potencia. La medición realizada por los codificadores incrementales, junto con la medición del consumo de corriente de cada motor, la medición del voltaje de la batería del vehículo, más otros datos similares medidos en el manipulador conforman el sistema propiosensor del robot o de su estado interno. Igualmente se requiere disponer de sensores que permitan realizar la navegación y el desarrollo de las tareas de alto nivel ejecutadas por el robot en su entorno, generalmente estos sensores van desde simples fines de carrera en forma de *bompers* hasta sofisticados sensores infrarrojos, ultrasónicos, radares láser y sistemas de visión mono o estéreo, complementados con la información de la localización medida por medio de compases magnéticos, giróscopos, sistemas de navegación inercial y, en algunos casos, sistemas de localización global (GPS). Estos sensores conforman el sistema exterosensor del robot. El conjunto sistema propiosensor y exterosensor conforma el sistema sensorial del robot móvil [24], ver figura 1.5.

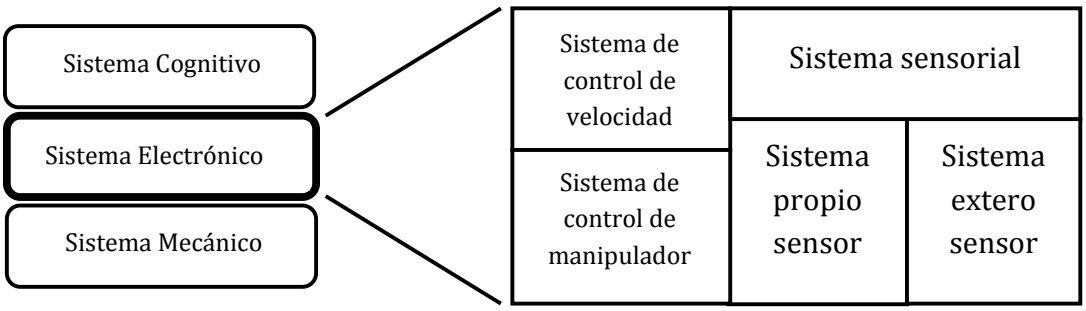


Figura 1. 5 División en bloques del sistema electrónico de un robot móvil

En la actualidad cada uno de los componentes (controlador – actuador y sensores) arriba mencionados, tanto de los sistemas de control como del sensorial, pueden ser módulos autónomos con interfaces de comunicación sofisticadas, por lo que es común encontrar algunos de ellos interconectados en una red de comunicación tan simple como I2C (*Inter Integrated Circuit*), una industrial como CAN (*Controller Area Network*) o tan versátil como *Ethernet* [1]. El que algunos o todos estos componentes dispongan o no de estas interfaces determina la configuración o arquitectura del sistema electrónico del robot móvil: centralizado o distribuido. Esto es, un único nodo central de procesamiento de tiempo real (RT) con los sensores y actuadores conectados directamente a él, o un conjunto de nodos especializados interconectados en una red de comunicaciones de tiempo real, ver figura 1.6. Independiente de sí el sistema

electrónico es centralizado o distribuido debe existir un mecanismo de intercomunicación entre éste y el sistema de procesamiento y toma de decisiones de alto nivel del vehículo, ver figura 1.7 [25][26].

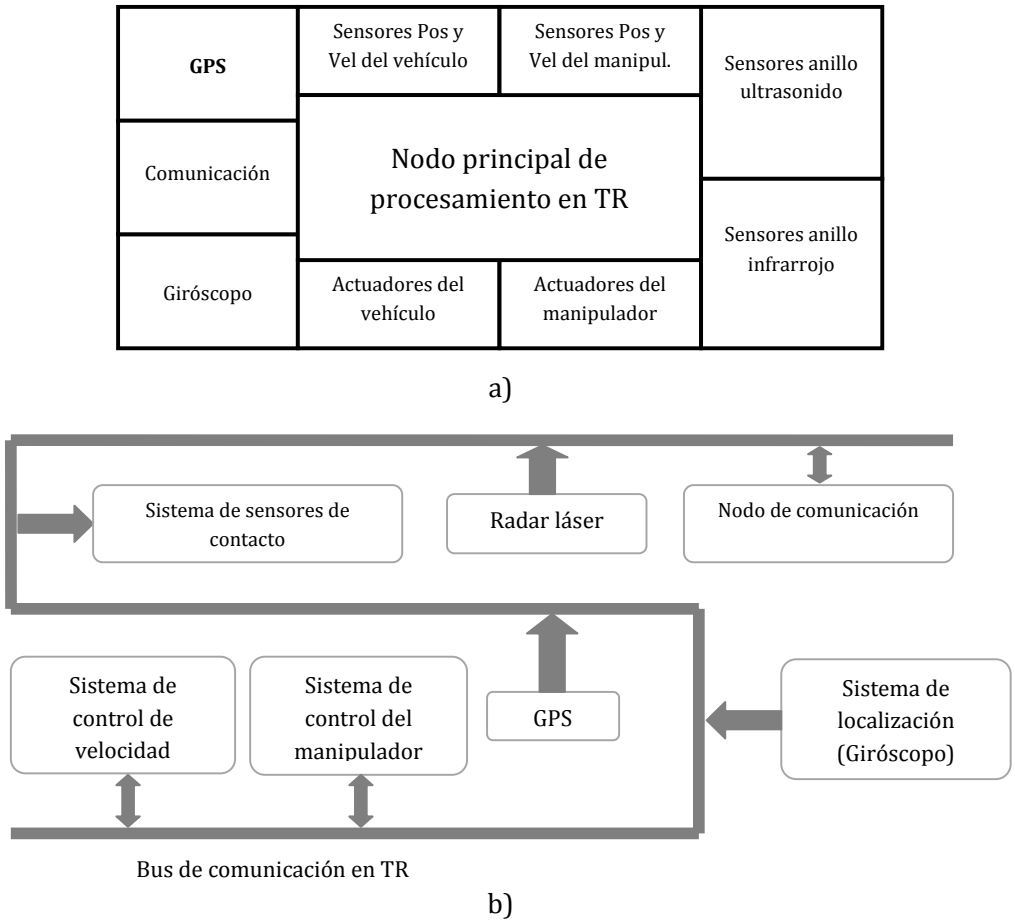


Figura 1.6. Arquitecturas del sistema electrónico de un robot móvil: a)centralizada o b)distribuida, ambas con respuesta en tiempo real (TR)

La autonomía se logra con un sistema sensorial que permite captar tanto el interior como el entorno alrededor del robot y un "Sistema de procesamiento y toma de decisiones de alto nivel", organizado en una determinada arquitectura cognitiva o de control; finalmente una interacción directa con el entorno se logra cuando el vehículo dispone de manipuladores y/o efectores finales. Cuando el Sistema de procesamiento y toma de decisiones de alto nivel recibe el flujo de datos del sistema sensorial, dispone de capacidades de procesamiento como fusión sensorial, generación de mapas, planificación de rutas, toma de decisiones inteligentes, etc., todo esto organizado según

la funcionalidad de una determinada arquitectura cognitiva y, adicionalmente, en intervalos de tiempo acotados, genera comandos útiles para el sistema de control brindando capacidad de autonomía al robot móvil, ver figura 1.7 [1].

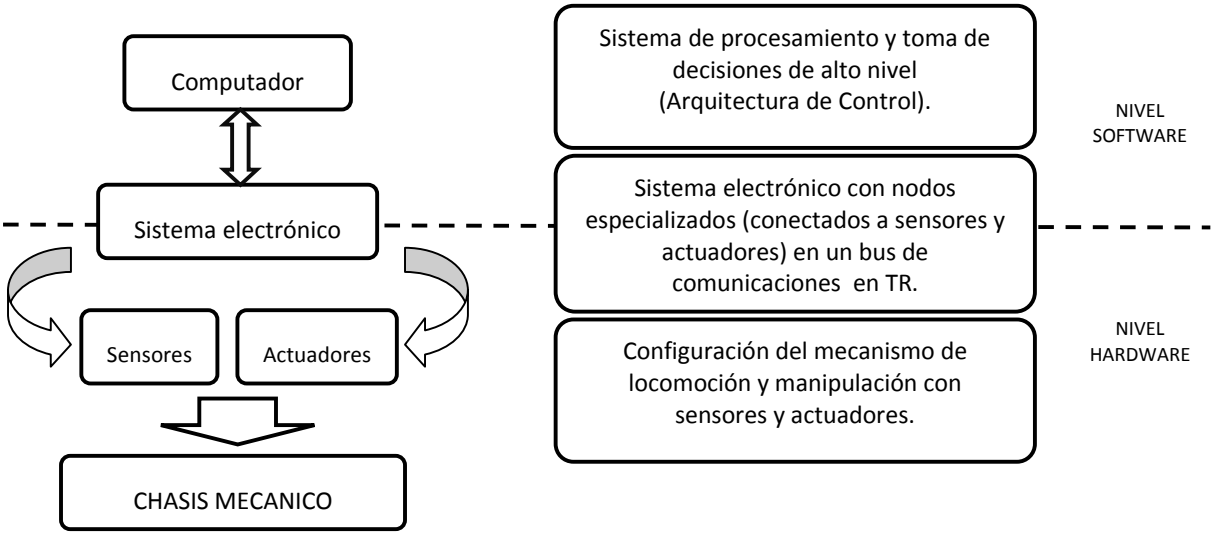


Figura 1.7. Integración mecanismo de locomoción, sistema electrónico y sistema de procesamiento

Como se mencionó previamente la autonomía de un robot móvil es una de sus características clave. Ésta se refleja en su habilidad más importante, la navegación vista desde dos perspectivas: una reactiva evadiendo obstáculos y una deliberativa planificando rutas. La navegación se puede definir como la combinación de tres funciones fundamentales: la construcción de mapas, que es el proceso de realizar mapas a partir de las medidas de los sensores del robot en diferentes posiciones; la localización del robot, que es el proceso de conocer la posición y orientación actual del robot a partir de las medidas de los sensores y el mapa actual; y la generación de trayectorias, que es el proceso de planificar una trayectoria factible y segura desde una determinada localización a otra objetivo a partir del mapa actual. Precisamente la principal parte de la investigación que actualmente se realiza en robótica móvil se concentra en la combinación de las dos primeras funciones fundamentales: localización y generación de mapas, más conocido como SLAM (*Simultaneos Localization And Mapping*)[3] por sus siglas en inglés y en el control y planificación de las trayectorias o movimientos que debe seguir el robot móvil. El desafío en el área de SLAM se concentra en el visual SLAM de ambientes dinámicos, mientras que el desafío en la segunda área es el desarrollo de marcos de trabajo computacionalmente eficientes que permitan un diseño de control

sistemático que se adecúe a las restricciones y complejidad del robot, mientras al mismo tiempo se brinda una buena expresividad al lenguaje de alto nivel, similar al humano, empleado en la especificación de la tarea que debe realizar el robot [3].

Finalmente el disponer de un sistema de locomoción y un sistema electrónico (actuadores y sensores) integrados en una máquina no conforman *per se* un robot móvil. La inteligencia para permitir la navegación autónoma debe estar organizada según una arquitectura de procesamiento cognitivo, también conocida como arquitectura de control, y adicionalmente, esta "inteligencia" debe permitir su monitoreo y comunicación expedita. Esta arquitectura de control se implementa en una determinada arquitectura software de procesamiento que normalmente reside, en la mayoría de los robots, en uno o, en pocos casos, varios nodos de computación. Este nodo es un sistema hardware de cómputo externo al sistema electrónico, ver figura 1.7. Este sistema de cómputo normalmente es un computador industrial, o simplemente un computador portátil, comunicado con el sistema electrónico a través de un puerto de comunicaciones estándar, normalmente RS232 o USB o bien, en ciertos casos, es una red de computadores. En este sistema de cómputo no reside, necesariamente, todo el sistema de procesamiento cognitivo software del robot móvil. Una parte importante, la relacionada con los algoritmos de bajo nivel para el control de velocidad y posición residen en el nodo principal o en nodos especializados del sistema electrónico.

En este computador o red de computadores se instala una plataforma de trabajo en robótica móvil, esto es, un entorno de desarrollo software especializado en robótica móvil, el cual en primera instancia debe interactuar con el sistema electrónico del robot en forma transparente; en segunda instancia, debe brindar herramientas para diseñar e implementar cualquier arquitectura de control, independiente de si los elementos de la arquitectura se encuentran en un mismo nodo o distribuidos en varios nodos de cómputo y; en tercera instancia, proporcionar aplicaciones para simulación 2D y 3D del robot, comunicación hombre - máquina de alto nivel (identificación visual del usuario, generación de voz artificial e identificación de comandos orales) y una ejecución, monitoreo y supervisión en línea del estado de todo el sistema.

1.2 Arquitecturas de Control

Históricamente el diseño de arquitecturas de control aplicadas a la robótica móvil se vio influenciado por los principios de la Inteligencia Artificial, de las décadas del cincuenta y

sesenta, en cabeza de los investigadores John McCarthy y Marvin Minsky [27]. Desde esa época el hombre busca emular en máquinas de cómputo electrónico el razonamiento humano diseñando arquitecturas para organizar el conocimiento y toma de decisiones apoyadas en teorías cognitivas tomadas de la sicología del ser humano y de la lógica matemática de primer y segundo orden. Básicamente la influencia de los investigadores de la IA se advierte en dos enfoques: primero, sensar para modelar lo más fielmente el mundo del robot, y segundo, deliberar o discutir, empleando razonamiento lógico puro, sobre la información contenida en el modelo del mundo con el fin de diseñar planes y estrategias a largo plazo que debería, posteriormente, ejecutar el robot estrictamente en lazo abierto. Si un cambio ocurría, se debía actualizar el modelo del mundo y desarrollar un plan que incluyera el cambio para ejecutarlo. Las arquitecturas de control organizadas de esta manera se denominan deliberativas.

La arquitectura de control deliberativa inicialmente referenciada en la literatura fue la diseñada para el robot Shakey del *SRI International* (*Standford Research Institute International*) en Estados Unidos [28]. Este proyecto pionero se realizó en dos fases, la primera en los años sesenta y la segunda en los setenta, los resultados obtenidos en las dos fases fueron recopilados en el informe técnico 323 por Nils Nilsson en 1984 [28]. En este robot móvil se diseñó un complejo sistema de modelamiento lingüístico – simbólico, de su limitado y diseñado mundo en un entorno tipo *indoor*, el cual se actualizaba periódicamente y sobre el cual se razonaba lógicamente para diseñar los planes a largo plazo que debía realizar el robot. Entre las décadas del setenta y ochenta, en forma independiente al *SRI International*, la división de sistemas inteligentes del NIST (*National Institute of Standards and Technology*) en Estados Unidos coordinada por James Albus trabaja en el sistema de toma de decisiones de un robot móvil, para el cual propone a finales de los setenta un sistema software jerárquico de control de tiempo real denominado NIST/RCS o simplemente RCS [29]. Durante los ochenta este sistema fue aplicado en el control de varios vehículos militares y hacia 1986, en conjunto con la NASA, es empleado como soporte para el desarrollo de una arquitectura de control de sistemas telerobotizados que se llamo: NASREM (NASA/NBS *Standar Reference Model*). Esta arquitectura fue empleada tanto para el control en forma autónoma como el telecontrol del brazo robótico de la estación espacial [30]. Los anteriores sistemas son ejemplos clásicos y representativos de arquitecturas deliberativas caracterizadas por el minucioso modelamiento del entorno, la elaboración de planes y su estricta ejecución.

La arquitectura deliberativa tiene dos características esenciales. La primera: el flujo de control es unidireccional y lineal. Esto es, la información fluye desde los sensores hacia el modelo del mundo, de ahí a los planes y finalmente hacia los actuadores. La segunda,

la ejecución de un plan en esta arquitectura es similar a la ejecución de un programa de computador. Ejecutar un plan o un programa es fácil cuando se compara a su generación. El contenido de la información está en la estructura de la composición y no en las instrucciones empleadas. La inteligencia del sistema reside en el sistema de generación de planes (planificador) y no en el mecanismo de ejecución. Esta tendencia prevaleciente de la IA generó que durante los ochenta los investigadores se enfocaran exclusivamente en planificación y modelamiento del mundo [31]. Para mediados de los ochenta, varios autores identificaron que este enfoque tiene sus limitaciones, ya que cuando un robot móvil con sensores ruidosos se enfrenta a un entorno incierto e impredecible, la ejecución rígida de un plan en lazo abierto no tiene sentido cuando el mundo ha cambiado rápidamente o éste ha sido modelado a partir de información parcial y/o ruidosa [32], [33]. James Firby sugiere mejorar el esquema deliberativo con una propuesta que denomina "Planificación Reactiva". Esta variante en el esquema básicamente sugería modificaciones a la arquitectura deliberativa que incluían hacer uso de elementos reactivos, que denominó paquetes de acción reactiva (RPAs), para secuenciar comportamientos [32], mientras que Philip E. Agre propone emplear una teoría general de la actividad, que permite la interacción inmediata con el entorno, para adecuar los planes sin requerir hacer uso de modelos explícitos del mundo [33].

El esquema de la *Subsumption* es propuesto por Rodney Brooks en 1986 como una arquitectura solo basada en comportamientos [34]. Este esquema inicialmente hardware propone hacer una organización jerárquica en forma de comportamientos reactivos para las tareas que debe realizar un robot móvil, fue lanzado como una alternativa para mejorar la eficiencia del esquema deliberativo aplicando restricciones a las tareas ejecutadas en las diferentes capas hardware, sin embargo la diferencia notable de la propuesta de Brooks con el enfoque deliberativo es el no hacer uso de planes ni mucho menos de modelos del mundo, el sistema simplemente se compone de comportamientos organizados jerárquicamente que pueden inhibir o no a comportamiento inferiores. El mayor éxito de *subsumtion* fue logrado en el área de la navegación de robots libres de colisiones. A pesar del éxito en esta área el esquema de *subsumption* no permitía realizar metas complejas por parte de los robots debido a su carencia de una estructura modular, lo que impedía manejar de forma controlada la complejidad incremental de las diferentes tareas requeridas en otros ámbitos de aplicación [35]. Por la época siguiente a la presentación de la *Subsumption* varios autores desarrollaron nuevas arquitecturas de control, en forma más o menos independiente, [36], [37], [38], [39], [40] y [41], unos conducentes a mejorar el acercamiento de la *subsumption* como en [41] u otros integrando las mejores características de los acercamientos deliberativo y de *subsumption* en un nuevo esquema como en [38] y [40].

En 1989 Ronald C. Arkin propone hacer uso de los esquemas sensor - motor, conceptos propios de la sicología, para implementar una capa de comportamientos reactivos para el diseño de sistemas de navegación [42]. En el mismo año Ronald C. Arkin propone integrar una capa adicional donde se pueda realizar replanificacion dinámica según las mediciones que se realicen por el robot [43]. Para 1990 Ronald C. Arkin organiza sus ideas en la propuesta de una nueva arquitectura "híbrida" que integra: comportamientos, percepción del entorno y un conocimiento básico del mundo aplicado a la navegación reactiva de robots móviles [38]. Por la misma época, y durante muchos años posteriormente hasta la actualidad, varios autores iniciaron una prolífica publicación de resultados producto de implementar su propia concepción de arquitectura híbrida y a la cual etiquetaron con rimbombantes nombres o simplemente con las siglas: Erann Gat en 1992 y la arquitectura ATLANTIS [44], Jonathan Connell en 1992 y la arquitectura SSS [45] David J. Musliner en 1993 y la arquitectura CIRCA [46] , Reid G. Simmonds en 1994 y la arquitectura TCA de control estructurado [47], Charles .A. Malcom en 1995 y la arquitectura SOMASS [48], Kurt Konolige en 1996 y la arquitectura Saphira [49], el mismo Ronald C. Arkin, desde 1987, a su arquitectura híbrida le llamó AuRA [50], Peter Bonnaso en 1991 y la arquitectura 3T [51], y el listado podría aún continuar. Todas las anteriores arquitecturas "híbridas" se caracterizaban por una organización jerárquica de tres capas (a excepción de Saphira): una capa inferior donde se ejecutan mecanismos de control reactivo realimentados basados en comportamientos, que se comunican en tiempo real con el sistema electrónico del robot, una capa intermedia o de secuenciación donde se encuentra un mecanismo de ejecución de planes reactivos a base de comportamientos y una capa superior dedicada la ejecución de pesados y no acotados algoritmos deliberativos de planificación [31].

1.3 Lenguajes, Plataformas y Herramientas

La investigación con robots móviles requiere la definición conceptual de un marco cognitivo que organice el conocimiento, toma de decisiones y la actuación del robot, esto es lo que se denomina Arquitectura Cognitiva en IA o de Control en robótica móvil. Una vez se define este marco conceptual se debe llevar a cabo una implementación del mismo, esto significa, diseñar y construir un sistema software que exhiba la funcionalidad de la arquitectura de control. Normalmente este sistema software se denomina plataforma de trabajo (*framework*), la cual normalmente consiste de un sistema operativo (SO), con ciertas características temporales mejoradas a los SO convencionales, un sistema de comunicaciones y un conjunto de herramientas de apoyo al investigador en diseño, programación, depuración, simulación y supervisión de los

módulos que conforman el sistema de cognición del robot [52]. Las primeras plataformas de trabajo, desarrolladas en los ochenta y noventa, empleaban herramientas que simplemente consistían en lenguajes de programación robóticos, agrupados en lenguajes genéricos, lenguajes de propósito especial o una combinación de ambos [53] [54].

Una plataforma de trabajo software que implementa una arquitectura híbrida consiste de tres componentes uno para cada nivel: un mecanismo de control reactivo realimentado, un mecanismo de ejecución de planes reactivo y un mecanismo deliberador para realizar cálculos deliberativos [31]. Estos componentes son principalmente implementados en lenguajes de programación usando un SO multitarea o multihilos, pero también pueden ser hechos codificando los algoritmos de tal manera que estos puedan ser interlazados dentro de un proceso computacional único ejecutándose en un computador. Una plataforma de trabajo de una arquitectura híbrida organiza los algoritmos de acuerdo a si ellos contienen o no estados, contienen memorias que reflejan un estado sobre el pasado o contienen estados que reflejan predicciones sobre el futuro. Los algoritmos basados en sensores, sin estado, integran el componente de control reactivo. Los algoritmos que contienen memoria sobre el pasado integran el secuenciador. Los algoritmos que hacen predicciones sobre el futuro integran el deliberador.

Lenguajes y herramientas robóticas

El mecanismo de ejecución de planes reactivo se implementa como un lenguaje de secuenciamiento condicional de comportamientos o una programación funcional reactiva (FRP) como lo definen otros autores [54]. Este se puede realizar en un lenguaje de propósito general como C o un lenguaje especializado como Haskell o Lisp, sin embargo se pueden requerir constructores especiales, por lo que los primeros investigadores diseñan lenguajes de propósito especial, ejemplos de estos son: RAPs de James Firby [55], PRS de Michael Georgeff [39], ALFA de Erann Gat [56], REX/GAPPS de Leslie Pack Kaelbling [36], Colbert de Kurt Knonolige [57], SIGNAl de P. LeGuernic [58], CHARON de Jean Luc Fleureau [59], finalmente los lenguajes FROB de Jhon Peterson [60] y Yampa de Hudak [61] se han construido con Haskell, y los lenguajes ESL de Erann Gat [62] y Behavior de Rodney Brooks [63] se soportan en el lenguaje Lisp.

Para el mecanismo de cálculos deliberativos, se ejecutan algoritmos dedicados a la

planificación, algoritmos de búsqueda en navegación (exponenciales o polinomiales con grandes constantes de tiempo) y en ciertos casos algoritmos de procesamiento digital de imágenes. El deliberador puede correr como uno o más hilos separados de ejecución. Generalmente no hay restricciones en los algoritmos que se pueden ejecutar en este mecanismo, por lo que puede ser implementado usando un lenguaje de programación estándar. El deliberador puede comunicarse con el resto de los elementos del sistema de dos formas diferentes: la primera, generando planes para que el mecanismo de secuenciamiento los ejecute o, la segunda, respondiendo a solicitudes específicas del secuenciador [31]. Esta característica deliberativa de este último elemento lo hacía propicio para que fuese programado con lenguajes procedurales convencionales como C, C++ y Ada, sin embargo se empleaba Lisp y en algunos casos se implementaron lenguajes de alto nivel basados en objetivos como RPL, PRS [64], TDL[65] y el lenguaje MAESTRO del INRIA [66].

Los robots móviles de finales de los ochenta y principios de los noventa buscaban experimentar con programación reactiva funcional en diferentes arquitecturas híbridas. La plataforma software de estos primeros robots consistía básicamente en el entorno de programación propio de un lenguaje de programación genérico (C), de un lenguaje especializado (Lisp) o de un lenguaje específico (RAP), sin mayores herramientas de apoyo que las ofrecidas por estos entornos como por ejemplo el *Debugger*. Sin embargo a medida que se diseñaban lenguajes específicos para esta área de aplicación se desarrollaron nuevas y mejoras herramientas por los diferentes centros de investigación.

La arquitectura ATLANTIS de Erann Gat se empleó en varios robots, en ésta el nivel controlador se implementaba con el lenguaje ALFA en un PC *Gespack* 68000 y los niveles superiores con *Macintosh Common Lisp* (MCL3.0) multi hilos sobre un PC *PowerBook Duo* 230. Los computadores se comunicaban por enlace Rs232 [31]. La arquitectura AuRA de Ronald C. Arkin fue empleada en múltiples robots utilizando el lenguaje Lisp, el lenguaje BHDL y el *MissionLab Simulator* [50]. La arquitectura CIRCA proponía un diseño de dos sistemas cooperativos: un sistema de Tiempo Real y un sistema de Inteligencia Artificial, empleando interfaces de comunicación entre los dos sistemas y utilizando Lisp [46]. La arquitectura 3T de Peter Bonnaso empleaba tres herramientas software para implementar la arquitectura en cada nivel: el *Skill Manager* que se compilaba en las tarjetas de los robots y un interpretador de RAPs y un planificador AP para los niveles de secuenciamiento y planificación que se comunican por TCP/IP [51]. La arquitectura TCA de Reid Simmonds es otro ejemplo de arquitectura de tres capas, pero la plataforma software que permite su implementación provee de constructores de control (módulos C o Lisp) para el desarrollo de

comportamientos deliberativos y reactivos, en particular los constructores brindan soporte para: 1) una comunicación interprocesos distribuida, 2) descomposición de tareas y restricción temporal entre subtareas, 3) separación y administración de recursos, 4) monitoreo de la ejecución y 5) manejo de excepciones [47]. La herramienta *GenoM* del LAAS-CNRS es una herramienta para especificar e integrar módulos funcionales en una arquitectura híbrida distribuida, cada módulo es especificado en un lenguaje y es automáticamente generado por *GenoM*, éste también produce un programa de test interactivo y librerías de interfaz para controlar el módulo y leer los datos resultantes. El código producido puede ser ejecutado en una estación de trabajo Unix para simulación o correr embebido en el sistema operativo de tiempo real *VxWorks* [67]. El lenguaje MAESTRO y sus herramientas se incrustaron en el proyecto ORCCAD del INRIA, constituyéndose en una completa plataforma para robótica de tiempo real, soportado en componentes básicos como las *Robot Task* (RT) y los *Robot Procedures* (RP) junto a una metodología de diseño orientada a objetos, permite diseñar (*Orccad Interface*), verificar (*Orccad Verif* con las herramientas *Auto* y *Autograph*), simular (con el simulador *Simparc*) y ejecutar una aplicación robótica (*Orccad Exec* generando código de C++ para integrar con librerías RT del sistema operativo RT *VxWorks*) [68], [66]. La arquitectura Saphira se implementa como un sistema de control por agentes móviles que se comunica con el hardware por un modelo de comunicaciones cliente – servidor. De interés particular en Saphira son sus elaboradas interfaces hombre-máquina consistentes en reconocimiento de voz (herramienta CORONA del SRI), seguimiento visual de personas y un programa estándar de conversión de texto a voz [49].

Plataformas y herramientas robóticas

Para finales del noventa y principios del siglo veintiuno se presenta una explosión de plataformas de trabajo y herramientas software en robótica móvil, en formas diversas, desde completos entornos de programación software (IDE), plataformas *middleware* para comunicación de componentes robóticos, herramientas de simulación 2D y 3D hasta simples librerías especializadas. Todos estos enmarcados en proyectos libres de fuente abierta (FOSS) realizados por centros de investigación en proyectos de doctorado o financiados por proyectos de I+D internacionales hasta proyectos de interés comercial por parte de la industria.

Dentro de los proyectos realizados por universidades y centros de investigación se encuentran:

- El proyecto francés GenoM desde 1997 dirigido por Sara Fleury en el LAAS [67][69].
- La plataforma de trabajo MissionLab desde 1998 dirigida por el profesor Ronald C. Arkin del Georgia Tech [70].
- La plataforma sueca BERRA de 1999 dirigida por Anders Oreback del Centro de Sistemas Autónomos del Instituto Royal de Tecnología [71].
- El proyecto alemán *middleware* para aplicaciones robóticas y de control MiRPA de 1999 dirigido por el profesor Torsten Kroeger de la Universidad Técnica de *Braunschweig* [72].
- El repositorio software del proyecto CLARAty en el 1999 dirigido por Issa Nesnas de la NASA [73].
- El proyecto alemán SmartSoft en el 1999 dirigido por Christian Schlegel de la Universidad de Ciencias Aplicadas de *Ulm* [74].
- El proyecto alemán OROCOS de automatización robótica de 1999 dirigido por Christian Schlegel [75].
- El popular proyecto PSG - Player/ Stage / Gazebo en el 2000 dirigido por Brian Gerkey [76][77][78].
- El proyecto de herramientas y paquetes TeamBots en el año 2000 dirigido por el investigador Tucker Balch del *Carnegie Mellon University* (CMU) [79].
- El proyecto Pyro en el 2000 dirigido por Douglas Blank del *Bryn Mawr College* [80].
- El proyecto de integración MARIE en el 2000 dirigido por Carle Côté de la Universidad de Sherbrooke[81][82].
- El proyecto japonés RT-Middleware en el 2000 dirigido por el profesor Kazuo Tanie del instituto japonés de Ciencia y Tecnología Industrial Avanzada [83][84].
- El proyecto alemán MCA(2) del 2000 dirigido por Kay U. Scholl en el Departamento de Informática de la Universidad Krlsruhe[85].
- El proyecto Frances OpenRobots desde el 2000 dirigido por Sara Fleury en el Laboratorio de Análisis de Arquitecturas de Sistemas del Centro Nacional Frances de Investigación Cientifica (LAAS/CNR) [86].
- El proyecto RTAI del 2000 dirigido por el profesor Paolo Mantegazza del Politécnico de Milano en Italia [87].
- La herramienta de simulación ThinkingCap-II en el 2001 desarrollada por Humberto Martinez en la Universidad de Murcia [88].

- El proyecto alemán de futbol robótico MIRO en el 2002 dirigido por Hanz Utz de la Universidad de *Ulm* [89].
- La plataforma SC-Agent en el 2003 de Juan Luis Posada en la Universidad Politécnica de Sevilla [90].
- El proyecto de integración CARMEN en el 2003 dirigido por Michael Montemerlo del *Carnegie Mellon University* [91].
- La plataforma de trabajo ROCI del 2003 dirigido por Luiz Chaimowicz de la Universidad de Pennsylvania [92].
- El proyecto ORCA, posteriormente ORCA2, de 2004 dirigido por Anders Oreback [93], [94].
- La plataforma de agentes cognitivos robóticos ADE en el 2004 dirigido por Mattias Sheutz de la Universidad de Indiana [95].
- La plataforma de trabajo CoolBOT en el 2004 dirigido por Antonio Domínguez de la Universidad de las Palmas de Gran Canaria [96].
- El kit de herramientas RobotFlow del 2005 dirigido por Dominic Létourneau de la Universidad Sherbrook [97].
- El proyecto alemán RoboFrame del 2005 dirigido por el profesor Sebastian Petters en la Universidad Técnica de *Darmstadf* [98].
- El ambiente de desarrollo robótico WURDE en el 2005 dirigido por Frederick Heckel de la Universidad de Washington [99].
- La plataforma robótica Urbi en el 2006 dirigido por Jean C. Baillie de la empresa francesa Gostai [100].

El anterior listado organizado cronológicamente debe verse como una limitada muestra de la amplia variedad de herramientas que, a la fecha, se encuentran disponibles, en forma libre, para solucionar desde el componente software más elemental hasta el más complejo en el trabajo investigativo y práctico con robots móviles.

