Análisis de arquitecturas existentes para robótica colectiva y desarrollo de nuevas soluciones que mejoren las identificadas

Autor: Matías A. Villanueva Sampayo

Tesis doctoral UDC / 2020

Directores: Juan Jesús Romero Cardalda
Adrián Carballal Mato
Tutor: Juan Jesús Romero Cardalda

Programa de doctorado en Tecnoloxías da Información e as Comunicacións

UNIVERSIDADE DA CORUÑA

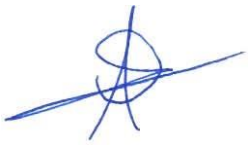
D. Juan Jesús Romero Cardalda y **Dr. Adrián Carballal Mato**, profesores de la Universidade da Coruña del Departamento de Ciencias de la Computación y Tecnologías de la Información como directores de la presente Tesis doctoral:

HACEN CONSTAR QUE:

La tesis titulada "**Análisis de arquitecturas existentes para robótica colectiva y desarrollo de nuevas soluciones que mejoren las identificadas**" realizada por D. **Matías A. Villanueva Sampayo,** bajo nuestra dirección en el Departamento de Ciencias de la Computación y Tecnologías de la Información de la Universidade da Coruña cumple con los requisitos para optar a grado de Doctor.

Y para que conste, firman la presente a 13 de noviembre de 2020

Adrián Carballal Mato



Juan Jesús Romero Cardalda

Matías A. Villanueva Sampayo

Agradecimientos

Expreso mi sincera gratitud a mi tutor de tesis, J. Romero, que me ha apoyado y animado durante el desarrollo de este trabajo. Su método me ha permitido avanzar con libertad y nunca me he sentido perdido en la dirección que debía tomar. Esto es así porque su perspicaz propuesta de líneas acaba por ser asumida naturalmente como propia y hace más fácil y agradable la tarea.

Agradezco también a Adrián Carballal su didáctica del método científico, y el hecho de que la lleve a cabo con inagotable paciencia.

Quiero dar también innumerables gracias a Dani, sin cuya opinión crítica y apoyo nunca hubiera llegado hasta donde estoy.

El personal de la UADI y del Programa de Doctorado merecen también mi gratitud por su perfecta atención.

Son también dignos de mención mis inmejorables compañeros durante mi tiempo como estudiante: Marcos Baltar, Eva Marcos, Mónica Rial, Alfonso Rivero, Raúl Rodil, Francisco Rodríguez, Marina Rodríguez, Francisco Ruido, Pablo Simón, Diego Souto, Juan Valiño y tantos otros que han contribuido en mi búsqueda de conocimiento y que han hecho gratos esos años.

Expreso mi agradecimiento también a mi familia que nunca ha dejado de apoyarme; a mis padres, Juan Luis y María Hermitas, por su ejemplo de esfuerzo y constancia en el trabajo; a mis hermanos, Juan y Marita, por aconsejarme siempre bien.

Por último, pero no menos importante, agradezco a Elena, además de sus observaciones lingüísticas, todo su apoyo y comprensión.

Resumo

A presente tese está orientada a arquitecturas de sistemas multi-robot en xeral, sen restricións de tamaño do colectivo de robots ou do eido de aplicación. En primeiro lugar identifica e describe os atributos destes sistemas así como das arquitecturas que lles dan soporte para logo propoñer un sistema de avaliación que permite cuantificar as capacidades destas últimas. Este sistema está baseado en probas e simulacións de diferentes aspectos como son (entre outros) a coordinación, a adaptación ou a interoperabilidade.

Os resultados do sistema de avaliación permiten a comparación e selección da arquitectura máis axeitada dependendo das necesidades do novo sistema a desenvolver. Ademais este sistema permite a avaliación de forma rápida de modificacións na arquitectura.

A arquitectura proposta trata de cubrir todas as posibilidades útiles en sistemas multi-robot permitindo a reutilización de calquera compoñente do sistema. Ademais facilita a interoperabilidade con outros sistemas e define un protocolo de mensaxes (a modo de linguaxe) que deben usar os membros do colectivo. Esta linguaxe representa o conxunto mínimo de operacións para que un colectivo poida desenvolver calquera tarefa. A esta proposta aplícaselle o sistema de avaliación dando mellores resultados na maioría das métricas que as outras arquitecturas avaliadas.

Resumen

La presente tesis está orientada a las arquitecturas de sistemas multi-robot en general, sin restricciones de tamaño del colectivo de robots o del ámbito de aplicación. Primeramente identifica y describe los atributos de estos sistemas así como los de las arquitecturas que les dan soporte para luego desarrollar un sistema de evaluación que permita identificar las capacidades de estas últimas, basado en la prueba y simulación de diferentes aspectos como son (entre otros) la coordinación, la adaptación o la interoperabilidad.

Los resultados del sistema de evaluación permiten comparar y seleccionar de entre las arquitecturas la más idónea según las necesidades un nuevo sistema a desarrollar. Además permite evaluar, de forma rápida, modificaciones.

La arquitectura propuesta trata de cubrir todas las posibilidades útiles en sistemas muti-robot permitiendo la reutilización de cualquier componente del sistema. Además facilita la interoperabilidad con otros sistemas y define un protocolo de mensajes básico, a modo de idioma, que deben usar los miembros del colectivo multi-robot. Este idioma representa el conjunto de operaciones mínimo para que un colectivo pueda cooperar desarrollando cualquier tarea. A esta propuesta se le ha aplicado el sistema de evaluación con resultados que, en su mayoría, mejoran las arquitecturas existentes.

Abstract

This thesis is aimed to multi-robot architectures in general, without size of collective or application environment restrictions. The attributes that describe the architectures and multi-robot systems are identified and documented. These attributes contribute to generate metrics of the architectures' relevant aspects that, in turn, are used to quantify the capacities. The metrics compose an evaluation system that rely on the simulation of the relevant aspects such as coordination, adaption, or interoperability.

Results from the evaluation system allow architectures comparison and selection in order to implement a new multi-robot system. The evaluation system enables the quick evaluation of an architecture modifications.

This thesis also propose a new architecture trying to cover all multi-robot systems based on reusing the system components as needed; This architecture makes easy to operate with other systems. A message protocol (a kind of language between the robots) is defined containing the minimal set of operations that the individuals must implement in order to carry any task. The evaluation system is applied to this new architecture yielding better results in most of the metrics than the other architectures evaluated.

Prefacio

Es innegable que un individuo (ya sea un robot o un miembro de alguna especie biológica) tiene unas capacidades limitadas que pueden ser insuficientes para acometer una determinada tarea. Además, la capacidad para adaptarse a las necesidades de dicha tarea suele tener también ciertos límites.

Por oposición, un colectivo de individuos heterogéneo puede adaptarse de múltiples formas a la misma tarea, por ejemplo, reduciendo, aumentando o cambiando el grupo de individuos asignados a la misma. Para optimizar esta adaptación se requiere un cierto tipo de coordinación; esta necesidad puede extenderse también al desarrollo de la tarea.

Cuanto más heterogéneo y numeroso es el grupo, más posibilidades tiene de adaptarse a una determinada tarea pero también incrementa la complejidad de la distribución del trabajo.

Objetivos

Objetivo general

El objetivo general es la mejora en la comprensión del funcionamiento (incluyendo sus ventajas e inconvenientes) de las técnicas y arquitecturas de coordinación para avanzar en la dirección de un sistema más eficiente e integrado que permita su uso en múltiples entornos con colecciones heterogéneas de robots y con múltiples objetivos a conseguir.

Objetivos específicos

- Identificación de las principales técnicas de coordinación, arquitecturas y cualquier sistema multipropósito que permita el desarrollo de sistemas multi-robot.

- Caracterización de las distintas técnicas usando los atributos que permitan clasificar las mismas.
- Localización y análisis de las metodologías de evaluación, métricas, etc. que permitan valorar las arquitecturas identificadas y cualquiera que pudiera surgir.
- Diseño o adaptación de una metodología para la evaluación de las distintas técnicas.
- Definición de una nueva técnica que trate de mejorar los resultados de las técnicas existentes

Estructura del texto

La presente memoria comienza con una introducción a los sistemas multi-robot en donde se analizan las características o atributos de este tipo de sistemas, así como las taxonomías que las clasifican basándose en estos atributos. Es importante distinguir entre las arquitecturas y los sistemas multi-robot que implementan / usan estas.

Continuamos con un análisis del estado del arte en el campo de las técnicas de robótica cooperativa. En esta sección se repasarán las arquitecturas relacionadas existentes y sus características, concluyendo con un resumen del estado general de la tecnología en este campo. También se incluye una comparativa y un listado de las características no disponibles en las arquitecturas identificadas. En este capítulo, además, incorporamos un listado de las implementaciones disponibles de las técnicas descritas.

El capítulo siguiente analiza el estado del arte de los sistemas, metodologías, métricas, etc. de evaluación para sistemas multi-robot.

Se continúa con la descripción del sistema de evaluación a emplear para, por un lado, aplicarlo sobre las arquitecturas encontradas de forma que los resultados sean comparables y, por otro lado, poder evaluar cualquier arquitectura.

En el siguiente capítulo, se muestran los resultados de aplicar el sistema de evaluación a las distintas implementaciones de las arquitecturas seleccionadas.

En el Capítulo 6 se define la nueva arquitectura para sistemas multi-robot donde se especifica el diseño y se describen los elementos más importantes del mismo, haciendo hincapié en la comunicación de las distintas instancias distribuidas.

Tras la presentación de la nueva arquitectura se incluye un capítulo con los resultados de aplicar el sistema de evaluación a la nueva arquitectura.

El presente documento finaliza con un resumen de las conclusiones de esta tesis y una descripción de posibles desarrollos futuros.

Índice

Parte I

Estado del arte

Capítulo 1

Introducción a los sistemas multi-robot

El creciente número y variedad de robots que cada día usamos, tales como los robot-aspiradora, los vehículos autónomos o las mascotas artificiales, hace cada vez más deseable que dichos robots se coordinen entre sí para ser más eficientes en el desempeño de sus tareas y puedan acometer tareas más complejas. Esta coordinación puede ir desde reaccionar a la presencia / acciones de los otros miembros del colectivo (similar a la cooperación entre individuos de un enjambre de abejas) hasta la integración de los cooperantes en un mismo robot físicamente distribuido, pasando por compartir información sobre el estado de la tarea en la que se colabora. Estos niveles de coordinación, necesarios para permitir que varios robots colaboren en la resolución de una tarea determinada (por ejemplo: la limpieza de la casa por parte de un grupo heterogéneo de robots aspiradora / fregona), requieren el soporte de un conjunto de herramientas que se encargue de las funciones de comunicación, coordinación, etc. independientes de la tarea objetivo del colectivo.

El nivel de "coordinación" o de integración entre los distintos robots es solo uno de los atributos que caracterizan un sistema con múltiples robots (Multi Robot System o MRS), habiendo otros muchos tales como la centralización/descentralización del control, el sistema de toma de decisiones, etc. Describiremos todos estos atributos más adelante.

1.1 Sistemas multi-robot

Un sistema multi-robot fue definido por [1] como una versión de un Sistema de Múltiples Agentes (MAS, por sus siglas en inglés) donde cada agente representa

un robot. Por su parte, los sistemas de robótica colectiva son definidos también como la especialización de los sistemas colectivos [2], definiendo estos últimos como "consistentes en muchos individuos que interactúan, tales como moléculas, insectos, animales, robots, agentes de software e incluso humanos. A través de sus interacciones, los individuos de un sistema colectivo son capaces de lograr comportamientos, funcionalidades, estructuras y otras propiedades que estos individuos no pueden lograr solos".

Un caso especial, tanto por su inspiración como por su desarrollo, es el de la Robótica Enjambre (Swarm Robotics), definida por [3] como "robots autónomos con capacidades locales de detección y comunicación, que carecen de control centralizado o acceso a información global, situados en un entorno posiblemente desconocido para ejecutar una acción colectiva".

Como podemos ver en estas definiciones, existe una dualidad entre los individuos/robots y el sistema resultante. Esto lleva a dos visiones o niveles diferenciados:

- **Nivel microscópico**: el de los robots es decir, su comportamiento como individuos. A este nivel las propiedades o comportamientos resultantes del colectivo no son visibles.
- **Nivel macroscópico:** el del sistema o colectivo, siendo aquí importantes los comportamientos del colectivo obviando los de los individuos que lo componen.

Estos niveles han sido ampliamente descritos en la literatura, ya sea como formas de análisis (por ejemplo en [3]) o una descripción de diferentes visiones del fenómeno de colectividad (por ejemplo en [2]) y son de gran importancia ya que permiten describir las dos caras de la dualidad existente en estos sistemas.

Otra forma para etiquetar los distintos campos de investigación dentro de los sistemas multi-robot es el espacio de evolución con cuerpos y cerebros como dimensiones ([5]).

Esta visión, tremendamente simplificada, clasifica los sistemas multi-robot de forma visual y con unas clases muy similares a las aproximaciones definidas por [2]:

- **Control centralizado** (**aproximación cooperativa** según [2]): sistemas donde se usan sensores y actuadores distribuidos. El sistema de control reside en un equipo o conjunto de equipos.
- **Sistemas de robótica distribuida** (**aproximación de red** según [2]): se caracteriza por compartir intensivamente datos de los sensores y capacidad computacional gracias a un gran ancho de banda de comunicaciones, propiciando que los agentes dispongan de conocimiento común que se usa en la toma de decisiones.
- **Robótica enjambre** (**aproximación de robótica enjambre** según [2]): Inspirada en los insectos sociales; su principal característica es la reducida comunicación explicita, lo que les impide disponer de conocimiento común y, por lo tanto, de una planificación basada en él.
- **[2]** propone también una **aproximación del mundo microscópico** basándose en la química artificial al considerarla interesante ya que la maquinaria química en sistemas biológicos puede ser vista como la unión de datos y procesador. Además esta aproximación es un gran ejemplo de auto-ensamblado y auto-organización.

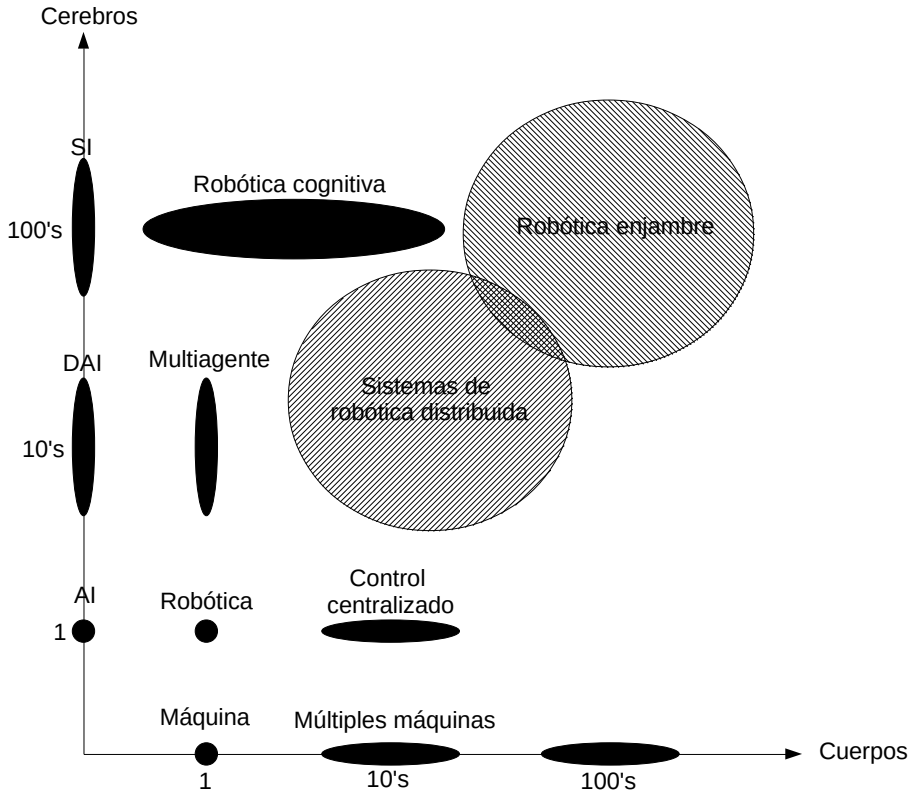


Ilustración 1: Evolución del número de cuerpos y cerebros, y su relación

1.2 Características de los sistemas multi-robot

Existen muchos atributos tanto del sistema en su conjunto como de cada individuo que son destacables, bien para clasificar una aproximación/solución concreta, bien para describir sus capacidades, etc. No todas las características son aplicables a todos los sistemas y muchas de ellas están íntimamente ligadas. En este apartado trataremos de describir aquellas comúnmente identificadas como relevantes y, en caso de que sea posible, se describirán las clases dentro de cada una.

1.2.1 Del sistema

En este punto describiremos aquellos atributos que caracterizan al colectivo o sistema en su conjunto.

1.2.1.1 Multifuncional / Monofuncional

Un sistema se denomina **multifuncional** (según [2]) cuando es capaz de ejercitar varias funciones, patrones o estructuras suficientemente diferenciadas. Por oposición, un sistema **monofuncional** solo es capaz de mostrar una única función, patrón o estructura.

1.2.1.2 Flexibilidad

La flexibilidad ha sido definida en [6] como "la capacidad de generar soluciones modulares para diferentes tareas". Esto se refiere a la posibilidad de que el colectivo realice varias tareas (sea multifuncional según la definición anterior).

Otros autores definen la flexibilidad como "la capacidad de hacer frente a un amplio espectro de diferentes entornos y tareas" [3].

1.2.1.3 Tamaño del colectivo

Define el número de individuos del colectivo, [4] distingue entre:

- Alone 1 robot. El colectivo más pequeño.
- Pair 2 robots. El grupo más simple.
- LIM Múltiples robots (n). El número n debe ser pequeño en relación con el tamaño o complejidad de la tarea o el entorno.

- INF n robots, con n >> 1. Existe, a efectos de la tarea, un número infinito de robots

1.2.1.4 Escalabilidad

El tamaño del colectivo, descrito anteriormente, está relacionado con la escalabilidad del sistema. La escalabilidad ha sido definida por [6] como la propiedad que "requiere que un sistema robótico pueda operar bajo una amplia gama de tamaños de grupo. Es decir, los mecanismos de coordinación que aseguran el funcionamiento del enjambre no deberían verse afectados por los cambios en el tamaño del grupo".

1.2.1.5 Robustez

[6] Define la robustez de un sistema robótico como la característica que le permite "continuar funcionando, aunque con un rendimiento inferior, a pesar de fallos en los individuos o perturbaciones en el ambiente". Y continúa evaluando los factores que contribuyen a esta propiedad:

- "**Redundancia en el sistema**; es decir, cualquier pérdida o mal funcionamiento de un individuo puede ser compensado por otro. Esto hace que los individuos no sean indispensables".
- "**Coordinación descentralizada**; es decir, destruir una determinada parte del sistema no impedirá el funcionamiento del sistema. La coordinación es una propiedad emergente de todo el sistema".
- "**La simplicidad de los individuos**; es decir, en comparación con un único sistema complejo que podría realizar la misma tarea, los individuos serían más simples, lo que los haría menos propensos a fallos".
- "**Multiplicidad de detección**; es decir, la detección distribuida por un gran número de individuos puede aumentar la relación señal / ruido total del sistema".

1.2.1.6 Coordinación

De la definición de [7] de coordinación podría resumirse esta como: un modo de cooperación en el cual los agentes (en nuestro caso, los robots miembros del colectivo) tienen en cuenta las acciones en ejecución por parte del resto de agentes durante la ejecución de las propias. Esto permite que el resultado conjunto sea coherente y (presumiblemente) más eficiente.

Algunos autores [1] proponen clasificar el nivel de coordinación en:

- **Fuerte**: Forma de coordinación que se apoya en un protocolo de coordinación
- **Débil**: Forma de coordinación que no se apoya en un protocolo de coordinación

[1] basándose en [8] y otros autores [2, 9] además clasifican la coordinación en:

- **Centralizada**: Organización que se caracteriza por poseer un agente (en nuestro caso un robot) que realiza de forma exclusiva la función de distribuir el trabajo para el resto de miembros del colectivo (este individuo suele denominarse líder). El resto de individuos del colectivo se limitan a seguir las instrucciones del líder.
- **Distribuida / Descentralizada**: La toma de decisión está distribuida o es completamente autónoma, no existiendo un agente líder.

Obviamente, existen múltiples niveles entre los dos extremos definidos.

En concreto, con respecto a la distribución de la toma de decisiones muchos autores ([11], [12]) definen categorías aparte de centralizada/descentralizada:

- **Híbrido**: en este enfoque se combina un control descentralizado a nivel local (lo que incrementa la robustez del sistema) y, gracias a una estructura jerárquica, el sistema se dota de la capacidad para distribuir y coordinar las acciones, tareas y objetivos en el colectivo.
- **Jerárquico**: la organización de los individuos se constituye con un superior jerárquico para cada individuo, a excepción del situado en la parte más alta de la pirámide, que es responsable de distribuir y coordinar las acciones entre sus subordinados, de forma similar a una cadena de mando militar clásica. Al igual que en esta última, puede haber varios niveles, con cada subordinado teniendo a su vez subordinados propios. La principal ventaja de esta organización es la alta escalabilidad al poder añadir niveles según sea necesario. Sin embargo, un fallo en un individuo no solo substrae sus capacidades al conjunto, sino que también roba las capacidades de sus subordinados.
- **Holárquico**: Los individuos influyen en las decisiones de forma que no hay una jerarquía vertical entre los individuos, sino entre los roles que estos

desempeñan. Además los individuos se organizan en círculos (grupos con responsabilidad sobre un círculo mayor, pero con capacidad de toma de decisiones acerca de la tarea encomendada).

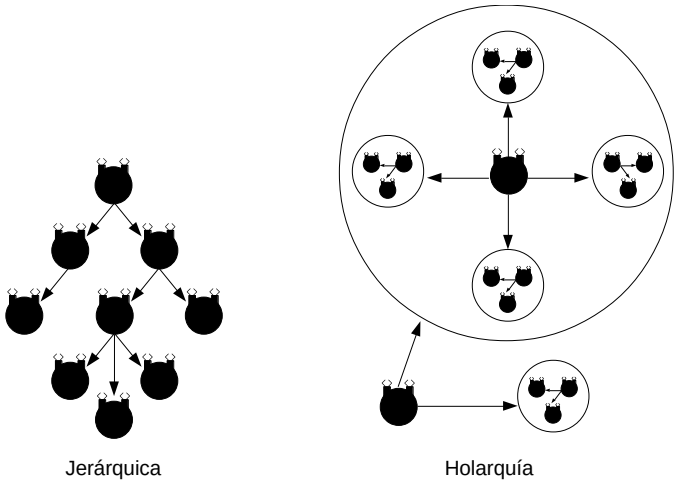


Ilustración 2: Representaciones de jerarquía y holarquía

- **Basado en coaliciones:** Una coalición es un grupo de individuos constituido con un objetivo y cuya existencia (la del grupo) cesa con la consecución del mismo (otras razones para que la coalición se deshaga son: deja de ser necesario, su objetivo se convierte en inalcanzable, etc.), por lo que su esperanza de vida se considera corta. Dentro de cada coalición los robots se comportan como iguales, aunque pueda haber un líder o intermediario que sirva de puente a la coalición como conjunto.
- **Equipo:** Un equipo está formado por un grupo de individuos que han llegado al acuerdo de trabajar conjuntamente para la consecución de un objetivo. Dentro del equipo los individuos, a diferencia de en una coalición, colaborarán de forma coordinada y se espera que maximicen el objetivo común, no el de cada individuo.

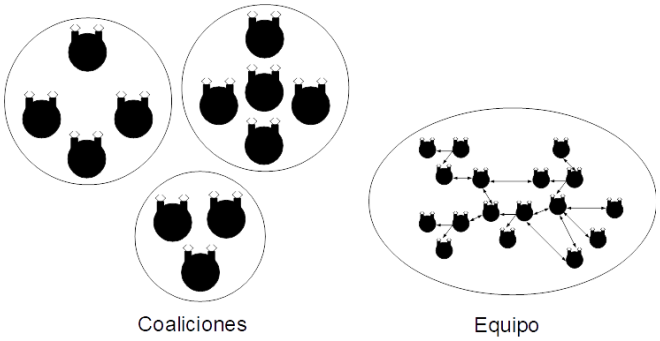


Ilustración 3: Representaciones de coaliciones y equipo

- **Basado en congregaciones:** Al igual que los equipos y las coaliciones, son grupos de individuos, pero a diferencia de ellos, no están orientados a un objetivo concreto, los grupos se forman con individuos con características parecidas o que se complementan entre sí.
- **Sociedades:** La mayor diferencia con los modelos anteriores radica en que las sociedades son grupos abiertos en los que los individuos pueden entrar y salir sin que la sociedad deje de existir. La otra gran diferencia es la imposición sobre los miembros de un conjunto de restricciones sobre su comportamiento, conocidas como leyes, normas o convenciones (estas restricciones son las que permiten el equilibrio entre los objetivos de cada individuo).

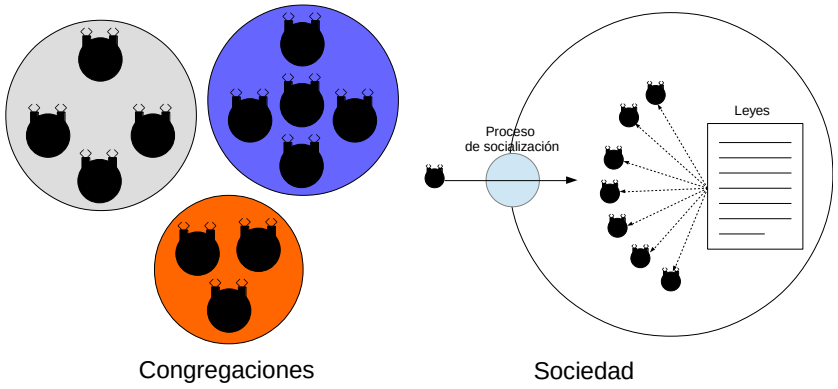


Ilustración 4: Representaciones de congregaciones y sociedad

- **Federaciones:** Grupos de agentes que han cedido cierta autonomía a un delegado (mediador o bróker). Este último actúa como intermediario entre el grupo y el mundo exterior (cada delegado de cada grupo es el responsable de interactuar con los otros delegados).
- **Mercados:** En un mercado, los individuos (compradores) hacen ofertas en régimen de competencia para obtener ciertos elementos (recursos, tareas…). Otros individuos (vendedores) pueden proporcionar estos elementos, seleccionado el comprador (el que proporcione la mejor oferta).

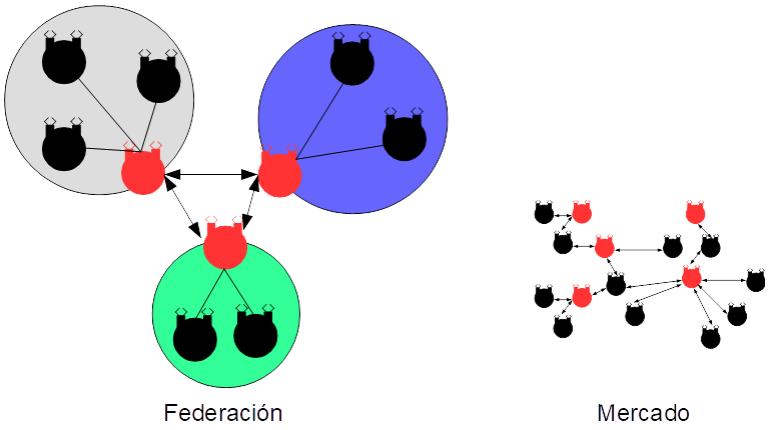


Ilustración 5: Representación de federaciones y mercado

- **Matricial:** Similar a las organizaciones jerárquicas en donde un individuo (o grupo) depende de varios "jefes" (la idea de la que viene el nombre, fija el número en 2), de esta forma, los individuos deben arbitrar los objetivos o instrucciones que les proporcionan estos otros.
- **Compuesto**: Es el resultado de componer varias de las estructuras vistas anteriormente.

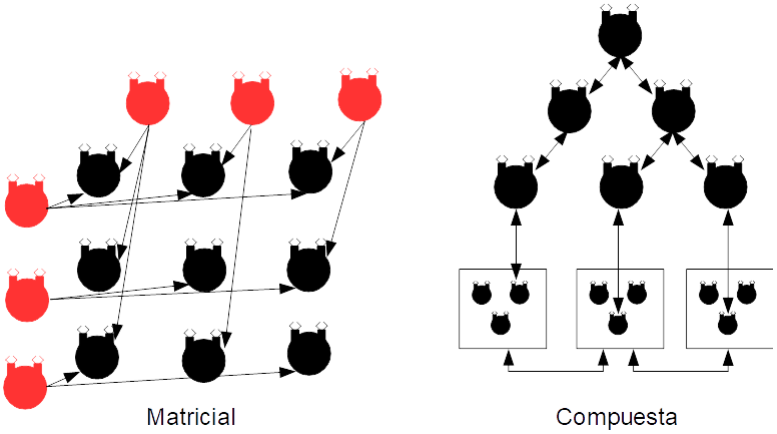


Ilustración 6: Representación de organización matricial y compuesta

1.2.1.7 Tipos de conocimiento colectivo

[2] Define el conocimiento colectivo como "un tipo especial de conocimiento relacionado con el grupo requerido para la coordinación y el control en un grupo de agentes". Y divide este conocimiento en 3 niveles:

- **Incompleto**: cuando un único individuo del grupo tiene conocimiento de un hecho.
- **Relacionado con el individuo:** cuando todos los individuos del grupo tiene conocimiento de un hecho.