Dentro de las plataformas o herramientas comerciales para el trabajo con robots móviles, la oferta es igualmente amplia: ARIA de la empresa *MobileRobots* antes conocida como *Actimedia* [101]; la plataforma *Robotic Developer Studio* de Microsoft [102]; la plataforma Webots de la empresa *Cyberbotics* [103]; el ambiente de desarrollo PC-BOT de la empresa *White Box Robotics* [104]; la plataforma de desarrollo robótico ERSP de la empresa *Evolution Robotics* [105]; la plataforma lego Mindstorms de la compañia LEGO [106]; por citar solo unas pocas referencias de las múltiples opciones comerciales disponibles en el mercado.

1.4 Esfuerzos de Comparación y Estandarización

La década del ochenta se constituyó en la caída del enfoque dominante de la IA sobre la robótica móvil con la arquitectura deliberativa, la fugaz aparición de la arquitectura reactiva y el nacimiento de la arquitectura híbrida que combinaba lo mejor de las arquitecturas previas. La década del noventa fueron los años de posicionamiento de la arquitectura híbrida, como las múltiples implementaciones de la misma así lo indican, el despertar del interés comercial en la robótica de servicios, pero también fueron los años de la identificación de importantes problemas en la investigación y trabajo en robótica móvil.

Estos problemas se pueden resumir en la carencia de una formalización y estandarización de las diferentes estructuras modulares y estilos de comunicación empleados tanto en las arquitecturas de control como en las implementaciones de las mismas. Es importante ver las diferentes plataformas de trabajo planteadas no como entidades monolíticas aisladas, sino como partes de una solución más completa. De esta forma los mejores aspectos de las diferentes implementaciones pueden ser identificados, aislados y empaquetados, de tal forma que ellos puedan ser empleados de una forma *plug and play* por otros investigadores en otras plataformas con otras arquitecturas. En la actualidad esto no es posible. La principal razón es que el nivel de abstracción empleado en la implementación de las diferentes arquitecturas está lejos de ser compatible. La comunidad de investigadores en robótica móvil requiere con urgencia definir y estandarizar "interfaces" entre los diferentes niveles de las arquitecturas independiente de la estructura de la arquitectura, esto es su organización modular, y del estilo de la arquitectura, esto es los diferentes mecanismos de comunicación empleados en la implementación de la arquitectura [107].

Esfuerzos de estandarización para atender los problemas planteados se han realizado desde diversos frentes. Un primer frente es la academia, donde los investigadores en aras de resaltar algún tipo de aporte han realizado estudios comparativos, tanto cualitativos como cuantitativos, de lenguajes, arquitecturas y *middlewares*. Continuando con la academia, otros investigadores han propiciado la realización de páneles, encuentros, comités y similares en conferencias, simposios, etc. con el fin de compartir inquietudes y proponer soluciones al tenor. Fruto de esto eventos se han formado diferentes asociaciones, grupos de trabajo y similares con el fin de definir estándares o simplemente reunir y analizar información útil. Ejemplos de tales asociaciones son: la organización de estándares y arquitecturas de referencia para robótica- RoSta [108], la red europea de investigación robótica – EURON [109], y el comité de actividades de estandarización de la RAS de la IEEE [110]. Un segundo frente es el del sector

gubernamental y militar de los países del primer mundo, específicamente en Estados Unidos, donde también se han ocupado de esta situación entidades como el NIST proponiendo su arquitectura por modelo de referencia 4D/RCS [111] para vehículos terrestres no comandados (UGV) inteligentes y el Departamento de Defensa (DoD) de los Estados Unidos quien propuso un marco de trabajo arquitectural llamado C4ISR (*Command, Control, Communications, Computers, Intelligence, Surveillance y Reconnaissance*), con el propósito de permitir la creación e integración de las arquitecturas que se usan en los diferentes niveles jerárquicos de las entidades gubernamentales, militares y civiles que hacen parte del DoD de los Estados Unidos [112]. Dentro de estas arquitecturas se incluye JAUS (*Joint Architecture for Unmanned Systems*) para los UGV ahora manejado por SAE (*Society of Automotive Engineers International*) [113].

Estudios de Lenguajes en Robótica

En los años ochenta y noventa se diseñaron una multiplicidad de lenguajes de programación tanto de propósito general como de dominio específico para la robótica móvil y con ellos se implementaron una amplia variedad de arquitecturas de control. El cuestionamiento que se hacían los investigadores es ¿cómo hacer mediciones entre lenguajes y arquitecturas y en qué dimensiones con el fin de obtener métricas confiables que permitiesen hacer una evaluación justa de herramientas o de algoritmos? Esta es aún una pregunta sin una respuesta definitiva y es la razón de ser de las diferentes propuestas de estandarización no solo aplicadas a la robótica sino a la integración empresarial en general. Los primeros intentos de comparación entre lenguajes de programación aplicados a la robótica móvil se realizaron en los ochenta con el trabajo de Tomas Lozano Perez en el MIT [114]. En aquel entonces los robots básicamente se encontraban en ambientes industriales y este estudio se concentra en lenguajes de sistemas de programación de robots industriales. Varios años después aparece el trabajo de Izzet Pembeci realizado en el *Computational Interaction and Robotics Lab* (CIRL) en la Universidad Johns Hopkins en el año 2002 [115], donde hace una evaluación comparativa de características clave de ocho (8) lenguajes de programación del nivel intermedio para robots móviles: RPAs, *Behavior Languaje*, Charon, Colbert, PRS, Signal, TDL y Haskell. Finalmente el trabajo de revisión de programación de robots de Geoffrey Biggs y Bruce MacDonald de la Universidad de *Auckland* en el año 2003 [116], en este trabajo se realiza una clasificación de la programación de los robots en: manual y automática. Dentro de la clasificación manual

destaca la importancia que tiene la programación basada en texto, principalmente de los lenguajes basados en comportamiento, y la nueva programación gráfica.

Estudios de Arquitecturas en Robótica

En la primera década del siglo veintiuno se iniciaron los trabajos comparativos entre arquitecturas. El primer y más completo realizado a la fecha es el Anders Oreback y Henrik I. Cristensen en el Centro de Sistemas Autónomos del Instituto de Tecnología Real en Suecia [117]. En este trabajo los autores comparan tres implementaciones de las arquitecturas: BERRA, TeamBots y Saphira sobre una misma plataforma robot, el *Super Scout* de la empresa Nomadic. Un trabajo algo similar se realizó en Inglaterra en el año 2005 a raíz del proyecto cognitivo COSY, donde realizaron una prueba de selección entre las arquitecturas MARIE y OAA [118]. En el año 2004 varios autores de las Universidades de Bielefeld y de Viena realizan un estudio de cuatro implementaciones de arquitecturas: ImaLab, ICE, SmartSoft y Zwork, buscando resaltar sus fortalezas y debilidades para desarrollar sistemas de visión cognitivos [119]. En el año 2006 Pat Langley y otros en el *Computational Learning Laboratory* en la Universidad de Stanford realizan una investigación sobre los desafíos y aspectos por cubrir en arquitecturas cognitivas [120]. En este trabajo los autores analizan las arquitecturas PRODIGY, Icarus, Soar y ACT-R. En el año 2007 los profesores James Kramer y Matthias Scheutz del *Artificial Intelligence and Robotics Lab* de la Universdad Notre Dame, realizan un exhaustivo análisis de nueve (9) ambientes de desarrollo software que permiten implementar diversas arquitecturas de control [121], comparándose: TeamBots, ARIA, Player/Stage, Pyro, CARMEN, MissionLab, ADE, MIRO y MARIE. Finalmente en el año 2007 los autores Azamat y Erwin realizan, en el Centro para la Tecnología de la Información del Instituto de Ciencias Aplicadas en Alemania, una evaluación comparativa quantitativa de tres sistemas de integración software robóticos: GenoM, Orca2 y Go [122].

Estudios de *Middlewares* en Robótica

A finales del noventa se hizo palpable que el origen de los problemas prácticos del

trabajo e investigación en robótica se centraban en la infraestructura de comunicación y distribución de las plataformas software de programación empleadas en la implementación de las arquitecturas de control. El caso representativo de esta situación es el de la popular arquitectura de los años noventa Saphira, la cual en su implementación software integraba los aspectos de control distribuido y comunicación interprocesos de tipo propietario con los aspectos propios de la arquitectura de control, lo cual hacía imposible el uso de la plataforma por parte del investigador sin que tuviese un dominio de la programación de aplicaciones distribuidas [123]. Esto condujo a determinar que el enfoque que se debía asumir por parte de los diseñadores de las plataformas de programación robóticas, consistía en realizar una clara separación entre el nivel de los módulos o componentes propios de la arquitectura de control y el nivel de la infraestructura de comunicación distribuida requerida por los componentes de la arquitectura. A esta infraestructura de comunicaciones se le denomina *middleware*. Uno de los primeros trabajos comparativos de *middleware* es el realizado por Jay Gowdy en el Instituto de Robótica del Carnegie Mellon en el 2000 [124], en este trabajo se comparan varias herramientas para comunicación interprocesos empleadas en los noventa: IPT, RTC, NML, NDDS, MPI y CORBA. Posteriormente en el año 2002 el profesor William D. Smart en el Departamento de Ciencias de la Computación de la Universidad de Washington realiza una comparación entre TAO y una típica transmisión de datos vía *sockets* [125]. En el año 2005 los investigadores Greg Broten y otros del *Defense Research and Development Canada*(DRDC), realizan una investigación aplicada sobre sistemas software para robótica móvil [126], en este trabajo se estudiaron especialmente las tecnologías *middleware* de las plataformas Claraty, Miro y Player, entre otras, con el fin de seleccionar el marco de trabajo para las variadas plataformas robóticas militares canadienses del DRDC. En el año 2008 se realizan más estudios comparando cualitativamente las tecnologías *middleware* subyacentes en varias plataformas de programación robóticas. Uno de estos es el de Nader y otros en la Universidad de los Emiratos Árabes Unidos [127], donde repasan las tecnologías *middleware* de quince (15) plataformas: Miro, Orca, UPnP *robot Middleware*, RT-Middleware, Aseba, Player, el kernel PEIS, Orin, Marie, RSCA, el *middleware* AWARE y otros. Otro trabajo es el de Molaletsa Namoshe y otros en el Departamento de Ciencia de Materiales y Manufactura de Sudáfrica en el 2008 [128], donde realizan un estudio de selección de la *middleware* más adecuada de las plataformas Miro, Marie, Orca2 y Player para emplearlas como marco de trabajo software en sus robots de laboratorio.

Eventos y Organizaciones de Estandarización en Robótica

Los estudios comparativos llevaron a evaluar más objetivamente tanto las arquitecturas como los *middleware*, buscando identificar cuáles deberían ser las métricas adecuadas que debían tenerse en cuenta para comparar herramientas o solucionar otros tipos de problemas. Varios eventos en el mundo se organizaron con tal finalidad, dos ejemplos de tales reuniones fueron: B-IT *Tutorial Robot Middleware and Integration Frameworks*, realizado de la Universidad de Ciencias Aplicadas en Alemania en el 2005, donde se invitaron autores de las plataformas más representativas, según los organizadores (Orca, Miro, Player, MCA2, Marie y SmartSoft), con el fin de presentar un caso de uso donde integraban movilidad y manipulación en un vehículo. El segundo evento es una sesión especial sobre estándares en robótica móvil que se abrió en el IROS 2007, en esta sesión se presentaron nuevas *middlewares* para robots: RoboFrame [98], Aseba [25], MiRPA [72] y diversos trabajos sobre medición de parámetros comparativos [129], [122], [130], [131] y propuestas de estándares [132].

Igualmente con los eventos se han formado varias organizaciones académicas, en algunos casos vinculadas con la industria, con el fin de conformar grupos de trabajo que reúnan información y propongan recomendaciones técnicas. Ejemplos de tales organizaciones son: TCU Planet, RoSTa y EURON. En el caso de TCU Planet (*Technical Coordination Unit for On Line Planning and Scheduling*) es la red europea de excelencia en *planning* la cual ayuda a coordinar a los miembros de la comunidad europea con un interés en tecnologías de *planning* y *scheduling*. Trabajó hasta el 2002 y estaba conformada por nodos de varias universidades europeas, emitió varios documentos sobre sistemas de planificación en línea, donde varios de los proyectos reportados eran en robótica móvil [133]. La organización de estándares y arquitecturas de referencia para robótica RoSta (*Robot Standards and Reference Architecture*) se conformó a partir de una propuesta, de acción coordinada europea para trabajar en un arquitectura de referencia y estándares para robots móviles, que se presentó al sexto programa marco de trabajo de la Unión Europea. Es un grupo aún activo aunque tiene mucho trabajo por realizar [108]. Finalmente está la red académica EURON, esta es una red europea activa que busca financiar estudios de prospectiva en diferentes áreas de la robótica. Esta tiene varios grupos de interés especial, particularmente: ROSE (*a Robotics Ontology for the semantic web*), *Network Robot Systems* y *Cooprob* (*Cooperative robotics*) [109].

A pesar de la carencia de estándares generales en la robótica móvil se cuenta con algunos antecedentes, por ejemplo el grupo de trabajo JAUS, que son las siglas en inglés para *Joint Architecture for Unmanned Systems*, inició actividades en 1998 como una

organización que combinaba la industria, el gobierno y la academia para proveer interoperabilidad e inserción de tecnología en el campo de los UGV, un tipo de robot móvil militar. Este grupo produjo las especificaciones JAUS, un grupo de documentos consistentes en una arquitectura de referencia, una especificación de conformidad al igual que un modelo de dominio. Sin embargo en el 2004 JAUS inició el proceso de transición a una organización internacional de estándares civiles SAE. En este momento JAUS está definido como un conjunto de los múltiples estándares de SAE, y a pesar de que el trabajo de estandarización en JAUS continúa, ahora los documentos no son de libre acceso [113]. La División de Sistemas Inteligentes del NIST en los Estados Unidos también ha estado trabajando desde los ochenta en una arquitectura por modelo de referencia, inicialmente conocida como RCS y ahora 4D/RCS, para robots móviles militares [111]. Otros ejemplos de estándares, que son aplicados a la robótica móvil de robots cooperativos, se encuentran en los sistemas multi agentes (MAS), de los cuales existen dos: el estándar FIPA (*Foundation for Intelligent Physical Agents*) [134] siendo el más ampliamente usado y el estándar MAF (*Mobile Agent Facility*) de la OMG [135]. En ambos estándares mediante la definición y caracterización de un conjunto de componentes software, se proporciona la especificación de una plataforma software necesaria para la ejecución y movilidad de los agentes. Finalmente un ejemplo de estándar gubernamental y militar es el marco arquitectural C4ISR (*Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance*) del Departamento de Defensa de los Estados Unidos. El DoD tiene una gran cantidad de entidades que generan y comparten información vital para la defensa de ese país. Cada una de estas entidades tiene su propio problema de comunicación y procesamiento por lo que emplean arquitecturas específicas. JAUS hace parte de estas arquitecturas, en este caso para robots móviles militares del DoD. C4ISR fue creado para proveer un estándar y un acercamiento arquitectural coordinado para que las diferentes agencias y servicios militares del DoD desarrollen su arquitectura particular de soporte y combate, incluyendo sistemas no comandados (UGV) y cualquier otra clase de sistemas robóticos [112]. Este se basa en el modelo de referencia LISI (*Level of Information Systems Interoperability*) [136], con lo cual busca que las arquitecturas que lo apliquen puedan ser fácilmente integradas y se permitan operaciones conjuntas, que las mismas sean presentadas en una forma consistente y para que sus componentes arquitecturales puedan ser reusados.

1.5 Resumen

Como se puede apreciar por lo expuesto en este primer capítulo un robot móvil puede llegar a ser un sistema con elementos mecánico, electrónico y cognitivo altamente complejo. Un robot es un sistema donde interactúan tanto sistemas da datos continuos como sistemas a eventos discretos. Este debe cumplir con exigentes objetivos de alto nivel, interactuar en un ambiente dinámico y complejo, manejar datos sensoriales con ruido e incertidumbre y, en muchas situaciones, ser reactivo a eventos no esperados. Toda esta información debe fluir rápida y oportunamente entre todos los niveles del sistema de control y toma de decisiones. Estas características influyen en la manera en cómo son diseñadas las arquitecturas de control, cómo pueden operar y cómo son validadas. Adicionalmente muchos de estos sistemas robóticos pueden catalogarse como sistemas de misión crítica, por ejemplo los UGV o los robots espaciales, por lo que es de obligatorio cumplimiento la total garantía en la seguridad del sistema, por lo que su diseño se debe enfocar en nunca fallar en detectar una situación anómala más que a evitar las fallas, y esta característica de diseño se debe ver reflejada en la arquitectura de control.

El problema de la complejidad presente en un robot móvil se maneja a través de una arquitectura cognitiva o de control adecuada para el sistema de procesamiento y toma de decisiones de alto nivel del robot. Otra manera de tratar la complejidad es que junto con las arquitecturas se proporcionen lenguajes de descripción y diferentes herramientas de análisis, verificación y validación. Tales lenguajes y herramientas, cuando se soportan en marcos teóricos sólidos, proporcionan constructores que facilitan el diseño, mientras ocultan la complejidad de los conceptos subyacentes.

Existen tres paradigmas de control robótico desde las que van orientadas a objetivos hasta las enfocadas a comportamientos y finalmente las híbridas, con diferentes formas de realizar la descomposición de sus módulos internos: temporal, espacial o por tareas. De cada paradigma hay múltiples implementaciones, aunque en la actualidad prevalecen las implementaciones de paradigmas híbridos, y con variados lenguajes, desde lenguajes simbólicos para descripción del comportamiento en tiempo real o ambientes de programación gráficos para especificación de la misión. Igualmente con diferentes herramientas, unas que permiten la detección y manejo de condiciones excepcionales, análisis de la respuesta del sistema en tiempo real o la validación individual o global del sistema. En los principales marcos de trabajo robóticos de la actualidad es palpable el enfoque a no atarse o ceñirse a un paradigma en particular, se especializan en disponer una adecuada *middleware* o infraestructura de comunicaciones distribuida y facilitar el uso de diversos tipos de módulos genéricos y la creación de

nuevos módulos especializados, permitiendo la implementación de cualquier arquitectura de control.

Lamentablemente la falta de recomendaciones técnicas universalmente aceptadas por la comunidad robótica, dificultad la reusabilidad de algoritmos y herramientas, por lo que existen múltiples trabajos en realizar la integración de estas herramientas dispersas en *toolkits* robóticos, hacer estudios comparativos de lenguajes, arquitecturas y entornos de desarrollo robóticos, igualmente abundan los trabajos de métricas de medición para lenguajes, arquitecturas y herramientas. Solamente en la actualidad se han conformado organizaciones de estandarización, entre las cuales ROsTA (*Robot Standards and Reference Architecture*) es la que realiza el trabajo más relacionado con el tema expuesto en este documento. No existe entonces un panorama claro que permita la fácil selección entre las innumerables implementaciones de arquitecturas, *middlewares* o entornos de desarrollo software disponibles, ni mucho menos medir los alcances de las diferentes herramientas presentes con el sistema, con indicadores cuantitativos del comportamiento del sistema, y aún más importante , no se cuentan con mecanismos universalmente aceptados para compartir e integrar los diferentes módulos software entre realizaciones arquitecturales.

1.6 Organización de la tesis

El trabajo está organizado en cinco capítulos. En el primer capítulo se realiza inicialmente una descripción de lo que implica física y lógicamente un robot móvil, posteriormente se realiza una revisión histórica del estado del arte de las arquitecturas de control y se finaliza con un segundo recorrido histórico del estado del arte de los lenguajes, herramientas, entornos de desarrollo, de estudios comparativos y de los comités de estandarización relacionados desde la década del ochenta hasta la primera década del siglo veintiuno, dentro de las que se enmarcan las aportaciones del trabajo realizado.

En el segundo capítulo, se presentan un estudio de las características representativas en cada uno de los tres paradigmas de control disponibles para los robots móviles. Para el paradigma deliberativo: la arquitectura de control del robot Shakey y la arquitectura NASREM; en el paradigma reactivo las arquitectura de *subsumption* y DAMN; y en el híbrido la arquitecturas AURA, 3T y Saphira. Finalmente en la nueva clasificación de arquitecturas multiagente propuesta: APOC, esquemas de COSY y 4D/RCS.

En el capítulo tres, se exponen las características más sobresalientes de los principales entornos de desarrollo software disponible para la implementación de arquitecturas de control. Se estudian: TCA-IPC/TDL, TeamBots, P/S/G, GeNom, SmartSoft, Balt&Cast, RT-Middleware, MIRO, MARIE, ORCA2 y ADE.

El cuarto capítulo, está dedicado al diseño de la propuesta Arquitectura de Control Genérica Multiagente (AGC-MAS) para robótica móvil. Se divide esta parte en una corta sección de trabajos antecedentes en los que se apoya la propuesta y finalmente se realiza el diseño de esta arquitectura genérica.

El capítulo cinco contiene una exposición de las conclusiones y futuros trabajos que se pueden abordar sobre temas relacionados con el desarrollo presentado en este trabajo de maestría.

El capítulo seis se compone de todas las referencias consultadas para la elaboración del presente trabajo.

En el anexo 1 se detallan las características técnicas de las arquitecturas del paradigma deliberativo: la arquitectura de control del robot Shakey y la arquitectura NASREM; del paradigma reactivo las arquitectura de *subsumption* y DAMN; y en el híbrido las arquitecturas AURA, 3T y Saphira.

2. Arquitecturas de Control

Los diversos elementos funcionales que hacen parte del sistema de procesamiento y toma de decisiones de alto nivel de un sistema inteligente, se organizan en una arquitectura de procesamiento denominada cognitiva. Históricamente la arquitectura cognitiva se vio influenciada por los principios de la Inteligencia Artificial, la cual buscaba que las máquinas emulasen las capacidades cognitivas del ser humano [27]. Debido a que la arquitectura de un sistema inteligente tipo robot móvil es más elemental que la requerida para un sistema altamente cognitivo, a la arquitectura cognitiva de un robot móvil se le denomina Arquitectura de Control [120]. Son pocos los artículos en robótica móvil que definen el significado de la palabra arquitectura que emplean, por lo que es importante definir su significado para este documento como: "la estructura de componentes, sus interrelaciones y principios de diseño; la asignación de funciones a los subsistemas y la especificación de las interfaces entre subsistemas" [111].

Los primeros grupos de investigación en robótica móvil dividieron la arquitectura de control en tres etapas de procesamiento elementales: Sensado, Planificación y Actuación [31]. Estas etapas se describen a continuación:

- Etapa de Sensado (S): capacita al robot para disponer de la información del entorno capturada de los sensores y producir, a partir de ésta, información útil en un modelo del entorno, para los otros elementos funcionales.
- Etapa de Planificación (P): capacita al robot para producir las tareas a ejecutar (vg. ir a A hasta B evadiendo obstáculos, cruzar la puerta, etc.) con base en la información sensorial y el conocimiento disponible que posee del entorno que lo rodea.
- Etapa de Actuación (A): capacita al robot para la ejecución de tareas mediante la interacción con los actuadores presentes en los diferentes lazos de control del robot.

Con el paso del tiempo estas tres etapas de procesamiento se organizaron en niveles de procesamiento que dieron creación a tres enfoques en las arquitecturas de control de

un robot móvil. Estos enfoques se soportan en una reorganización particular de la interacción entre cada uno de los niveles de acuerdo a las directrices de tres paradigmas diferentes: Deliberativo; Reactivo e Híbrido [31]. Estos paradigmas se diferencian en la comunicación y organización que proponen entre los diferentes niveles presentes en la Tabla 2.1.

Tabla 2.1. Niveles de procesamiento de la arquitectura de control clásica de un robot móvil (Sensado-Planificación -Actuación) en términos de entradas y salidas

	ARQUITECTURA DE CONTROL		
Niveles	SENSADO	PLANIFICACION	ACTUACION
Entrada	Señales de los sensores	Información (sensorial y/o cognitiva)	Información sensorial o directivas - tareas
Salida	Información sensorial	Directivas - acciones	Ordenes de actuación

El paradigma que da origen a la arquitectura de control Deliberativa establece que antes de generar las ordenes para los lazos de control del robot móvil en el nivel de actuación, primero se debe procesar la información sensorial del nivel de sensado, por parte del nivel de planificación, para producir un mapa del entorno lo más fiel posible a la realidad, con el fin de apoyarse en este mapa y la información actual de los sensores, y planear la estrategia de movimientos más idónea para la ejecución de la tarea del robot móvil. Una vez se determina el plan de ejecución se entregan las ordenes al nivel de actuación. A este paradigma se le conoce como SPA (Sensado-Planificación-Actuación).

El paradigma asociado al control Reactivo establece que el nivel de sensado se comunica directamente con el nivel de acción, no existe el nivel de planificación, se organizan pares de relación sensor-motor con el fin de conformar diversos comportamientos que deben competir entre ellos con el fin de imponer su comportamiento en el robot. No hay modelos del mundo ni planificadores de trayectorias. A este paradigma se le conoce como SA (Sensado-Actuación).

El paradigma híbrido establece que la arquitectura de control hace uso de los comportamientos del control reactivo, pero también emplea, inicialmente, el nivel de planificación del control deliberativo con el fin de planear la ejecución de

comportamientos más idónea para la realización, posteriormente, de la tarea por parte del nivel reactivo. Por lo que a este paradigma se le conoce como P, SA (Planificación, Sensado-Actuación).

A continuación se describen las características representativas de las arquitecturas de control más representativas en cada uno de los anteriores paradigmas.

2.1 Arquitectura Deliberativa

El paradigma Deliberativo fue el primer enfoque empleado en el control de los robots móviles, se apoyó en las ideas de la IA y fue el que clásicamente propuso la organización secuencial de los tres niveles presentes en la tabla 2.1, también se le conoce como SPA (Sensado – Planificación - Acción). Su aplicabilidad predominó principalmente desde 1967 hasta 1990. Según el paradigma deliberativo un robot autónomo, obligatoriamente y en forma secuencial, primero tiene que sensar su entorno, construir un modelo de su entorno, después deliberar y planificar la acción a realizar y finalmente ejecutar dicha Acción (SPA). Si un cambio ocurría en el entorno, se debía actualizar el modelo del mundo y desarrollar un nuevo plan que incluyera el cambio. De esta forma el ciclo tiene que repetirse hasta completarse cada una de las tareas del robot. Los dos ejemplos representativos de esta arquitectura son la arquitectura de control del Robot Shakey [28] del SRI *International* (*Stanford Research Institute International*) y la arquitectura NASREM [30] de la NASA, la descripción técnica de las mismas se detallan en el anexo 1.

2.2 Arquitectura Reactiva

El paradigma reactivo fue el enfoque empleado en el control de los robots móviles que elimina el nivel de planificación para evitar los problemas de lentitud en la respuesta o simple bloqueo del sistema, del paradigma deliberativo. Su concepción se le debe el profesor Rodney Brooks y a sus estudiantes del MIT (*Masachussets Institute Technology*) [34], en él se propone la organización del control en una jerarquía de niveles de Sensado – Actuación, presentes en la tabla 2.1, que conforman comportamientos cada vez más complejos. También se le conoce como SA (Sensado - Acción), su aplicabilidad

predominó principalmente hasta 1992. No son muchos los ejemplos de este paradigma, el más representativo es la arquitectura de *Subsumption* de Brooks [34] y la arquitectura DAMN de Rosemblat [137], las cuales se describen en el anexo 1.

2.3 Arquitectura Híbrida

El paradigma Híbrido establece que la arquitectura de control hace uso de los comportamientos del control reactivo, pero también emplea el nivel de planificación del control deliberativo con el fin de planear la ejecución de comportamientos más idónea para la realización de la tarea. A este paradigma se le conoce como P, SA (Planificación, Sensado-Actuación). Es el paradigma actualmente más empleado en el control de robots móviles y su formulación se debe al investigador Ronald C. Arkin [50]. Las arquitecturas híbridas representativas en la década del noventa son AuRA [50], 3T [51] y Saphira [49], las cuales se describen tecnicamente en el anexo 1.

2.4 Arquitectura Multiagente

En años recientes el uso del término agente [27] ha ganado importancia en muchos campos diferentes y especialmente dentro de la comunidad de la IA. Inicialmente el término fue aplicado a los sistemas software de planificación que interactuaban en un ambiente cambiante en tiempo real. La idea detrás de esto era que los agentes tienen capacidades deliberativas mientras de igual forma son hábiles para realizar tareas en el mundo real. Posteriormente apareció el término agente racional acotado para referirse a aquellos sistemas software con capacidades de toma de decisión que tienen restricciones temporales y/o de recursos.

La noción tanto de un agente embebido en un entorno como de su capacidad de toma de decisiones con restricciones temporales o de recursos fue bien aceptada en el campo de la robótica, particularmente en los robots móviles autónomos que operan en ambientes naturales. Como las capacidades de procesamiento hardware de los robots van mejorando y se pueden hacer más inteligentes, la terminología de los agentes se hizo parte de la literatura de la robótica para referirse a este aspecto o simplemente para emplearla como sinónimo de robot móvil. De esta forma los investigadores de este

campo empezaron a usar el término agente para referirse a sus robots. La idea detrás de esto era simple, los agentes también pueden ser sistemas físicos (los robots) que toman decisiones en un mundo real. Pero por otra parte los agentes son sistemas software "inteligentes", por lo que no pasó mucho tiempo hasta que el campo de la robótica se fue inundando de ideas de combinar agentes software en arquitecturas multi nivel con comportamiento emergente [27].

Otros campos también se beneficiaron del concepto de los agentes. Con el crecimiento de la Internet en la década del noventa, se creó un nuevo modelo software que lleva la encapsulación más allá del nivel de objetos y le brinda a los módulos una extensión temporal. Este modelo software de agentes es particularmente útil cuando se combina con código móvil. Cuando estos agentes móviles trabajan a través de la red para desarrollar tareas útiles para sus usuarios aparecen algunos términos nuevos: agente buscador, el cual ayuda a encontrar cosas en la red, agente enrutador, el cual ayuda a mover más rápido los paquetes, el agente filtrador de datos, el cual observa por condiciones particulares en una red o en recurso y alerta al usuario o toma otra acción. De esta forma el término agente también se relaciona con interfaces de usuario avanzadas. A pesar de que el agente éste limitado a un contexto de información tecnológica, éste puede referirse a muchos conceptos software. Esta misma situación también se observa reflejada en el campo de la robótica móvil, donde el sistema de procesamiento y toma de decisiones de alto nivel se empezó a modelar como entidades tipo agente, interrelacionadas para controlar no solo un simple robot móvil sino equipos de robots que interactúan en ambientes reales. Esta es la nueva bienvenida que el campo de la robótica le hace a la Inteligencia Artificial, específicamente al área de la Inteligencia Artificial Distribuida (DAI por las siglas en ingles) [27].

El DAI está dividido en dos disciplinas: la solución de problemas distribuidos (DPS) y los sistemas multi agentes (MAS)[27]. Los temas principales considerados en DPS son los aspectos relacionados con la administración de información como descomposición de tareas y síntesis de solución. Los sistemas multi agentes ayudan brindando los principios para la construcción de sistemas complejos involucrando agentes múltiples y los mecanismos de coordinación de agentes independientes. De esta forma, MAS permite que los sub problemas en los que se ha dividido un problema sean atendidos por diferentes agentes solucionadores los cuales tienen sus propios intereses y objetivos.

Los MAS son aplicables en muchos dominios, particularmente son necesarios cuando un sistema tiene diferentes elementos con variados objetivos, posiblemente conflictivos, y debe manejar interacciones. Los MAS también son aplicables en sistemas que no son

distribuidos, por ejemplo un dominio que fácilmente se pueda dividir en componentes de tal manera que tareas independientes puedan ser atendidas por agentes separados. Un beneficio de los MAS es su escalabilidad, ya que al ser inherentemente modulares, es mucho más fácil adicionar nuevos agentes en un sistema multi agente que adicionarle nuevas capacidades a un sistema monolítico. Para el campo de la robótica móvil el paradigma de los sistemas multi agente es particularmente útil tanto en robots simples como en sistema multi robots. Por ejemplo la arquitectura híbrida puede ser separada en diferentes agentes que pueden verse como un sistema multi agente.

El paradigma MAS ha dado pie para la creación de arquitecturas cognitivas con capacidades de uso como arquitecturas de control multi agente para robots móviles. A continuación se describen dos de ellas, APOC [138][139] y la arquitectura de esquemas de Cosy (CAS) [140].