- **Relacionado con el grupo:** cuando todos los individuos del grupo saben que todos los individuos tienen conocimiento de un hecho, es decir, los individuos disponen de información sobre la distribución del conocimiento en el grupo.
- **Eventual y continuo:** Relaciona los individuos con conocimiento de un determinado hecho con dicho hecho, permitiendo conocer aquellos individuos que nos pueden proporcionar una determinada información.

1.2.1.7 Cooperatividad

[10] [11] Dividen los MRS en 2: Sistemas de Colectivos Enjambre y Sistemas Intencionalmente Cooperativos. Sobre estos últimos los caracteriza según su cooperatividad en:

- **Fuertemente cooperativos**: los sistemas MRS de esta clase se caracterizan por una fuerte coordinación entre todos los individuos para conseguir de forma cooperativa un objetivo común. Esta forma de cooperación requiere una estructura y capacidades de comunicación y sincronización de gran calidad, lo que lleva a un mayor ancho de banda y un menor retardo (entre otros parámetros de calidad).
- **Débilmente cooperativos**: en los sistemas débilmente cooperativos los individuos se reparten las tareas y, cada individuo, ejecuta su parte de las mismas con una cierta independencia a nivel operativo. En este tipo de sistemas los requisitos mencionados para los sistemas fuertemente cooperativos se relajan enormemente.

1.2.1.8 Heterogeneidad

Casi todos los autores coinciden en clasificar los colectivos según su homogeneidad (también denominada diferenciación) en:

- **Idénticos**: colectivo cuyos miembros son idénticos, tanto a nivel hardware como a nivel software.
- **Homogéneo**: colectivo cuyos miembros son prácticamente iguales, tanto a nivel hardware como a nivel software. La diferencia hecha con la categoría anterior por [4] aporta un matiz interesante: los individuos de una misma casta de un enjambre de abejas no son necesariamente idénticos.

- **Heterogéneo**: colectivo cuyos miembros no son idénticos, existiendo diferencias en la configuración hardware y/o en el software.

1.2.1.9 Estructura de comunicación

[8] propone tres tipos (o niveles) de comunicación que define de la siguiente forma:

- **Interacción vía el entorno**: "El tipo de interacción más simple y limitado se produce cuando el entorno mismo es el medio de comunicación (en efecto, una memoria compartida), y no hay comunicación o interacción explícita entre los agentes. Esta modalidad también ha sido llamada 'cooperación sin comunicación' por algunos investigadores".
- **Interacción a través de detección**: "Correspondiente a las relaciones de alcance-de-la-mano en teoría de la organización [13], la interacción a través de la detección se refiere a las interacciones locales que ocurren entre los agentes como resultado de que los agentes se detecten entre sí, pero sin comunicación explícita. Este tipo de interacción requiere la capacidad de los agentes para distinguir entre otros agentes en el grupo y otros objetos en el entorno, lo que se denomina 'reconocimiento de parentesco' en algunas publicaciones [14]".
- **Interacción a través de las comunicaciones**: "La tercera forma de interacción implica la comunicación explícita con otros agentes, ya sea mediante mensajes intencionales dirigidos o transmitidos (es decir, el (los) destinatario (s) del mensaje puede ser conocido o desconocido). Debido a que las arquitecturas que permiten esta forma de comunicación son similares a las redes de comunicación, surgen muchos problemas comunes del campo de las redes, incluido el diseño de topologías de red y protocolos de comunicación".

Otros autores ([15]) dividen estos 3 niveles en dos: **comunicación implícita** (interacción vía al entorno e interacción a través de la detección) y **comunicación explícita** (interacción a través de las comunicaciones).

1.2.1.10 Auto-regeneración o auto-reparación

Según [16] se refiere a la habilidad de recuperar propiedades perdidas del colectivo, en concreto la formación o la diferenciación de robots dentro de la

propia formación (el patrón espacio temporal de roles). Según su interpretación, al mismo tiempo que se recuperan estas propiedades se recuperan las capacidades que puedan emanar de mantener una determinada formación.

Esta característica puede ser el resultado de una serie de comportamientos activos que aíslan al individuo o individuos cuyo comportamiento daña el objetivo del colectivo (como los descritos en [17]) o tratando de que los individuos del colectivo (todos) recuperen las propiedades perdidas como puede ser la formación [16].

1.2.1.11 Auto-reconfiguración

Muy relacionada con el auto-ensamblado, es la capacidad de un colectivo de cambiar la configuración (normalmente la forma) del auto-ensamblado.

1.2.1.12 Auto-replicación

Según [18] la auto-replicación es una de las características de los enjambres de robots modernos, en la que varios módulos se conectan entre sí para formar una copia exacta de un robot original determinado.

1.2.1.13 Canibalización

Característica que permite a uno o varios robots del colectivo usar piezas, elementos o materiales de otro como parte de su auto-modificación.

1.2.2 De los individuos

Describiremos aquí aquellos atributos que se aplican a uno o varios de los miembros del colectivo. En el caso de colectivos homogéneos, estas propiedades podrían entenderse como propiedades del colectivo.

1.2.2.1 Capacidad de procesamiento

Casi todos los robots empleados en MRSs tienen una capacidad computacional reducida y, en muchos casos, están conectados a plataformas computacionales fijas (no móviles) sobre las que descargan parte de las tareas más costosas (modelos cloud y fog).

1.2.2.2 Parámetros del sistema de comunicación

1.2.2.2.1 Rango de comunicación

Distancia (u otro factor equivalente, como puede ser una ventana temporal) que limita la posibilidad de establecer una comunicación entre dos individuos que en otras condiciones de distancia podrían establecerla.

Está muy relacionada con la estructura de comunicación anteriormente descrita. Puede dividirse (usando una clasificación similar a [4]) en:

- **Ninguno**: Los robots no pueden comunicarse directamente, pero sí pueden observar el comportamiento de otros o detectar su presencia. Es el caso de los tipos de estructura de la comunicación: interacción vía el entorno e interacción a través de detección, respectivamente.
- **Limitado**: La comunicación de los robots está limitada por la distancia (u otros factores equivalentes) y estos solo pueden comunicarse cuando su distancia es inferior al límite. Entre los otros factores tenemos: la propia topología de la posible red de comunicaciones, ventanas temporales,… Es importante tener en cuenta que muchos de estos factores, como por ejemplo el alcance, tienen impacto sobre el consumo de energía, y acostumbran ser regulables por decisiones de cada robot. Esta limitación suele dar lugar a comportamientos que permiten superarla, como la formación de una cadena.
- **Infinito**: Los robots pueden comunicarse sin ningún tipo de limitación práctica. Esto no quiere decir que, por ejemplo, puedan comunicarse a una distancia física infinita, sino que el alcance es suficientemente mayor que la distancia máxima alcanzable dentro del escenario donde los robots operan.

1.2.2.2.2 Ancho de banda de comunicación

Medida de los datos que se pueden transmitir por unidad de tiempo.

- **Ninguno**: Los robots no pueden comunicarse, ni siquiera observando el comportamiento de otros o detectando su presencia. En este caso los robots solo reaccionan al entorno (caso de estructura de la comunicación: interacción vía el entorno).

- **Limitado**: El ancho de banda está limitado por algún factor, como puede ser el propio método de comunicación, el medio usado, el consumo de energía, interferencias, etc. Existen varios subtipos de interés:
    - **Movimiento**: (propuesto por [4]) La comunicación se produce por el movimiento (similar a la danza de las abejas para indicar la dirección y la distancia a las flores, por ejemplo).
    - **Energía**: El incremento del ancho banda incrementa el consumo de energía, siguiendo esta relación una determinada función (que no suele ser lineal).
    - **Distancia**: Un incremento en la distancia reduce el ancho de banda. Al igual que en el caso anterior este incremento no tiene por qué ser lineal.
    - **Muy limitado**: El ancho de banda es muy bajo. Puede ser debido a un alto coste energético o de procesamiento o a una limitación del hardware de comunicaciones usado. Esto lleva a robots bastante independientes.
- **Infinito**: Sin limitaciones aplicables al ancho de banda.

1.2.2.2.3 Topología de comunicación

Vamos a considerar las siguientes topologías; sin embargo se usan otras topologías de redes clásicas (anillo, bus, estrella…) sobre todo en el entorno de los sistemas multi-robot de robots en red (por ejemplo en la industria):

- **Malla**: Cada robot puede comunicarse con un número limitado de sus vecinos. De querer comunicarse con otros robots fuera de este subconjunto deberá delegar la comunicación en uno de ellos (que a su vez puede tener que hacer lo mismo). Este tipo de topología requiere algoritmos de enrutado y, de igual forma que con las limitaciones en el rango, comportamientos que permiten reconfigurar la malla de forma que mejore las posibilidades de conseguir un determinado objetivo (moviendo los robots, por ejemplo). Algunos autores se refieren a esta topología como grafo.
- **Árbol**: Podría verse como un subtipo de la malla donde existe una comunicación jerárquica. Este tipo de topologías es común en sistemas con coordinación centralizada o jerárquica.

- **Totalmente conexa**: Cada individuo puede comunicarse con cualquier otro de forma directa y direccional (unicast) sin necesidad de pasar por otros individuos. Esto no evita que pueda enviar mensajes a un grupo determinado (multicast) o a todos los miembros del colectivo (broadcast).
- **Totalmente conexa sin direccionamiento**: Un caso particular de la topología anterior en donde solo se puede emitir mensajes en broadcast, por lo que solo un individuo puede estar transmitiendo en un momento determinado.

1.2.2.2 Conciencia de la existencia de otros (awareness)

La conciencia, tal como la define [1][19], "es la propiedad de un robot en el colectivo para poseer conocimiento de la existencia de otros miembros".

1.2.2.3 Auto-ensamblado

[2][20] [21] (Entre otros) Proponen el concepto de auto-ensamblado, entendido como: la unión física de varios individuos del colectivo para formar una estructura cuyas capacidades superen las limitaciones de los individuos por separado. Esto permite, por ejemplo, superar un desnivel en el terreno, transportar un objeto demasiado pesado, etc.

La característica de auto-ensamblado es, por un lado, una capacidad de uno o varios individuos y, por el otro, un comportamiento del colectivo. Entendido como el comportamiento o propiedad del colectivo es definido por [2] como “una clase específica de auto-organización mediante la cual un conjunto de unidades preexistentes forman de forma autónoma patrones espaciales o estructuras sin ninguna orientación externa”.

1.2.2.4 Auto-modificación, auto-adaptación o auto-reconfiguración

Esta característica permite a un robot concreto reconfigurar las partes que lo constituyen modificando las conexiones físicas entre ellas.

1.3 Comportamientos básicos

Una de las técnicas de diseño más usadas incluye la definición de comportamientos básicos [14] para la creación de otros más complejos que cumplan con un objetivo determinado. Estos comportamientos básicos pueden

combinarse de forma automatizada y no supervisada o como parte del proceso de diseño. Esta técnica de diseño se fundamenta en la creencia de que los comportamientos complejos se originan mediante interacciones locales basadas en reglas simples, siendo los comportamientos básicos la forma de estructurar dichas reglas.

En este punto presentaremos un listado de los comportamientos más empleados usando las categorías descritas en [3] para agruparlos.

1.3.1 Comportamientos de organización especial

En este subapartado se incluyen aquellos comportamientos que se ocupan mayormente de la distribución y organización en el espacio de los individuos del colectivo.

En este grupo también incluiremos lo que puede verse como una forma concreta de distribución en el espacio en la cual los individuos se conectan físicamente a otros (auto-ensamblado).

1.3.1.1 Agregación

La agregación es el comportamiento más sencillo de los que se ocupan de la organización y distribución espacial. Consiste en concentrar a todos los individuos en la misma región del espacio.

A pesar de su simplicidad, este comportamiento es muy interesante para poder alcanzar otros fines como son el poder mover un objeto entre todos los miembros del colectivo, etc.

1.3.1.2 Dispersión

Comportamiento contrario al anterior en donde [14] los individuos se dispersan desde un punto inicial.

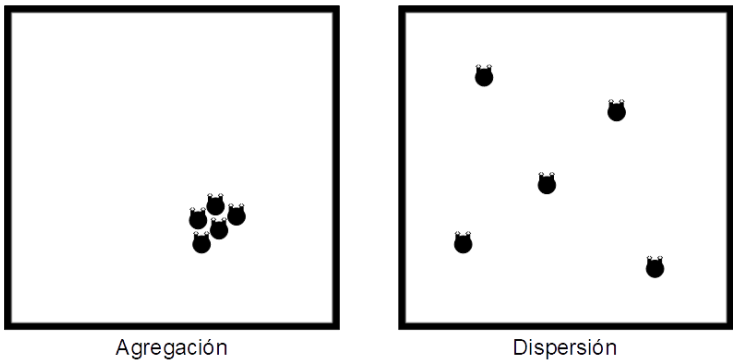


Ilustración 7: Ejemplos del resultado de los comportamientos de agregación y dispersión

1.3.1.3 Formación de un patrón

Consiste en distribuir en el espacio los individuos siguiendo un determinado patrón. Esta distribución puede incluir limitaciones de distancia entre distintos individuos, diferenciación en la distribución de individuos en un colectivo heterogéneo, etc.

Este comportamiento es útil en muchas circunstancias como es la exploración, la vigilancia de un área,...

1.3.1.4 Cadena

Este comportamiento supone un subconjunto dentro del anterior en el que el patrón es una cadena entre dos puntos.

Este comportamiento es útil en una gran variedad de escenarios como puede ser la exploración, mantener la conexión con un grupo de individuos deslazados más allá de la distancia de comunicación, etc.