Arquitectura APOC

APOC (*Activating*, *Processing, Observing, Components*) es un marco arquitectural de agentes genérico propuesto por los profesores Virgil Andronache y Matthias Scheutz del Laboratorio de Robótica e IA de la Universidad de NotreDame. Este marco tiene como propósito que cualquier otra arquitectura, sea deliberativa o híbrida, pueda ser expresada de una forma única. Su enfoque está basado en modelar los diferentes elementos funcionales que aparecen en los niveles de las arquitecturas de control de un sistema agente único, esto es un robot autónomo, como componentes distribuidos empleando agentes MAS y un sólido formalismo matemático tipo DES [138].

APOC propone hacer uso de un tipo básico de componente y cuatro tipos básicos de enlaces: A(*Activating*), P(*Proccesing*), O(*Observing*) y C(*Component*). Estos enlaces tienen el propósito de permitir conectar componentes básicos y derivados, en la especificación de cualquier arquitectura compleja de agentes. Una arquitectura APOC consiste de componentes computacionales llamados nodos los cuales pueden tener cualquiera de los cuatro enlaces listados, ver figura 2.1.

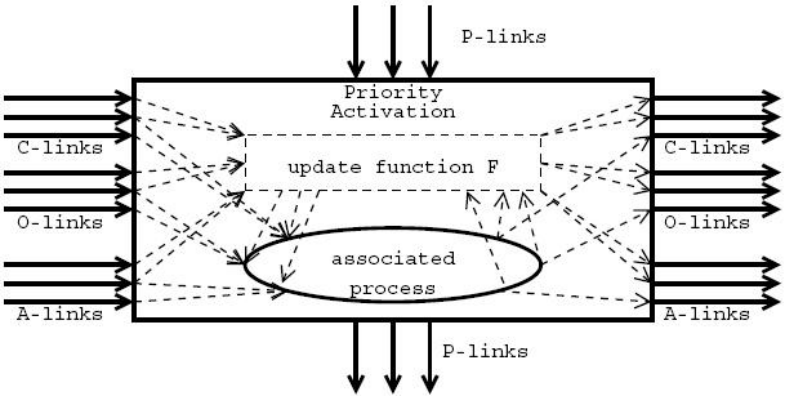


Figura 2.1 Estructura básica de un componente APOC que ilustra enlaces entrantes y salientes A,P,O y C, al igual que los niveles de activación y prioridad junto con la función de actualización F y el proceso asociado. Fuente [138]

Los cuatro tipos de enlaces definidos en APOC tienen como objetivo cubrir los tipos de interacción importantes que se presentan entre los componentes de una arquitectura de agentes. El enlace Activación (A) permite a los componentes enviar estructuras de datos y o recibir estructuras de datos de otros componentes; el enlace observación (O) permite a los componentes observar el estado de otros componentes; el enlace control de procesos (P) permite que los componentes controlen e influyan en los cálculos que se realizan en otros componentes y el enlace componente (C) permite que un componente instancie otros componentes y enlaces entre ellos.

Cada componente en APOC tiene un nivel de prioridad y de activación y puede recibir entradas y enviar salidas desde y hacia otros componentes vía cualquiera de sus enlaces. Las entradas son procesadas y las salidas son producidas de acuerdo a una función de producción F, la cual determina la funcionalidad que el componente implementa, mapeando desde sus entradas y componente internos (los valores de activación y prioridad actuales) a las salidas y estados internos actualizados(los nuevos valores de activación y prioridad). De esta forma la función de actualización proporciona la especificación de un proceso computacional que, en un componente instanciado, continuamente actualiza el estado total del componente. El algoritmo particular para implementar la función F se define en forma separada para cada componente empleado en la arquitectura. Adicionalmente un componente en APOC tiene un proceso asociado, el cual arranca, interrumpe, reasume o finaliza. El proceso asociado puede ser un proceso externo a la arquitectura (por ejemplo un proceso que controla las ruedas de un robot móvil) o bien un proceso computacional auto contenido que toma entradas del componente y entrega sus salidas (por ejemplo un proceso que

implementa un algoritmo de análisis de imágenes que dada una imagen retorna una representación de alto nivel 3D de todos los objetos encontrados en la imagen)[139].

El marco arquitectural APOC puede ser empleado para tres propósitos: 1) como una herramienta de análisis formal para la evaluación de arquitecturas, 2) como una herramienta de diseño de componentes arquitecturales, y 3) como una plataforma para la definición de arquitecturas agentes.

Arquitectura de Esquemas de CoSY (CAS)

La Arquitectura CAS (COSY *Architecture Schema*) es uno de los principales resultados del proyecto europeo COSY realizado en la Universidad de Birmingham en el Reino Unido [140]. CAS está basado en la idea de integrar una arquitectura como una colección de sub arquitecturas de bajo acoplamiento entre las mismas, donde una sub arquitectura puede ser considerada como un subsistema de la arquitectura global.

Como se ilustra en la parte izquierda de la figura 2.2, cada sub arquitectura contiene un número de componentes de procesamiento los cuales vía memorias de trabajo y un componente de control, llamado administrador de tareas, comparten información. Algunos componentes de procesamiento dentro de la sub arquitectura son administrados y otros no lo son. Los componentes no administrados realizan procesamiento simple sobre los datos, por lo que corren constantemente y emiten sincrónicamente sus datos en la memoria de trabajo. Los procesos administrados, en contraste, monitorean los cambios de contenido en las memorias de trabajo y sugieren posibles tareas de procesamiento para los datos de la memoria de trabajo. Como estas tareas pueden llegar a ser costosas, y la potencia computacional normalmente es limitada, las tareas son seleccionadas con base en las necesidades del sistema global. El administrador de tareas es básicamente un conjunto de reglas para realizar tales asignaciones. Cada memoria de trabajo de la sub arquitectura puede ser leída por cualquier componente en cualquier sub arquitectura, pero solo puede ser escrita por los procesos que están dentro de la sub arquitectura y por un número limitado de componentes privilegiados. Estos últimos pueden colocar información en cualquier sub arquitectura, permitiendo de esta forma una creación de objetivos tipo *top-down* y una coordinación cruzada para la arquitectura.

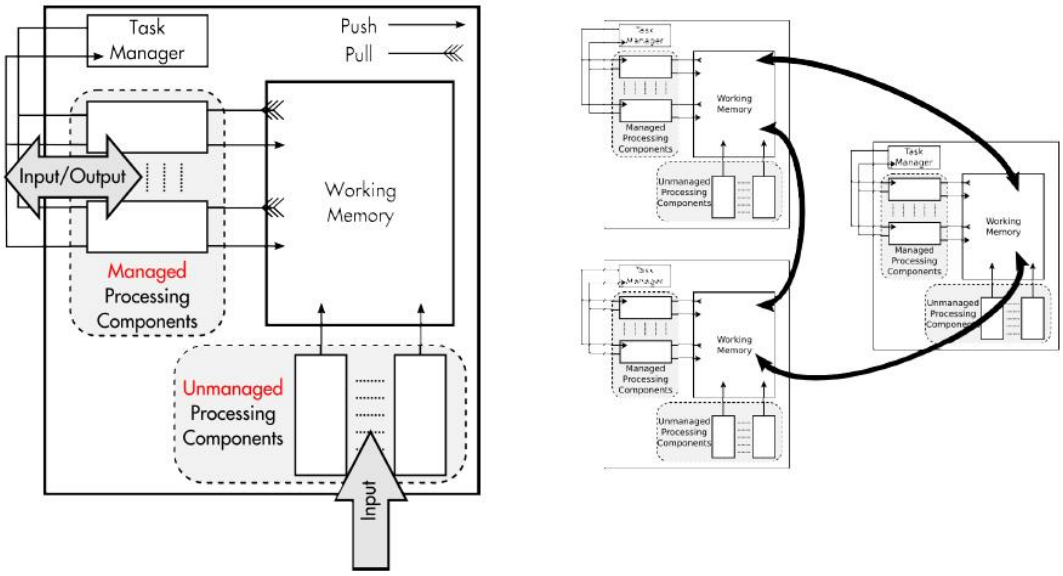


Figura 2.2 Dos vistas de la Arquitectura de Esquemas COSY. La vista izquierda es el esquema al nivel de una sub arquitectura simple. La vista derecha muestra como un sistema es construido a partir de un número de tales sub arquitecturas. Fuente [140]

Si hay varios objetivos de procesamiento que una sub arquitectura requiere cumplir se hace una mediación entre éstos por parte del administrador de tareas. Esta mezcla de diseño *top-down*, procesamiento conducido por datos y objetivos buscados, para su buen manejo, requiere una coordinación dentro de la sub arquitectura o a través de múltiples sub arquitecturas. En un extremo el número de componentes privilegiados se puede limitar a uno (coordinación centralizada) y en el lado opuesto todos los componentes pueden ser privilegiados (coordinación completamente descentralizada).

2.5 Resumen

En este capítulo, se han considerado las características de los tres paradigmas empleados en la robótica móvil en el diseño de arquitecturas de control: deliberativo, reactivo e híbrido y se ha propuesto, en este trabajo de maestría, una nueva clasificación denominada multiagente que se puede considerar como un sistema cognitivo distribuido. De cada uno de ellos se realizó un análisis de arquitecturas representativas: Shakey y Nasrem en el paradigma deliberativo, Subsumption y DAMN en el paradigma reactivo, AuRa, Saphira y 3T en el paradigma híbrido y finalmente APOC y CAS en el paradigma multi agente.

Como se puede apreciar de la arquitectura de control de Shakey y Nasrem, descriptas en el anexo 1, el paradigma deliberativo implica la planificación explícita de cada una de las acciones del robot. Igualmente se caracteriza por obtener y utilizar un modelo interno del mundo real lo más exacto y completo posible para emplearlo como base de razonamiento para todos los planes y acciones. De manera que el procesamiento de la información sensorial se orienta a la consecución de este modelo detallado del entorno del robot. No solo la NASA, el NIST y el SRI en Estados Unidos trabajaron empleando este paradigma durante los setenta y ochenta, sino todo grupo de investigación fuese de IA o de Robótica. En general todas esas arquitecturas deliberativas realizan un flujo de información consistente en mediciones de los sensores, realizar un procesamiento durante tres o seis niveles hacia arriba, donde se planea y se regresa una orden, que pasa nuevamente por todos los niveles, hasta llegar a los actuadores. La rigidez del paradigma, el considerar que se debía modelar con minuciosidad el mundo y las limitaciones, oclusiones y ruido de los sensores generaron graves dificultades para las implementaciones de este paradigma, las cuales se definen en dos situaciones conocidas como:

Problema Marco o Persistencia: el efecto de las acciones propuestas no puede ser exactamente conocido. Esto genera que los planes a largo plazo no sean útiles debido a la desviación de los estados reales del robot con los supuestos al momento de programar las acciones a largo plazo. La razón: los sensores no son perfectos; sufren de incertidumbre, no siempre funcionan bien y fallan.

Presunción de un mundo cerrado: Cada cosa relevante en el entorno del robot debe estar en el modelo. Como esto es imposible debido a la razón del punto anterior, genera

que el Planificador genere planes erróneos basados en modelos con información faltante o errónea.

Adicionalmente, a los dos problemas anteriores, hay que agregar que los algoritmos empleados en el nivel de planificación, heredados de la IA, se caracterizan por no garantizar un tiempo acotado de respuesta, por lo que su ejecución en tiempo real no se podía garantizar.

A pesar del tiempo que ha pasado desde el trabajo con Shakey y NASREM en los setenta y ochenta hay elementos interesantes para rescatar de los dos proyectos. En la actualidad increíbles progresos en reducir el tamaño y costo de los computadores, sensores y actuadores, hace que el uso práctico de los robots sea una realidad más de lo que podía ser en los sesenta. Por lo que en la actualidad nuevos proyectos robóticos se pueden beneficiar del legado de Shakey, no por su paradigma deliberativo, sino porque este proyecto condujo importantes avances en técnicas de IA, muchas de las cuales fueron reportadas en la literatura, pero un gran nivel de información específica nunca apareció más que en una serie de reportes SRI relativamente inaccesibles. Este listado es de particular interés por: 1) las técnicas usadas en las rutinas de acción que permiten una recuperación flexible generada a partir de acciones ejecutadas inapropiadamente, 2) el método de integración de percepción con acción, y 3) las técnicas para planificación y ejecución de secuencias complejas de acciones. Con respecto a NASREM a pesar de ser una arquitectura estrictamente jerárquica y diseñada en los ochenta, aún se encuentra operacional y su uso es obligatorio para quienes obtienen contratos en la NASA relacionados con el manipulador espacial robótico.

El paradigma reactivo, inspirado en la observación del comportamiento instintivo de los seres vivos, más conocido con el nombre de arquitectura de *Subsumption*, tiene su principal interés en la incorporación de comportamientos, en la forma de pares de esquemas sensor-motor, para modelar las habilidades básicas de la máquina. Originalmente *Subsumption* fue de implementación hardware con máquinas de estado finito modelando cada comportamiento, donde el nivel sensor se comunica directamente con el nivel de actuación, pero posteriormente se diseñó un lenguaje llamado *Behaviour*. A pesar de que se pueda pensar que en esta arquitectura los sensores están comunicados directamente con los actuadores, se emplean las conexiones sensor-motor para determinar la consigna a seguir por parte de los sistemas de control que gobiernan a los actuadores. El enfoque se caracteriza por: 1) su aversión a los planes; no utiliza un nivel de planificación que coordine antes de actuar, 2) la ejecución concurrente de múltiples comportamientos, 3) un mecanismo de composición

de comportamientos que los inhibe, suprime o combina y, 4) la no generación de modelos del entorno; en palabras de Brooks "el mundo real es el mejor modelo".

El paradigma Reactivo permitió aumentar la velocidad de navegación de los robots, al eliminar los lentos algoritmos de planificación, y dotarlos de una eficiente y eficaz capacidad de evasión de obstáculos. Sin embargo el enfoque no modular en la organización de los comportamientos y la ausencia de modelos del entorno dificulta la capacidad de planificar y ejecutar tareas de más alto nivel por parte del robot. En este sentido la arquitectura DAMN fue más flexible al permitir la coexistencia de múltiples comportamientos tanto reactivos como de tipo deliberativo, que hicieran uso de modelos del entorno y fuesen coordinados por un mecanismo central, de este modo todos los elementos de control presentes tanto en los niveles reactivos como deliberativos se podían ver como comportamientos.

El paradigma Híbrido de Ronald C. Arkin en 1987 y su arquitectura AuRA, al igual que 3T y Saphira aprovechan lo mejor de los enfoques previos. En primera instancia se tiene un nivel inferior de múltiples comportamientos (SA) encargados del control directo del robot, pero se hace uso, en segunda instancia, de un nivel superior de Planificación (P) (sin embargo no tan exigente o pesado como el usado en Shakey o NASREM) y finalmente un nivel intermedio de secuenciamiento que conecta el nivel inferior y superior (SA, P). No todas las arquitecturas híbridas proponen una capa de secuenciamiento, simplemente un nivel reactivo en primer plano y un nivel deliberativo en segundo plano, por lo que en este detalle se diferencian las arquitecturas híbridas.

En el paradigma híbrido la información sensorial es utilizada tanto a nivel de Planificación, para la construcción de un modelo interno del entorno del robot que se utiliza para la descomposición de tareas y la selección de los comportamientos que implementarán estas tareas, como a nivel de la ejecución de dichos comportamientos, donde el valor procesado o no de cada sensor es utilizado por cada comportamiento que lo necesite. El enfoque de delegar al nivel reactivo el control del robot, mientras el nivel deliberativo, en segundo plano, modela y planea permitió evitar los problemas del paradigma deliberativo. En esta comunicación entre los niveles reactivo – deliberativo hay también diferencias, mientras algunas arquitecturas proponen que el nivel reactivo encueste asincrónicamente al nivel deliberativo el qué hacer, en otros el nivel deliberativo sincrónicamente le comunica al nivel reactivo el qué hacer. A pesar de las diferencias, el paradigma híbrido es el enfoque adecuado para permitir una rápida respuesta de los robots en su entorno y modelar un nivel cognitivo superior útil.

Finalmente en este trabajo de maestría se ha propuesto un nuevo enfoque para clasificar las nuevas arquitecturas de control empleadas en robótica, se le ha denominado paradigma multi agente.

El paradigma multi agente es un paradigma híbrido donde se hace una distribución del nivel cognitivo en múltiples nodos denominados agentes, idea tomada de los sistemas multi agentes (MAS) de la IA. Simplemente es otra forma de organizar la estructura de procesamiento y toma de decisiones de alto nivel del robot. En lugar de considerar que se tienen dos niveles centralizados responsables del control del robot: uno reactivo y otro deliberativo, estas dos responsabilidades se reparten en múltiples agentes, donde se podría proponer desde algo tan elemental, como un agente reactivo y un agente deliberativo, uno por cada nivel, hasta un modelo más complejo con múltiples agentes, cada uno responsable de funciones reactivas, deliberativas o de supervisión del sistema. Cada agente contaría con inteligencia, acceso a recursos y capacidad de gestión, lo que permitiría ver la funcionalidad total de la arquitectura como el fruto de las interrelaciones de los diferentes agentes. Cuando se llega a este nivel es más evidente que la arquitectura requiere la definición del tipo de comunicación y significado preciso de los mensajes empleados por los agentes. En esta nueva clasificación se han seleccionado dos arquitecturas APOC y CAS, ambas provienen de grupos de investigación en arquitecturas cognitivas de la IA, pero que las han empleado en el control de robots. De mayor interés en el análisis realizado es la arquitectura APOC por su fundamentación matemática en sistemas a eventos discretos, y que la habilita para poder ser empleada como una herramienta de verificación de nuevas arquitecturas.

3. Herramientas y Plataformas

En este capítulo se presenta un análisis cualitativo de las características sobresalientes de las principales herramientas y plataforma de trabajo, de fuente abierta – libres no comerciales, empleadas en el ámbito investigativo y académico para diseñar e implementar sistemas de control de robots móviles.

3.1 Introducción

Todo trabajo práctico o aplicado, incluyendo el campo de la investigación en robótica, se ve impactado por las herramientas que son usadas. Buenas herramientas ayudan a simplificar tareas comunes mientras que malas las hacen complicadas. La disponibilidad de herramientas flexibles, confiables y reusables para la programación robótica es un aspecto clave para la comunidad investigadora de este campo [77]. Las hipótesis sobre las cuales son construidas un conjunto de herramientas desplaza al investigador que las usa hacia una clase particular de solución, por eso es importante conocer los fundamentos arquitecturales teóricos sobre los cuales están fundamentadas estas herramientas.

En este trabajo de maestría a este conjunto de herramientas se les denomina "Plataforma de trabajo para programación de robots" o simplemente "Plataforma de trabajo robótica".

3.2 Plataformas de Trabajo Robóticas

Para este análisis se han seleccionado las plataformas que aparecen en la tabla 4.1. La elección de cada una se ha realizado de acuerdo a la sonoridad que han tenido en varios estudios comparativos [117], [118], [121], [122] por ser fuente de estandarización [84]

y finalmente por ser el fruto de recientes e importantes proyectos de investigación [145], [138].

Tabla 3.1 Listado de las principales plataformas de trabajo robóticas de fuente abierta y libre.

Nombre plataforma	**Institución de origen**	**Página web**
TCA – IPC/TDL	Carnegie Mellon University	www.cs.cmu.edu/~tdl/,
TeamBots	Carnegie Mellon University	www.cs.cmu.edu/~trb/TeamBots/
Player /Stage / Gazebo	Universidad de California del Sur	http://playerstage.sourceforge.net/
GenoM	CNRS - LAAS	https://softs.laas.fr/openrobots/
SmartSoft	Universidad de Ulm	www.rz.fh-ulm.de/~cschlege/orocos/
Balt & Cast	Universidad de Birgminham	www.cognitivesystems.org
RT-Middleware	METI, JARA, AIST en Japón	www.is.aist.go.jp/rt/OpenRTM-aist/html-en/index.html
MIRO	Universidad de Ulm	http://miro-middleware.berlios.de/
MARIE	Universidad de Sherbrooke	http://marie.sourceforge.net
ORCA2	Universidad de Sydney	http://orca-robotics.sourceforge.net
ADE	Universidad de Notre Dame	www.nd.edu/ airolab/software

TCA (*Task Control Architecture*) – IPC/TDL

TCA es una arquitectura de control del CMU (*Carnegie Mellon University*) [47]. Su implementación es en esencia una especie de sistema operativo robótico de alto nivel ubicado sobre una distribución Unix, que proporciona un conjunto integrado de mecanismos comúnmente empleados para soportar comunicaciones distribuidas,

descomposición de tareas, administración de recursos, monitoreo de la ejecución del sistema y recuperación de errores.

Las comunicaciones interproceso en TCA están basadas en una comunicación soportada en *sockets* anónimos usando el protocolo TCP/IP que permite el paso de mensajes tanto en el modo *publish/subscribe* y cliente/servidor. TCA también soporta el *marshaling* y *unmarshalling* de datos basado en un lenguaje de definición de formatos. Todas las comunicaciones en TCA son enrutadas a través de un servidor central, el cual puede generar registros tipo *log* de todos los mensajes de tráfico.

El control de nivel de tareas de TCA incluye capacidades para realizar descomposición jerárquica de tareas, secuenciación y sincronización de tareas, administración de recursos, monitoreo de la ejecución del sistema y manejo de excepciones. Por nivel de tareas se entiende lo relacionado con la integración y coordinación del sensado, la planificación y el control en tiempo real con el fin de alcanzar un conjunto dado de objetivos. Un servidor central es el responsable de despachar las tareas.

Las propiedades de la arquitectura TCA fueron inicialmente programadas en un sistema operativo Unix pero posteriormente fueron reescritas y separadas en dos herramientas, o paquetes software, IPC (Inter Process Communicatios) y TCM (*Task Control Management*). IPC es un *middleware* propietario desarrollado en el CMU con características similares a TCA pero con comunicación *peer to peer* al igual que otras mejoras. TCM es una re implementación completa de las capacidades del control de nivel de tareas de TCA y fue escrito en C++.

Posteriormente se desarrolló un lenguaje de descripción de tareas, denominado TDL (*Task Description Language*) para reemplazar a TCM. TDL es un super conjunto de C++ que viene con una sintaxis para describir capacidades de control de nivel de tareas. El objetivo fue facilitar el escribir programas de control de nivel de tareas empleando un lenguaje con una sintaxis familiar a los investigadores de la robótica. TDLC es un programa en Java que traslada el código TDL a C++. La última mejora que se le agregó a TDL fue la capacidad de trabajar en forma distribuida, ahora se denomina MTDL (Multi TDL)[141].

Finalmente con las herramientas IPC y TDL se pueden diseñar sistemas de control para robot móviles autónomos simples supeditados a una arquitectura especifica TCA, aunque IPC suministra un componente de distribución, su acercamiento basado en *socket* lo hace limitante para considerarlo como un marco de trabajo general.

TeamBots

TeamBots es una colección basada en Java de programas de aplicación y paquetes Java para apoyar la investigación en robótica móvil, tanto de agente simples como múltiples, diseñado por Tucker Balch en el CMU (*Carnegie Mellon University*). Una gran colección de clases e interfaces está disponible para desarrollar nuevo software [79].

Una selección de tales clases se llama Clay, este es un paquete de clases de Java que puede ser combinado para crear sistemas de control de robots basados en comportamiento. Clay toma ventaja de la sintaxis de Java para facilitar la combinación, el *blending* y la abstracción de comportamientos. Clay puede ser usada para crear sistemas reactivos simples o configuraciones jerárquicas complejas con capacidad de aprendizaje y memoria. Una interface básica se le hereda a todas las clases robot. Teambots fue diseñado para propósitos de simulación, y tiene una buena interface grafica para tales propósitos. Es empleado en investigación y educación. Teambots no impone una arquitectura como tal, aunque se especializa en sistemas reactivos facilita métodos para realizar secuenciamiento de tareas.

Ya que TeamBots ha sido enteramente construido en Java, con un código fuente cuidadosamente agrupado en una colección fina y granular de clases, éste puede correr básicamente en cualquier SO que soporte Java. Claro está que con el fin de que se pueda emplear con robots reales, se debe disponer de los manejadores *hardware* de los diferentes dispositivos del robot. Estos manejadores normalmente se implementan usando JNI (*Java Native Interface*) el cual envuelve la funcionalidad especifica de la plataforma.

Finalmente TeamBots puede considerarse como un marco de trabajo general para robots móviles, pero que no se restringe a una arquitectura de control determinada. Tiene facilidades para ejecutar un sistema de control en una aplicación de un proceso simple lo que lo hace no distribuido y menos robusto. TeamBots puede ser empleado para el control de robots simples y múltiples [79].

Player / Stage / Gazebo (P/S/G)

Player [77] es un proyecto surgido de los laboratorios de investigación robótica de la USC (Universidad de California del Sur). En forma básica Player es un servidor de dispositivos robóticos, basado originalmente en *sockets*, que hace las veces de una capa

de abstracción hardware para permitir el control y lectura de una amplia variedad de actuadores y sensores. Player se ejecuta en una máquina que está físicamente conectada a una colección de estos dispositivos y ofrece una interface *socket* TCP a los clientes que se desean comunicar con ellos.

Los clientes se conectan a Player para comunicarse con los dispositivos intercambiando mensajes vía un *socket* TCP. En este sentido Player es similar a cualquier servidor de dispositivos estándar como los que se encuentran sobre los sistemas Unix y Linux. Al igual que ellos, Player soporta múltiples clientes concurrentemente, cada uno en su propio *socket* TCP, por lo que los clientes pueden ser escritos en cualquier lenguaje de programación que provea soporte para *sockets*. En la actualidad Player permite otras capas de transporte como JINI y CORBA, para ser usadas en lugar de TCP. Adicionalmente también se puede implementar un Player monolítico, sin capa de transporte, que usa solo intercambio de mensajes internos.

Con el fin de proveer una abstracción uniforme para una amplia variedad de dispositivos, Player sigue el modelo Unix de tratar todos los dispositivos como archivos. Por ejemplo, para iniciar la recepción de la lectura de los sensores, el cliente abre el dispositivo apropiado con acceso leer; igualmente, antes de enviar una orden a un actuador, el cliente debe abrir el dispositivo apropiado con acceso escribir. Adicional al flujo de comandos y datos asíncronos, se cuenta con un mecanismo de solicitud/réplica, *akin to ioctl()*, que los clientes pueden usar para obtener o fijar información de configuración para los dispositivos de Player. Varios clientes pueden conectarse simultáneamente a un dispositivo ya que no hay mecanismo de aseguramiento. Player maneja tres conceptos bien diferenciados: el *driver*, un módulo que puede producir y consumir mensajes, y el dispositivo, que es un *driver* unido a una interface.

Stage es una herramienta gráfica que permite simular una población de robots móviles moviéndose y sensando en un ambiente 2D. Varios modelos de sensores están disponibles, incluyendo sonar, barrido por radar laser, cámaras con *zoom* y montaje *pan-tilt*, detección de color y odometría. Los dispositivos de Stage presentan una interface estándar Player así que pocos o ningún cambio requieren para operar entre la simulación y el hardware en un ambiente real.

Gazebo es una herramienta para simulación de múltiples robots en ambientes en exteriores. Al igual que Stage es capaz de simular una población de robots, sensores y objetos pero en un mundo 3D. Éste genera tanto realimentación sensorial de lo sucedido al robot como la interacción física posible entre los objetos, para ello dispone de un simulador de física de cuerpos rígidos. Gazebo presenta una interface estándar

Player adicional a su propia interface nativa. Los controladores escritos para el simulador Stage pueden generalmente ser usados con Gazebo sin modificaciones y viceversa.

EL proyecto P/S/G ofrece una capa de abstracción de dispositivos hardware para plataformas robóticas, sean éstas simuladas o físicamente disponibles. Sin embargo no impone restricción alguna en la arquitectura de control, esto queda libre al diseñador robótico, tanto así que no brinda interfaces para los niveles superiores ni reactivos ni deliberativos, por lo que puede ser usado para diseñar cualquier tipo de arquitectura [78].

GeNom

GeNom fue desarrollado por el grupo RIA (*Robotics and Artificial Intelligence*) del LAAS - CNRS (Laboratorio de Análisis de Arquitecturas de Sistemas del Centro Nacional de Investigación Científica) en Francia. Éste hace parte de la arquitectura software robótica del LAAS donde GeNom es la implementación de la capa funcional. Éste brinda soporte para la generación de componentes software y viene con un conjunto de utilidades, por ejemplo: librerías de comunicación, servidor de aplicaciones y herramientas de construcción, las cuales en conjunto permiten implementar aplicaciones software robóticas. En GeNom las abstracciones de la capa funcional son representadas en la forma de componentes software llamados módulos GeNom. Para implementar una aplicación específica estos módulos son combinados en una red de nodos de comunicación los cuales pueden hacer uso de servicios de otros nodos. La funcionalidad más útil para el cliente de un módulo proviene de sus servicios, por ejemplo las solicitudes que el módulo proporciona pueden ser accesados por medio de una librería de interfaces solicitudes/réplicas. Los servicios que se encuentran en ejecución se conocen como actividades. Puede haber varias actividades corriendo simultáneamente. Con el fin de permitir el monitoreo y administración de las solicitudes el módulo GeNom organiza el código en la forma de "*codels*". Un *codels* es la entidad más pequeña que el módulo puede manejar y de esta forma el *codel* puede ser considerado como atómico, esto es que debe finalizar su ejecución dentro de un corto intervalo de tiempo conocido. Estos *codels* son implementados como funciones estándar de C.

Con respecto a los mecanismos de comunicación, GeNom hace uso de dos métodos para establecer comunicación entre módulos: acercamiento basado en RPC (*Remote Procedure Calls*) y un acercamiento soportado en *script* TCL. El primero confía en una serie de métodos para escribir/leer datos en *posters* y realizar entrega de mensajes

entre módulos soportado en las librerías *posterLib* y *csLib*. El segundo método, aunque con menor rendimiento respecto al primero, tiende a ser más simple y escalable. Éste usa un servidor de aplicaciones para comunicarse con los módulos y el interpretador TCL [69].

SmartSoft

SmartSoft [94][74] es un proyecto liderado por Schelegel y Worst en el Instituto de Investigación para el Conocimiento Aplicado en la Universidad de Ulm en Alemania. SmartSoft es un marco de trabajo software por componentes para soportar una arquitectura de robots móviles.

Con SmartSoft es posible definir un sistema como un conjunto de módulos los cuales interactúan intercomunicando entre ellos. Los módulos tienen una estructura uniforme organizada en un área de usuario y en un área del marco de trabajo. Un módulo es un proceso, el cual puede ser multihilo. Existe un hilo proporcionado y controlado por el marco de trabajo, el cual es transparente a los diseñadores de módulos, que es responsable de todas las intercomunicaciones entre los hilos internos de cada módulo y también para soportar las comunicaciones entre los módulos. El usuario puede organizar la funcionalidad de un módulo usando múltiples hilos si se considera necesario.

Cada módulo proporciona diferentes servicios y estos servicios se proveen por medio de objetos proxy que se denominan “Objetos Servidores”. El modelo usado para la intercomunicación entre módulos es el cliente-servidor. Si un módulo desea accesar los servicios de otro módulo, éste necesitará ser asistido por el objeto servidor desde el segundo módulo. Igualmente, él actuara de la misma manera para ofrecer sus servicios a otros módulos.

Un aspecto clave de SmartSoft es el soportar el núcleo de su marco de trabajo en la definición y uso de los patrones de comunicación: *AutoUpdated Timed, AutoUpdated Newest, Command, Query, Event y Configuration*.

El patrón *AutoUpdate* es a menudo conocido como el servicio *Push*. Es un concepto basado en subscripción usado cuando los clientes desean nuevos datos tan pronto como estos están listos. Su uso típico se da cuando los comportamientos deben consumir datos provenientes de servidores de datos de sensores.

La versión temporizada del patrón *AutoUpdate* es empleada cuando los clientes, durante la solicitud de subscripción, envían un valor indicando el intervalo de tiempo entre actualizaciones. Esta versión también puede ser llamada modo síncrono. La versión *newest* del patrón *AutoUpDate* se emplea cuando los clientes desean actualizaciones tan pronto cuando los datos están disponibles, sin importar cuan largo o corto sea el intervalo de tiempo. Este tipo de patrón se puede llamar modo asíncrono. El patrón *Query* es similar a un método que retorna un valor por llamada. La llamada puede opcionalmente tener un parámetro. Este patrón puede verse como el servicio *Pull*. El patrón *Command* es un método de llamada simple que no retorna valor (similar a las funciones con parámetro *void* de C o C++).