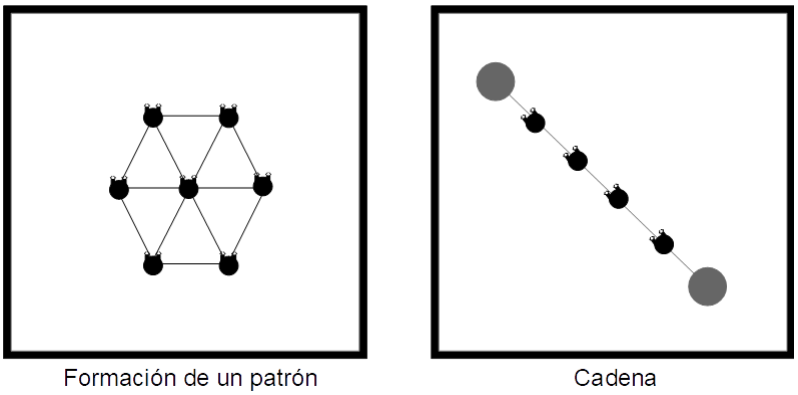


Ilustración 8: Ejemplo del resultado de los comportamientos de formación de un patrón y cadena

1.3.1.5 Auto-ensamblado y morfogénesis

El auto-ensamblado lo hemos descrito anteriormente como "la unión física de varios individuos del colectivo para formar una estructura". La morfogénesis es el comportamiento que lleva a un colectivo de robots a auto-ensamblarse siguiendo un patrón concreto de utilidad para una tarea concreta.

La utilidad de estos comportamientos reside en el cambio de las capacidades del grupo que se interconecta, permitiendo, por ejemplo, disponer de más energía para poder transmitir a más distancia, superar desniveles en el terreo que cada individuo no podría por sí mismo, transportar objetos de gran tamaño o peso, etc.

1.3.1.6 Agrupación y ensamblado de objetos

Este es un grupo extenso de comportamientos cuya línea común reside en el desplazamiento coordinado de varios objetos colocados en el escenario en donde se mueven los individuos del colectivo. Los comportamientos típicos que se incluyen son:

- **Agrupación de objetos**: situar todos los objetos de un determinado tipo en un lugar concreto, unos cerca de los otros. Este comportamiento puede incluir varios puntos de agrupación para distintos tipos de objetos.
- **Ensamblado de objetos**: situar uno o varios objetos en una posición concreta relativa a los demás objetos y forzar su conexión (esto puede ir desde simplemente colocarlos de una determina forma hasta actuar sobre los propios objetos para que permitan o realicen la unión).

La utilidad de este grupo de comportamientos puede ser un objetivo en sí misma (clasificar objetos o ensamblar un determinado elemento), o parte de un comportamiento más complejo (como podría ser el ensamblado de objetos para su posterior transporte).

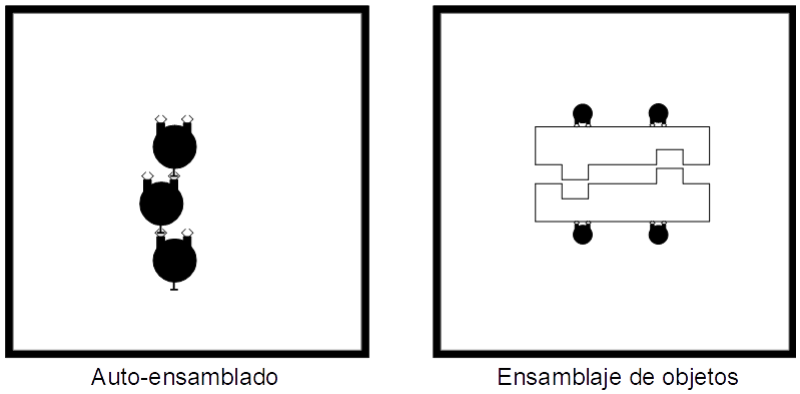


Ilustración 9: Ejemplo del resultado de los comportamientos de auto-ensamblado y ensamblaje/agrupación de objetos

1.3.1.7 Homing

Comportamiento relacionado con la capacidad de un individuo del colectivo para ir a una ubicación concreta designada previamente. Normalmente implica volver a un lugar después de viajar a una gran distancia de él.

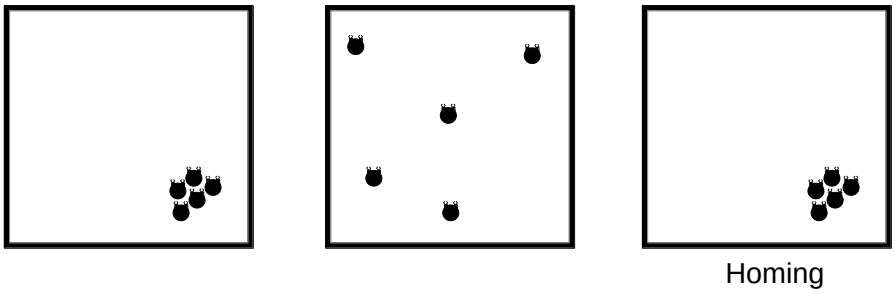


Ilustración 10: Ejemplo en el que los individuos se dispersan por el entorno tras lo que exhiben el comportamiento de homing

1.3.2 Navegación

En este grupo incluimos aquellos comportamientos relacionados con el desplazamiento coordinado de los individuos del colectivo.

1.3.2.1 Exploración colectiva

En este subgrupo se incluyen dos comportamientos: la cobertura de área y la navegación guiada por el colectivo. La combinación de estos dos comportamientos da lugar al comportamiento de exploración colectiva.

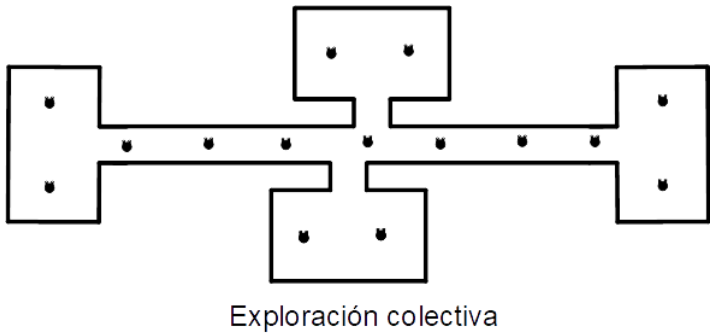


Ilustración 11: Ejemplo de comportamiento de exploración colectiva

1.3.2.1.1 Cobertura de un área

Este comportamiento tiene como objetivo distribuir los individuos del colectivo de manera que cubran una determinada zona del espacio formando un patrón. De esta forma los individuos estarán lo suficientemente cerca para comunicarse pero cubriendo el área designada.

Esta red de robots puede ser usada para controlar variables del entorno en dicha área o para guiar a otros individuos hacia un destino concreto.

No debe confundirse la cobertura de un área con la dispersión, ya que esta última no implica que los robots ocupen un determinado espacio.

1.3.2.1.2 Navegación guiada por el colectivo

[3] Llama al ejemplo del punto anterior de guiar otros individuos: navegación guiada por el colectivo.

1.3.2.2 Movimiento coordinado

Este comportamiento, que también se conoce como flocking (bandada de pájaros), tiene como objetivo el movimiento coordinado de los individuos manteniendo un determinado patrón muy similar al comportamiento de una bandada de pájaros o un banco de peces.

Este comportamiento permite, por ejemplo, que se mantengan las capacidades de movimiento de los individuos maximizando la maniobrabilidad del colectivo, evitando colisiones entre los individuos, protegiendo al colectivo de la exposición a un agente externo dañino, etc.

Un caso concreto del movimiento coordinado es el seguimiento, en donde un robot sigue los movimientos de otro.

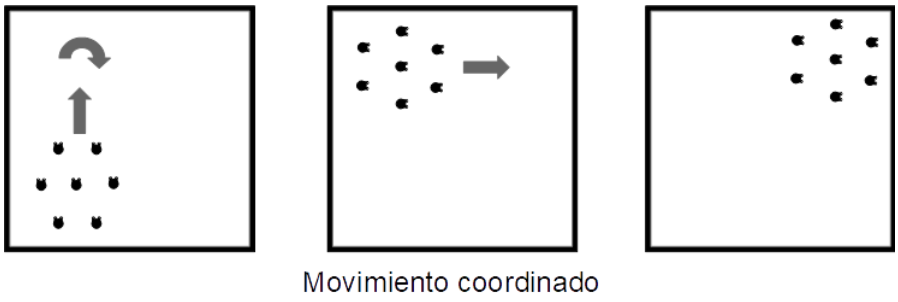


Ilustración 12: Resultado del comportamiento de movimiento coordinado

1.3.2.3 Transporte colectivo

Este comportamiento define un patrón de cooperación para el transporte de un objeto entre varios miembros del colectivo. Este comportamiento también se llama recuperación colectiva de una presa.

La utilidad de este comportamiento es evidente, pero tiene particular interés cuando el objeto que se va a transportar es un miembro del colectivo incapaz de moverse por el tipo de terreno, dañado o sin energía, o simplemente carente de actuadores que le permitan desplazarse.

Este comportamiento puede entenderse que incluye el de manipulación de un objeto (que se explica a continuación) añadiendo la necesidad de planificar la navegación (el desplazar al grupo) manteniendo la formación (comportamiento anteriormente descrito).

1.3.2.4 Manipulación de un objeto

La manipulación de un objeto es un comportamiento que puede ser visto como parte del anterior, donde un grupo de robots pertenecientes al colectivo cooperan para cambiar la posición, orientación, etc. de un objeto.

Para conseguir los objetivos de este comportamiento por parte de un colectivo de robots sin actuadores capaces de agarrar el objeto, [9] define 3 técnicas o tipos de comportamiento:

- **Agarrar**: En este tipo de manipulación, el colectivo completo de robots se colocan de forma que todos puedan "agarrar" el objeto y son la única fuerza que actúa sobre dicho objeto. En este caso "agarrar" el objeto significa que los robots empujan el objeto desde direcciones opuestas de manera que quede sujeto entre ellos o mantienen una formación suficientemente cerrada como para que el objeto no se pueda desplazar dentro de la misma.

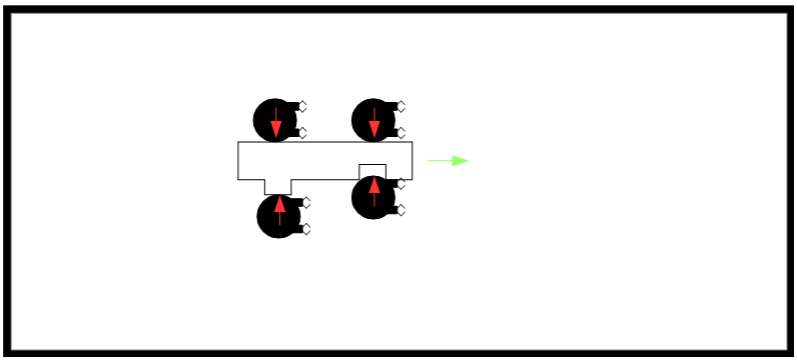


Ilustración 13: Ejemplo de comportamiento de agarre, donde las flechas rojas representan la fuerza ejercida sobre el objeto y la verde, la dirección del movimiento

- **Empujar**: Implica que los robots no hacen fuerza para "agarrar" ni su formación impide el movimiento del objeto dentro de la misma. Para que esta técnica funcione se requieren otras fuerzas, como la fricción que frene el objeto una vez que se deja de empujar. Esta técnica tiene la ventaja sobre la anterior de poder ser aplicada a casi cualquier objeto e incluso a varios objetos a la vez.

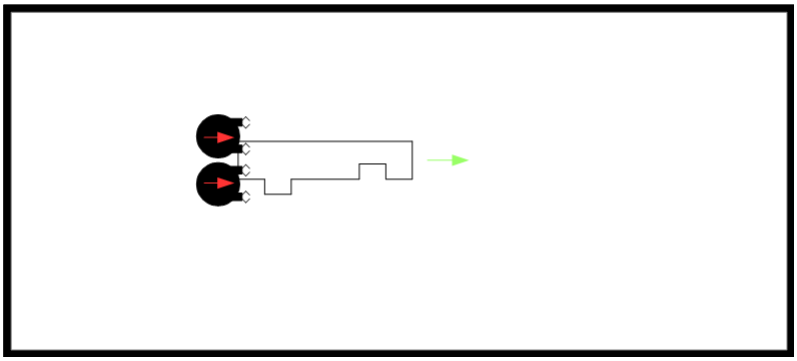


Ilustración 14: Ejemplo de comportamiento de empujar, donde las flechas rojas representan la fuerza ejercida sobre el objeto y la verde, la dirección del movimiento

- **Encajonar**: Los robots mantienen una formación alrededor de todo el objeto, de manera similar a cuando lo agarran pero permitiendo un cierto grado de movimiento dentro de dicha formación. Esta técnica reduce los requisitos, tanto computacionales como de sensores, comparada con la de agarre.

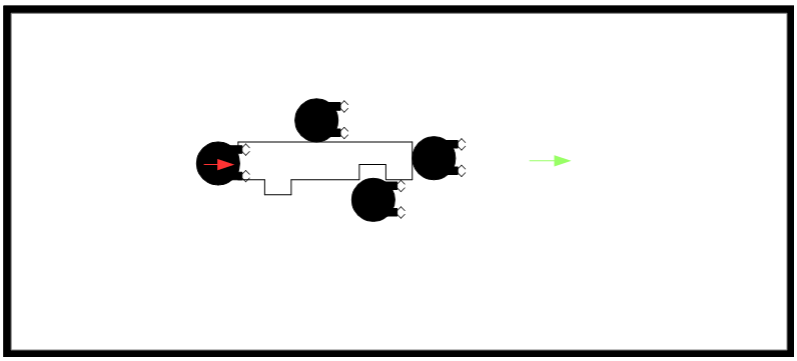


Ilustración 15: Ejemplo de comportamiento de encajonar, donde las flechas rojas representan la fuerza ejercida sobre el objeto y la verde, la dirección del movimiento

1.3.3 Toma de decisiones colectiva

Comportamientos relacionados con la influencia entre los individuos del colectivo en la toma de decisiones.

1.3.3.1 Consenso

Permite a los individuos del colectivo alcanzar un consenso sobre la opción a tomar, supuestamente, maximizando el rendimiento del colectivo o beneficiándolo de alguna forma. Obviamente, la opción que maximiza el rendimiento cambia con el tiempo y no todos los individuos perciben el entorno de igual modo, esto dificulta alcanzar un consenso.

1.3.3.2 Asignación de tareas

Comportamiento colectivo por el cual se asignan tareas a los distintos individuos del colectivo. De igual forma que en el comportamiento anterior, el objetivo es maximizar el rendimiento dando flexibilidad a los individuos para seleccionar la tarea a ejecutar.