Las partes principales de la implementación software de SmartSoft están basadas en la plataforma ACE (*Adaptive Communication Environment*) [142] que permite la portabilidad entre diferentes plataformas. Las rutinas de comunicación de bajo nivel se construyen al tope de TCX [143]. Este marco de trabajo no impone estructura alguna en el área de usuario dentro de cada módulo, es competencia del diseñador hacer uso correcto de las primitivas que provee la herramienta [144].

BALT & CAST

Balt & Cast es un *toolkit* fruto del proyecto de investigación COSY realizado en la Universidad de Birmingham en el Reino Unido en el 2005 [145]. Estas herramientas están soportadas en la arquitectura CAS. Ésta básicamente es una metodología de diseño de arquitecturas, que consiste en realizar un análisis de escenarios detallados para derivar requerimientos que conducen a unos principios de diseño para arquitecturas que pueden ser expresados en términos de esquemas arquitecturales. Estos definen un gran espacio de diseño que contiene muchos diseños específicos: las instancias arquitecturales. El *toolkit* tiene dos partes: una capa software BALT que actúa como *middleware* de conexiones, y una capa de estructura sobre BALT que implementa la arquitectura de esquemas CAST.

BALT (*Boxes and Lines Toolkit*) es una plataforma de comunicaciones implementada sobre CORBA, que proporciona conexiones tipo *push y pull* entre componentes que corren en hilos individuales. Soporta C++ y Java. Los componentes pueden estar interconectados entre máquinas y lenguajes diferentes sin necesitar cambios en la estructura usada dentro del código del componente. Todas las conexiones están basadas en interfaces tipo de tal manera que la validez de la conexión puede ser verificada durante la conexión.

CAST (*CAS Toolkit*) es una herramienta que proporciona clases abstractas de C++ y Java para cada uno de los componentes de CAS: componentes administrados y no administrados (componentes de procesamiento), sub arquitecturas de manejo de tareas o sub arquitecturas de memorias de trabajo. CAST extiende la interface de configuración de BALT para proporcionar un mecanismo que combine componentes en sub arquitecturas y para que éstas sean combinadas en una arquitectura completa. Los componentes en CAST más que interactuar directamente comparten información vía memorias de trabajo. Un componente de procesamiento puede escribir datos a su memoria de trabajo por varios mecanismos, todos los cuales requieren que el componente llamado proporcione el dato con un ID y el tipo de información de interés. Cuando los datos son escritos a una memoria de trabajo, los objetos cambiados son propagados a todos los componentes de la sub arquitectura administrada y a todas las memorias de trabajo conectadas, las cuales envían los objetos a los componentes administrados de sus sub arquitecturas.

El mecanismo de control basado en tareas de la arquitectura de esquemas se realiza en conexiones entre los componentes administrados y una sub arquitectura de administración de tareas. Los componentes deben proporcionar la tareas que se desean ejecutar. Esta propuesta puede ser aceptada o rechazada por el administrador de tareas [146].

Balt & Cast es un herramienta útil para investigar arquitecturas de procesamiento de información aplicadas a robots móviles, debido a su enfoque en la arquitectura de esquemas y a que CAST se soporta en memorias de trabajo para compartir datos, esto significa que algunos modelos de procesamiento serán más fáciles de implementar que otros [146].

RT-Middleware

RT-Middleware (*Robot - Technology - Middleware*) es una plataforma para aplicaciones robóticas japonesa desarrollada en conjunto por el Ministerio de Economía Japonés, el METI (*Ministry of Economy, Trade and Industry*) Japonés, la Asociación Robótica Japonesa (JARA) y el Instituto Nacional Japonés de Tecnología Industrial y Ciencia Avanzada (AIST) del año 2005. RT-Middleware es una infraestructura software basada en CORBA que se implementó usando un número determinado de especificaciones al nivel de la interface *middleware* distribuida, a partir de lo cual se desarrolló un prototipo llamado OpenRTM [83].

El principal objetivo de este *middleware* es construir los elementos funcionales de un robot en una estructura modular a nivel software con el fin de que el diseño del sistema de control se limite simplemente a combinar los módulos seleccionados. Los componentes usados para construir los sistemas se denominan componentes RT. Estos objetivos permiten que los diseñadores de sistemas o simplemente los integradores construyan sistemas de control de robots optimizados para una amplia variedad de aplicaciones de una forma eficiente y efectiva en costo. Otro objetivo importante es hacer los robots más inteligentes distribuyendo los recursos necesarios en red. RT-Middleware proporciona todos los servicios necesarios para facilitar que las aplicaciones robóticas operen en forma distribuida.

En la actualidad se realizan varios esfuerzos para emplear RT-Middleware como un estándar para componentes robóticos por parte de OMG (*Objetc Management Group*). Estos esfuerzos permitirían una rápida integración de los componentes robóticos implementados por diferentes fabricantes [84].

MIRO

MIRO (*Middleware for Robots*) fue desarrollado en el trabajo de doctorado de Hans Utz en la Universidad Alemana de Ulm en el 2005[89]. Éste es un marco de trabajo orientado a objetos distribuidos para control de robots móviles, que tiene como propósito facilitar la integración de software heterogéneo y mejorar la portabilidad y mantenibilidad del sistema software del robot. Los componentes claves de MIRO han sido desarrollados en C++ para Linux soportado sobre tecnología CORBA(*Common Object Request Broker Architecture*), empleando como infraestructura de comunicación el *middleware* ACE (*Adaptive Communication Environment*)[142]. Debido a la independencia del lenguaje de programación que proporciona CORBA, los componentes pueden ser escritos en cualquier y para cualquier SO que disponga de una implementación CORBA.

En la actualidad MIRO provee soporte hardware para tres plataformas: B21 de *iRobot*, el *Pionner* de *ActiMedia* y algunas plataformas no comerciales. Las interfaces incluyen abstracciones hardware para odometría, movimiento, sensores de rango (sonar, infrarrojo, bómper, láser), *stall*, video, sistema *pantilt*, botones GUI y voz. Los componentes intercambian datos a partir de suscripciones, por medio de las cuales se les hace llegar notificaciones conducidas por eventos.

MIRO incluye una máquina de comportamientos que permite especificar

comportamientos reactivos, la cual permite una descomposición temporal jerárquica de conjuntos de eventos y comportamientos – tareas. Hay dos tipos de transiciones locales y globales, que pueden ser editadas por medio de una interface gráfica. La configuración del hardware, las subscripciones de datos y las especificaciones de datos tipo *log* se guardan en archivos XML. Dos tipos de datos *log* se definen: "log de niveles" y "log de categorías", que permiten a los desarrolladores variar la granularidad de los datos de *logging*, mientras la herramienta gráfica LogPlayer, permite observar los datos de *logging* [147].

MARIE

MARIE (*Mobile and Autonomous Robotics Integration Environment*)[81] fue desarrollado en el trabajo de doctorado de Carle Côté en la Universidad de Sherbrooke en Canada en el 2003. Es un ambiente de programación específicamente diseñado con el firme propósito de ser usado para integrar y distribuir componentes, herramientas y aplicaciones robóticas desarrolladas por otros grupos de investigación. MARIE está implementado en C++ y emplea como plataforma integradora la plataforma de comunicación ACE. Gracias al mediador de patrones de diseño MARIE suministra un componente centralizado que conecta una amplia variedad de software diferente.

MARIE tiene cuatro componentes funcionales: adaptadores de aplicación, adaptadores de comunicación, administradores de aplicación y administradores de comunicación. Los adaptadores de aplicación actúan como *proxies* entre el componente central y las aplicaciones. El adaptador de comunicaciones traslada datos entre los adaptadores de aplicaciones. Los administradores de comunicaciones crean y administran enlaces y finalmente los administradores de aplicaciones coordinan los estados del sistema y configuran y controlan los componentes del sistema en cualquier nodo de procesamiento.

Con el fin de mantener su objetivo de ser un ambiente de integración de herramientas, se han desarrollado componentes para Player/Stage, CARMEN y ARIA. Adicionalmente MARIE viene con un potente par de herramientas: *FlowDesigner* y *RobotFlow* [148]. El primero es una librería para el procesamiento de flujo de datos acoplada con un despliegue gráfico que permite a los desarrolladores crear bloques software reusables enlazados en una red. Esta librería incluye soporte para procesamiento de señales, procesamiento de audio, cuantización de vectores, redes neuronales, lógica difusa y un *plugin* para la herramienta matemática Octave. Por su parte RobotFlow es el *toolkit* para robots móviles de *FlowDesigner* que incluye soporte para los robots *Pioneer2* de la

empresa *MobileRobots*, múltiples dispositivos hardware, comportamientos, máquinas de estado finito, procesamiento de imágenes e interfaces para ser usadas con MARIE [82].

ORCA2

ORCA2 es desarrollado en el departamento de Robótica de Campo de la Universidad de Sydney en Australia por Alex Brooks en el 2004 . ORCA2 se puede definir como un marco de integración de software robótico. Éste hace uso de los conceptos de la ingeniería de software basada en componentes (CBSE) y soporta su infraestructura de comunicación en el *middleware* ICE (*Internet Communication Internet*)[149] de la empresa ZeroCs. El concepto clave de ORCA2 es representar una aplicación software robótica como una red de comunicación de componentes software independientes. ORCA2 habilita un acercamiento flexible en el desarrollo de una aplicación mientras al mismo tiempo dirige este proceso definiendo unas líneas guías que se deben seguir con el fin de alcanzar resultados eficientes. Por ejemplo que en todos los componentes se deben implementar interfaces de comunicación.

En la infraestructura de comunicaciones ORCA2 usa como mecanismo de comunicación ICE. Éste permite métodos de comunicación tanto síncronos como asíncronos, permitiendo que tanto los clientes como los servidores tengan la oportunidad de procesar y comunicarse en ambos modos. Los componentes de ORCA2 usualmente basan su comunicación en un modelo *Publisher – Subscriber* usando un servicio de notificación.

ADE

ADE (*APOC Development Environment*) es un proyecto liderado desde el 2004 por Matthias Scheutz del Laboratorio de Interacción Robot Humano de la Universidad de Notre Dame [95]. Es un ambiente de programación que combina: 1) soporte para desarrollar e implementar arquitecturas cognitivas basadas en agentes, con 2) la infraestructura necesaria para implementarlas como componentes distribuidos. Un objetivo explícito de ADE es combinar las características de los sistemas multi agentes (tratando los componentes de la arquitectura como agentes en un sentido MAS), con *toolkits* y un ambiente de programación para diseño e implementación de agentes complejos. ADE es una implementación Java de la arquitectura de agentes universales APOC (*Activating, Processing, Observing Components*) [138] de Scheutz, la cual provee niveles arbitrarios de abstracción e interconexión de componentes. La comunicación

entre los componentes de ADE se realiza con las facilidades RMI (*Remote Method Invocation*) de Java. ADE proporciona toda una infraestructura de componentes para un servicio de nombres mejorados, monitoreo y mediación de conexiones, características de seguridad (control de acceso y autenticación), y la habilidad para almacenar el estado en tiempo de ejecución del sistema, lo cual permite la detección y recuperación de fallas en componentes.

ADE está limitado a robots de MobileRobots y de Arrick Robotics, pero dispone de un conjunto de abstracciones para sensores y actuadores comunes, lo que permite extender su soporte para otros tipos de plataformas. Los archivos de configuración pueden ser hechos en forma de archivos de texto o XML y deben incluir una descripción abstracta de la arquitectura y/o la especificación en tiempo de ejecución de la distribución del componente. Existen representaciones gráficas de componentes individuales, accesibles vía una interfaz gráfica multi usuario distribuida, la cual proporciona una vista completa de la arquitectura de agentes y el medio para controlar componentes individuales. Facilidades de *logging* permiten que cualquier componente en ADE escriba a múltiples archivos. ADE proporciona varios componentes predefinidos para definición de comportamientos, procesamiento de imágenes, reconocimiento y generación de voz, un interpretador de reglas de propósito general, una interface con Prolog y envoltorios para permitir el uso de software externo.

ADE tiene como herramientas gráficas una implementación Java de un cliente Player que provee una interfaz con el simulador 2D de multi robots Stage y con otros componentes de la plataforma Player/Stage.

3.3 Análisis de herramientas de las plataformas de trabajo

El listado de plataformas de trabajo de la tabla 3.1 y expuesto en la sección 3.2 no pretende ser definitivo. Como se puede apreciar de la sección de *Plataformas y herramientas robóticas* del capítulo uno, existe una amplia variedad de herramientas desarrolladas en los noventa y es aún mayor el número de las herramientas desarrolladas en la primera década del siglo veintiuno. Herramientas como MissionLab, IPC, TCM, TDL, GenoM, SmartSoft, ACE, ICE, TCX, CORBA, fueron desarrolladas en los noventa, mientras que TeamBots, Player, Stage, Gazebo, Balt & Cast, OpenRTM, Miro, Marie, FlowDesigner, RobotFlow, Orca2, ADE y muchas otras que no se incluyeron en la tabla 3.1 fueron desarrolladas en la primera década del siglo veintiuno.

Una situación bastante confusa con las herramientas es qué funcionalidad cumplen cada una de ellas, para ello se definirán una serie de términos comunmente empleados cuando se habla de herramientas en robótica móvil:

- Plataforma de trabajo robótica: es un conjunto de herramientas diversas que en conjunto permiten diseñar, depurar, validar, ejecutar y monitorear una arquitectura de control. Ejemplos: ADE, Marie, Miro, OpenRTM y SmartSoft.
- *Middleware* robótico: es una herramienta que proporciona una infraestructura de comunicaciones y procesamiento distribuido. Ejemplos: IPT, RPC, NML, MPI, ICE y las implementaciones suaves de Corba ACE y TAO y, claro está, CORBA.
- Capa de abstracción *hardware*: es una herramienta que proporciona un nivel de abstracción *software* de dispositivos sensores y actuadores. Ejemplos: Player.
- Simuladores gráficos: son herramientas que permiten visualizar en ambientes 2D y 3D el comportamiento del robot en un entorno simulado. Ejemplos: Stage y Gazebo.

El anterior listado no constituye los únicos tipos de herramientas, se requieren herramientas para realizar la validación y verificación, la supervisión y monitoreo del sistema, entre otras. De este grupo de herramientas la más importante es la que hace de *middleware*, que no solo debe suplir un soporte de procesamiento distribuido sino que además debe hacer que este mecanismo sea transparente para los elementos que los emplean, en particular para los algoritmos de control del robot que se encuentran en diferentes módulos o agentes, suministrándoles interfaces de comunicación genéricas. A continuación se expondrán una serie de seis aspectos que se resaltan al analizar las diferentes herramientas.

Un primer aspecto con la variada oferta de plataformas de fuente abierta es que la mayor parte de las mismas básicamente son un middleware especializado para robótica, que, en forma ideal, debe suministrar el soporte para la distribución de módulos de procesamiento con multiplicidad de modelos de comunicación. Por lo que todo el soporte de algoritmia en procesamiento de señales, fusión sensorial, diseño de comportamientos, visión de máquina, SLAM, modelamiento del mundo, sistemas de planificación, etc. y la misma arquitectura de control, deben ser diseñados por los interesados. Este es el caso de la concepción de Player lanzada en el año 2000, y que la hizo tan popular, ya que a diferencia de otras herramientas del noventa no se restringía a una arquitectura de control en particular, por lo que una gran cantidad de grupos de investigación en el mundo la hicieron su plataforma de trabajo. Esto a su vez redundó en el beneficio de Player ya que los diversos grupos fueron creando manejadores de dispositivos y algoritmos empleando la definición de interface estándar de Player, lo que promueve la reusabilidad de código dentro de la plataforma.

Un segundo aspecto es la importancia que tienen las herramientas de simulación, en este sentido Player demostró que este tipo de herramientas diseñadas en forma transparente con el *middleware*, esto es, no hay que hacer cambios en el diseño de la arquitectura para trabajar con el hardware real del robot o el robot virtual en el ambiente de simulación, son claves para el posicionamiento de cualquier plataforma de trabajo.

Un tercer aspecto es la carencia de plataformas que suministren una gama completa de herramientas para cubrir todas las etapas del desarrollo que requiere un proyecto de robótica móvil, en este sentido cobran importancia los proyectos de integración de herramientas como MARIE, que buscan reunir en una plataforma las mejores herramientas de apoyo existentes. Obviamente MARIE no es el único proyecto de este tipo, otros como CARMEN también buscan el mismo propósito. Otros proyectos por ejemplo ADE, SmartSoft, etc, tratan de cubrir todos los aspectos pero la tarea es tan monumental que es imposible realizarla por un único grupo de investigación en robótica o en inteligencia artificial.

Un cuarto aspecto es la imposibilidad aún de compartir código especializado entre estas diferentes plataformas, y la dificultad no se debe al *middleware*, realmente herramientas como CORBA permiten que se puedan comunicar componentes (módulos o agentes) entre diferentes infraestructuras de comunicación, sino al hecho de no compartir una misma interfaz de estructuras y tipo de datos para todos los algoritmos diseñados en las diferentes plataformas, lo que dificultad pasar información en forma transparente entre componentes y obviamente afecta la reusabilidad del código [150].

Un quinto aspecto es que el "tiempo real" raramente es un criterio de diseño principal en las actuales plataformas por lo que muchas de éstas emplean Java como lenguaje de implementación, por ejemplo TeamBots y ADE. Muchos investigadores consideran a Java inapropiado para sistemas de misión crítica como lo son los robots, sin embargo debe tenerse en cuenta que estudios a finales de los noventa y en la actualidad han demostrado [151] que una gran porción del tiempo que el sistema gasta en un lazo de control consiste en llamadas al *hardware*. Esto es totalmente independiente del lenguaje de programación usado. Adicionalmente siempre está la opción de usar JNI (*Java Native Interface*) para incrustar código crítico de tiempo real escrito en otros lenguajes o hacer uso de un sistema operativo de tiempo real como RTAI o VxWorks, como diversas plataformas lo hacen.

Un aspecto final es el hecho de que son pocas las plataformas que están integradas con un formalismo matemático definido que permita hacer análisis rigurosos al diseño de la

arquitectura de control, con el fin de evitar situaciones de bloqueo u otras propiedades y se puedan diseñar mecanismos de recuperación ante errores o fallos, etc., que son tan o más valiosos que el simple hecho de que el sistema funcione. Esta es una preocupación vital para agencias como la NASA y las entidades militares como el *Army* o el DoD en los Estados Unidos, y son pocos los grupos de investigación en la academia que atienden esta necesidad, por esta razón plataformas como ADE o las nuevas mejoras en verificación formal realizadas a la herramienta GeNom [152] adquieren importancia en este aspecto.

3.4 Resumen

En este capítulo se ha realizado un repaso por un listado de once plataforma de trabajo robóticas seleccionadas en este trabajo: IPC/TDL, TeamBots, Player, GeNom, SmartSoft, Balt&Cast, RT-Middleware, Miro, Marie, Orca2 y ADE. A continuación se expondrá brevemente la generalidad con respecto a las arquitecturas de control de las plataformas estudiadas más relevantes citadas en la literatura.

Con las herramientas IPC y TDL se pueden diseñar sistemas de control para robots móviles autónomos simples supeditados a una arquitectura específica TCA. Aunque IPC suministra un componente de distribución, su acercamiento basado en *socket* lo hace limitante para considerarlo como un marco de comunicaciones general para robots móviles.

TeamBots puede considerarse como un marco de trabajo general para robots móviles, pero que no se restringe a una arquitectura de control determinada. Tiene facilidades para ejecutar un sistema de control en una aplicación de un proceso simple lo que lo hace no distribuido y menos robusto. TeamBots puede ser empleado para el control de robots simples y múltiples [79].

EL proyecto P/S/G ofrece una capa de abstracción de dispositivos hardware para plataformas robóticas sean éstas simuladas o físicamente disponibles. Sin embargo no impone restricción alguna en la arquitectura de control, esto queda libre al diseñador robótico, tanto así que no brinda interfaces para los niveles superiores ni reactivos ni deliberativos, por lo que puede ser usado para diseñar cualquier tipo de arquitectura [78].

Las partes principales de la implementación software de SmartSoft están basadas en la plataforma ACE (*Adaptive Communication Environment*) [142] que permite la portabilidad entre diferentes plataformas. Las rutinas de comunicación de bajo nivel se construyen al tope de TCX [143]. Este marco de trabajo no impone estructura alguna en el área de usuario dentro de cada módulo, es competencia del diseñador hacer uso correcto de las primitivas que provee la herramienta [144].

Balt & Cast es un herramienta útil para investigar arquitecturas de procesamiento de información aplicadas a robots móviles, debido a su enfoque en la arquitectura de esquemas y a que CAST se soporta en memorias de trabajo para compartir datos, esto significa que algunos modelos de procesamiento serán más fáciles de implementar que otros [146].

4. Arquitectura Genérica de Control Multiagente AGC-MAS

En esta sección se propone la concepción y diseño de una arquitectura genérica de control basada en Sistemas Multi Agente, denominada AGC-MAS (Arquitectura Genérica de Control Multiagente), para emplearla en la especificación de cualquier tipo de arquitectura de control de un robot móvil.

4.1 Antecedentes

La temática abordada en este capítulo se soporta en: a) dos trabajos previos sobre sistemas multiagente aplicados a control de robots móviles, b) las propuestas civiles y militares sobre arquitecturas de referencia, y finalmente c) los trabajos de estandarización relacionados con la temática que en la actualidad se encuentran en discusión por la comunidad de investigadores.

Sistemas Multiagente aplicados a robots móviles

En esta área se destacan dos trabajos doctorales realizados en España en el presente siglo. El primero titulado "Una arquitectura Distribuida para el control de robots autónomos móviles" [88] de Humberto Martínez Barbera de la Universidad de Murcia del año 2001 y el segundo el trabajo titulado "Arquitectura para el control de robots móviles mediante delegación de código y agentes" [90] de Juan Luis Posadas de la Universidad Politécnica de valencia del año 2003.

En la propuesta de la Universidad de Murcia se define una Arquitectura Distribuida de Control (BGA) empleada como base de los diferentes elementos de control. En esta arquitectura se pueden definir procesos deliberativos y comportamientos reactivos. Una particularidad llamativa de este trabajo es que los diferentes elementos de la

arquitectura se comunican por medio de un protocolo basado en KQML, este era el lenguaje de comunicación de agentes empleado en el estándar de facto FIPA [134] sobre UDP. Esto significa que BGA es una arquitectura basada en agentes que no emplea un *middleware* de apoyo. Finalmente en el trabajo se ha definido un lenguaje (BG), similar en su sintaxis al lenguaje de programación C, que hace uso de la lógica difusa para especificar y definir los distintos módulos de control. Estos módulos se limitan simplemente a dos agentes: el reactivo y el deliberativo comunicados por un sistema de pizarra.

En la propuesta de la Universidad Politécnica de Valencia se diseña una arquitectura híbrida y distribuida basada en agentes, donde se tiene en cuenta las características de las redes y la capacidad de cómputo de los nodos de procesamiento, para determinar el lugar más adecuado de ejecución de los agentes. Esto significa que los agentes pueden moverse entre los nodos del sistema con el fin de reducir los requerimientos de comunicación en el acceso a los datos. Un agente puede decidir su ubicación óptima en función de índices de medida no convencionales basados en la antigüedad de los datos que reciba. Un segundo elemento innovador de la arquitectura es que permite mediante el uso de agentes móviles aplicar técnicas de delegación de código para realizar el control de los robots minimizando los problemas relacionados con la latencia de la red. El diseño de esta arquitectura se sustenta sobre un sistema de comunicaciones basado en una estructura de pizarra distribuida de objetos. El sistema de comunicaciones permite la interacción entre los distintos agentes de la arquitectura mediante una interfaz común para el acceso a los objetos de la pizarra. Los agentes podrán transmitir mensajes o enviar información al resto de agentes mediante la escritura y lectura de los objetos de la pizarra. Los datos o valores de dichos objetos están caracterizados temporalmente por el sistema a través de su antigüedad.

Arquitecturas de referencia y estandarización

En esta área el primer trabajo que se destaca es el realizado desde la década del setenta por el NIST (*National Institute Standars and Tecnnology*) en los Estados Unidos. Éste ha sido todo un proceso de evolución que inicio con el desarrollo de la arquitectura de control de tiempo real RCS, dirigida no sólo a robótica móvil sino también para automatización. Esta arquitectura evolucionó de la mano de la NASA en la década del ochenta en la jerárquica y pesada arquitectura NASREM, ya descripta en detalle en el

anexo 1, sin embargo un elemento no mencionado entonces es que, según James Albus en [153], hay cinco elementos principales requeridos para el desarrollo de un sistema robótico inteligente:

1. Una arquitectura conceptual.
2. Una arquitectura funcional.
3. Una arquitectura software.
4. Una arquitectura hardware.
5. Un ambiente de desarrollo software.

Esta mención es interesante ya que la gran mayoría de los artículos, sino todos aquellos relacionados con arquitecturas de control o cognitivas, sólo se limitan a esbozar un dibujo de cajas y flechas de la arquitectura funcional del sistema, pero detrás de esta arquitectura conceptual o funcional existen otros tipos de arquitecturas que deben integrarse para que esta arquitectura pueda ser operativa.

Cuando se hizo palpable a inicios de los noventa que las arquitecturas temporalmente rígidas y jerárquicas no eran adecuadas y se debía realizar un diseño arquitectural equilibrado entre reactividad y deliberación, el NIST rediseñó RCS para adecuarlo a los nuevos paradigmas de control de robots, integrándolo con una plataforma cognitiva alemana de procesamiento digital de imágenes VaMoRs 4-D de donde se originó 4D/RCS, Arquitectura por Modelo de Referencia para UGV [154]. Ésta es básicamente una arquitectura por modelo de referencia que provee unos fundamentos teóricos para el diseño, la ingeniería, la integración y las pruebas de sistemas software inteligentes para sistemas de vehículos no comandados o UGV, arquitectura que ha sido empleada por el *Army* en los Estados Unidos para el diseño de varios UXV en el programa Demo III, pero que para la comunidad académica no ha sido de interés. En el trabajo doctoral del año 2000 de Jay Gowdy en el CMU [155] se puede encontrar una potencial explicación a esa situación. En su tesis Gowdy establece que las arquitecturas de control existentes se pueden clasificar en dos grupos: las emergentes y las de referencia. Las del primer grupo son todas aquellas arquitecturas que se adecuan a las condiciones cambiantes de las tareas que se deben o podrían ejecutar, mientras que las del segundo grupo no permiten esta opción. RCS, NASREM y 4D/RCS pertenecen a este último grupo.

A pesar de las diferencias existentes en la forma de abordar los diseños de las arquitecturas por parte de los grupos de investigación en las universidades (flexibles) y las instituciones gubernamentales (rígidas), es de rescatar que estos últimos realizan

esfuerzos de integración arquitectural más eficientes que los que se logran en la industria o la academia. El ejemplo palpable es el de la arquitectura de referencia para AGV denominada JAUS [113] y el del marco arquitectural C4ISR [112] ambos apoyados por el DoD de los Estados Unidos.

C4ISR son las siglas en inglés de Comando, Control Comunicaciones, Computadores, Inteligencia, Vigilancia y Reconocimiento. El objetivo de C4ISR es proporcionar un estándar para que las agencias y servicios militares del DoD desarrollen su arquitectura particular de combate y soporte, incluyendo cualquier tipo de vehículo no comandado y cualquier otra clase de sistemas robóticos, en estas últimas situaciones es donde el DoD recomienda el empleo de JAUS. Con C4ISR se busca fácilmente integrar y habilitar operaciones conjuntas, para esto las arquitecturas deben presentarse en forma consistente permitiendo que los componentes arquitecturales sean reusables. C4ISR define tres tipos de arquitecturas:

1. Una Arquitectura Operacional (AO) la cual describe las tareas, unidades operacionales y los flujos de información requeridos con el fin de ejecutar una misión militar.
2. Una Arquitectura Técnica (AT) la cual contiene un conjunto de reglas que gobiernan la organización, interacción e interdependencia de los componentes del sistema. AT especifica paradigmas conceptuales del procesamiento, de las bases de datos y de las comunicaciones. AT también especifica estándares u diccionarios de datos.
3. Una arquitectura de sistemas (AS) la cual describe los componentes e interconexiones del sistema físico que se integran para una misión militar particular. La AS se construye para satisfacer los requerimientos de la arquitectura operacional según los estándares definidos en la arquitectura técnica.

Con C4ISR se ha logrado una integración de las diferentes arquitecturas, tanto militares como civiles, que participan de los programas de defensa del DoD.

La comunidad académica se ha preocupado por ejecutar iniciativas para formalizar estándares para robótica de servicios, a pesar de que esto aún no se ha logrado, varias organizaciones como la OMG [135], la red académica EURON [109], el comité de

estandarización de la IEEE RAS [110] y principalmente RoSta [108] realizan esfuerzos en este sentido.

RoSta es una acción coordinada ejecutada bajo el sexto programa marco (FP6) de la Unión Europea, cuyo propósito es reunir a los principales investigadores de la robótica de servicios para trabajar en estándares de robots y una arquitectura de referencia para robótica de servicios. Los objetivos que tiene por delante RoSta son monumentales:

1. Creación de un glosario/ontología para manipulación móvil y robots de servicios.
2. Especificación de una arquitectura de referencia para manipulación móvil y robots de servicios.
3. Especificación de una *middleware* para manipulación móvil y robots de servicios.
4. Formulación de *benchmarks* (para componentes, métodos, *middleware* y arquitecturas) para manipulación móvil y robots de servicios.

A nivel de Ontologías se busca definir toda la terminología técnica encontrada en las áreas temáticas que se encuentran en este campo. Igualmente el trabajo alrededor de la definición de una arquitectura de referencia aún se encuentra bastante incipiente, contando simplemente con un plan de trabajo inicial. De la misma forma las actividades alrededor de una especificación *middleware* genérica para robótica móvil y manipulación, junto con el diseño de pruebas de *benchmarks* para estos sistemas están aún en la fase de especificación de requerimientos.

Estos frentes de trabajo iniciaron formalmente a inicios del 2007 cuando, RoSta y el comité de estandarización RAS de la IEEE, establecieron un grupo de trabajo conformado por diversos investigadores del campo que con el apoyo de la organización de una serie de *WorkShops* abiertos y reuniones de expertos tratan temas como mediciones, comparabilidad, interoperabilidad y reusabilidad de conceptos arquitecturales y componentes en robótica. Las conclusiones de estos encuentros y las recomendaciones de este grupo de expertos se encuentran en un *white paper*, el cual se encuentra circulando desde principios del 2009 entre un grupo mayor de miembros de RoSta con el propósito de hacer realimentación hasta la fecha de su publicación final esperada para finales del 2009 [156].

A pesar de los recientes esfuerzos de RoSta por definir una arquitectura de referencia para robótica de servicios y manipuladores, ésta iniciativa ya se viene ejecutando desde 1997 para AGV por el DoD. Inicialmente por parte de la organización civil JAUS y desde el año 2003, cuando el grupo de estandarización de civil y comercial de JAUS decidió trasladar su trabajo, a la sociedad privada SAE [113]. El principal estándar JAUS es la arquitectura de referencia para AGV, el cual a pesar de limitarse sólo a vehículos militares con énfasis en tele operación puede extenderse a robots de servicios. Esta arquitectura de referencia consiste en la definición de un lenguaje de comunicación con una sintaxis y semántica bien definida para que sea empleada por todos los elementos de la arquitectura de control del AGV [157].

Por su parte la OMG (*Object Management Group*) inició a mediados de 2004 un proyecto de estandarización pero limitado sólo a componentes de tecnología robótica, en cuya iniciativa se apoya en la experiencia adquirida en la plataforma japonesa RT-*Middleware* [84]. Para principios del 2006 emitió un documento de *Request for Proposal* en este sentido [158].

A partir de los dos trabajos citados, de sistemas multiagente aplicados a robots móviles, la idea de datos de comunicación caracterizados por una estampa temporal y la definición de arquitectura de referencia para AGV de JAUS como un lenguaje de comunicación de mensajes, se propone la concepción y diseño de la arquitectura AGC-MAS para robots de servicios.