1.3.4 Otros comportamientos colectivos

En esta categoría podemos ver comportamientos de interés pero que no caben en ninguna de las categorías anteriores.

1.3.4.1 Detección/tolerancia de fallos

Conjunto de comportamientos que permiten la detección y tratamiento de fallos a nivel del colectivo. Como ya hemos visto anteriormente, estos comportamientos se pueden basar en dos técnicas:

- **Ignorar**: gracias a algún tipo de diferenciación, los individuos sanos dejan de percibir a los dañados como parte del colectivo y tratan de resolver la tarea sin tenerlos en cuenta. Esta técnica permite a un conjunto de robots "dañados" darse cuenta de que no forman parte del colectivo original e, incluso, formar su propio colectivo.
- **Aislar**: a los individuos dañados, por ejemplo, rodeándolos de individuos "sanos" que les impiden interactuar con el resto del colectivo de alguna forma (por ejemplo, cortando sus comunicaciones o inmovilizándolos).

1.3.4.2 Regulación del tamaño de grupo

Este comportamiento incluye aquellas capacidades que permitan seleccionar los individuos del colectivo de forma que su número sea el deseado. Existen muchas motivaciones para este tipo de regulación, como pueden ser las limitaciones de recursos o espacio.

1.3.4.3 Interacción con humanos

Comportamientos relacionados con la capacidad de un operador humano de modificar el comportamiento del colectivo.

1.3.4.4 Resolución de conflictos de recursos

[8] Describe un tipo de comportamiento que podríamos relacionar con aquellos relativos a alcanzar un consenso pero que tiene cierta entidad propia, en cuanto puede producirse de múltiples formas, incluso como parte de otros comportamientos. Este comportamiento es la resolución de conflictos de recursos, definidos estos como la selección de qué individuo tiene acceso a un recurso individual e indivisible en un determinado momento.

Un recurso es cualquier elemento que sea necesario para realizar una tarea determinada: espacio, energía, ventana de comunicaciones, un objeto,…

Este tipo de conflictos pueden ser más comunes dependiendo del tipo de coordinación entre los individuos del colectivo, haciendo más o menos necesaria la existencia de comportamientos que los resuelvan.

1.4 Comportamiento emergente

De la interacción entre los distintos individuos del colectivo emerge un comportamiento atribuible al colectivo (nivel macroscópico) y no necesariamente al resultado de la suma de comportamientos individuales o reglas impuestas durante el diseño del sistema. En [14] se define este tipo de comportamiento como aquel caracterizado por la siguiente propiedad: "se manifiesta por estados globales o patrones extendidos en el tiempo que no están programados explícitamente pero que resultan de interacciones locales entre los componentes de un sistema".

Este tipo de comportamiento ha sido observado y ampliamente descrito en la literatura científica para seres biológicos, habiendo ejemplos en las bandadas de pájaros, bancos de peces, etc. Estos ejemplos (entre otros) de comportamientos emergentes entre seres biológicos se ha tratado de analizar y reproducir en sistemas robóticos (por ejemplo: [22], [23]).

[14] afirma que "en un sistema suficientemente complejo se podrían observar comportamientos emergentes" (esto plantea un problema para su estudio, ya que para calificar un determinado patrón como un comportamiento emergente se requiere de un observador que lo identifique como tal). En [24] el autor coincide en esta afirmación y define la emergencia como: "surgimiento de estructuras novedosas y coherentes, patrones y propiedades durante el proceso de auto-organización en sistemas complejos".

En [2] se distingue entre dos niveles:

- "La **emergencia débil** describe nuevas propiedades que surgen en los sistemas. como resultado de las interacciones entre agentes colectivos. La emergencia, en este caso, es parte del modelo, que describe el comportamiento de un sistema"
- **"Emergencia fuerte** es un tipo de emergencia en el que la propiedad emergente es irreducible a sus agentes individuales"

La generación de comportamientos emergentes suele hacerse mediante la aplicación de un conjunto local de reglas sobre los individuos del colectivo siendo las distintas formas de crear y aplicar estas reglas las que dan lugar a los distintos

métodos de diseño (de arriba abajo, de abajo arriba, evolutivo inspirado en la naturaleza).

1.4.1 De arriba abajo

Es una de las aproximaciones de diseño tradicional (discutida ampliamente en [25]) basada en la suposición de que los sistemas se pueden descomponer en: observación, estimación del estado y control.

El proceso de diseño se inicia al nivel más alto de abstracción asumiendo la posibilidad de acceso a los recursos por parte de los elementos de niveles inferiores (control centralizado).

El sistema se modela a partir de una especificación que solo tiene en cuenta el estado global del colectivo. Cada individuo del colectivo será capaz de estimar o recuperar para su uso los recursos de otros individuos, todo ello con unos ciertos requisitos de tiempo y precisión dependientes del recurso y el objetivo.

Con estas condiciones satisfechas, las propiedades de la especificación antes mencionada de un sistema con control centralizado se mantienen, con una cierta pérdida, en sistemas con control distribuido.

1.4.2 De abajo arriba

En otra de las metodologías clásicas ([25]), de forma opuesta a la técnica descrita en el punto anterior, el proceso de diseño se realiza desde el mínimo nivel de abstracción, considerando siempre que no se dispone de conocimiento del estado global del sistema y confiando en que los comportamientos colectivos emerjan a partir de las interacciones de los individuos (o entre los individuos y el entorno).

Con la aplicación de esta metodología se diseñan los comportamientos y reacciones a los estímulos de cada uno de los individuos del colectivo. Dichos estímulos incluyen elementos del entorno, acciones, presencia o ausencia de otros individuos, estado interno del individuo…

El sistema resultante suele ser altamente escalable y requerir un bajo nivel de comunicaciones (o incluso ninguna).

1.4.3 Métodos automáticos

1.4.3.1 Robótica evolutiva

En esta metodología se emplean técnicas de computación evolutiva para sintetizar un conjunto de reglas capaces de producir un determinado comportamiento objetivo. Para evaluar las soluciones (conjuntos de reglas) durante la búsqueda de dicho objetivo se usa una función de fitness.

1.4.3.2 Aprendizaje reforzado

Aplicación de las técnicas de aprendizaje automático en donde cada robot recibe una recompensa por acciones que van en la dirección del objetivo marcado y una penalización por aquellas que lo alejan.

En este caso el objetivo es del colectivo y las acciones deben ponerse en el contexto del mismo para poder evaluarse. Esto genera un problema para poder asignar una recompensa para cada individuo, aunque hay trabajos para dividir una recompensa entre los individuos del colectivo ([26], [27] y [28]).

1.4.4 Inspirado por la naturaleza

Durante el proceso de diseño, ideas observadas en la naturaleza tratan de implementarse en el sistema. Estas ideas van desde comportamientos de insectos (abejas y hormigas sobre todo) y otros animales, hasta las propiedades físicas que les proporcionan una cierta ventaja.

La imitación de determinados comportamientos de insectos sociales es usada especialmente en el campo de robótica enjambre y ha dado lugar a varias propiedades como el auto-ensamblado, la comunicación vía el entorno/detección, etc.

1.5 Taxonomía

Existen múltiples intentos para la definición de una taxonomía para sistemas de robots colectivos, muchas de ellas se limitan a cubrir una pequeña parte de las características anteriormente descritas agrupándolas en un determinado número de ejes.

En este punto vamos a analizar las distintas características implicadas en algunas de las taxonomías más conocidas.

Una de las primeras taxonomías de sistemas multi-robot [8], clasifica los sistemas usando lo que denomina 5 ejes de investigación:

- La arquitectura del grupo: Infraestructura sobre la que se sustenta el comportamiento colectivo.
- Conflictos de recursos: Al teorizar sobre la necesidad de que varios individuos compartan recursos declara la necesidad de contar con un método de resolución de estos.
- Origen de la cooperación: Como se obtiene dicha cooperación, es decir, qué lo motiva y cómo se articula.
- Aprendizaje: Capacidades de adaptación y flexibilidad.
- Problemas geométricos: Todo lo relativo a poner una serie de individuos en un espacio con 2 o 3 dimensiones (búsqueda de camino, formaciones…).

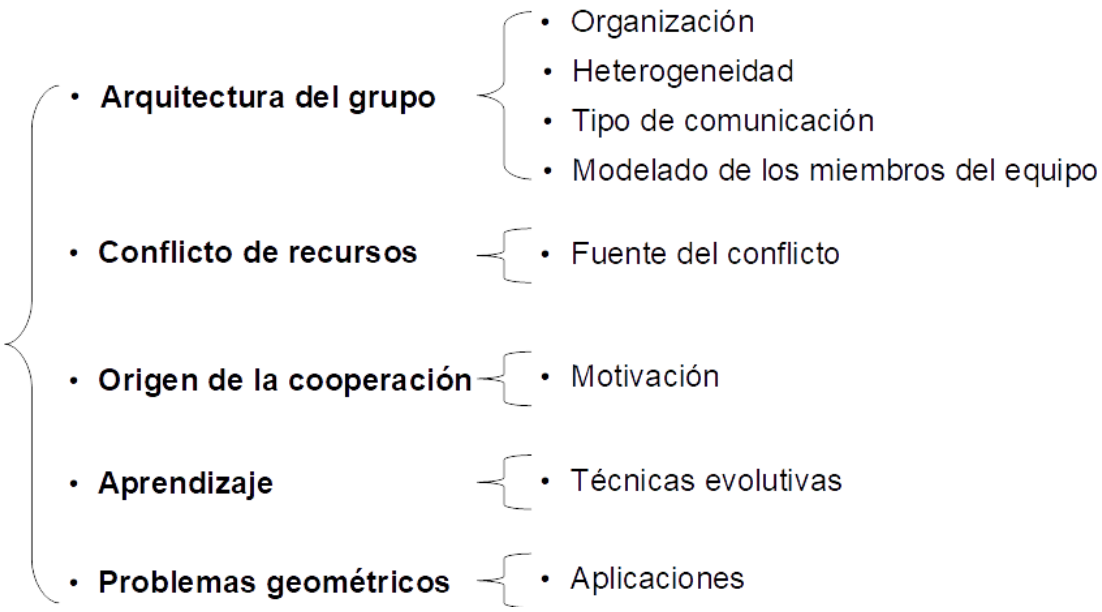


Ilustración 16: Representación gráfica de la taxonomía propuesta por Y. UNY CAO y otros en 1997

Una taxonomía específica para un objetivo (recolección de forraje) concreto ha sido propuesta en [29] definiendo 8 "cantidades":

- Tamaño de grupo: Un individuo vs múltiples individuos.
- Heterogeneidad: Homogéneo vs heterogéneo.

- Sumideros: Un sumidero vs múltiples sumideros.
- Límites del entorno: Campo abierto vs espacio cerrado.
- Fuentes: Una fuente vs múltiples fuentes.
- Número de objetos: Un solo tipo vs varios tipos.
- Posición inicial de los objetos: situados en áreas concretas vs colocados aleatoriamente.
- Comunicación: disponible vs no disponible.

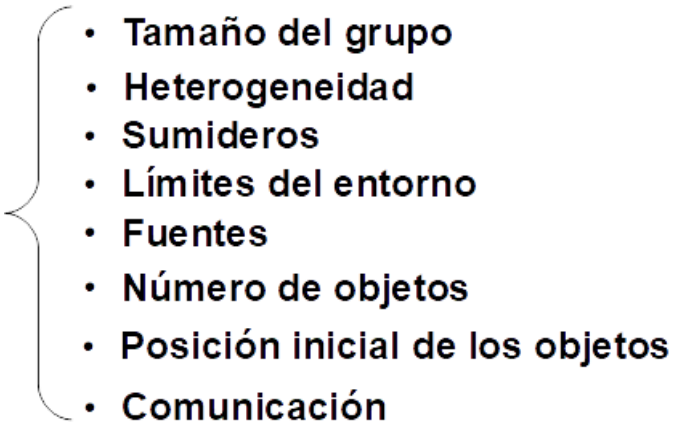


Ilustración 17: Representación gráfica de la taxonomía propuesta por Esben H. Østergaard en 2001

En [30] los autores adoptan la visión de que la tarea y su función de evaluación son la misma cosa y tienen en consideración tres ejes:

- Rendimiento: Métricas del rendimiento en la ejecución de la tarea.
- Entorno de la tarea: Características del entorno en donde se acomete la tarea y otras restricciones externas.
- Plataforma robótica: Propiedades y capacidades de los robots que conforman el colectivo.

Esta taxonomía recuerda en la descripción de sus clases a [4], pero tiene una aproximación diferente

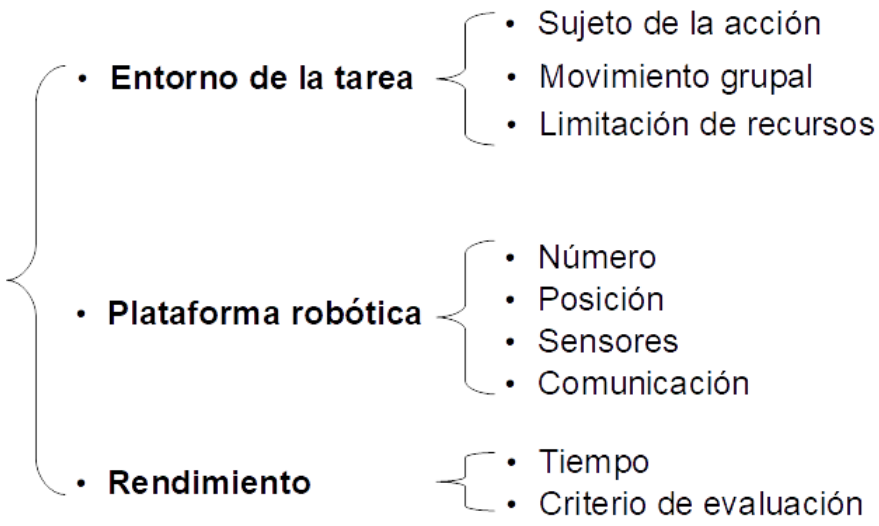


Ilustración 18: Representación gráfica de la taxonomía propuesta por Tucker Balch en 2002

En [31] los autores hacen una revisión de la taxonomía planteada en [4] en la que definen 7 dimensiones:

- Tamaño del colectivo: entendido como el número de miembros del mismo clasificado en 1, 2, límite e infinito.
- Rango de las comunicaciones: clasificado en: ninguno, cercano e infinito.
- Topología de las comunicaciones: broadcast, direccionable, árbol o grafo.
- Ancho de banda de las comunicaciones: infinito, movimiento (coste similar al del movimiento), bajo o cero.
- Reconfigurabilidad de la organización del colectivo: estática, coordinada o dinámica.
- Capacidad de proceso: unidad de suma no lineal, autómata de estados finitos, autómata de pila o máquina de Turing.
- Diferenciación: idénticos, homogéneos o heterogéneos.