4.2 Arquitectura Genérica de Control Multiagente

La AGC-MAS es una arquitectura que toma la idea de modelar un sistema cognitivo como la interacción de múltiples elementos individuales de cómputo inteligentes propio de los sistemas multiagente. Éstos elementos son agentes que se comunican por medio de mensajes con una semántica bien definida y a los que adicionalmente se les agrega una estampa temporal. AGC – MAS consiste básicamente en la definición tanto de los diferentes mensajes que conforman el lenguaje de comunicación como en la definición de los diferentes agentes que hacen parte de la arquitectura.

4.2.1 Introducción AGC-MAS

El propósito de AGC-MAS es servir como una capa de abstracción superior ubicada sobre un *middleware* que implementa la infraestructura de comunicación distribuida de un sistema *software* pero independiente del mismo. Esta capa de abstracción permite el diseño de sistemas robóticos empleando la inserción y reuso de módulos inteligentes. Módulos que no sólo deben realizar deliberación sobre la funcionalidad realizada sino que además deben deliberar sobre las condiciones del sistema de comunicaciones y de la máquina de cómputo donde se encuentran en ejecución. En AGC –MAS se definen estos módulos inteligentes como un conjunto de agentes reusables y sus interfaces de comunicación. Con AGC – MAS se puede diseñar sistemas robóticos tanto tele controlados como autónomos sean estos deliberativos, híbridos o reactivos.

4.2.2 Topología de AGC-MAS

Un primer principio de AGC-MAS es el enfoque lógico y jerárquico que se propone en la concepción de un robot móvil concibiéndolo como un sistema organizado como una red de subsistemas y estos, a su vez, organizados en una red de nodos de computo y al interior de los nodos múltiples agentes interactuando por medio de mensajes, ver figuras 4.1 y 4.2.

En AGC-MAS se parte que un robot móvil es un sistema complejo conformado por el agrupamiento lógico de subsistemas, por ejemplo una plataforma móvil y un manipulador. Cada uno de estos subsistemas realiza una o más funciones del robot como una entidad única y localizada dentro de la estructura física del robot. Cada subsistema está conformado por una red de nodos de procesamiento, cada uno de estos nodos es responsable de permitir la ejecución de un conjunto coherente de funciones y brindar un mecanismo de coordinación para el flujo y control de los mensajes de AGC-MAS. Cada una de las funciones dentro del nodo son realizadas por agentes, responsables cada uno de una única capacidad funcional del robot móvil. Los mensajes de AGC-MAS se definen con respecto a cada una de estas capacidades. Los agentes sólo

pueden residir dentro de los nodos, pero no necesariamente deben estar asociados a un determinado nodo. Finalmente los mensajes de AGC-MAS están conformados por una cabecera y un campo de datos.

En el párrafo anterior se introdujeron varios términos que constituyen un robot móvil bajo AGC-MAS: Sistema, Subsistemas, Nodos y Agentes. Una representación gráfica de cómo cada uno de estos elementos AGC-MAS conforma un sistema robot se puede ver en las figuras 4.1 y 4.2.

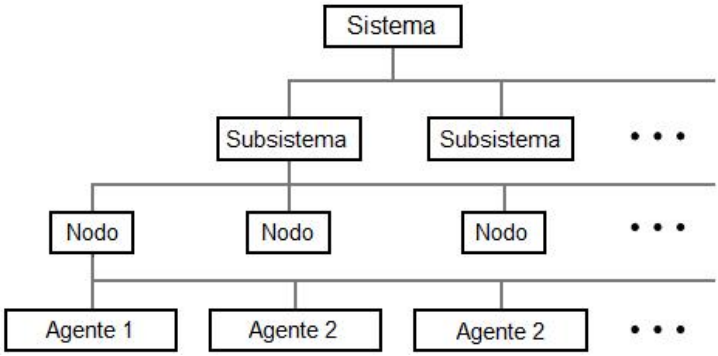


Figura 4.1 Topología jerárquica de un robot móvil bajo AGC-MAS

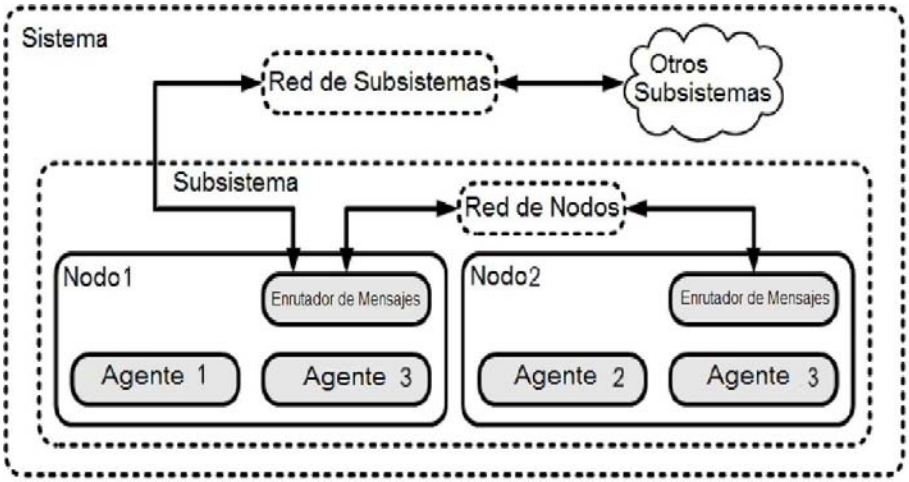


Figura 4.2 Topología de comunicaciones de un robot móvil bajo AGC-MAS

Una forma alternativa de ilustrar como los elementos de AGC-MAS se ensamblan para

conformar un robot se observa en la figura 4.2. En esta figura se observa que cada nodo consiste de dos o más agentes, donde por lo menos uno de ellos es un agente enrutador de mensajes, responsable de comunicar los mensajes entre los agentes residentes en un mismo nodo o entre agentes de diferentes nodos.

Nivel de Sistemas

Un sistema es un agrupamiento lógico de uno o más subsistemas. Un sistema puede verse como un agrupamiento que logra una sinergia a partir de los subsistemas constituyentes. Un ejemplo puede ser una plataforma móvil (subsistema móvil), que contiene un manipulador de varios grados de libertad (subsistema manipulador), los cuales se pueden supervisar y controlar a distancia desde una estación remota (subsistema monitoreo remoto), la integración de estos tres subsistemas conforma el sistema robot móvil.

Nivel de Subsistemas

Un subsistema es una unidad independiente. La unidad consiste al menos un nodo o una red de nodos de computo y el software necesario para permitir los requerimientos de funcionalidad. Son ejemplos de subsistemas una unidad de monitoreo remoto, una plataforma móvil, un manipulador o una red de sensores de vigilancia.

Nivel de Nodos

Un nodo está compuesto del *hardware* y *software* necesario para permitir el desarrollo de una capacidad de cómputo dentro del subsistema. Un nodo se puede ver como una caja negra que proporciona un determinado servicio. Ejemplos de nodos que pueden hacer parte de un subsistema son: nodo modelador del entorno, nodo de visión de máquina, nodo controlador de la plataforma móvil, etc. No es requisito que un nodo realice un único y determinado servicio, en un mismo nodo se pueden prestar, llegado el caso, varios servicios. Desde un punto de vista del hardware un nodo puede ser un computador y, si se cuenta con varios nodos, éstos pueden estar comunicados en una forma tan elemental como un enlace RS232 punto a punto o una red Ethernet.

Nivel de Agentes

Un agente es el nivel más bajo de la descomposición de un sistema AGC-MAS. Un agente es una unidad cohesiva de software que no sólo proporciona una funcionalidad bien definida sino que adicionalmente puede deliberar, sobre el comportamiento temporal del sistema de comunicaciones y de los recursos de cómputo de la máquina donde reside, con el fin de determinar su mejor ubicación. Cada agente tiene una interfaz de comunicaciones que básicamente es su capacidad de comunicarse por medio de los mensajes de AGC-MAS. Cada agente debe tener una dirección que permita ubicarlo dentro de la red de nodos y esta hace parte de los mensajes.

Finalmente se puede finalizar la explicación de la topología de niveles propuesta para AGC-MAS con la idea básica que cualquier subsistema está compuesto de agentes software, distribuidos en uno o más nodos de cómputo.

4.2.3 Definición de Agentes en AGC-MAS

El agente representa una o varias capacidades funcionales tanto hardware como software del sistema robótico. Un agente tiene las siguientes características:

1. Un agente reside solo en los nodos.
2. Un agente puede duplicarse, redundar o emigrar.
3. Los agentes se direccionan usando identificadores de subsistemas, nodos y agentes.
4. El identificador de un agente se compone de un nombre para el agente y un número de identificación.
5. Un agente intercambia mensajes con respecto a sus capacidades con otros agentes.

Un segundo principio de la Arquitectura Genérica de Control MAS es la definición de un conjunto determinado de mensajes y unas reglas de intercambio para todos los agentes.

4.2.3.1 Número de identificación de agentes

Cada agente tiene un número de identificación (ID) el cual se asigna de acuerdo a una clasificación a priori que se hace en AGC-MAS. Los agentes básicos son los asociados a los sensores exteroreceptores a estos se les asignan los números 00 a 09, los siguientes agentes son los reactivos a estos se les asignan los números 10 a 19, a los agentes de control de movimiento de la plataforma móvil se les asigna los números de 20 a 29, los agentes de comunicación van de 31 a 39, los agentes deliberativos van del 40 a 49 y los agentes de mando van del 50 a 59. Un caso especial es el agente que representa la plataforma móvil al cual se le asigna el ID30.

4.2.3.2 Clasificación de agentes en AGC-MAS

AGC-MAS propone una serie de agentes identificados a partir de los componentes básicos requeridos en el diseño de un sistema de control de movimientos general para un robot móvil. El sistema robótico puede combinar algunos de estos agentes de tal manera que se cumpla la funcionalidad del sistema dado. Los agentes se han organizado en grupos funcionales con números de identificación de acuerdo al siguiente listado:

a) Agentes extero receptores(ID 00 a 09)
b) Agentes reactivos (ID 10 a 19)
c) Agentes de control de movimiento (ID 20 a 29)
d) Agente plataforma móvil (ID 30)
e) Agentes comunicación (ID 31 a 39)
f) Agentes deliberativos (ID 40 a 49)
g) Agentes de mando (ID 50 a 59)

El agente plataforma móvil representa tanto el vehículo, sus actuadores de movimiento, los manejadores de potencia de los mismos y los intero receptores. Los agentes extero receptores representan todos los sensores externos necesarios para asistir en la navegación del vehículo, los agentes de control de movimiento representan todos los agentes necesarios para implementar un sistema de control de movimientos cuya consigna de movimientos es establecida por los agentes deliberativos, los agentes reactivos son los responsables de darle capacidad de reacción inmediata a la plataforma ante la presencia de obstáculos, los agentes de mando permiten el control de alto nivel donde se establece el objetivo o misión para el sistema o subsistema robótico. Los

agentes de comunicación son los responsables de brindar la facilidad de comunicación de mensajes al resto de los agentes.

Agentes Extero Receptores

Éstos agentes representan los sensores responsables de la medición de las variables físicas necesarias para evaluar el desempeño de la navegación de la plataforma. Se constituyen en los sensores de las variables de proceso del sistema de control de movimientos de la plataforma. Inicialmente se han identificado cinco agentes sensores, de acuerdo al siguiente listado donde se consignan con sus respectivos nombres e identificación numérica:

1. Sensor_Rango(ID00): es el sensor de medición de rango normalmente es un radar laser. El Mensaje asociado es Reporte_Rango.
2. Sensor_Posición _Local(ID01): es el sensor o grupo de sensores que permiten calcular la localización (posición + orientación) de un punto de la plataforma, vg. centro de gravedad, con respecto a un marco de referencia local. El mensaje asociado es Reporte_Posición _Local.
3. Sensor_Velocidad(ID02): es el sensor que permite medir la velocidad de desplazamiento de la plataforma. El mensaje asociado es Reporte_Velocidad.
4. Sensor_Posición_Global(ID03): es el sensor GPS que entrega la localización de la plataforma con respecto a un marco de referencia global del mundo. El mensaje asociado es Reporte_Posición _Global.
5. Sensor_Visión(ID07): es el sensor que representa el sistema de visión de máquina por procesamiento digital de imágenes normalmente estéreo. El Mensaje asociado es Reporte_Imagen.

Todos los anteriores agentes son responsables de las interfaces internas para recibir los datos crudos de los sensores y mapearlos a mensajes AGC-MAS.

Agentes Reactivos

Estos agentes representan los comportamientos reactivos que habilitan al sistema para responder en forma inmediata ante una situación que ponga en riesgo la ejecución de la misión, en este caso la presencia de un obstáculo en el camino. Inicialmente se han

identificado dos agentes reactivos, de acuerdo al siguiente listado donde se consignan con sus respectivos nombres e identificación numérica:

1. Evadir_Obstaculos(ID10): es el comportamiento que permite sabiendo lo que sucede en el entorno modificar el esfuerzo de control que va dirigido al proceso, esto es el vehículo, con el fin de lograr la evasión del obstáculo o mantener la integridad del móvil. Su mensaje asociado es Fijar_Esfuerzo_Control_modificado.
2. Deambular(ID11): es el comportamiento que permite fijar un esfuerzo de control según la información recibida por los agentes extero receptores con el fin de deambular por el entorno. Su mensaje asociado es Fijar_Esfuerzo_Control.

Éstos agentes reactivos se organizan en forma jerárquica de acuerdo a la división propuesta por Rodney Brooks en la arquitectura *Subsumption*. En el anterior listado sólo se han implementado los dos primeros niveles. Éstos agentes son responsables de recibir el mensaje esfuerzo de control y los mensajes de los agentes extero receptores y modificar el esfuerzo de control convenientemente de tal forma que se logre el comportamiento reactivo deseado.

Agentes Control de Movimiento

Éstos agentes representan los módulos responsables de la implementación de sistemas de control de movimientos en lazo cerrado de la plataforma móvil. Se han propuesto tres tipos de controladores de movimientos tanto para marcos de referencia local como global, de acuerdo al siguiente listado donde se consignan con sus respectivos nombres e identificación numérica:

1. Controlador_Local(ID20): es el controlador que dada una consigna en forma de una localización local y velocidad deseada permite que se genere el esfuerzo de control que minimice la localización local real de la plataforma con respecto a la deseada. Su mensaje asociado es Fijar_Esfuerzo_Control.
2. Controlador_Trayectos_Local(ID21): es el controlador que dado un vector de localizaciones sucesivas locales permite generar los esfuerzos de control que logran que la plataforma transite por los diferentes trayectos. Su mensaje asociado es Fijar_Esfuerzo_Control.
3. Controlador_Caminos_local(ID22): es el controlador que dado un vector de

localizaciones sucesivas locales permite generar los esfuerzos de control que logran que la plataforma transite por los diferentes puntos siguiendo un camino particular definido. Su mensaje asociado es Fijar_Esfuerzo_Control.

4. Controlador_Global(ID29): es el controlador que dada una consigna en forma de una localización global deseada permite que se genere el esfuerzo de control que minimice la localización local real de la plataforma con respecto a la deseada. Su mensaje asociado es Fijar_Esfuerzo_Control.
5. Controlador_Trayectos_Global(ID28): es el controlador que dado un vector de localizaciones sucesivas globales permite generar los esfuerzos de control que logran que la plataforma transite por los diferentes trayectos. Su mensaje asociado es Fijar_Esfuerzo_Control.
6. Controlador_Caminos_Global(ID27): es el controlador que dado un vector de localizaciones sucesivas globales permite generar los esfuerzos de control que logran que la plataforma transite por los diferentes puntos siguiendo un camino particular definido. Su mensaje asociado es Fijar_Esfuerzo_Control.
7. Robot_Móvil(ID30): es el agente que representa la plataforma móvil con sus sensores internos, actuadores de movimiento y manejadores de potencia de los mismos. En el contexto del sistema de control de movimientos este agente representa la planta donde se ejecuta el proceso a controlar.

Todos los anteriores agentes por medio de sus interfaces reciben los mensajes que representan la localización(es) deseada(s) y velocidad deseada y entregan un mensaje esfuerzo de control hacia el agente Robot_Móvil. Este mensaje esfuerzo de control codifica el desplazamiento angular y la potencia que los manejadores de los actuadores de la plataforma deben aplicar. Esta codificación se debe interpretar y modificar según sea la arquitectura mecánica del robot móvil, esto se debe realizar al interior del agente Robot_Móvil.

Agentes de Comunicación

Estos agentes representan la doble funcionalidad de comunicación de mensajes entre agentes en un mismo nodo o enrutando los mensajes según el subsistema y nodo donde se encuentren los agentes que deben recibirlos. Se ha identificado un único agente de comunicación denominado Enrutador_Mensajes(ID35) que realiza esta doble funcionalidad. Estos agentes enrutadores de mensajes se encuentra ubicados dentro de cada nodo en cada subsistema.

Agentes Deliberativos

Éstos agentes representan la capacidad deliberativa del sistema representada en dos funcionalidades: el modelamiento y análisis del entorno del robot y la planificación de la misión. Inicialmente se han identificado dos agentes deliberativos de acuerdo al siguiente listado donde se consignan con sus respectivos nombres e identificación numérica:

1. Cartográfico(ID40): es un agente bastante complejo que a partir de la información suministrada por los diversos agentes extero receptores construye un modelo del entorno y a partir del mismo proporciona información valiosa para planificar las actividades que desarrollan la misión. Los mensajes asociados son Reporte_Objetos, Reporte_Límites y Reporte_Características.
2. Planificador_Misión(ID45): es el agente responsable de organizar el plan de actividades, vista como una secuencia temporizada de mensajes comandos en forma de árbol con posiciones y velocidades deseadas, que permiten desarrollar dinámicamente la navegación del robot a partir del objetivo de la misión. Los mensajes consigna de posiciones y velocidades deseadas deben ser ejecutados por el agente controlador de acuerdo a la coordinación secuencial y temporal que realice el agente Mando_Subsistema.

El planificador de misión se puede concebir como un sistema de supervisión robótico que a partir de la información actualizada del desempeño de la ejecución de la misión, de las perturbaciones existentes y del plan de navegación deseado fija las diferentes consignas necesarias para cumplir con la misión fijada.

Agentes de Mando

Éstos agentes realizan la coordinación general de las actividades y representan la capacidad de integración de los agentes de un subsistema con otros subsistemas o bien la interfaz hombre máquina para fijar la misión. Se han identificado dos agentes de acuerdo al siguiente listado donde se consignan con sus respectivos nombres e identificación numérica:

1. Mando_subsistema(ID50): es un agente coordinador de todas las actividades de un subsistema, responsable de la coordinación de los agentes de un subsistema con los agentes de otros subsistemas. En el caso de sólo existir un único subsistema también es el agente que hace de interfaz hombre – máquina, permitiendo que se fije la misión deseada o se realice un control manual directo del robot móvil. Sus mensajes asociados pueden ser: Fijar_Esfuerzo_Control, Fijar_Punto_Global_Deseado, Fijar_Trayectos_Global_Deseado, Fijar_Velocidad _Deseada, Fijar_Caminos_Global_Deseado, Fijar_Punto_Local_Deseado, Fijar_Trayectos_Local_Deseado, y Fijar_Caminos_Local_Deseado según sea el tipo de agente controlador.
2. Mando_Sistema(ID45): es el agente coordinador de los subsistemas de un sistema con el fin de cumplir una determinada función. También hace de interfaz hombre máquina con el fin de fijar la misión a ejecutar.

Normalmente cuando se tiene un subsistema sencillo como un robot móvil, el agente Mando_Subsistema recibe del agente Planificador_Misión el listado temporal de mensajes comando que constituyen la misión, el cual debe secuenciar y coordinar para que el respectivo agente controlador los ejecute.

4.2.4 Especificación de Mensajes en AGC-MAS

Los mensajes en AGC-MAS proporcionan las palabras que todos los agentes de la arquitectura genérica emplean para atender o solicitar servicios de otros agentes o bien solicitar que se ejecute una acción. Por esta razón los mensajes se encuentran organizados en tres clases: Reportes y Solicitudes para el primer caso y Comandos para el segundo caso. Los del primer caso se emplean para compartir información entre agentes y los del segundo caso se emplean para gobernar los cambios de estado en los cuales puede estar el robot móvil. Todos los mensajes se encuentran organizados en grupos funcionales: básicos, extero sensores, plataforma móvil y misión.

Mensajes Básicos

Este grupo funcional reúne los mensajes que todo agente debe soportar. Se encuentran

divididos en subgrupos.

1. Control de estado: éstos son los mensajes que determinan los estados en los cuales puede estar la plataforma móvil: Pendiente, Retomar, Apagar, Reiniciar, Parada_Emergencia y Salida_Emergencia.
2. Control de acceso: éstos son mensajes que permiten que un agente externo, un usuario humano, tome control en el sistema: Solicitud_Control y Liberar_ Control.
3. Configuración dinámica: son mensajes para permitir la configuración dinámica. Por ejemplo un agente solicita a otro agente por la lista de servicios que presta y así saber la funcionalidad prestada o bien indicar la presencia de un agente en un subsistema.
4. Configuración de comunicación: son los mensajes que permiten que un agente controlador configure el tipo de comunicación con los agentes extero sensores. Se pueden configurar dos tipos eventos periódicos: periódico sin reemplazo y servicio de conexión. En el primer caso los mensajes se van evacuando por parte del agente enrutador de mensajes a medida que los consume el agente destino, pueden quedar en cola si es el caso. En la segunda situación los mensajes no se colocan en cola y los nuevos pueden reemplazar a los mensajes viejos que no han sido consumidos. Otro tipo de evento que se puede configurar es el evento basado en un cambio. Ésto es al agente destino sólo se le envía un mensaje cuando ha sucedido un evento en el agente emisor.
5. Mensaje de funcionamiento: es un mensaje periódico de normal funcionamiento, que todo agente envía a su agente enrutador de mensajes con el fin de notificar que se encuentra vivo, y el agente enrutador de mensajes, a su vez, se lo comparte a los otros agentes por medio de un mensaje tipo *broadcasts*. Éstos mensajes pueden ser empleados para permitir una configuración dinámica según una temporización de perro guardián típica de los sistemas hardware basados en microprocesadores.

Mensajes de los extero sensores

Éstos son los mensajes que son enviados por los agentes extero sensores con el fin de reportar la información que ha sido registrada por el sensor y que ha sido solicitada por otro agente, o bien pueden ser mensajes asociados a configuraciones particulares de

funcionamiento para el sensor. Ejemplo Solicitud_Posición_Global, Reporte_Posición_ Global y Fijar_Orientación_Cámara.

Mensajes de la plataforma móvil

Éstos son los mensajes asociados al control de lazo cerrado realizado por el agente controlador con los agentes extero sensores y el agente robot móvil. Por ejemplo: Fijar _Esfuerzo_Control, Fijar_Velocidad, Reporte_Velocidad, Reporte_Posición_Global, Modificar_Esfuerzo_Control, etc.

Partes de un mensaje en AGC-MAS

En AGC-MAS se propone el formato básico para los mensajes típico de todo protocolo: cabecera más cuerpo. En la cabecera se consigna datos relacionados al tipo de mensaje y en el cuerpo la información de interés que se desea compartir. Para la cabecera se propone el siguiente formato:

a) Propiedades del mensaje: versión, acuse de recibo/no acuse de recibo, bandera servicio de conexión y bandera de mensaje expirado.
b) Código del comando: único para el mensaje, determina el comportamiento y los datos incluidos en el cuerpo.
c) Identificación de la fuente: es la dirección AGC-MAS del emisor del mensaje en la forma subsistema.nodo.agente.
d) Identificación del destino: es la dirección AGC-MAS del receptor del mensaje en la forma subsistema.nodo.agente.
e) Estampa de tiempo: es la hora global de emisión del mensaje.
f) Otros: Longitud del mensaje y campos de reemsamble de mensajes con múltiples partes.

Para el cuerpo del mensaje se debe asociar una longitud máxima a priori y contener un vector de presencia. Donde cada bit indique la presencia de un campo particular en el cuerpo del mensaje.

Agentes y Mensajes en AGC-MAS

La tabla 4.1 resume la relación entre cada agente con sus mensajes en la arquitectura de control genérica propuesta.

Tabla 4.1 Relación de los agentes con sus mensajes en AGC-MAS

Agente AGC-MAS	Mensajes Entrada	Mensajes Salida
Planificador_Misión(ID45)	Objetivo de la Misión	Trayectorias y comandos organizadas en forma de árbol
Mando_Sistema(ID55), Mando_Subsistema(ID50)	Trayectorias y comandos organizadas en forma de árbol.	Secuencia de trayectorias o puntos deseados y comandos
Sensor_Posición_Global(ID03), Sensor_Posición_Local(ID01), Sensor_Velocidad(ID02), Sensor_Rango(ID00), Sensor_Visual(ID07)	Datos, Datos, Datos, Datos, Datos Fijar_Configuración_Camara.	Reporte_Posición_Global, Reporte_Posición_Local, Reporte_Rango, Reporte_velocidad, Reporte_Visión
Robot_Móvil(ID30),	Fijar_Esfuerzo_Control	
Enrutador_Mensajes(ID35)	Los mensajes de todos los agentes de un nodo	Los mensajes enrutados
Controlador_Global(ID29), Controlador_Trayectos_Global(ID28), Controlador_Caminos_Global(ID27), Controlador_Local(ID20), Controlador_Trayectos_Local(ID21), Controlador_Caminos_Local(ID22)	Fijar_Punto_Global_Deseado, Fijar_Trayectos_Global_Deseado, Fijar_Caminos_Global_Deseado, Fijar_Punto_Local_Deseado, Fijar_Trayecto_Local_Deseado, Fijar_Camino_Local_Deseado, Reporte_Posición_Global, Reporte_Velocidad	Fijar_Esfuerzo_Control
Evadir_Obstaculos(ID10), Deambular(ID11)	Reporte_Posición_Global Reporte_Rango Reporte_Velocidad Fijar_Esfuerzo_Control	Fijar_Esfuerzo_Control_Modificado Fijar_Esfuerzo_Control
Cartográfico(ID40)	Reporte_Posición_Global Reporte_Rango Reporte_Velocidad y comandos del Planificador_Misión y de Mando_Subsistema, ppalmente.	Reporte_Objetos, Reporte_Limites, Reporte_Características, etc

4.2.5 Diagramas de Control Gráficos de AGC-MAS

Las secciones previas han delineado un grupo de agentes y sus mensajes de comunicación para una arquitectura de control genérica para robots móviles basados en agentes que se comunican por medio de mensajes. A continuación se explicará un método gráfico empleando AGC-MAS que permite diseñar diagramas de arquitecturas de control en forma incremental, partiendo desde un simple robot móvil telemanipulado hasta llegar a las clásicas arquitecturas de control empleadas en robótica móvil.

Para la representación gráfica de un diagrama de control en AGC-MAS se propone que los agentes se dibujen como bloques rectangulares con el nombre del agente en su interior sin incluir su identificación numérica. Las flechas que entran o salen de cada bloque representan los mensajes de solicitud, reporte o comando, preferiblemente dibujadas en el sentido arriba - abajo que izquierda - derecha. Para darle significado a la flecha se coloca el nombre del mensaje al lado derecho o en su parte inferior. Dependiendo de la configuración de la comunicación entre agentes a cada mensaje de solicitud le debe corresponder su mensaje de reporte pero, con el fin de no aumentar la complejidad de la representación gráfica, se recomienda dibujar sólo las flechas necesarias para interpretar rápidamente el gráfico. Por ejemplo un reporte de un sensor puede deberse a un mensaje de solicitud que llega periódicamente o que se origino una única vez cuando se solicito un servicio de conexión al agente sensor. Sea cual fuere la situación se recomienda dibujar sólo la flecha asociada al mensaje reporte del sensor. En AGC-MAS no se establece que cada grupo funcional de agentes tenga en su representación gráfica un color determinado, sin embargo los agentes sensores se representan con un color diferente al resto de los agentes para ayudar en la lectura del diagrama.

Los diagramas de control gráficos que a continuación se explicarán ilustran sólo el intercambio de mensajes solicitud y reporte realizados entre los agentes, constituyéndose sólo en una vista de las múltiples que debe tener un diseño. Estos diagramas de control AGC-MAS propuestos en el presente trabajo de maestría se deben complementar con otras gráficas de apoyo, por ejemplo diagramas de cambio estado

y/o diagramas de secuencia.

Escenario A: robot móvil tele operado

Requerimiento: Control manual directo de un robot móvil.
Agentes: Mando_Subsistema y Robot_Móvil
Diagrama de Control: ver figura 4.3

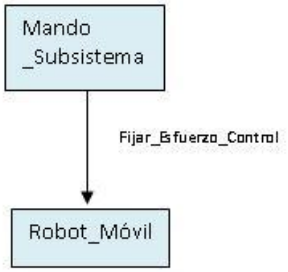


Figura 4.3 Robot móvil operado en forma manual

En el escenario A de la figura 4.3 se aprovecha el hecho de que el agente Mando_Subsistema está en capacidad de hacer de interfaz hombre-máquina y permitir que no sólo se fije la misión sino permitir un control manual del robot móvil entregando directamente la señal esfuerzo de control que debe ejecutar el agente Robot_Móvil.

Escenario B: control de la localización global de un robot móvil

Requerimiento: Control del movimiento en lazo cerrado de un robot móvil fijando una localización global deseada.
Agentes: Mando_Subsistema, Controlador_Global, Robot_Móvil, Sensor_Velocidad y Sensor_Posición_Global.
Diagrama de Control: ver figura 4.4

En el escenario B de la figura 4.4 el agente Mando_Subsistema es empleado para fijar una localización deseada al sistema de control de movimientos de lazo cerrado conformado por los agentes Controlador_Global, Robot_Móvil y Sensor_Posición_Global. En este lazo el agente Controlador_Global compara la consigna deseada con el reporte

de localización real y genera una solicitud de esfuerzo de control hacia el robot móvil con el fin de minimizar el error de localización. A pesar de que no se fija una consigna de velocidad el agente Sensor_Velocidad se comunica con el agente Controlador_Global, lo que permitiría al agente controlador monitorear la velocidad de ejecución, la cual pudo haber dejado en un valor por defecto, o controlar la velocidad en el caso de que llegue un mensaje de Fijar_Velocidad.

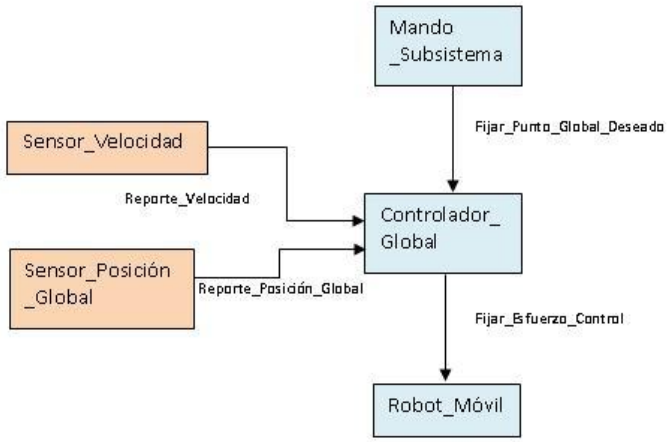


Figura 4.4 Control en lazo cerrado de un robot móvil hacia una localización deseada

Escenario C: control de la trayectoria global de un robot móvil

Requerimiento: Control del movimiento en lazo cerrado de un robot móvil fijando un trayecto global deseado.
Agentes: Mando_Subsistema, Controlador_Trayectos_Global, Robot_Móvil, Sensor_Velocidad y Sensor_Posición_Global.
Diagrama de Control: ver figura 4.5

En el escenario C de la figura 4.5 el agente Mando_Subsistema es empleado para fijar una serie de trayectos deseados, con su respectiva velocidad deseada, al sistema de control de movimientos de lazo cerrado conformado por los agentes Controlador _Trayectos_Global, Robot_Móvil, Sensor_Posición_Global y Sensor_Velocidad. En este lazo el agente Controlador_Global compara la consigna deseada de un trayecto con el reporte de localización real y genera una solicitud de esfuerzo de control hacia el robot móvil con el fin de minimizar el error de localización. En cada trayecto el objetivo del

controlador es llegar a la consigna deseada que marca el final de un trayecto y el inicio del siguiente sin importar la ruta seguida entre las dos localizaciones pero si guardando cumplir la velocidad deseada.

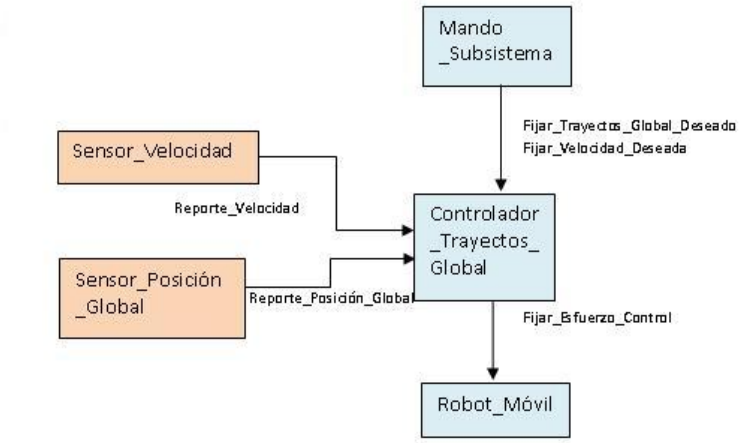


Figura 4.5 Control en lazo cerrado de un robot móvil de acuerdo a un trayecto deseado

Escenario D: control del camino global de un robot móvil

Requerimiento: Control del movimiento en lazo cerrado de un robot móvil fijando un camino global deseado.
Agentes: Mando_Subsistema, Controlador_Caminos_Global, Robot_Móvil, Sensor_Velocidad y Sensor_Posición_Global.
Diagrama de Control: ver figura 4.6

En el escenario D de la figura 4.6 el agente Mando_Subsistema es empleado para fijar una serie de caminos deseados con su respectiva velocidad deseada al sistema de control de movimientos de lazo cerrado conformado por los agentes Controlador _Trayectos_Global, Robot_Móvil, Sensor_Posición_Global y Sensor_Velocidad. En este lazo el agente Controlador_Global compara la consigna deseada de un camino con el reporte de localización real y genera una solicitud de esfuerzo de control hacia el robot móvil con el fin de minimizar el error de localización y seguir lo más fielmente posible las curvas de conexión para los puntos fijados para el camino.

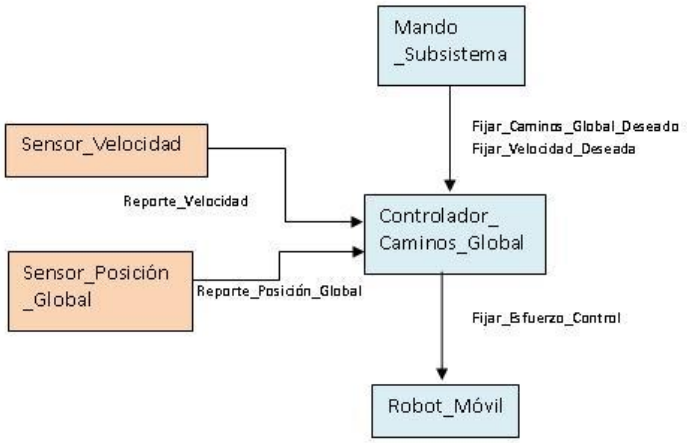


Figura 4.6 Control en lazo cerrado de un robot móvil de acuerdo a un camino deseado

Escenario E: control del camino global de un robot móvil con reactividad

Requerimiento: Control del movimiento en lazo cerrado de un robot móvil fijando un camino global deseado pero incluyendo un comportamiento de evasión de obstáculos.
Agentes: Mando_Subsistema, Controlador_Caminos_Global, Robot_Móvil, Sensor_Velocidad, Sensor_Posición_Global, Sensor_Rango y Evasión_Obstaculos.
Diagrama de Control: ver figura 4.7

En los diagramas de control de los escenarios B, C y D el agente Mando_Subsistema simplemente fija una misión a priori, consistente en una localización, trayectos o caminos deseados respectivamente, y espera que el agente controlador la ejecute lo más fielmente posible. Pero eso sólo sucederá si no hay presencia de perturbaciones para el robot, situación ideal que no sucede en la realidad. Por lo que un excelente complemento a los escenarios C, B y D cuando se presenten obstáculos en el recorrido, y evitar que el robot móvil pase a un estado de parada de emergencia y potencial bloqueo, es adicionarle un agente reactivo que autónomamente subsuma el esfuerzo de control del agente controlador hasta que se logre la evasión del obstáculo y el controlador pueda continuar ejecutando la misión, como se observa en el escenario E de la figura 4.7.

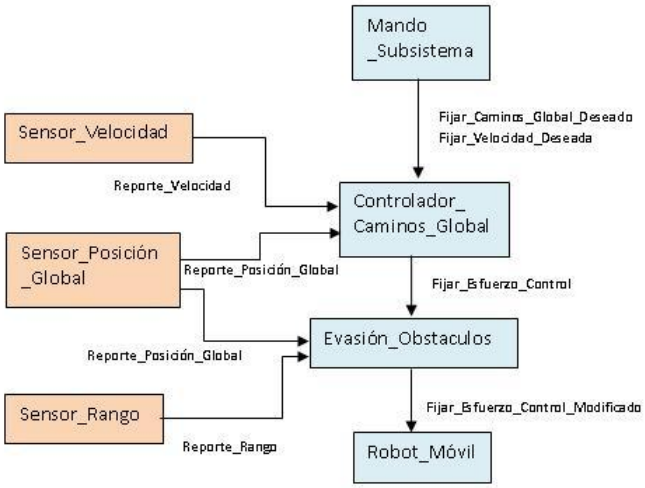


Figura 4.7 Control en lazo cerrado de un robot móvil de acuerdo a un camino deseado con capacidad de evasión de obstáculos

Con el fin de que el agente reactivo Evasión_Obstáculos realice adecuadamente su función debe contar con la información de los sensores Posición_Global y, principalmente, con el radar laser Sensor_Rango que le proporciona información exacta de la localización de puntos en un radio determinado del robot móvil en un frente angular de 0 a 190 grados. La anterior información también puede complementarse con el dato de la velocidad a la cual se desplaza el robot móvil, con el fin de modificar el esfuerzo de control del controlador permitiendo evadir el peligro, manteniendo la integridad del robot y evitando una replanificación de la misión.

Escenario F: control de camino con reactividad y planificación de misión

Requerimiento: Control del movimiento en lazo cerrado de un robot móvil fijando un camino global deseado con evasión de obstáculos, pero el camino obedeciendo a una planificación de una misión de alto nivel según un modelo del entorno.
Agentes: Mando_Subsistema, Controlador_Caminos_Global, Robot_Móvil, Sensor_ Velocidad, Sensor_Posición_Global, Sensor_Rango, Evasión_Obstaculos, Cartografico y Planificador_Misión.
Diagrama de Control: ver figura 4.8

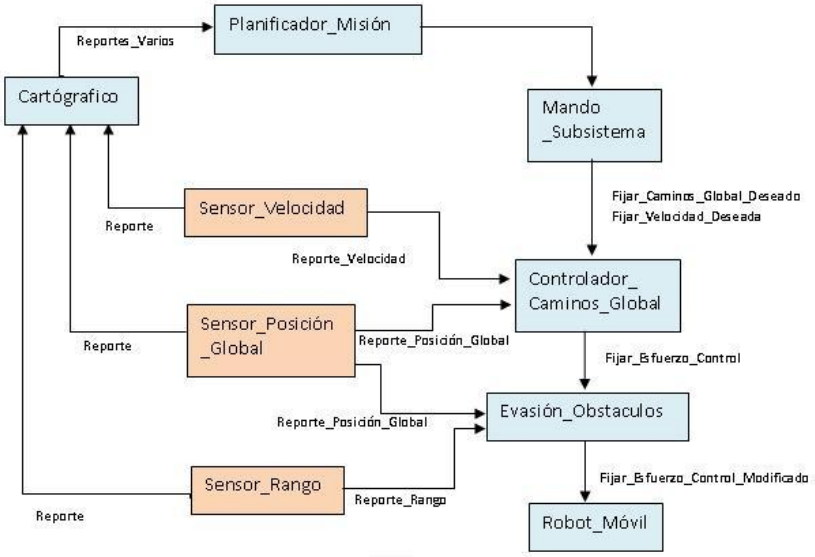


Figura 4.8 Control en lazo cerrado de un robot móvil de acuerdo a un camino deseado con evasión de obstáculos y planificación de la misión

En el escenario E de la figura 4.7 el agente Mando_Subsistema simplemente fija una misión a priori, consistente en un camino, dado de antemano por un operador humano. Sin embargo en la robótica de servicios el objetivo es que el usuario fije una misión de alto nivel al robot, en forma de una orden similar a la que ejecutaría un ser humano, por ejemplo limpie la cocina o vigile los alrededores de la casa. Por esta razón se requiere un agente Planificador_Misión que dada la misión a ejecutar en forma de una orden de alto nivel, diseñe un plan de actividades consistente en un árbol de mensajes que determina el camino a seguir por el robot y que debe coordinar en su ejecución el agente Mando_Subsistema, esto se puede ver en la figura 4.8. Sin embargo el planificador de misión debe apoyar su diseño en un modelo del entorno, construido a partir de un mapa a priori complementado dinámicamente con las lecturas de los agentes Sensor_Rango, Sensor_Posición_Global y Sensor_Velocidad, ver figura 4.8. El suministro de esta información es responsabilidad del agente Cartográfico. Un Sistema de control así diseñado, con capacidad de evasión de obstáculos, cumple con el enfoque del paradigma híbrido.

Escenario G: control de camino con planificación de misión

Requerimiento: Control del movimiento en lazo cerrado de un robot móvil fijando un camino global deseado, sin evasión de obstáculos, pero el camino obedeciendo a una planificación de una misión de alto nivel según un modelo del entorno.
Agentes: Mando_Subsistema, Controlador_Caminos_Global, Robot_Móvil, Sensor_Velocidad, Sensor_Posición_Global, Sensor_Rango, Cartografico y Planificador_Misión.
Diagrama de Control: ver figura 4.9

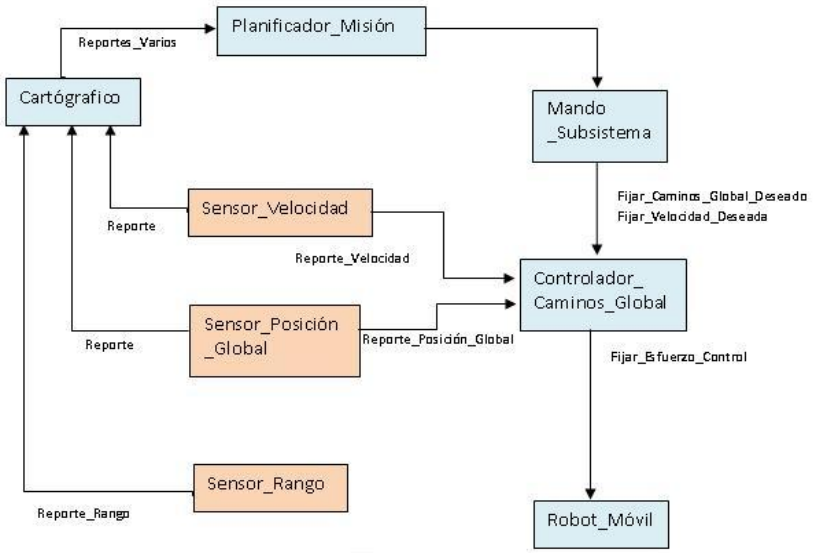


Figura 4.9 Control en lazo cerrado de un robot móvil de acuerdo a un camino deseado, sin evasión de obstáculos, pero incluyendo re planificación de la misión

En el escenario G de la figura 4.9 se mantiene la misma configuración del escenario F, con la excepción de que el agente Evasión_Obstaculos ha sido retirado. Esta sencilla modificación en el diagrama de control del escenario F hace que se pase de un enfoque de un paradigma híbrido a un paradigma deliberativo. Lo que obliga a que dada la situación de la presencia de un obstáculo en el camino recorrido por el robot móvil, el estado del robot móvil pasa a parada de emergencia mientras el planificador de la misión evalúa la situación de acuerdo a la información registrada en el modelo del entorno del agente cartográfico, esto puede ser un proceso computacionalmente

costoso, se realiza una re planificación de la misión, el robot móvil pasa a un estado de salida de emergencia y posteriormente pasa a un estado de retomar su actividad de acuerdo a un nuevo camino planificado.

Escenario H: control reactivo de un robot móvil

Requerimiento: Permitir que el robot móvil deambule por el entorno evadiendo obstáculos.
Agentes: Deambular, Evasión_Obstáculos, Robot_Móvil, Sensor_Velocidad, Sensor_Posición_Global y Sensor_Rango.
Diagrama de Control: ver figura 4.10

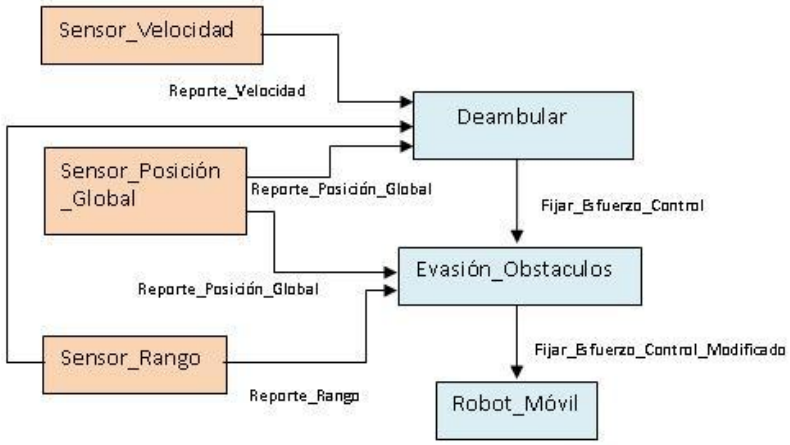


Figura 4.10 Robot Móvil deambulando por el entorno evadiendo obstáculos.

En el escenario H de la figura 4.10 se hace un quiebre fundamental con los diseños previos ya que el diagrama de control del robot móvil se diseña principalmente con la interelación de los agentes reactivos Deambular y Evasión_Obstaculos apoyados con la información suministrada por los agentes extero sensores. El agente Deambular, de acuerdo a los reportes de los agentes sensores, determina un recorrido arbitrario para el cual suministra el esfuerzo de control requerido al robot móvil. Esfuerzo de control que puede ser subsumido por el agente Evasión_Obstaculos con el fin de mantener la integridad del vehículo. Este diagrama de control así diseñado obedece a un enfoque del paradigma reactivo.

4.3 Resumen

En el presente capítulo se ha realizado la propuesta y diseño de una arquitectura de control genérica para robótica de servicios basada en el paradígma de los sistemas multiagentes, denominada AGC -MAS. Los sustentos teóricos en los que se apoya la propuesta de AGC-MAS son:

- Sistemas Multiagentes: enfoque tomado de la Inteligencia Artificial Distribuida, en la cual se diseña un sistema cognitivo a partir de la interrelación de módulos inteligentes que están en capacidad de deliberar sobre el desempeño de la función o servicio prestado.
- Las arquitecturas de control basadas en datos con estampa de tiempo: esta importante relación entre el valor de un dato y su relación con su antigüedad, ha sido identificada en el mundo de las redes de telecomunicación, sin embargo no había sido aplicado a los sistemas robóticos hasta el trabajo de Juan Luis Posadas de la Universidad Politécnica de valencia del año 2003. Por medio de esta relación se puede tomar decisiones acerca del sitio más adecuado para la ejecución del agente.
- Lenguaje de comunicación basado en mensajes para comunicar tanto interna como externamente un robot móvil. Esta es un fundamento tomado de JAUS que tiene como propósito mejorar la interoperabilidad entre UGV de diferentes fabricantes y facilitar la modularidad y reusabilidad del código.

AGC-MAS se puede ver como una capa de abstracción superior ubicada sobre un *middleware* que implementa la infraestructura de comunicación distribuida de un sistema *software* pero que es independiente del mismo. Esta capa de abstracción permite el diseño de sistemas robóticos, sean estos teleoperados o autónomos deliberativos, híbridos o reactivos, empleando la inserción y reuso de agentes. Agentes que no sólo realizan deliberación sobre la funcionalidad realizada sino que además deben deliberar sobre las condiciones del sistema de comunicaciones y de la máquina de cómputo donde se encuentran en ejecución.

En AGC –MAS se define un sistema robótico como una topología de subsistemas y estos a su vez como redes de nodos de cómputo, en las cuales un grupo de agentes, apoyándose en un conjunto de mensajes, realizan las diversas funcionalidades del

sistema robótico.

Los agentes de AGC-MAS se han organizado de acuerdo a la identificación de los componentes principales requeridos en el diseño genérico de un sistema de control de movimientos, sea este un robot tele controlado o un robot autónomo deliberativo, híbrido o reactivo. Estos agentes son: a) Extero receptores, b) Reactivos, c) Control de movimiento, d) Plataforma móvil, f) Deliberativos y g) Mando. Estos agentes requieren un lenguaje de comunicación para realizar su trabajo, por esta razón en AGC-MAS se propone un lenguaje basado en grupos funcionales de mensajes que los diferentes agentes se intercambian en la realización de sus tareas.

Los mensajes en AGC-MAS proporcionan las palabras que todos los agentes de la arquitectura genérica emplean para atender o solicitar servicios de otros agentes o bien solicitar que se ejecute una acción. Por esta razón los mensajes se encuentran organizados en tres clases: Reportes y Solicitudes para el primer caso y Comandos para el segundo caso. Los del primer caso se emplean para compartir información entre agentes y los del segundo caso se emplean para gobernar los cambios de estado en los cuales puede estar el robot móvil. Todos los mensajes se encuentran organizados en grupos funcionales: básicos, extero sensores, plataforma móvil y misión.

Finalmente a los diseños de arquitecturas de control que empleen AGC-MAS se les pueden dar una representación gráfica empleando una sistema de cajas y flechas. Donde cada caja tiene su significado asociado a un agente y las flechas a los mensajes de entrada y salida entre los agentes. De esta forma diseñar una arquitectura de control sea esta tele operada o autónoma se puede realizar rápidamente si se conocen los agentes involucrados y los mensajes de comunicación fundamentales.

5. Conclusiones y Recomendaciones

El desarrollo de esta tesis consideró como tema de trabajo una arquitectura genérica para robótica móvil, las conclusiones se realizan de acuerdo a los temas abordados en cada capítulo.

En el capítulo 1 se puede concluir que la complejidad del trabajo en robótica móvil es altamente creciente y lo será aún más cuando se integre un concepto global de plataforma móvil con brazos manipuladores, y plataforma móvil no solo con ruedas sino con piernas, incluyendo aérea y acuática. Igualmente se concluye la necesidad de que la comunidad académica se organice para realizar propuestas de estandarización al respecto y que a pesar de no existir aún estándares masivamente aceptados en robótica, existen recomendaciones militares al igual que completas herramientas de trabajo software robótico, que bien seleccionadas pueden ahorrar mucho trabajo a los nacientes grupos de investigación en Robótica e Inteligencia Artificial.

En el capítulo 2 se analiza el problema de la creciente complejidad del trabajo en robótica móvil y concluye en la necesidad de definir una arquitectura de control para manejarla. Se establece que las arquitecturas de control híbridas ofrecen la solución de mejor equilibrio entre reactividad y deliberación. No todas las arquitecturas híbridas proponen una capa de secuenciamiento, simplemente un nivel reactivo en primer plano y un nivel deliberativo en segundo plano. La información sensorial es utilizada tanto a nivel de Planificación, para la construcción de un modelo interno del entorno del robot que se utiliza para la descomposición de tareas y la selección de los comportamientos que implementarán estas tareas, como a nivel de la ejecución de dichos comportamientos, donde el valor procesado o no de cada sensor es utilizado por cada comportamiento que lo necesite. El enfoque de dejar al nivel reactivo el control del robot, mientras el nivel deliberativo, en segundo plano, modelaba y planeaba permitió evitar los problemas del paradigma deliberativo. En esta comunicación entre los niveles reactivo – deliberativo hay también diferencias, mientras algunas arquitecturas proponen que el nivel reactivo encueste asincrónicamente al nivel deliberativo el qué hacer, en otros el nivel deliberativo sincrónicamente le comunica al nivel reactivo el qué hacer. A pesar de las diferencias el paradígma híbrido es el enfoque adecuado para permitir una rápida respuesta de los robots en su entorno y modelar un nivel cognitivo

superior útil.

Finalmente en este capítulo 2 se ha propuesto un nuevo enfoque para clasificar las nuevas arquitecturas de control empleadas en robótica, se le ha denominado paradígma multi agente. El paradígma multi agente es un paradígma híbrido donde se hace una distribución del nivel cognitivo en múltiples nodos denominados agentes, idea tomada principalmente de los sistemas multi agentes (MAS) entre otras de la IA. Simplemente es otra forma de organizar la estructura de procesamiento y toma de decisiones de alto nivel del robot. En lugar de considerar que se tienen dos niveles centralizados responsables del control del robot: uno reactivo y otro deliberativo, estas dos responsabilidades se reparten en múltiples agentes, donde se podría proponer desde algo tan elemental, como un agente reactivo y un agente deliberativo, uno por cada nivel, hasta un modelo más complejo con múltiples agentes, cada uno responsable de funciones reactivas, deliberativas o de supervisión del sistema. Cada agente con inteligencia, acceso a recursos y capacidad de gestión lo que permitiría ver la funcionalidad total de la arquitectura como el fruto de las interrelaciones de los diferentes agentes.

Capitulo 3. En el capítulo tres se hace una selección de once plataformas de trabajo software robóticas con el fin de realizar un análisis cualitativo, sin embargo se identifica la dificultad del trabajo a realizar debido a la carencia de una terminología técnica adecuada unificada entre todos los proyectos, lo que obliga a realizar definiciones de términos y a concluir que todas las plataformas no son lo mismo: algunas son *middleware*, otras son librerías robóticas, otras son proyectos de integración y muy pocas son completas plataformas de trabajo robótico. Esto solo evidencia la urgente necesidad de estandarizar estos diferentes tipos de software que presentan estos proyectos con el fin de aumentar la reusabilidad de componentes, sistemas, y herramientas robóticas entre la comunidad de investigadores.

Finalmente en el capítulo 4 se propone la concepción y diseño de una arquitectura genérica de control basada en Sistemas Multi Agente, denominada AGC-MAS (Arquitectura Genérica de Control Multiagente). La cual se puede emplear en la especificación de cualquier tipo de arquitectura de control de un robot móvil sea este tan elemental como teleoperado o tan complejo como una arquitectura híbrida. Para ello AGC-MAS se soporta en tres pilares:

a) La concepción topológica de un sistema robotico como la unión de subsistemas,

un subsistema como una red de nodos de cómputo y en cada nodo un conjunto de agentes que realizan una funcionalidad específica.

b) La propuesta y división de los agentes necesarios para implementar un sistema genérico de control de movimientos para un robot móvil. Estos agentes son: a) Extero receptores, b) Reactivos, c) Control de movimiento, d) Plataforma móvil, f) Deliberativos y g) Mando.
c) Un lenguaje de comunicación basado en mensajes organizados en tres clases: Reportes, Solicitudes y Comandos. Los dos primeros se emplean para compartir información entre agentes y el tercero se emplea para gobernar los cambios de estado en los cuales puede estar el robot móvil. Todos los mensajes se encuentran organizados en grupos funcionales: básicos, extero sensores, plataforma móvil y misión.

Un valor agregado de AGC-MAS es el brindarle una semántica a los clásicos diagramas de cajas y flechas empleados en la representación gráfica de arquitecturas de control en robots moviles. Ya que gracias a una definición clara de la funcionalidad tanto del agente AGC-MAS como de los mensajes de entrada y salida, se evita la ambigüedad o vacios en la interpretación de la función del bloque y el significado de las flechas del mismo.

Con respecto a las recomendaciones y trabajos futuros hay importantes puntos a tener en cuenta como a) arquitecturas de apoyo complementarias a AGC-MAS, b) trabajos futuros a realizar en AGC-MAS y c) la implementación de AGC –MAS en plataformas de trabajo *software* robotico existentes.

El trabajo con robótica móvil es tan complejo que no se puede simplemente hacer uso de una arquitectura de control para manejar todas las dimensiones del sistema. Se debe hacer uso de un marco arquitectural de varias partes:

a) Un completo Glosario /Ontologia que cubra todos los aspectos del trabajo con manipuladores y robots de servicios.
b) Una arquitectura del sistema software que permita modelar y analizar todos los aspectos relacionados con las limitaciones temporales, sistemas operativos y herramientas software delsistema robotico.
c) Una arquitectura del sistema hardware que permita modelar ydocumenar todos los aspectos de los nodos de cómputo y las redes de comunicación empleadas en el sistema robotico.

En el anterior marco arquitectural de tres partes AGC-MAS se puede emplear como la parte cuatro referente a la especificación de una arquitectura de control. AGC-MAS requiere de otros estándares para cubrir los diversos aspectos de un sistema robotico como lo hace el marco arquitectural C4ISR del DoD de los Estados Unidos. Adicionalmente este marco arquitectural de cuatro partes debe ser complementado con una especificación que permita modelar el dominio del sistema robótico, para ello algunos centros de investigación emplean UML, pero debido a la complejidad de un sistema robótico en el presente trabajo de maestria se recomienda emplear SySML.

La propuesta y diseño de AGC-MAS realizado en el presente trabajo de maestria debe entenderse como un primer acercamiento donde se han definido las bases de una arquitectura de control genérico multiagente para sistemas robóticos. Sin embargo en esta definición hay áreas en las cuales se debe específicar aún más la arquitectura AGC-MAS. Estas áreas deberán ser fruto de trabajos futuros:

a) Diseño de los diagramas de cambio de estado o de secuencia para permitir modelar el comportamiento de los agentes en el intercambio de los mensajes básicos: control de estado, control de acceso, configuración dinámica, configuración de comunicación y mensaje de funcionamiento.
b) Diseño de un diagrama que permita modelar la movilidad de los agentes entre nodos de computo según la evaluación realizada sobre los recursos del sistema hardware, esta diseño debe involucrar el análisis del campo estampa temporal de los mensajes.
c) Diseñar en forma detallada todos los campos tanto de la cabecera como del cuerpo de un mensaje AGC-MAS. Para cada mensaje definir el contenido de los diferentes campos del cuerpo del mensaje.

Finalmente con respecto a las plataformas software existentes que pueden ser libremente empleadas a la fecha la más popular y con mejor soporte técnico es Player, sin embargo herramientas como MARIE y ADE deben ser igualmente tenidas en cuenta al igual que RT-*Middleware*. Sin embargo no se puede hacer uso de ninguna de estas herramientas *software* sin tener el apoyo de una especificación como AGC-MAS que brinda el soporte semántico para el diseño de una arquitectura de control. En la actualidad las plataformas de trabajo *software* robóticos que por su concepción facilitarían este trabajo son CAST&BALTS, ADE y OpenRT-aist, por lo que un potencial trabajo futuro estaría encaminado a integrar AGC-MAS con una de estas herramientas.

6. Bibliografía

[1] Ollero, A. (2001). Robotica Manipuladores y robots móviles. Barcelona, España, Marcombo.

[2] Karlsson, J. (2000). Un world robotics statistics 1999. *Industrial Robotic*, 27 (1), 14-18.

[3] Garcia, E., Jimenez, M., De Santos, P. & Armada, M. (2007). The evolution of robotics Research , from industrial robotics to field and service robotics. *IEEE Robotics & Automation Magazine*, 14 (1), 90-103.

[4] Robot terrestre de la NASA, imagen tomada de la página web: - http://www.vnunet.com /vnunet/news/2194728/nasa-sends-robots-north-pole, consultada en abril de 2009.

[5] Robot móvil con orugas de la empresa LYNXMOTION, imagen tomada de la página web: - http://www.lynxmotion.com/Category.aspx?CategoryID=120, consultada en abril de 2009.

[6] Robot humanoide ASIMO de la compañía HONDA, imagen tomada de la página web: - http://world.honda.com/ASIMO/history/asimo.html , consultada en abril de 2009.

[7] Robot articulado para investigación, imagen tomada de la página web: - http://www.iearobotics.com/personal/juan/doctorado/Modulos-Y1/modulos-y1.html, consultada en abril de 2009.

[8] Robot acuático de la empresa IROBOT, imagen tomada de la página web: - http://www.irobot.com/sp.cfm?pageid=428, consultada en abril de 2009.

[9] Robot PENGUIN de la empresa FESTO, imagen tomada de: - http://www.prensalibre.com/pl/ domingo/archivo/revistad/2009/abril/26/?cont=todo , consultada en abril de 2009.

[10] Robot aéreo UAV, de la USA ARMY, imagen tomada de la página web: - http://airvoila.com/vehiculo-aereo-no-tripulado-uav/, consultada en abril de 2009.

[11] Robot aéreo UAV de elevación vertical, imagen tomada de la página web: - http://www.roboserv.net/apid55, consultada en abril de 2009.

[12] Bekey, G. & Yuh J. (2007). The Status of Robotics: report on the WTEC international study: part I. *IEEE Robotics & Automation Magazine*, 14 (4), 76-81.

[13] Plataforma locomoción ackerman, imagen tomada de: - http://www.egr.uh.edu/courses/ece /ece4437/support/notes_hardware/Robot_Design_of_Base.pdf, consultada en abril de 2009.

[14] Configuración de locomoción tipo triciclo, imagen tomada de la página web: - http://muchotrasto.com/TiposDePlataformas.php, consultada en abril de 2009.

[15] Robot educativo diferencial, imagen tomada de la página web de Superrobotica: - http://www.superrobotica.com/Catalogo.htm, consultada en abril de 2009.

[16] Robot con locomoción skid steer por ruedas, imagen tomada de la página web:- http://tec.nologia.com/2008/01/23/, consultada en abril de 2009.

[17] Robot con locomoción tipo skid steer por orugas, imagen tomada de la página web:- http://www.robhaz.com/about_rescue_features_eng.asp, consultada en abril de 2009.

[18] Plataforma base, producto lynxmotion, imagen tomada de la página web de Lynxmotion - http://www.lynxmotion.com/, consultada en abril de 2009.

[19] Robot de exploración con objetivos espaciales, imagen tomada de la página web - http://www.gmv.es/b2_gmv/index.php?blog=5&m=2008, consultada en abril de 2009.

[20] Plataforma base educativa, imagen tomada de la página web : - http://todoelectronica.com /robot-programable-carbot-p-6406.html, consultada en abril de 2009.

[21] Plataforma con pinza de la empresa lynxmotion, imagen tomada de la página web: - http://www.lynxmotion.com/Category.aspx?CategoryID=110, consultada en abril de 2009.

[22] Plataforma con pinza y manipulador de lynxmotion, imagen tomada de la página web: - http://www.lynxmotion.com/Category.aspx?CategoryID=110, consultada en abril de 2009.

[23] Plataforma militar con manipulador de Irobot, imagen tomada de la página web: - http://www.irobot.com/sp.cfm?pageid=171, consultada en abril de 2009.

[24] Everett, H. (1995). Sensors for Mobile Robots: Theory and Application. A. K. Peters, Ltd., Natick, MA.

[25] Magnenat, S., Longchamp, V. & Mondada, F. (2007). Aseba, an event-based middleware for distributed robot control. In International Conference on Intelligent Robots and Systems (IROS).

[26] Alvira, C., Segura, J., Velasco, O. & Flórez, J. (2007). Atibot: UGV SKID STEERING de arquitectura distribuida. *III IEEE Colombian Workshop on Robotics and Automation*, Cartagena.

[27] Russell, S. & Norvig, P. (1996). *Inteligencia Artificial un enfoque moderno*. Prentice Hall Hispanoamerica.

[28] Nilsson, N. (1984). Shakey the robot. Technical Report 323. Artificial Intlligence Center, SRI international.

[29] Barbera, A., Albus J., Firzgerald, M. & Haynes, L. (1984). RCS: The NBS Real-Time Control System. Proceedings of the Robots 8 Conference and Exposition, 2.

[30] Albus, J., McCain, H. & Lumia, R. (1986). NASA/NBS Standard Reference Model for Telerobot control System Architecture (NASREM). NASA Document SS-GSFC-0027.

[31] Gat, E. (1998). On Three-Layer Architectures. Artificial Intelligence and Mobile Robots. chapter 8, 195 – 210, MIT Press, Cambridge, MA.

[32] Firby, J. (1987). An investigation into reactive planning in complex domains. Proceedings Of the Sixth National Conference on Artificial Intelligence, 1, 202-206.