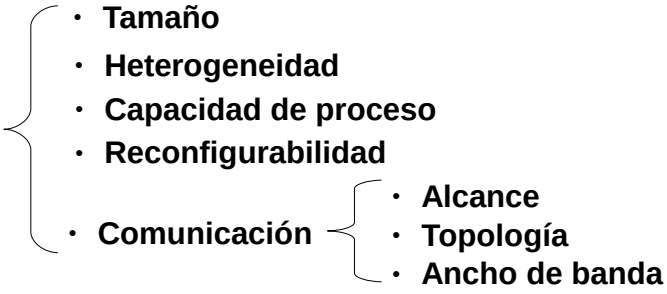


Ilustración 19: Representación gráfica de la propuesta de Gregory Dudek (y otros) en 2002

Por último veremos la propuesta de [2] que define la taxonomía con un problema en 3D donde los ejes son:

- Aplicación: Objetivo y entorno de aplicación.
- Tipo: Inspiración o técnica de diseño.
- Principios subyacentes: Características del sistema o individuos.

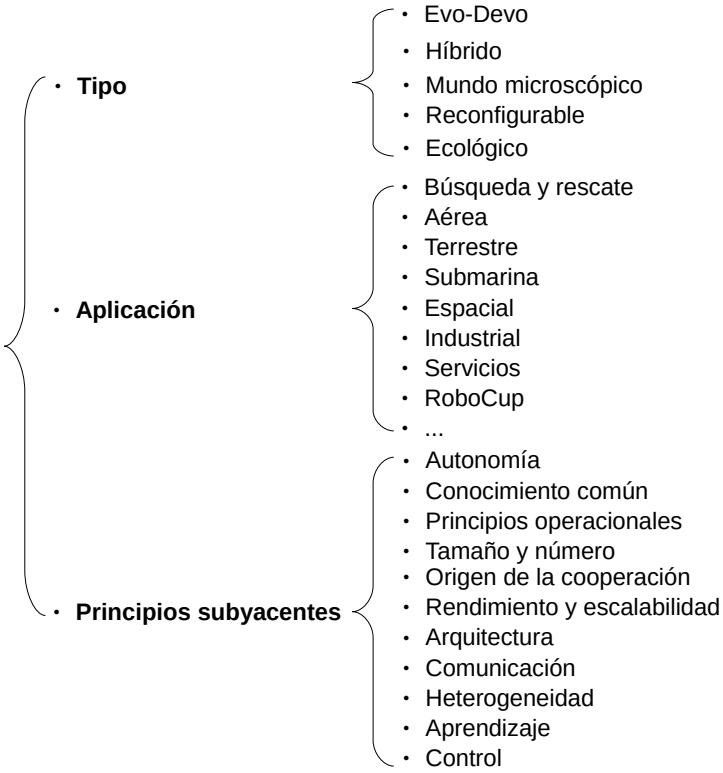


Ilustración 20: Esquema de la taxonomía propuesta por Serge Kernbach en 2013

Capítulo 2

Estado del arte en arquitecturas

En [8], basándose en la definición de arquitectura para un sistema de información ("la parte del sistema que permanece sin cambios a menos que un agente externo lo cambie" [32]), se define como aquello que “proporciona la infraestructura sobre la cual se implementan los comportamientos colectivos y determina las capacidades y limitaciones del sistema”.

Debemos distinguir entre arquitecturas diseñadas para ejecutarse sobre un conjunto concreto de individuos (sobre un hardware específico) —y, por lo tanto, difíciles de aplicar en otros sistemas— y aquellas diseñadas incluyendo los requisitos funcionales y técnicos necesarios para ser portables.

2.1 CEBOT

El Celular Robotic System [33] (o CEBOT) está inspirado en la idea de célula de los sistemas biológicos, en donde una o más células se unen para formar tejidos que a su vez se integran en un individuo. De igual forma las *células* del CEBOT se unen para realizar una misión.

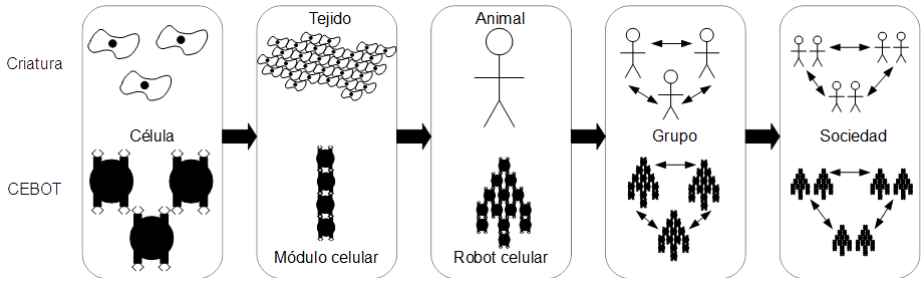


Ilustración 21: Analogía entre criatura y CEBOT (tal como la presentan sus autores)

Esta idea se materializa en una arquitectura descentralizada y jerárquica que se ejecuta sobre un colectivo homogéneo de robots. Una de las características destacadas de este sistema es el auto-ensamblado, posible gracias a la capacidad de reconfigurar su estructura física (conexión entre las *células*).

Cada *célula* de este sistema es un robot autónomo con cierta capacidad para desplazarse y conectarse a otros con un mecanismo simple. Dicho mecanismo no es universal y no está pensado para unirse a cualquier otro robot.

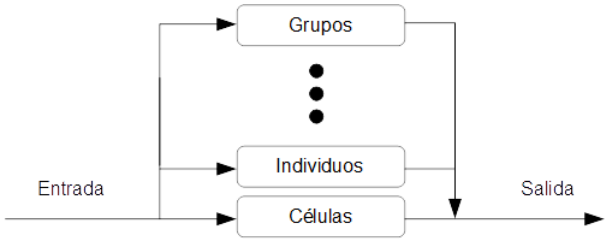


Ilustración 22: Entrada y salida del sistema con comportamientos desde células a grupos

El diseño de la arquitectura software [34] de este sistema se basa en comportamientos estructurados en capas, cada uno de los cuales se corresponde con un comportamiento básico o tarea. Dada la capacidad de auto-ensamblado del sistema, la arquitectura debe ser capaz de modificar su comportamiento según la configuración actual del colectivo (los comportamientos del *grupo* deben ser una extensión de los comportamientos de la *célula*). De esta forma, los autores proponen una arquitectura de procesos paralelos (que se corresponden con comportamientos capaces de desempeñar una tarea). Cada uno de estos procesos produce un vector de comportamiento que debe combinarse con la salida del resto de procesos.

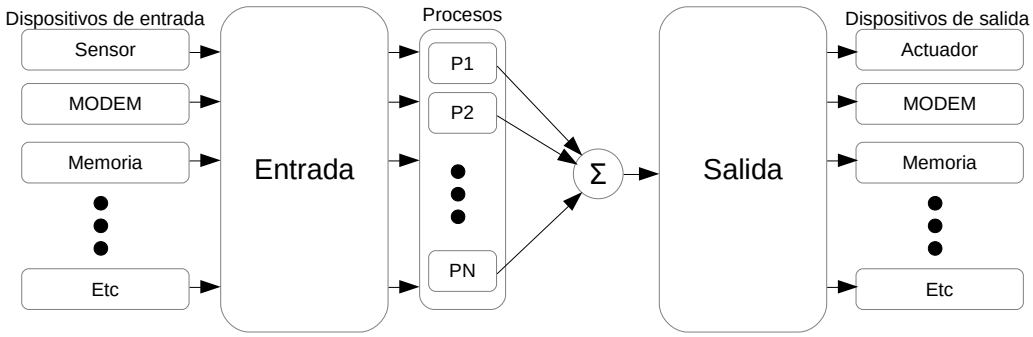


Ilustración 23: Diagrama conceptual de la arquitectura jerárquica propuesta para CEBOT

De esta forma los dispositivos de entrada (entre los que se hallan los sistemas de comunicaciones) disparan que los procesos (que se ejecutan paralelamente y de forma asíncrona) produzcan sus vectores de comportamientos. Estos últimos se componen de forma que emerja el comportamiento óptimo para la situación actual.

Dada la necesidad de seleccionar un comportamiento sobre otros, los autores proponen dos matrices: la matriz de prioridad y la matriz de interés. Gracias a estas matrices se puede adaptar el resultado de la combinación de los vectores de comportamientos mediante aprendizaje.

Dependiendo del número de procesos las relaciones entre procesos se hacen más numerosas. Para reducir estas, los autores organizan los procesos en capas de forma jerárquica. Así definen *unidades* como elementos que sintetizan los resultados de procesos o unidades de capas inferiores, correspondiéndose una unidad de la capa de abajo de todo con un proceso.

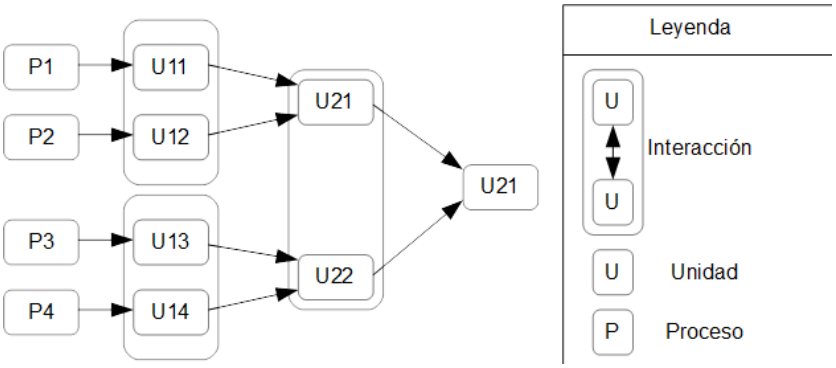


Ilustración 24: Arquitectura jerárquica con 3 capas

2.2 ACTRESS

ACTor based Robot and Equipment Synthetic System (ACTRESS) [35], se basa en el formalismo de Universal Modular ACTOR [36] (en el cual, un único objeto, el *actor* y en el intercambio de mensajes entre *actores,* representa las estructuras de datos y de control).

Bajo la definición de ACTRESS, los componentes robóticos se denominan *robotors*: un *actor* robótico capaz de manipular datos y, posiblemente, también

objetos físicos. El intercambio de mensajes antes mencionado se realiza usando un sistema de comunicaciones. Un *robotor* debe, además, cumplir dos condiciones:

- Ser capaz de tomar decisiones y ejecutar acciones de forma autónoma.
- Ser capaz de comunicarse con otros componentes.

ACTRESS plantea múltiples tipos de *robotors* según sus capacidades. Podemos ver algunos ejemplos en la siguiente imagen.

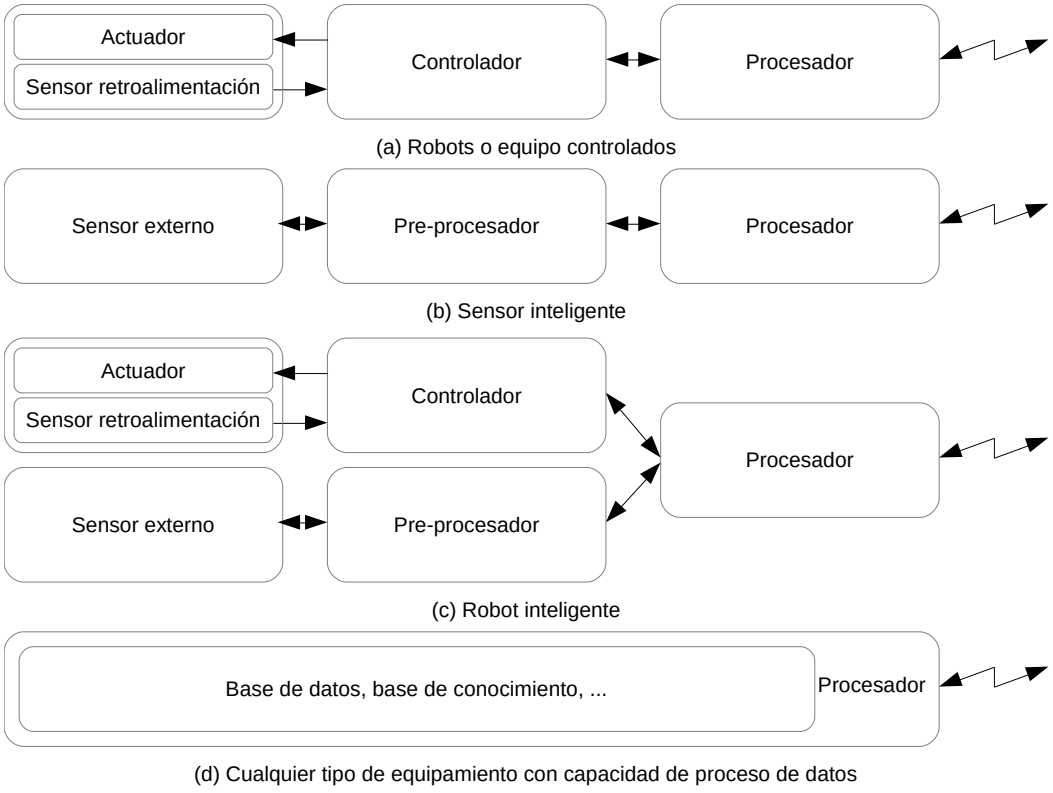


Ilustración 25: Ejemplos de robotors

No hay restricciones en el uso de *robotors* heterogéneos, de hecho, la arquitectura está diseñada específicamente para grupos de este tipo.

2.2.1 Comunicaciones

Tal como vimos anteriormente, esta arquitectura requiere que los individuos del colectivo posean capacidad de comunicación entre ellos (haciendo gran hincapié y describiendo en detalle el sistema usado). El uso de esta capacidad de comunicación, sin embargo, estaría condicionado por la naturaleza de la tarea: aquellas que no requieran coordinarse con otros *robotors* se desarrollarán con

comunicaciones esporádicas, aquellas que sí requieran la coordinación de varios *robotors* deberán compartir señales de control de forma frecuente.

ACTRESS propone dos tipos de protocolos:

- **Protocolo de comunicaciones**: (concepto equivalente a un protocolo de red clásico) permite el establecimiento y cierre de conexión, la corrección de errores, enrutado, etc.
- **Protocolo de mensajes**: define la sintaxis de la información intercambiada. Su nombre se deriva del concepto de intercambio de mensajes entre *actores*.