[33] Agre, P. & Chapman, D. (1987). Pengi: An implementation of a theory of activity. Proceedings Of the Sixth National Conference on Artificial Intelligence, 1, 268-272.

[34] Brooks, R. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2 (1), 14–23.

[35] Connell, J. (1989). A Colony Architecture for an Artificial Creature. Technical Report 1151. Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

[36] Kaelbling, L. (1987). REX: A symbolic Languaje for the design and parallel implementation of embedded systems, Proceedings Of the AIAA Conference on Computers in Aerospace.

[37] Soldo, M. (1990). A reactive a preplaned control in a mobile robot. Proceedings Of the IEEE International Conference On Robotics and Automation, 2, 1128-1132.

[38] Arkin, R. (1990). Integrating Behavioral Perceptual and World Knowledge in reactive navigation, Robotics and Autonomous Systems, 6(1-2), 105-122.

[39] Georgeff, M. & Lanskey A. (1987). Reactive Reasoning and planning. Proceedings Of the Sixth National Conference on Artificial Intelligence, 1, 677-682.

[40] Simmonds, R. (1990). An architecture for coordinating Planning, Sensing and Action. *Proc. Of the DARPA Workshop on Innovative Approaches to planning, Scheduling and Control*, 292-297.

[41] Rosenblatt, J. & Payton, D. (1989). A Fine grained alternative to the Subsumption Architecture. *International Joint Conference on Neural Networks*, 2, 317-323.

[42] Arkin, R. (1989). Motor Schema based mobile robot navigation. *International Journal of Robotic Research*, 8 (4), 92-112.

[43] Arkin, R. (1989). Dynamic replanning for a mobile robot based on internal sensing. *In proceedings of the IEEE Int. Conf. Robotics and Automation*, 3, 1416-1421.

[44] Gat, E. (1992). Integrating planning and reacting in a heterogeneous asynchronous architecture for controling real world mobile robots. *In Proceedings of the Tenth National Conference on Artificial Intelligence*, 809-815.

[45] Connell, J. (1992). SSS: A hybrid architecture applied to robot navigation. *Proceedings of the IEEE International Conference on Robotics and Automation*, 3, 2719-2724.

[46] Musliner, D., Durfee, H. & Shin, G. (1993). CIRCA: A cooperative Intelligent Real Time Control Architecture. *IEEE Transactions on Systems, Man And Cybernetics*, 23(6), 1561-1574.

[47] Simmonds, R. (1994). Structured Control for Autonomous Robots. *IEEE Transactions on*

Robotics and Automation, 10(1), 34-43.

[48] Malcom, C. (1995). The SOMASS System: a hybrid symbolic and behavior based systems to plan and execute assemblies by robot. Proceedings of AISB Conference, 157-168.

[49] Knonolige, K., Myers K., Ruspini, E. & Saffioti, A. (1997). The Saphira Architecture: A design for Autonomy. Journal of Experimental and Theoretical Artificial Intelligence, 9, 215-235.

[50] Arkin, R. & Balch, T. (1997). AuRA: Principles and Practice in Review. Journal of Experimental and Theoretical Artificial Intelligence, 9, 175-189.

[51] Bonnaso, P. & Kortenkamp, D. (1997). Experiences with an architecture for intelligent, reactive agents. Journal of Experimental and Theoretical Artificial Intelligence, 9, 237-256.

[52] Kramer, J. & Scheutz, M. (2007). Development Environments for Autonomous Mobile Robots: A Survey. Autonomous Robots, 22(2), 101-132.

[53] Pembeci, I. & Hager, G. (2002). A comparative review of robot programming languages. Technical Report. Computational Interaction and Robotics Lab, Johns Hopkins University.

[54] Biggs, G. & MacDonald, B. (2003). A Survey of Robot Programming Systems. In Proceedings of Australasian Conference on Robotics and Automation, 1.

[55] Firby, J. (1989). Adaptive execution in dynamic domains. Technical Report YALEU/CSD/RR 672. Department of Computer Science, Yale University.

[56] Gat, E. (1991). ALFA: A lenguage for programming reactive robotic control systems. Proceedings of the IEEE Comference on Robotics and Automation, 2, 1116-1121.

[57] Konolige, K. (1997). Colbert: A language for reactive control in Saphira. In KI Advances in Artificial Intelligence, 31-52.

[58] LeGuernic, P., Gauthier, T., LeBurgne, M. & LeMarie, C. (1991). Programming real time applications with SIGNAL. In Proceedings of the IEEE, 79(9), 1321-1336.

[59] Alur, R., Grosu R., Hur, Y., Kumar, V. & Lee, L. (2000). Modular Specification of hybrid systems in CHARON. In Proceedings of the 3rd International Workshop on Hybrid Systems: computation and control, 6 - 19.

[60] Peterson J. & Hager, G. (1999). Monadic robotics. Proceedings of the 2nd Conference on Domain specific languages, 2, 95 – 108.

[61] Hudak, P. (200). Arrows, robots, and functional reactive programming. In Summer School on Advanced Functional Programming 2002, Oxford University, Lecture Notes in Computer Science. Springer-Verlag.

[62] Gat, E. (1994). ESL: A language for supporting robust plan execution in embedded autonomous agents. Proceedings of the IEEE Aerospace Conference, 1, 319-324.

[63] Brooks, R. (1989). The Behavior language User´s Guide. Massachusetts Institute of

Technology, Cambridge, MA.

[64] Ingrand, F., Chatila, R., Alami, R. & Robert, F. (1996). PRS: A high level supervision and control language for autonomous mobile robots. In Proceedings of the IEEE International Conference on Robotics and Automation, 1, 43-49.

[65] Simmonds, R. & Apfelbaum, D. (1998). A task description language for robot control. In Proceedings of the International Conference on intelligent robotics and systems, 3, 1931-1937.

[66] Coste-Maniqe, E. & Turro, N. (1997). The MAESTRO Language and its Environment: Specification, Validation and Control of Robotic Missions. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2, 836 – 841.

[67] Fleury, S., Herrb, M. & Chatila, R. (1997). GenoM: A tool for the specification and implementation of operating modules in distributed robot architecture. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1 , 842–848.

[68] Simon, D. Espiau, B., Kapellos, K. & Pissard-Gibollet, R. (1997). ORCCAD: Software Engineering for Real Time Robotics. A Technical Insight, Robotica, Special issues on Languajes and Software in Robotics, 15 (1), 111-116.

[69] Página web de LAAS open software for autonomous systems. https://softs.laas.fr /openrobots/wiki/, consultada en abril de 2009.

[70] Página web del proyecto MissionLab, http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/, consultada en abril de 2009.

[71] Lindstrom, M., Oreback, A. & Christensen, H. (2000). BERRA: A research architecture for service robots. *In Proceedings of international conference on robotics and automation*, 4, 3278–3283.

[72] Finkemeyer, B., Kroeger, T., Kubus, D., & Olschewski, M. (2007). MiRPA: Middleware for Robotic and Process Control Applications. In Proccedings of the IEEE/RSJ International comference on intelligent robots and systems.

[73] Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., & Das, H. (2001). The CLARAty architecture for robotic autonomy. In Proceedings of the IEEE aerospace conference, 1, 121-132.

[74] Schlegel, C. & Worz, R. (1999). The software framework SMARTSOFT for implementing sensorimotor systems. *In Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 3, 1610-1616.

[75] Página web del proyecto Orocos, www. orocos.org, consultada en abril de 2009.

[76] Gerkey, B., Vaughan, R., & Howard, A. (2003). The player/stage project: Tools for multirobot and distributed sensor systems. In Proceedings of the International Conference on Advanced Robotics, 1, 317–323.

[77] Collet, T. & MacDonald B. (2005). Player 2.0: toward a practical robot programing framework. In proceedings of the Australasian Conference on Robotics and Automation.

[78] Página web de PSG, http://playerstage.sourceforge.net/ consultada en abril de 2009.

[79] Página web de la plataforma Teambots, http://www.cs.cmu.edu/~trb/TeamBots/, consultada en abril de 2009.

[80] Blank, D., Kumar, D., Meeden L. & Yanco, H. (2004). Pyro: A Python-based Versatile Programming Environment for Teaching Robotics. Journal of Educational Resources in Computing , 3(4), 1-15.

[81] Côté, C., Brosseau, Y., L´etourneau, D., Raievsky, C. & Michaud, F. (2006). Robotic software integration using MARIE. *International Journal on Advanced Robotics Systems*, 3(1), 55–60.

[82] Página web de MARIE, http://marie.sourceforge.net/, consultada en abril de 2009.

[83] Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T. & Yoon, W. (2006). RTMiddleware: Distributed Component Middleware for RT (Robot Technology). The *IEEE/RSJ International Conference on Intelligent Robots and Systems ,* 1, 3933-3938.

[84] Página web de RT-Middleware, http://www.is.aist.go.jp/rt/OpenRTM-aist/html-en/index.html , consultada en abril de 2009.

[85] Página web del proyecto MCA2, www.mca2.sf.net, consultada en abril de 2009.

[86] Página web del proyecto OpenRobots, https://softs.laas.fr/openrobots/wiki/, consultada en abril de 2009.

[87] Dozio, L. & Mantegazza, P. (2003). Real-time distributed control systems using rtai. In Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 1, 11-18.

[88] Martínez, H. (2001). A distributed architecture for intelligent control in autonomous mobile robots an applied approach to the development of the quaky-ant platform, PhD Thesis,Dept. of Communications and Information Engineering, University of Murcia.

[89] Utz, H., Sablatnog, S., Enderle, S. & Kraetzschmar, G. (2002). Miro – middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, 18(4), 493–497.

[90] Posadas, J. L. (2003). Arquitectura para el control de robots móviles mediante delegación de código y agentes. PhD. Thesis. Universidad Politécnica de Valencia.

[91] Montemerlo, M., Roy, N. & Thrun, S. (2003). 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*, 3, 2436-2441.

[92] Chaimowicz, L., Cowley, A., Sabella, V. & Taylor, C. (2003). ROCI: a distributed framework for multi-robot perception and control. *In Proceedings of the IEEE/RSJ International Conference*

on Intelligent Robots and Systems, 1, 266- 271.

[93] Orebaeck, A. (2004). A Component Framework for Autonomous Mobile Robots. PhD thesis, Royal Institute of Technology, Numerical Analysis and Computer Science.

[94] Página web proyecto Orca robotics, http://orca-robotics.sourceforge.net/, consultada en abril de 2009.

[95] Andronache, V. & Scheutz, M. (2006). ADE - An Architecture Development Environment for Virtual and Robotic Agents. In International Journal on Artificial Intelligence Tools, 15(2), 251-286.

[96] Domınguez-Brito, A. C. (2003). CoolBOT: a Component-Oriented Programming Framework for Robotics. Tesis Doctoral, Universidad de Las Palmas de Gran Canaria.

[97] Página web de Robotflow, Open source robotics toolkit for flowdesigner, http://robotflow.sourceforge.net/, consultada en abril de 2009.

[98] Petters, S., Thomas, D. & Von-Stryk, O. (2007). RoboFrame - A Modular Software Framework for Lightweight Autonomous Robots. In Proccedings of the IEEE/RSJ International comference on intelligent robots and systems.

[99] Página web de WURDE robotics development environment , http://wurde.sourceforge.net/, consultada en abril de 2009.

[100] Página web de la empresa Gostai, www.gostai.com, consultada en abril de 2009.

[101] Página Web de la empresa MobileRobotics, www.mobilerobotics.com, consultada en abril de 2009.

[102] Página web de MRDS, msdn.microsoft.com/en-us/robotics/aa731520.aspx, consultada en abril de 2009.

[103] Página web Webots , http://www.cyberbotics.com/, consultada en abril de 2009.

[104] Página web de PC-BOT, http://whiteboxrobotics.com/, consultada en abril de 2009.

[105] Página web de ERSP 3.1 robotic development platform, http://www.evolution.com/ products/ersp/Evolution Robotics, consultada en abril de 2009.

[106] Página web de educational division – mindstorms for schools, http://www.lego.com/eng/ education/mindstorms /default.asp. LEGO, consultada en abril de 2009.

[107] Coste-Maniere, E. & Simmons, R. (2000). Architecture, the Backbone of Robotic Systems. In Proceedings of the IEEE International Conference on Robotics and Automation.

[108] Página web de RosTa, Robot Standards and Reference Architectures, http://www.robot-standards.org/ o wiki.robot-standars.org, consultada en abril de 2009.

[109] Página web de EURON European Robotics Research Network, www.euron.org, consultada en abril de 2009.

[110] Página web de RAS Standing committee on standard activities, www.ieee-ras.org/ industrial/standards, consultada en abril de 2009.

[111] Albus, J. (2002). 4D/RCS: A reference Model Architecture for Intelligent Unmanned Ground Vehicles. In Proceedings of the SPIE AeroSense Conference 4715 on Unmanned Ground Vehicle Technology IV.

[112] Página web de la Federación de Científicos Americanos (FAS), http://www.fas.org/irp/ doddir/dod/c4isr/es.htm, consultada en abril de 2009.

[113] Página web de JAUS, http://www.jauswg.org/index.shtml, consultada en abril de 2009.

[114] Lozano, T. (1983). Robot programming. In Proceedings of the IEEE Tutorial on Robotics, IEEEComputer Society, 71(7), 821-841.

[115] Pembeci, I. & Hager, G. (2002). A comparative review of robot programming languages. Technical report. *Computational Interaction and Robotics Lab*. Johns Hopkins University.

[116] Biggs, G. & MacDonald, B. (2003). A Survey of Robot Programming Systems. In Proceedings of Australasian Conference on Robotics and Automation.

[117] Oreback, A. & Christensen, H. (2003). Evaluation of Architectures for Mobile Robotics. *Autonomous Robots*, 14, 33 – 49.

[118] Project COSY (Cognitive Systems for Cognitive Assistants) (2005). Preliminary Report on Requirements for Architectures and Tools. Birmingham University, Report Intern DR.1.1.

[119] Wrede, S., Bauckhage, C., Sagerer, G., Ponweiser, W. & Vincze, M. (2004). Integration Frameworks for Large Scale Cognitive Vision Systems – An Evaluative Study. In 17th International Conference on Pattern Recognition, 1, 761-764.

[120] Langley, P. (2008). Cognitive architectures: Research issues and challenges. Technical Report, Computational Learning Lab, Center for the Study of Learning and Information, Standford University.

[121] Kramer, J. & Scheutz, M. (2007). Development Environments for Autonomous Mobile Robots: A Survey. *Autonomous Robots*, 22(2), 101-132.

[122] Shakhimardanov, A. & Prassler, E. (2007). Comparative evaluation of robotic software integration systems: a case study. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1, 3031-3037.

[123] Cañas, J., Matellán, V. & Montúfar, R. (2006). Programación de robots móviles. Revista Iberoamericana de Automática e Informática Industrial, 3(2), 99-110.

[124] Gowdy, J. (2000). A qualitative comparison of interprocess communications toolkits for robotics. Report Intern TR-00-1, The Robotics Institute, Carnegie Mellon University.

[125] Hill, C. & Smart, W. (2002). Middleware for robots?. In Intelligent Distributed and Embedded Systems: AAAI Spring Symposium, 1 , 1-5.

[126] Broten, G., Monckton, S., Giesbrecht, J. & Collier, J. (2006). Software Systems for Robotics – An applied research perpective. International Journal Advanced Robotic Systems, 3(1), 11-17.

[127] Mohamed, N., Al-Jaroodi, J. & Jawhar, I. (2008). Middleware for Robotics: A Survey. In Proceedings of the IEEE International Conference on Robotics, Automation, and Mechatronics, 1, 736-742.

[128] Molaletsa, N. (2008). Open middleware for robotics. In Proceeddings of the 15th International Conferencen on Mechatronics and Machine Vision in Practice, 1, 189-194.

[129] Baeuml, B. (2007). Towards the Evaluation of Software Concepts for Complex Mechatronic Systems. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

[130] Ng-Thow-Hing, V., List, T. & Thorisson, K. (2007). Design and Evaluation of Communication Middleware in a Distributed Humanoid Robot System. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

[131] Nayar, H. & Nesnas, I. (2007). Measures and Procedures: Lessons Learned from the CLARAty Development at NASA/JPL. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

[132] Nilsson, K. Olsson T. & Bruyninckx, H. (2007). Basic Robotics Standards(BRoS) - Motivations and examples. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

[133] Página web de TCU planet, www.informatik.uni-ulm.de/ki/Planet/TCU-olps/, consultada en abril de 2009.

[134] Página web de FIPA, www.fipa.org , consultada en abril de 2009.

[135] Página web Estándar CORBA para agentes de la OMG, www.omg.org , consultada en abril de 2009.

[136] Chatfield, J. (1998). New Architecture Directions. The Edge. The Mitre Advanced Technology Electronic Newsletter, 1(3).

[137] Rosenblatt, J. (1995). DAMN: A Distributed Architecture for Mobile Navigation. In *proceedings of the AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*. 1, 339-360.

[138] Scheutz, M. & Andronache, V. (2004). The APOC framework for the comparison and Evaluation of agent architectures. In proceedings of AAAI Workshop on Intelligent Agent Architecture, 1, 66-73.

[139] Página Web APOC , http://www.nd.edu/airolab, consultada en abril de 2009.

[140] Hawes, N., Wyatt J. & Sloman A. (2006). An Architecture Schema for Embodied Cognitive

Systems. Technical report CSR0612, School of Computer Science, University of Birmingham.

[141] Página web de TDL e IPC en el CMU, http://www.cs.cmu.edu/~ipc/, consultada en abril de 2009.

[142] Página web de ACE, http://www.cs.wustl.edu/ Schmidt/ACE.html, consultada en abril de 2009.

[143] Fedor, C. (1995). TCX. An interprocess communication system for building robotic architectures. Programmer Guide. Carnegie Mellon University.

[144] Página web de SmartSoft, http://www.rz.fh-ulm.de/ cschlege/orocos/, consultada en abril de 2009.

[145] Hawes, N., Zillich, M. & Wyatt, J. (2007). BALT & CAST Middleware for cognitive robotics. In *The 16th IEEE International Symposium on Robot and Human interactive Communication*, 1, 998 – 1003.

[146] Página web proyecto Cosy, www.cognitivesystems.org, consultada en abril de 2009.

[147] Página web de MIRO, http://www.ics.uci.edu/ sequti/miro/miro index.html, consultada en abril de 2009.

[148] Página web de RobotFlow y FlowDesigner, http://robotflow.sourceforge.net/ y http:// flowdesigner.sourceforge.net/, consultada en abril de 2009.

[149] Página web de ICE, http://www.zeroc.com/, consultada en abril de 2009.

[150] Broten, G., Mackay, D., Monckton, S. & Collier, J. (2009). The Robotics Experience: beyond components and middleware. IEEE Robotics & Automation Magazine, 16 (1), 46-54.

[151] Broten, G., Mackay, D. & Desgagnes, R. (2008). Middleware for robotics: aplications on real time systems. In proceedings 3rd International Workshop Software Development and Integration in Robotics, 1, 6-12.

[152] Bensalen, S., Gallien, M. & Ingrand, F. (2009). Designing autonomous robots toward a more dependable software architecture. IEEE Robotics & Automation Magazine, 16 (1), 67-77.

[153] Albus, J., Lumia, R., Fiala, J. & Wavering, A. (1989). NASREM: The NASA/NBS Standard Reference Model for Telerobot Control System Architecture. In Proceedings of 20th International Symposium on Industrial Robots.

[154] Albus, J. (2002). 4D/RCS A Reference Model Architecture for Intelligent Unmanned Ground Vehicles. In Proceedings of the SPIE 16 th Annual international symposium on aerospace/Defense Sensing, simulation and controls.

[155] Gowdy, J. (2000). Emergent Architectures: A Case Study for Outdoor Mobile Robots. technical report CMU-RI-TR-00-27, Robotics Institute, Carnegie Mellon University.

[156] Prassler, E. & Nilson, K. (2009). 1,001 robot architectures for 1,001 robots. Column

Industrial Activities, IEEE Robotics & Automation Magazine, 16(1), 113-113.

[157] Huang, H., Albus, J. Kotora, J. & Liu , R. (2003). Robotic Architecture Standars Framework in the Defense Domain with Illustratios Using the NIST 4D/RCS Reference Architecture. In Proceedings of IEEE/RJS International Conference on Intelligent Robots and Systems, 3, 2415-2420.

[158] Object Management Group (2005). Request for Proposal. Robot Tecnnology Components. Documento OMG ptc, disponible en la página de la OMG, www.omg.org.

Anexo

1. Arquitecturas Clásicas de Control de Robots Móviles

1. Arquitecturas Clásicas de Control de Robots Moviles

1.1 Arquitecturas Deliberativas

Arquitectura de Control Robot Shakey

Desde 1966 hasta 1972 el Centro de Inteligencia Artificial del SRI *International* condujo una investigación en un sistema robótico móvil llamado Shakey. Esta investigación fue financiada por DARPA bajo una serie de contractos con *Rome Air Development center*, la NASA y la oficina de investigación de la Armada, todos en Estados Unidos. Dos versiones completas del robot Shakey se desarrollaron. En 1969 se completó un primer sistema integrado por un vehículo móvil equipado con una cámara de TV y otros sensores, todo radiocontrolado por un computador SDS-940. En 1971 se completó un sistema robot más potente haciendo sustanciales mejoras al programa software y reemplazando el SDS-940 por una pareja de computadores DEC PDP-10 y PDP-15 [1] .

El propósito de la investigación fue diseñar el primer robot móvil con capacidad de razonar acerca de sus acciones, desarrollando conceptos y técnicas en inteligencia artificial que habilitaran al autómata a operar en forma independiente en ambientes reales. Las metas primarias fueron la solución de problemas especificados en forma incompleta, que requerían la creación de estrategias y metas intermedias, y el mejoramiento del desempeño con experiencia de entrenamiento, un proyecto bastante ambicioso para los avances tecnológicos de la época.

Shakey tenía una cámara de televisión, un sistema de triangulación *Range Finder*, sensores de contacto, y estaba conectado a dos computadoras DEC PDP-10 y PDP-15 a través de enlaces de radio y vídeo. Shakey usaba programas para realizar la percepción, el modelamiento del mundo y generar órdenes de actuación. Rutinas de acción de bajo nivel eran las responsables de movimientos simples, giros y la ejecución de la ruta. Acciones de nivel intermedio encadenaban las de bajo nivel de tal forma que permitían llevar a cabo tareas más complejas. Los programas del nivel superior podían formular y

ejecutar planes con el fin de lograr objetivos suministrados por un usuario. Asimismo, el sistema generalizaba estos planes y los guardaba para su posible uso futuro.

El corazón de la arquitectura de control de Shakey es el "modelo" del mundo que habita. Este modelo es una estructura de datos global que puede ser accesada y modificada por rutinas de los niveles superiores. El modelo del mundo es único para todas sus operaciones, es un modelo axiomático. En este modelo se han establecido convenciones para representar puertas, paredes, habitaciones, objetos y el estado del robot. Este modelo axiomático es una colección de proposiciones del cálculo de predicados almacenadas como clausulas en una estructura de datos indexada. El formato de almacenamiento permite el uso del modelo sin modificaciones como si fuese un conjunto de axiomas para las operaciones de planificación que realiza la herramienta STRIPS y para las tareas de comprobación de teoremas.

La arquitectura de control del robot Shakey es una estricta estructura jerárquica tipo *top – down* en la cual se identifican cinco niveles principales, ver figura 1.1. Aunque algunos de estos niveles fueron mucho más definidos que otros y algunos tienen una subestructura considerable, la efectividad de la arquitectura está fuertemente apoyada a partir de unas claras especificaciones de estos niveles y sus interconexiones.

Nivel Ejecutivo (PLANEX)
Nivel de Planificación
Nivel de ILAs(*Intermediate Level Actions*)
Nivel de LLAs(*Low level Actions*)
Nivel Físico

Figura 1.1 Arquitectura de control de cinco niveles de Shakey

El nivel inferior (físico): consiste del vehículo robótico y su conexión a los programas de usuario. Esta conexión incluye enlaces de comunicación vía radio y microondas, un computador PD-15 con su software y un canal de comunicaciones con su software de comunicación entre la PDP-15 y la PDP-10. Este nivel inferior puede ser visto como la definición de las capacidades físicas elementales del sistema.

El nivel de las LLAs : el segundo nivel consiste de lo que se denominó Acciones de Bajo Nivel o LLAs (*Low level Actions*). Estos son los programas de control del robot del más bajo nivel disponibles a los programas de usuario en el lenguaje LISP, la principal herramienta de programación empleada. Las LLAs son funciones programadas de las capacidades físicas del robot tales como "ROLL" y "TILL".

El nivel de las ILAs: con la finalidad de que el robot pueda exhibir un comportamiento interesante, éste fue equipado con una librería de Acciones de Nivel Intermedio o ILAs (*Intermediate Level Actions*). Estos elementos del tercer nivel son paquetes preprogramados de LLAs, incrustadas en un marco de trabajo tipo tabla de Markov con varias características de percepción, control y corrección de errores. Cada ILA representa una experiencia construida de alguna capacidad física significativa, tales como "PUSH" o "GO TO".

El principal sensor del sistema perceptual es la cámara de TV. Los programas que procesaban los datos de las imágenes fueron restringidos a unas pocas rutinas de visión básicas, las cuales orientan al robot y detectan y localizan objetos. Estos programas son incorporados en el sistema tanto al nivel de ILAs como de LLAs.

El nivel de Planificación: este nivel está relacionado con la planificación de la solución a los problemas. El mecanismo de planificación básico es STRIPS. STRIPS construye secuencias de ILAs necesarias para realizar las tareas especificadas. Tal secuencia, junto con sus efectos esperados, pueden ser representados por una tabla triangular llamada MACROP (Operación Macro).

El nivel Ejecutivo: el quinto nivel del sistema es el denominado Ejecutivo, el programa que realmente invoca y monitorea la ejecución de ILAs especificadas en una MACROP. El programa ejecutivo se denomina PLANEX.

Arquitectura de Control NASREM

NASREM fue la culminación hacia 1989 de más de quince años de investigación en el proyecto Sistemas de control de tiempo real (RCS) del NIST (*National Institute of Standards and Technology*) en los Estados Unidos con robots y máquinas inteligentes. La

primera versión de RCS se desarrolló para el laboratorio de Robótica del NIST y se adaptó para el control de manufactura en el centro de investigación durante los ochenta. Desde 1986, RCS fue usado para implementar un número variado aplicaciones adicionales, incluyendo el proyecto de vehículos acuáticos autónomos múltiples NBS/DARPA, el robot de manejo de materiales en campo de la *Army* (FMR) y el proyecto de vehículos semi autónomos terrestres TEAM de la *Army*. En 1987 RCS fue adaptada para su aplicación en el Servicio Telerobótico en Vuelo de la estación espacial, convirtiéndose en el llamado Modelo Estándar de Referencia para la Arquitectura de un Sistema de Control Telerobótico (NASREM)[2].

El paradigma fundamental de la arquitectura conceptual de NASREM se muestra en la figura 1.2. La arquitectura de control es representada como una jerarquía de tres pilares de módulos de computación, asistidos por un sistema de comunicaciones y una memoria global, a lo largo de los cuales se encuentran seis niveles tipo *top – down*: *service mission, service bay, task, e-move, primitive* y *servo*.

El pilar 1 del módulo de descomposición de tareas desarrolla planificación de tiempo real y funciones de monitoreo de tareas; éstos descomponen los objetivos de las tareas en espaciales y temporales. El pilar 2 del módulo de procesamiento sensorial filtra, correlaciona, detecta e integra información sensorial tanto espacial como temporal con el fin de reconocer y medir patrones, características, objetos, eventos y relaciones en el mundo externo. El pilar 3 del módulo del modelo del mundo responde inquietudes, hace predicciones y calcula funciones de evaluación sobre el espacio de estado definido por la información almacenada en la memoria global. La memoria global es una base de datos la cual contiene la mejor estimación del sistema acerca del estado del mundo externo. Los módulos del modelo del mundo mantienen la base de datos de la memoria global actualizada y consistente.

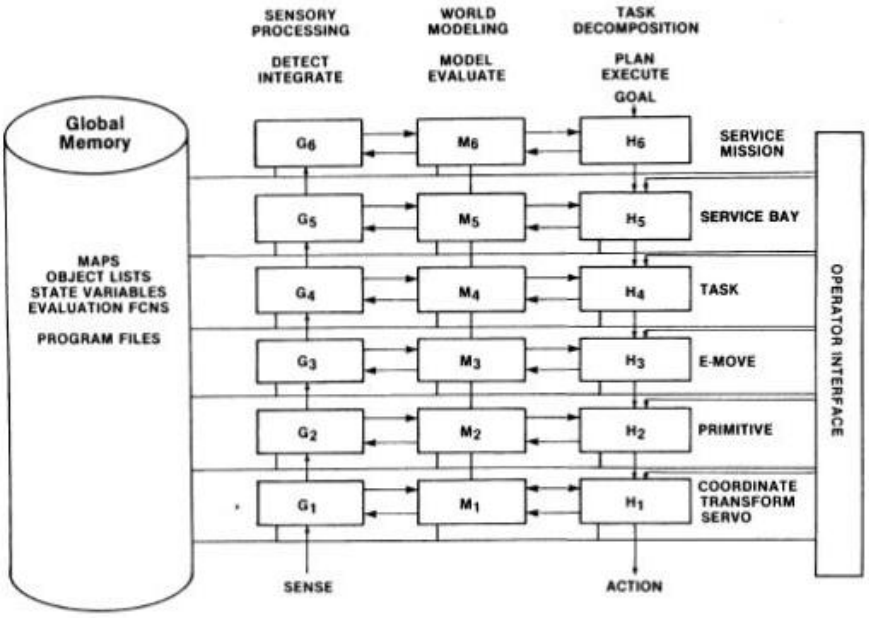


Figura 1. 2 Arquitectura de control de NASREM. Fuente[2]

Los niveles jerárquicos de NASREM se describen a continuación:

El nivel de tareas es el nivel de control más alto de NASREM relacionado solamente con el robot autónomo simple. El nivel de tareas recibe comandos desde el nivel *Service Bay*. Cada comando es descompuesto en una secuencia de "elementos de manipulación" incomprensibles para el nivel *E-move*. Para alcanzar esta descomposición, el módulo de planificación a nivel de *Tasks* puede usar técnicas de IA para planear autónomamente, o usar planes pre especificados ingresados por un operador.

El nivel *E-move* recibe comandos desde el nivel de tareas que representan "manipulaciones" elementales", tales como MOVE <objeto> TO <destino>. El nivel *E_move* planea todos los aspectos de la manipulación. Esto incluye encontrar un camino libre de colisiones y determinar cualquier estrategia de movimiento fino. Como sucede en el nivel de *Tasks*, parte de la descomposición de *E-move* puede ser obtenida a partir de planes pre especificados. Los segmentos de camino entregados por *E-move* son ejecutables por el nivel *Primitive*.

El nivel *Primitive* es responsable de la generación de la secuencia temporal de vectores de estado deseados necesarios para producir una trayectoria dinámica a partir del comando de *E-move*. El nivel *Primitive* determina el comportamiento del manipulador sobre la escala de tiempo entre el nivel *E-move* y el nivel *Servo*. Éste transforma los segmentos de camino en secuencias dinámicas de puntos de camino directamente ejecutables por el nivel *Servo*.

El nivel *Servo*, el nivel más bajo de la jerarquía de descomposición de tareas, calcula las señales de control para los actuadores. El trabajo del nivel servo es manejar un pequeño movimiento en un sentido dinámico. Este nivel ejecuta un algoritmo específico para acercarse a los vectores de estado deseados. El conjunto de vectores deseados es típicamente un punto en una trayectoria compleja calculada por el nivel *Primitive.* El nivel *Primitive* actualiza el conjunto deseado periódicamente.

La arquitectura de control tiene una interface de operador en cada nivel en la jerarquía. La interface provee un medio por medio del cual operadores humanos, en la estación espacial o en tierra, pueden observar y supervisar el telerobot. Cada nivel de la jerarquía de descomposición de la tarea provee una interface donde el operador humano puede asumir el control.

1.2 Arquitecturas Reactivas

Arquitectura de Subsumption

En 1985 Rodney Brooks proporcionó una descomposición al problema de control de un robot en capas hardware de niveles de comportamiento, en lugar del clásico planteamiento secuencial funcional. Dentro de esta configuración, presentó la idea de la *subsumption*, esto es, que las capas más complejas no solo dependen de sus capas inferiores, más reactivas, sino que también podían influenciar el comportamiento de las mismas. La arquitectura resultante podía simultáneamente atender múltiples y potencialmente conflictivos objetivos de una forma reactiva, proporcionado precedencia a los objetivos de alto nivel [3].