El protocolo de comunicaciones se basa en el Modelo de Referencia OSI (Open System Interconnection) y su división en capas, mientras que el protocolo de mensajes posee una estructuración en *niveles*, no siempre jerárquicos, y diferentes a las capas. Estos niveles son:

- **Nivel de control**: En este nivel se transmiten datos que, mayormente, dependen del hardware, tales como señales para los actuadores, datos de los sensores, etc.
- **Nivel físico**: La información referente al mundo físico (posición, velocidad, etc.) se transmite en este nivel.
- **Nivel procedural**: En este nivel se transmiten los procedimientos. Cada *robotor* debe proveer de un intérprete que procesará las solicitudes y comandos.
- **Nivel del conocimiento**: El conocimiento disponible en cada *robotor* se envía en este nivel.
- **Nivel conceptual**: En este nivel se transmiten conceptos, incluyendo las tareas e intenciones.

Algunos de los niveles requieren reducir el overhead resultante del uso de las capas superiores del Modelo de Referencia OSI por lo que los autores proponen el uso directo de capas inferiores a la de presentación dependiendo del nivel de protocolo de mensajes. Los niveles y capas propuestos pueden verse en la siguiente imagen:

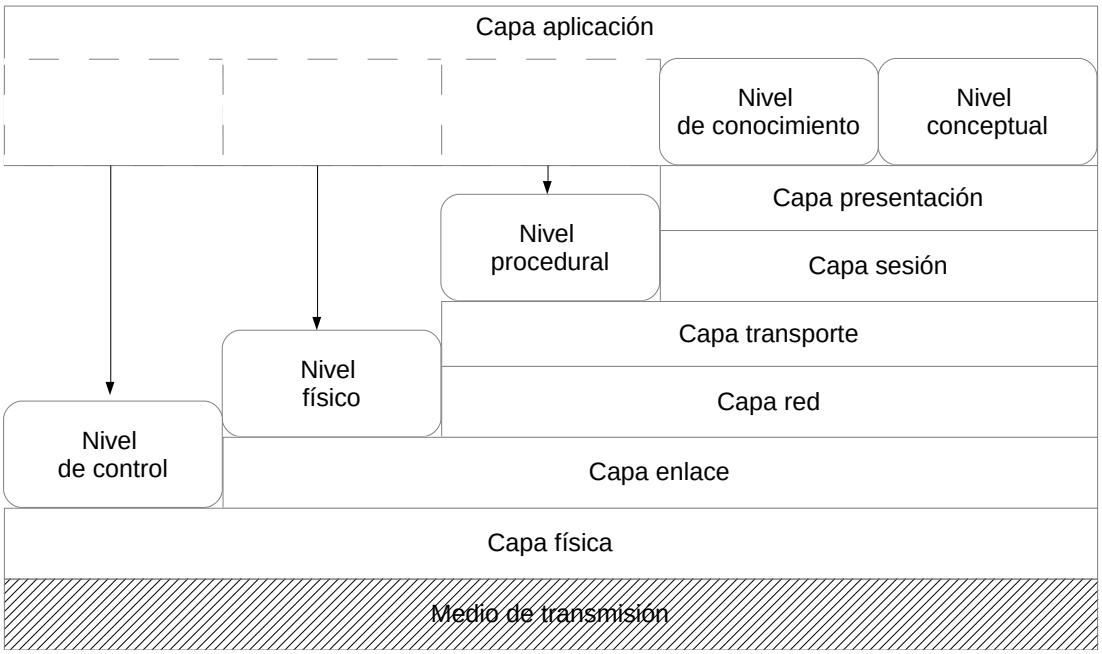


Ilustración 26: Uso de capas de los distintos niveles del protocolo de mensajes

2.3 GOFER

El proyecto GOFER [37] está orientado al control de un colectivo de robots, en el rango de las decenas, en interiores. El proyecto considera que las indicaciones que debe recibir el sistema han de incluir el objetivo (tarea) a realizar, en vez del cómo o el conjunto concreto de pasos a seguir. A partir de esta tarea / objetivo, el sistema debería ser capaz de decidir qué acciones son necesarias y cuándo deben ejecutarse; todo ello optimizando el consumo de recursos y cumpliendo con los requisitos de la tarea.

Para cumplir con estos objetivos, GOFER propone las siguientes actividades que planificarán y asignarán las tareas: planificación de tareas, asignación de tareas, planificación de movimientos y toma de decisiones en tiempo de ejecución.

Bajo esta arquitectura los robots pueden adquirir objetivos de dos formas: generándolos ellos mismos a partir de las necesidades del momento y recibiendo las solicitudes de otros agentes, ya sean otros robots o humanos (asignación de tareas).

2.3.1 Planificación de tareas

En esta arquitectura, la búsqueda de acciones para cumplir un objetivo se denomina "planificación de tareas", y permite hacer dos tipos de planificación: *estructuras* (que contienen variables) e *instancias* (aplicaciones de las estructuras donde las variables adquieren un valor).

Los autores definen una *estructura* como un agregado de una jerarquía de acciones (grupo de acciones donde la situada en la parte superior representa el conjunto del proceso y el resto pueden ser secuencias de acciones o acciones paralelizables) y de restricciones (definen los valores admisibles de las variables).

El proceso de planificación incluye tres pasos:

1. Determinar las *estructuras* de planificación para la consecución del/los objetivo/s (recuperadas desde una librería o construidas usando un algoritmo)
2. Generar la *instancia* de la *estructura*
3. Toma de decisiones finales (por ejemplo: seleccionar qué objetivo o estructura aplicar)

2.3.2 Asignación de tareas

GOFER usa un sistema que se clasifica como "parcialmente centralizado" para la asignación de tareas a robots disponibles. Sus autores han bautizado este sistema como CTPS (Central Task Planning and Scheduling System).

El CTPS recibe instrucciones (pedidos u objetivos) por un lado y, por otro, los robots disponibles hacen propuestas a los planes generados por el CTPS a partir de los objetivos. El CTPS toma una decisión entre las propuestas presentadas (lo cual puede incluir esperar por otras propuestas) asignando la tarea a la mejor solución.

2.3.3 Planificación de movimientos

GOFER asigna la responsabilidad de la planificación de movimientos a cada robot. Cada robot recibe una red de caminos, generada previamente, y usa A* para hallar el mejor camino a su destino.

2.3.4 Ejecución

Para ejecutar las acciones del plan, GOFER usa redes Petri jerárquicas generando una red (S T) a partir de la *instancia* del plan. Esta red está compuesta de estados, transición entre acciones y transición jerárquica.

2.4 Matarić

Matarić propone en su tesis [14] una arquitectura basada en *comportamientos básicos* demostrando su funcionamiento con un grupo de robots, "The Nerd Herd". El autor define un comportamiento como "una ley de control que agrupa un conjunto de restricciones para lograr y mantener una meta" y postula la teoría de que para cada dominio de aplicación existe un conjunto de comportamientos que se pueden denominar básicos, ya que "son necesarios para generar otros comportamientos, además de ser un conjunto mínimo que el agente necesita para alcanzar su repertorio de objetivos". Esta definición implica (al indicar que el conjunto debe ser mínimo) que un comportamiento del conjunto no es reducible a un grupo de otros comportamientos de dicho conjunto.

El autor define también el comportamiento conjunto, colectivo o grupal como el patrón temporal (definido por un observador) de las interacciones entre varios agentes.

2.4.1 Combinación de comportamientos

Dado que los comportamientos básicos deben combinarse (arbitrarse) de alguna forma para conseguir los comportamientos grupales deseados, el autor define dos tipos de combinación: directa y temporal.

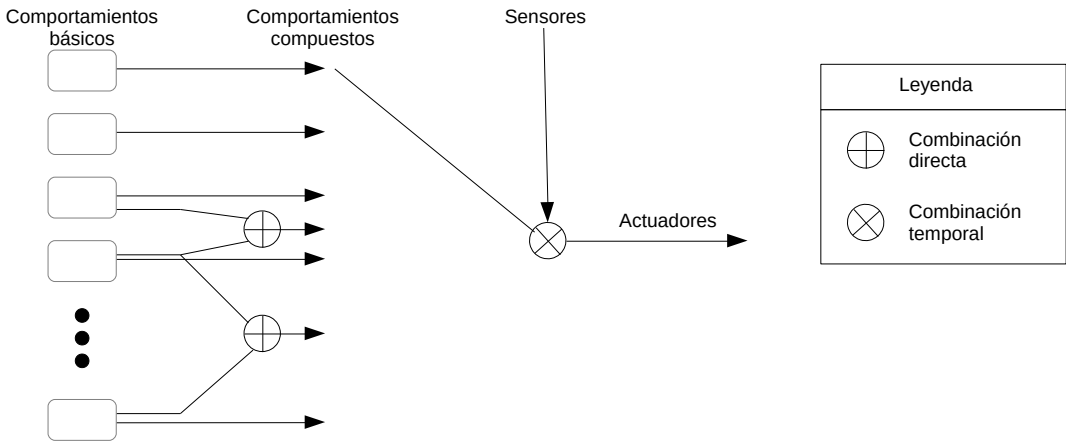


Ilustración 27: Arquitectura de control propuesta por Matarić

Con una combinación directa (basada en la suma), la salida de varios comportamientos contribuye a la salida de la combinación; mientras que una combinación temporal (basada en la activación) permite que la secuencia de salida sea coherente.

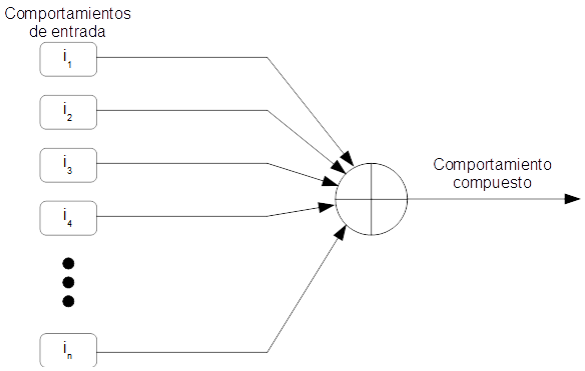


Ilustración 28: Combinación directa

La combinación directa se define como una función que toma como entrada los vectores de salida producidos por uno o varios comportamientos básicos y produce una salida. El autor ejemplifica este tipo de composición con los comportamientos espaciales que son fácilmente expresables con coordenadas de posición, vectores de velocidad, etc. En dicho ejemplo, la combinación directa se realiza usando una suma ponderada. El resultado de este tipo de combinación genera comportamientos compuestos tales como movimiento en formación, rodear o pastorear.

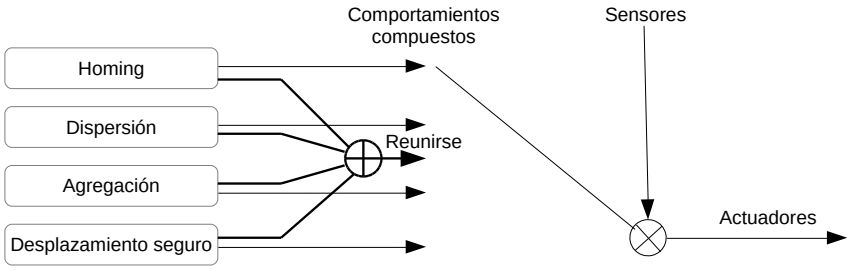


Ilustración 29: Ejemplo de comportamiento espacial compuesto con operador de combinación

Por su parte, el operador de combinación temporal genera una secuencia de comportamientos básicos. La forma más simple para el funcionamiento de este operador es la de un selector basado en un determinado patrón de los datos de los sensores.

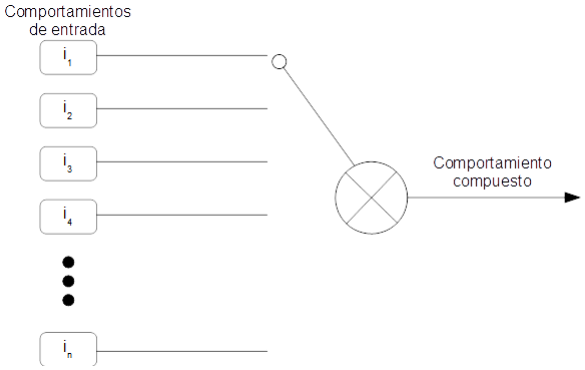


Ilustración 30: Combinación temporal

De la misma forma que se puede generar el comportamiento de rebaño con el operador combinación, podría hacerse con el operador temporal.

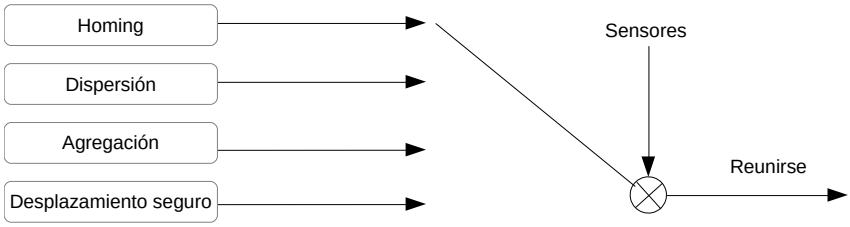


Ilustración 31: Ejemplo de comportamiento espacial compuesto con operador temporal

2.5 ALLIANCE

Sus autores definen ALLIANCE [38] como “una arquitectura de software que facilita el control cooperativo y la tolerancia a fallos de equipos” (de número de individuos pequeño a medio) “de robots móviles heterogéneos que realizan misiones compuestas de subtareas débilmente acopladas que pueden tener dependencias de orden”.

2.5.1 Supuestos

El diseño de esta arquitectura se basa en una serie de supuestos que pasamos a describir:

1. La probabilidad de que los robots del equipo detecten los efectos de sus propias acciones es > 0, en otras palabras: dicha detección es posible.
2. Un robot r_i tiene una probabilidad de detectar las acciones de otros robots con capacidades redundantes a r_i > 0. La detección de dichas acciones se puede producir por cualquier método, incluidos aquellos que impliquen la comunicación explicita (desde los robots que ejecutan las acciones u otro).
3.
    a. Los robots no se mienten entre sí.
    b. Los robots del colectivo no son adversarios (al menos intencionalmente) entre sí.
4. El sistema de comunicación puede no estar disponible.
5. Los sensores y actuadores no se consideran perfectos.
6. La probabilidad de fallo de cualquier subsistema de cualquiera de los robots es > 0.
7. Un fallo en un individuo puede implicar que este no pueda comunicar dicho fallo a sus compañeros de equipo.
8. No hay un sistema centralizado que almacene el conocimiento común.