Esta arquitectura predica sobre la sinergia entre la sensación y la actuación propia de los animales inferiores, tales como los insectos. Brooks propone que en lugar de construir robots complejos para mundos simples se debe seguir el camino evolutivo y empezar construyendo robots simples para un mundo real, complejo e impredecible. A partir de este argumento se destacan una serie de características claves de la arquitectura de *subsumption*:

1. No se hace uso de representación explicita de conocimiento.
2. Los comportamientos son distribuidos más que centralizados
3. Las respuestas a los estímulos es reflexiva, esto es, la secuencia percepción - acción no es modulada por una deliberación cognitiva.

La arquitectura como tal es implementada en hardware con máquinas de estado finito aumentadas (AFSM), esto es FSM (*Finite State Machine*) temporizadas, organizadas en capas, correspondiendo cada capa a comportamientos pre-alambrados. La característica aumentada de las FSM consiste de temporizadores los cuales, una vez finaliza la temporización, pueden bloquear una entrada a una FSM (una inhibición) o suprimir una línea de salida (una supresión). El rol principal de estos inhibidores y supresores es permitir a las capas de comportamiento de niveles altos subsumir aquellos de los niveles inferiores. Así el control del comportamiento general de los robots se distribuye a través de la interacción de las capas hardware de la arquitectura sin ningún mecanismo de control centralizado.

Brooks demostró que es a menudo ventajoso descomponer un sistema en tareas paralelas o comportamiento de niveles incrementales de competencia, más que la clásica descomposición funcional. Mientras una descomposición típica del paradigma deliberativo puede ser de la forma:

Sensores -> percepción -> modelamiento -> planificación -> reconocimiento de tareas-> control motor.

Rodney Brooks propuso descomponer el mismo dominio en la siguiente forma:

evitar objetos < deambular < explorar < construir mapas < monitorear cambios < identificar objetos < planear acciones < razonar sobre el comportamiento del objeto.

Donde < denota niveles incrementales de competencia. Los beneficios potenciales de este acercamiento incluyen robusticidad incrementada, soporte de concurrencia, construcción incremental y facilidad de realizar pruebas.

La figura 1.3 y el listado siguiente representan la organización por niveles de competencia de la arquitectura de *subsumption* que debe cumplir un robot móvil. De acuerdo a Brooks un nivel de competencia es una especificación informal de una clase de comportamiento deseada para un robot para todos los ambientes que éste pueda encontrar.

Nivel 0: Evitar contacto con objetos (sean éstos estáticos o en movimiento).

Nivel 1: Deambular sin un rumbo fijo evitando golpear objetos.

Nivel 2: Explorar el mundo observando lugares a la distancia que se vean alcanzables y accesibles.

Nivel 3: Construir un mapa del entorno y planear rutas para ir de un lugar a otro.

Nivel 4: Notificar de cambios que se presenten en el entorno "estático".

Nivel 5: Razonar sobre el mundo en términos de objetos identificables y realizar tareas relacionadas a ciertos objetos.

Nivel 6: Formular y ejecutar planes los cuales involucran cambiar el estado del mundo en una forma deseable.

Nivel 7: Razonar sobre el comportamiento de los objetos en el mundo y modificar los planes apropiadamente.

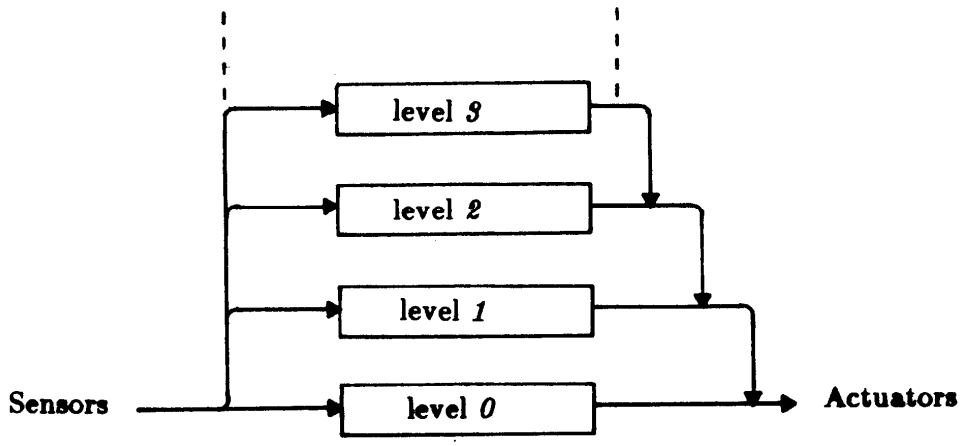


Figura 1.3 Arquitectura por capas de *Subsumption*. Los niveles superiores subsumen la función de las capas inferiores cuando desean tomar el control. Fuente [3]

Una de las hipótesis de *subsumption* es que el comportamiento complejo puede ser el producto de muchos comportamiento simples interactuando unos con otros en un ambiente complejo. Este enfoque en simplicidad conduce a un diseño ya explicado donde cada capa individual se compone de módulos de AFSM interactuando asincrónicamente sin ningún tipo de control central. En general las diferentes capas no son completamente independientes. En la descomposición anterior, el deambular y el explorar dependen de la habilidad del robot para evitar objetos. Pero el sistema puede estar capacitado para cumplir con múltiples objetivos en paralelo a pesar de la dependencia. Los objetivos de una capa ocasionalmente pueden entrar en conflicto con los de otra capa, en cuyo caso los objetivos de alta prioridad deberán restar valor a los de baja prioridad. Por esta razón la arquitectura de *subsumption* dispone de mecanismos por medio de los cuales las capas altas, más competentes, pueden observar el estado de las inferiores, inhibiendo sus salidas y cancelando sus entradas, y de esta forma ajustan su comportamiento. Las tareas de alta prioridad de las capas inferiores (tales como, detenerse cuando un objeto está al frente) seguirán teniendo una prioridad por defecto si el diseñador evita cualquier manipulación en esas tareas particulares.

Arquitectura DAMN

La gran mayoría de las arquitecturas deliberativas o híbridas, imponen una estructura *top – down* para alcanzar la simbiosis de elementos reactivos y deliberativos, la Arquitectura Distribuida para Navegación Móvil (DAMN, por sus siglas en inglés) toma un acercamiento diferente donde múltiples módulos de comportamientos reactivos o deliberativos comparten el control del robot concurrentemente, enviando votos que permitan seleccionar los comandos de acción para ser ejecutados [4].

DAMN (*Distribuited Architecture for Mobile Navigation*) es una arquitectura distribuida reactiva propuesta por Julio Rosemblat durante su doctorado en el CMU (*Carnegie Mellon University*) a mediados de la década del noventa. Está basada en la ejecución de múltiples comportamientos que acceden al control del robot interactuando con un mecanismo de votación. El objetivo es integrar mecanismos reactivos con componentes deliberativos sin tener que imponer una estructura jerárquica tipo *top – down*.

La arquitectura DAMN está conformada por los siguientes elementos:

- El sistema de control de votación o Arbitro DAMN.
- Los comportamientos.
- Un controlador de modo de operación.
- El controlador hardware del robot.

La figura 1.4 muestra las interconexiones entre los diferentes elementos que integran a DAMN.

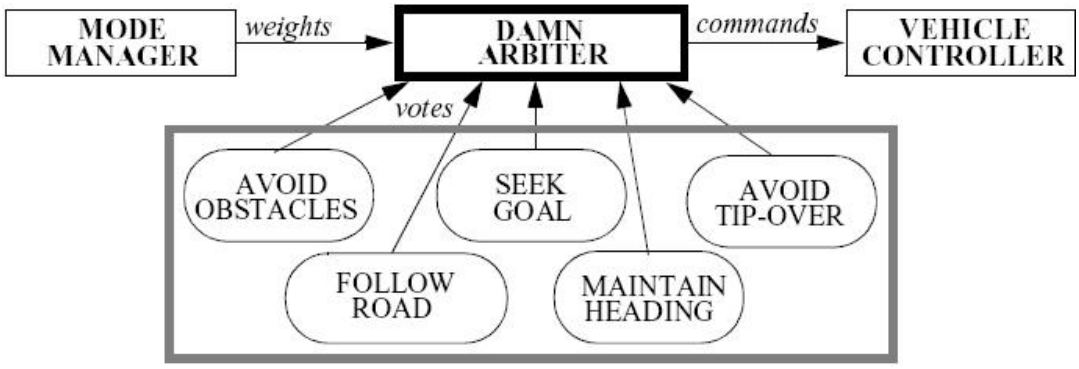


Figura 1.4 Arquitectura DAMN e interconexiones entre sus componentes. Fuente [4]

Los diferentes comportamientos con independencia de su nivel de competencia, proporcionan votos positivos o negativos en el espacio de comandos (girar, velocidad, área de visión, etc). El mecanismo de votación es el encargado de fusionar los resultados para elegir la opción más votada. Esta selección es el resultado de diferentes procesos que incluyen la combinación de las salidas (ponderadas según indique el controlador de modo de operación), el filtrado y la interpolación. La operación es similar a lo que sucede en la arquitectura de *subsumption*, aunque en DAMN se dispone de modelos del mundo que pueden ser empleados.

En esta arquitectura los comportamientos son agrupados en tres niveles de competencia:

1. Comportamientos orientados a seguridad
    - De la dinámica del vehículo: por ejemplo límites en los giros y velocidades.
    - En evasión de obstáculos: evaluación de colisiones en la ruta seguida.
    - Comportamientos auxiliares: movimientos y acciones de inercia por defecto.
2. Comportamientos orientados a acción
    - Monitoreo de caminos.
    - Operación remota.
    - Fin del camino.
3. Comportamientos orientados a objetivos
    - Sub objetivos.
    - Campos de gradiente.

- Planificación de caminos.

El diseño impuesto en DAMN le permite combinar comportamientos usando diferentes frecuencias de operación, como es el caso de comportamientos reactivos de alta frecuencia contra comportamientos deliberativos de baja frecuencia. La fusión de los comandos generados determina el comportamiento del sistema, de tal forma que las acciones producidas por los módulos de mayor peso son las que ejercen mayor influencia. A pesar de esto, mecanismos de operación de alto nivel pueden ser empleados para controlar la asignación de pesos, proporcionando un nivel de control superior.

DAMN tiene capacidad de ejecución adaptiva, puede aprender o adaptarse a las condiciones. Esto se soporta en la modificación dinámica de los pesos asignados a cada comportamiento. Es de esta forma que se hace posible introducir aprendizaje en el sistema, de tal forma que una configuración adecuada de pesos pueda ser registrada para su futura utilización en situaciones similares.

1.3 Arquitecturas Híbridas

Arquitectura AURA

El concepto de arquitectura Deliberativa híbrida con Reactiva se le atribuye a Ronald C. Arkin en el Georgia Tech, está basado en la "teoría de esquemas percepción - motor" y el acercamiento fue implementado en una arquitectura para robots autónomos denominada AuRA (*Autonomous Robot Architecture*), de dos niveles orientada a ejecutar tareas de navegación. AuRa implementa ideas tomadas del estudio de los sistemas biológicos y de la neurofisiología aplicada a los sistemas robóticos [5].

La estructura de AuRa comprende varios sistemas organizados en dos niveles: un nivel deliberativo superior y un nivel reactivo inferior. El nivel deliberativo es una jerarquía compuesta por un planificador de misión, un razonador espacial y un secuenciador de

planes, apoyados en una representación del ambiente del robot y de su estado interno, ver figura 1.5. El nivel reactivo se compone de los sistemas percepción y motor, éste último es diseñado como un administrador - monitor de esquemas. Cada esquema motor es asociado con un esquema de percepción, el cual brinda el estímulo que el comportamiento requiere. Los comandos de actuación generados por estos comportamientos, percepción – motor, son sumados y normalizados en un proceso especial para ser enviados al hardware del robot, ver figura 1.5. El nivel reactivo se basa en el modelo de comportamientos emergentes, se cuenta con un conjunto limitado de esquemas que integran unos comportamientos básicos, que por composición pueden dar lugar a comportamientos complejos. Los esquemas de la arquitectura AuRa se caracterizan por los siguientes aspectos listados:

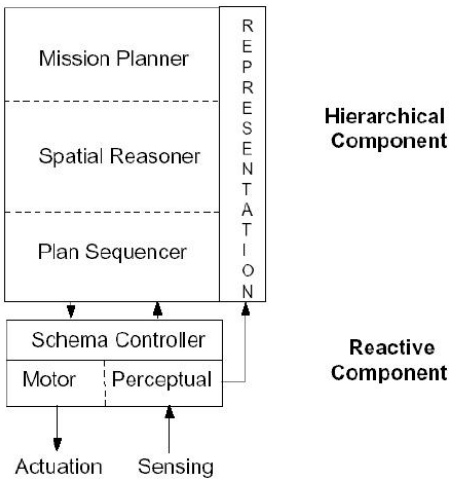


Figura 1.5 Arquitectura de AuRA. El componente reactivo conforma el nivel inferior y el componente jerárquico conforma el nivel deliberativo superior. Fuente [5]

- Los esquemas forman una red dinámica de procesos.
- Cada esquema es un módulo computacional independiente.
- La selección de los comportamiento básicos se realiza por el secuenciador en el nivel deliberativo empleando la información de la misión, el ambiente y el estado del robot.
- Con el listado de comportamientos básicos el administrador de esquemas, en el nivel reactivo, selecciona aquellos que formarán los comportamientos solicitados.
- El comportamiento presentado por el robot es una composición de los

comportamientos básicos ejecutados asíncronamente.

A continuación se explica la funcionalidad de cada uno de los sistemas presentes en los dos niveles de AuRA, según se observa en la figura 1.6.

Sistema de Percepción: es un repositorio de esquemas de percepción (ps), estos son responsable de adquirir la información captada por los sensores, procesarla y enviarla a los sistemas cartográfico y motor.

Sistema Cartográfico: gestiona dos zonas de memoria. La memoria a largo plazo (LTM), en la que se encuentra información inicial y fija del ambiente y la memoria a corto plazo (STM), en la que se crea un modelo dinámico del entorno según la información procesada de los sensores.

Sistema de planificación: tiene una estructura jerárquica de tres niveles, con un módulo por nivel: planificador de misiones, el razonador espacial y el secuenciador de planes.

1. Módulo planificador de misiones: éste realiza el plan de alto nivel y para ello dispone de una interfaz de usuario que permite la definición de las tareas a realizar.
2. Módulo razonador espacial (planificador de caminos): es el responsable de diseñar una ruta inicial para el robot interactuando con el subsistema cartográfico.
3. Módulo secuenciador de planes: recibe la ruta diseñada e interactúa con el administrador de esquemas motor (ms) con el fin de seleccionar los comportamientos básicos que permitan llevar al robot en la ruta establecida. La elección del esquema de percepción que lleva asociado un esquema motor no está definida a priori, se realiza en función del ambiente, el estado interno del robot y las tareas a cumplir.

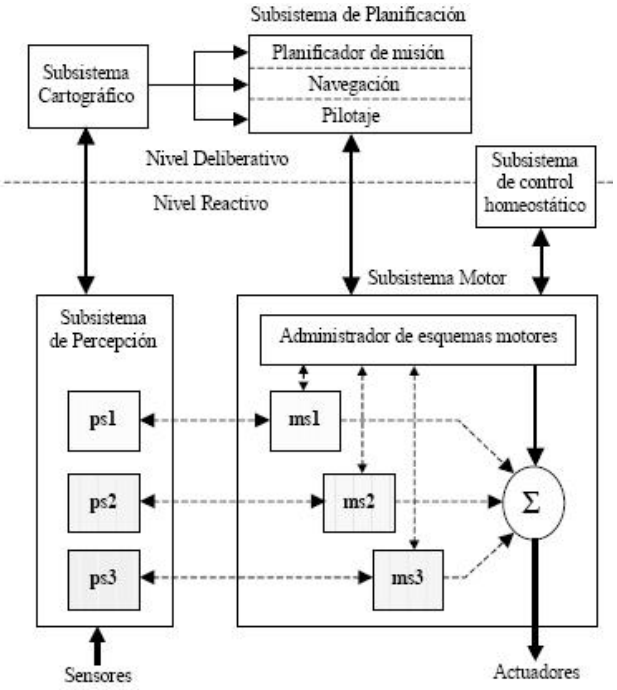


Figura 1.6 Arquitectura híbrida Aura, organizada en dos niveles: reactivo y deliberativo y cinco sub sistemas. Fuente [7]

Sistema Motor: es un repositorio de esquema motor gestionado por un administrador de esquemas que tiene acceso al repositorio del sistema de percepción para conformar los comportamientos básicos seleccionados por el módulo de conducción. Un mismo comportamiento puede estar formado por varios esquemas motor coordinados por FSM. Dentro de cada comportamiento se permite el uso de memoria, conocimiento y representaciones internas. Este sistema motor calcula la contribución de los comportamientos seleccionados al movimiento total del robot y realiza la composición de sus acciones motoras basándose en la teoría de campos de potencial o de fuerza.

Sistema Homeostático: es el responsable de garantizar la supervivencia del robot. Realiza esto manteniendo un estado de equilibrio entre los diferentes elementos que conforman el robot. Está en capacidad de alterar el comportamiento del robot en función de la evolución del ambiente y del estado interno del robot.

Arquitectura 3T

Esta arquitectura asociada a la NASA, resulta de la cooperación de varios investigadores como Peter Bonasso, James Firby, Erann Gat y David Korterkamp. Arquitecturas cercanas a esta son AAA y Atlantis. El principal objetivo de 3T es alcanzar un comportamiento robusto en la ejecución de tareas por medio de la combinación de reactividad y deliberación [6].

3T está organizada en tres capas como se observa en la figura 1.7. Una capa reactiva de habilidades o capacidades, una capa de secuenciamiento de tareas y una capa deliberativa donde se realiza la planificación de la misión.

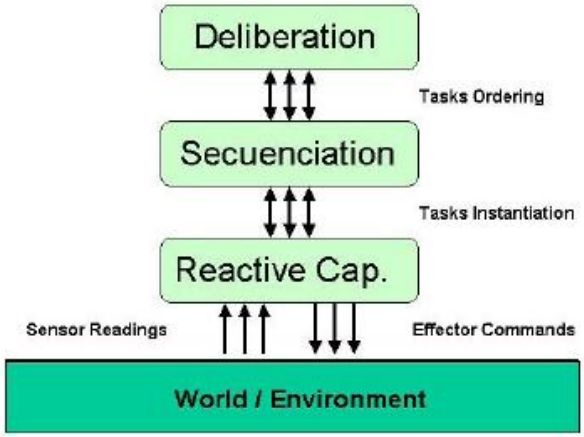


Figura 1.7 Arquitectura 3T de tres capas: deliberación, secuenciación y capacidades. Fuente [6]

En cada uno de las capas se encuentran diferentes componentes los cuales están ubicados dependiendo de las necesidades temporales de ejecución, ver figura 1.8.

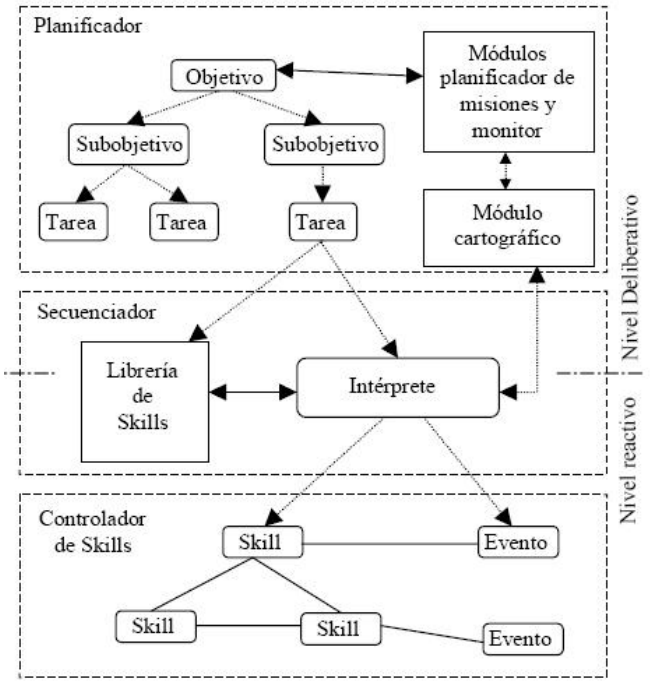


Figura 1.8 Arquitectura híbrida 3T, estructura de tres capas con dos niveles básicos: reactividad y deliberación. Fuente [7]

La capa deliberativa superior está formada por el Planificador, éste descompone la misión en una lista de tareas con base en los objetivos a cumplir por la misión. Éstas a su vez son descompuestas en uno o más conjuntos de acciones o RPA (Paquetes de Acción Reactiva). Para realizar su función esta capa se apoya en tres módulos: Planificador de misión, el Cartográfico o modelo del mundo y el Monitor, ver figura 1.8.

La capa de secuenciamiento es una especie de interface entre el nivel deliberativo y el nivel reactivo. Esta capa es la responsable de recibir los RAP activos y controlar su ejecución por medio de la selección de las habilidades (*skills*) requeridas en el nivel reactivo, para ello desde un repositorio selecciona las habilidades básicas para cada RAP, mientras simultáneamente un conjunto de monitores de eventos se activan indicándole al secuenciador cuando las habilidades se han activado. El secuenciador modifica la composición de RAP de las tareas en respuesta a los eventos, a violaciones temporales o a nuevos cambios provenientes del planificador.

La capa de habilidades está conformada por un controlador de habilidades, el cual

establece una interface uniforme con el secuenciador. Dos tipos de señales son enviadas por esta interface: comandos dirigidos a las habilidades y eventos de notificación para el secuenciador. El secuenciamiento es llevado a cabo por el interpretador de RAP. Este módulo usa un repositorio de habilidades indexadas por parámetros para mapearlos en diferentes configuraciones de habilidades. Los eventos constituyen un tipo especial de capacidad, dedicada a señalizarle al secuenciador sobre la detección de circunstancias relevantes sobre el progreso de la actividad, tales como finalización de tareas, cambios en el ambiente, etc.

Una condición importante que debe cumplir esta arquitectura es tener tres capas bien definidas operando concurrentemente en forma asíncrona. Otra condición clave de 3T es la definición de la localización adecuada por capas para cualquier actividad del sistema. Para ello propone el análisis de cuatro características:

1. El periodo de activación: los periodos típicos empleados en el nivel reactivo están en el orden de los milisegundos, en el orden de las decimas de segundo en el secuenciador y del rango de los segundos en el planificador.
2. Ancho de banda: las habilidades de bajo nivel pueden procesar altos volúmenes de datos mientras las comunicaciones entre niveles demandan un bajo ancho de banda.
3. Funcionalidad: si al interior de una actividad, en cualquier nivel, se hace uso de una funcionalidad ya incluida en la arquitectura, la actividad debe ser rediseñada con el fin de permitir que un mecanismo por defecto realice esta función (por ejemplo: habilidades que realizan selección de acciones o RAP que administran recursos).
4. Flexibilidad: las habilidades usadas ya se encuentran compiladas y no pueden ser modificadas en tiempo de ejecución. El secuenciamiento y la planificación, por otra parte, pueden ser modificados ya que están basados en interpretadores.

La arquitectura 3T provee soporte para la ejecución adaptiva permitiendo que varios métodos de solución puedan ser declarados para cada tarea. Cada método tiene asociado un conjunto de condiciones de aplicabilidad que deben ser tenidos en cuenta para su selección. Finalmente, cada tarea incluye una prueba de satisfacción para determinar cuando ésta ha sido completamente finalizada.

Arquitectura Saphira

En la arquitectura Saphira se han considerado tres objetivos específicos que deben ser considerados para un robot móvil autónomo: ejecución robusta de tareas, seguimiento de personas y construcción de mapas [8]. Para el cumplimiento de estos objetivos la arquitectura debe incorporar tres características básicas: coordinación, coherencia y comunicación. Saphira aplica estas premisas para constituirse en una arquitectura empleada en el diseño de aplicaciones de seguimiento, navegación e interacción con robots móviles [8].

La arquitectura de Saphira es una estructura por niveles, un deliberativo que procesa tareas y un nivel reactivo que procesa comportamientos, cuyos componentes están alrededor de un mecanismo central interno de representación de información, el espacio local perceptual (LPS). Alrededor del LPS hay componentes de percepción responsables de transferir datos de los sensores al LPS y extraer información de ellos, y componentes de acción en los cuales diferentes comportamientos son ejecutados. En estos componentes coexisten varios tipos de comportamientos: reactivos a bajo nivel, orientados a objetivos a un nivel intermedio y orientados a tareas en un alto nivel. La figura 1.9 presenta los diferentes elementos que integran esta propuesta.

El LPS es una representación geométrica del ambiente alrededor del robot y del estado interno del robot, organizada en diferentes niveles según la complejidad de los comportamientos que requieran de ella, y es fundamental para las tareas de fusión de datos, planificación de movimientos y la integración de la información del mapa. Todos los componentes de Saphira hacen constante referencia al LPS, lo que le proporciona coherencia a la arquitectura en su representación.

La organización de la arquitectura, como se puede ver en la figura 1.9, es parcialmente vertical y parcialmente horizontal. La organización vertical se puede observar tanto en la percepción (lazo izquierdo), como en la acción (lado derecho). Varios componentes de percepción se encargan de procesar datos sensoriales y adicionarlos al LPS, con el fin de que puedan ser empleados por los componentes de reconocimiento de objetos y de navegación. Los comportamientos más complejos que realizan acciones dirigidas a objetivos son utilizados para guiar el comportamiento reactivo. En el nivel deliberativo de tareas, los comportamientos son secuenciados y su progreso monitoreado por medio de eventos en el LPS. La estructura horizontal aparece debido a que los

comportamientos pueden elegir la información apropiada del LPS, por ejemplo los comportamientos críticos, como evadir obstáculos, confían en un procesamiento simple de los sensores ya que deben estar disponibles rápidamente.

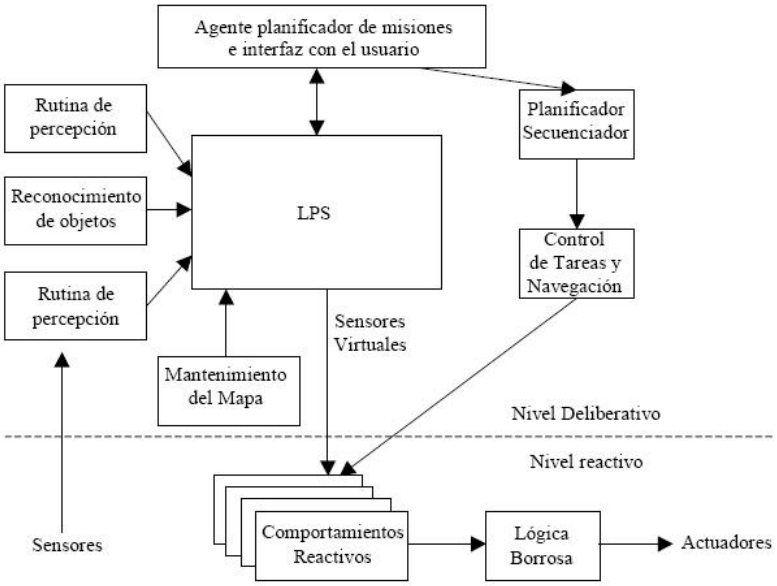


Figura 1.9 Arquitectura híbrida Saphira, estructura de dos niveles: reactivo y deliberativo organizados alrededor de un LPS. Fuente [7]

En el nivel de control de bajo nivel Saphira se soporta en comportamientos concurrentes. El problema de control se descompone en unidades de control llamadas comportamientos básicos, como evadir obstáculos o seguir un pasillo. Una de las características diferenciadoras de Saphira es que los comportamientos básicos están escritos y son combinados utilizando técnicas basadas en lógica difusa.

1.4 Resumen

En este anexo se ha presentado un resumen de las características técnicas de los tres paradigmas clásicos empleados en la robótica móvil en el diseño de arquitecturas de control: deliberativo, reactivo e híbrido. De cada uno de ellos se realizó un análisis de

arquitecturas representativas: Shakey y Nasrem en el paradigma deliberativo, Subsumption y DAMN en el paradigma reactivo y finalmente AuRa, Saphira y 3T en el paradigma híbrido.

Como se puede apreciar de la arquitectura de control de Shakey y Nasrem, el paradigma deliberativo implica la planificación explícita de cada una de las acciones del robot. Igualmente se caracteriza por obtener y utilizar un modelo interno del mundo real lo más exacto y completo posible para emplearlo como base de razonamiento para todos los planes y acciones. De manera que el procesamiento de la información sensorial se orienta a la consecución de este modelo detallado del entorno del robot. No solo la NASA, el NIST y el SRI en Estados Unidos trabajaron empleando este paradigma durante los setenta y ochenta, sino todo grupo de investigación fuese de IA o de Robótica. En general todas esas arquitecturas deliberativas realizan un flujo de información consistente en mediciones de los sensores, realizar un procesamiento durante tres o seis niveles hacia arriba, donde se planea y se regresa una orden, que pasa nuevamente por todos los niveles, hasta llegar a los actuadores.

El paradigma reactivo, inspirado en la observación del comportamiento instintivo de los seres vivos, más conocido con el nombre de arquitectura de *Subsumption*, tiene su principal interés en la incorporación de comportamientos, en la forma de pares de esquemas sensor-motor, para modelar las habilidades básicas de la máquina. Originalmente *Subsumption* fue de implementación hardware con máquinas de estado finito modelando cada comportamiento, donde el nivel sensor se comunica directamente con el nivel de actuación, pero posteriormente se diseñó un lenguaje llamado *Behaviour*. A pesar de que se pueda pensar que en esta arquitectura los sensores están comunicados directamente con los actuadores, se emplean las conexiones sensor-motor para determinar la consigna a seguir por parte de los sistemas de control que gobiernan a los actuadores. El enfoque se caracteriza por: 1) su aversión a los planes; no utiliza un nivel de planificación que coordine antes de actuar, 2) la ejecución concurrente de múltiples comportamientos, 3) un mecanismo de composición de comportamientos que los inhibe, suprime o combina y, 4) la no generación de modelos del entorno; en palabras de Brooks “el mundo real es el mejor modelo”.

El paradigma Híbrido de Ronald C. Arkin en 1987 y su arquitectura AuRA, al igual que 3T y Saphira aprovechan lo mejor de los enfoques previos. En primera instancia se tiene un nivel inferior de múltiples comportamientos (SA) encargados del control directo del robot, pero se hace uso, en segunda instancia, de un nivel superior de Planificación (P) (sin embargo no tan exigente o pesado como el usado en Shakey o NASREM) y

finalmente un nivel intermedio de secuenciamiento que conecta el nivel inferior y superior (SA, P). No todas las arquitecturas híbridas proponen una capa de secuenciamiento, simplemente un nivel reactivo en primer plano y un nivel deliberativo en segundo plano, por lo que en este detalle se diferencian las arquitecturas híbridas.

2. Bibliografía Anexo

[1] Nilsson, N. (1984). Shakey the robot. Technical Report 323. Artificial Intlligence Center, SRI international.

[2] Albus, J., McCain, H. & Lumia, R. (1986). NASA/NBS Standard Reference Model for Telerobot control System Architecture (NASREM). NASA Document SS-GSFC-0027.

[3] Brooks, R. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2 (1), 14–23.

[4] Rosenblatt, J. (1995). DAMN: A Distributed Architecture for Mobile Navigation. In *proceedings of the AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*. 1, 339-360.

[5] Arkin, R. & Balch, T. (1997). AuRA: Principles and Practice in Review. Journal of Experimental and Theoretical Artificial Intelligence, 9, 175-189.

[6] Bonnaso, P. & Kortenkamp, D. (1997). Experiences with an architecture for intelligent, reactive agents. Journal of Experimental and Theoretical Artificial Intelligence, 9, 237-256.

[7] Domınguez-Brito, A. C. (2003). CoolBOT: a Component-Oriented Programming Framework for Robotics. Tesis Doctoral, Universidad de Las Palmas de Gran Canaria.

[8] Knonolige, K., Myers K., Ruspini, E. & Saffioti, A. (1997). The Saphira Architecture: A design for Autonomy. Journal of Experimental and Theoretical Artificial Intelligence, 9, 215-235.