2.5.2 Descripción general

Dados los supuestos antes mencionados y la orientación a colectivos heterogéneos con tolerancia a fallos, el sistema se diseñó como una arquitectura

basada en comportamientos, distribuida y en la que cada individuo puede decidir las acciones a realizar basándose en los datos que posee.

Cada individuo recibe como entradas:

- Requisitos de la misión (objetivo del colectivo).
- Acciones de otros individuos (aquellas detectadas).
- Condiciones ambientales actuales (aquellas detectadas).
- Estado interno.

A partir de estos datos, el sistema en cada robot (que posee un conjunto de funciones de alto nivel) selecciona un conjunto de acciones usando "motivaciones" modeladas matemáticamente. Siguiendo el framework basado en comportamientos, los comportamientos que permiten realizar una tarea, reciben como entrada los datos de los sensores y poseen un cierto control sobre los actuadores.

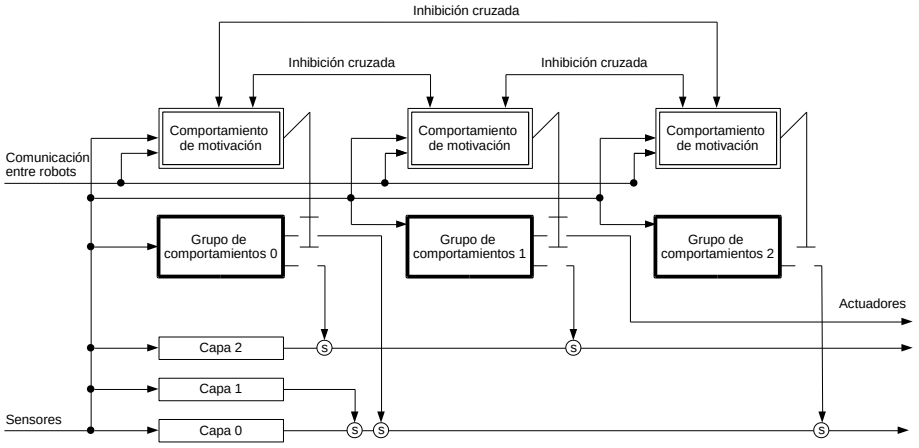


Ilustración 32: Implementación de ALLIANCE en cada robot

Las *competencias* (comportamientos de bajo nivel) representan comportamientos básicos tales como desplazarse o evitar obstáculos mientras que los comportamientos de alto nivel se corresponden con objetivos tales como generar el mapa del entorno. La salida de un comportamiento de bajo nivel puede ser inhibida por otros en capas superiores.

Dentro del mismo nivel de cada capa de *competencia* es posible que los comportamientos se inhiban entre ellos para seleccionar el deseado.

2.5.3 Comportamientos de motivación

Las motivaciones dotan a los robots de capacidades de respuesta a situaciones no esperadas, lo que permite a un individuo continuar aplicando un grupo de comportamientos hasta que estos no son productivos en la consecución de los objetivos.

Esta aproximación contrasta con otras basadas en la descomposición inicial de la tarea que cumple con el objetivo y la asignación de las sub-tareas de forma que se optimice el resultado. Comparativamente, al permitir la selección de acciones de forma individual en cada momento combinada con la detección de fallos en otros miembros del colectivo, se mejora la probabilidad de recuperarse ante un fallo.

Los comportamientos de motivación hacen uso de las motivaciones mencionadas e implementan la capacidad de selección de acciones adaptada en esta arquitectura. Estos comportamientos reciben como entradas la información recibida de otros robots (comunicaciones), los datos de los sensores, señales de inhibición de otros comportamientos y motivaciones internas; la salida define el nivel de activación del *grupo de comportamientos* asociado que se activa cuando alcanza un determinado valor.

ALLIANCE define dos tipos de motivaciones internas: impaciencia y aquiescencia.

El cálculo de la salida combina matemáticamente el estado anterior, la impaciencia, los datos de los sensores y la aquiescencia:

$$m_{ij}(0) = 0$$

$$m_{ij}(t) = [m_{ij}(t-1) + \text{impatience}_{ij}(t)] \times \text{sensory feedback}_{ij}(t) \times \text{activity suppression}_{ij}(t) \times \text{impatience reset}_{ij}(t) \times \text{acquiescence}_{ij}(t)$$

2.5.3.1 Impaciencia

La impaciencia está diseñada para permitir que un robot reaccione cuando otros no son capaces de realizar una determinada tarea.

La impaciencia ($\text{impatience}_{ij}(t)$) se ve afectada por 3 parámetros: el tiempo que el robot r_i permite que comunicaciones provenientes de otro afecten el

comportamiento ij; los otros dos son los niveles de impaciencia aplicables cuando hay un robot realizando la tarea relacionada con el comportamiento y cuando no hay ningún robot realizando dicha tarea.

Existe otro valor relacionado con la impaciencia (impatience $reset_{ij}(t)$) que permite que el nivel de impaciencia se reduzca a 0 desde el momento en que el robot recibe una comunicación de otro individuo afirmando que está realizando la tarea.

2.5.3.2 Aquiescencia

La aquiescencia es la motivación que permite a un robot inhibirse cuando considera que no es capaz de ejecutar una tarea de forma adecuada.

Dos parámetros (en concreto, dos timeouts) definen el nivel de aquiescencia: el tiempo máximo en que un robot tratará de ejecutar un determinado comportamiento antes de permitir que otro robot trate de ejecutarlo y el tiempo máximo en el que el robot ejecutará el comportamiento antes de cambiar a ejecutar otro grupo de comportamientos.

En el caso del primer parámetro, este timeout debe coincidir con otro robot tratando de realizar la misma tarea.

2.5.4 L-ALLIANCE

Existen múltiples variantes y adaptaciones de ALLIANCE. La más interesante es L-ALLIANCE [39].

Los parámetros definidos para cada motivación interna se consideran fijos a lo largo del tiempo. Idealmente estos parámetros deberían adaptarse según la experiencia obtenida por cada robot. Para ello L-ALLIANCE añade un nuevo componente (el *monitor*) para cada comportamiento de motivación.

El *monitor* "observa" la eficiencia con la que el robot desempeña un determinado comportamiento y adapta los parámetros definidos en ALLIANCE según dichas observaciones.

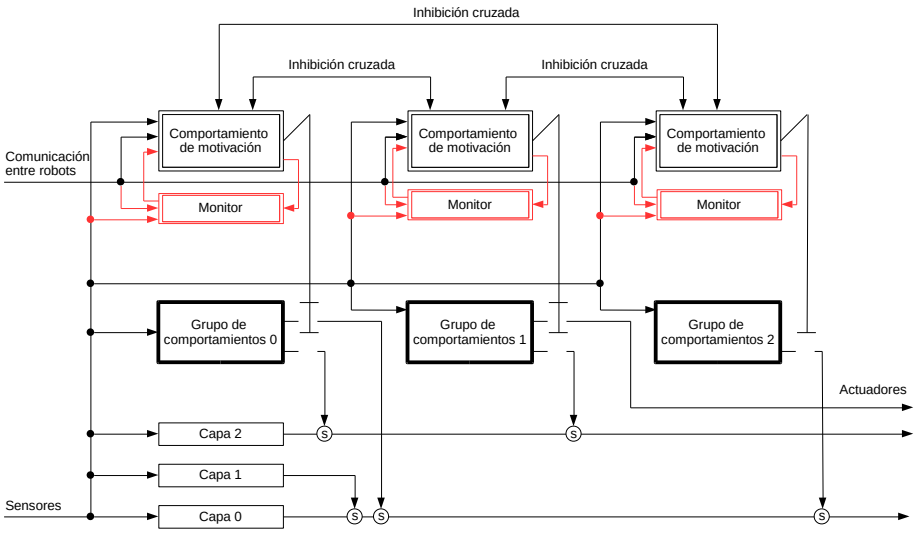


Ilustración 33: L-Alliance

2.6 ARTIS

Sus autores definen [40, 41] ARTIS (Architecture for Real-Time Intelligence Systems) como "una arquitectura para implementar sistemas de control inteligente cuya ejecución se desarrolla en tiempo real" e incluyen un modelo formal para sistemas multiagente [42].

La orientación de esta arquitectura es mucho más amplia que los colectivos de robots, permitiendo la creación de múltiples tipos de aplicaciones en donde el resultado de una operación es tan importante como el tiempo en el que se obtiene. En este texto nos centraremos en los aspectos relevantes para su aplicación en colectivos de robots.

ARTIS implementa el modelo de blackboard [43] bajo el cual deben existir 3 componentes: **Fuentes de conocimiento** (cada una de las cuales proporciona una parte del conocimiento necesario para resolver el problema); la estructura de datos del **blackboard** (base de datos global que contiene el estado del sistema y sobre la cual las fuentes de conocimiento hacen sus modificaciones); y **control** (permite a las fuentes de conocimiento reaccionar a los cambios en el blackboard según sea necesario).

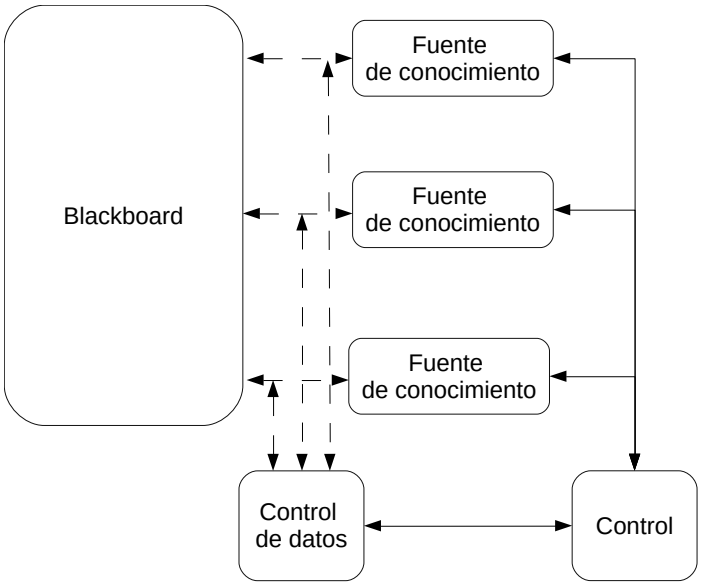


Ilustración 34: Modelo blackboard

2.6.1 Modelos de tareas

ARTIS define dos modelos de las tareas para facilitar el diseño e implementación: un modelo de alto nivel más cercano a la funcionalidad del sistema y otro modelo de bajo nivel que no está abstraído de las técnicas usadas para el cumplimiento de las restricciones de tiempo real.

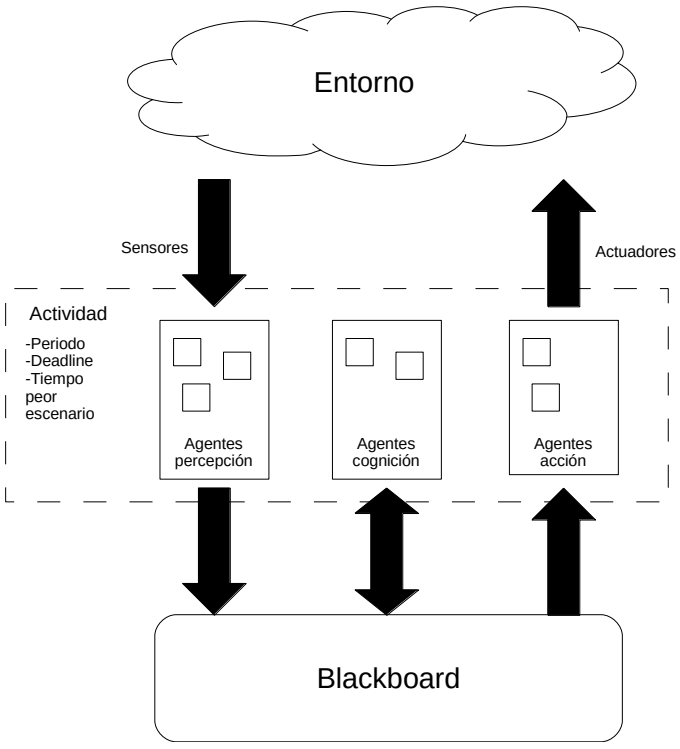


Ilustración 35: Modelo de alto nivel ARTIS

En el modelo de alto nivel se define un agente [41] como "la entidad inteligente mínima capaz de resolver un problema concreto". En este modelo, cada agente posee varios niveles con un orden particular y cada nivel es capaz de producir un resultado. Para obtener el resultado final se ejecutan las tareas pertenecientes a cada nivel hasta que se alcanza el tiempo máximo o se hayan ejecutado todos, siendo el resultado final el último obtenido. Dentro de este mismo modelo, una actividad se define como una entidad abstracta capaz de resolver una parte del problema global del sistema y consta de tres subconjuntos de agentes: percepción (son capaces de leer los datos de los sensores), cognición (son capaces de procesar los datos y tomar decisiones) y acción (son capaces de llevar a cabo acciones).

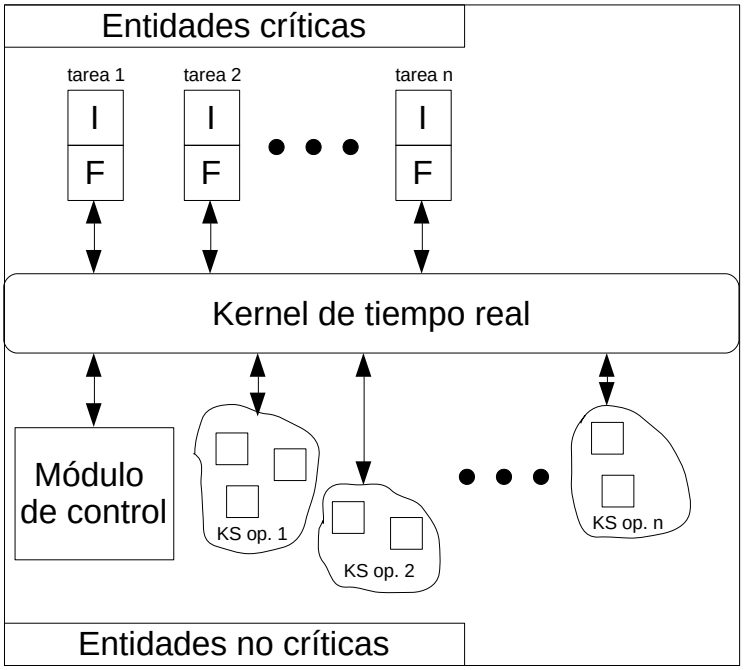


Ilustración 36: Modelo de bajo nivel ARTIS

El modelo de bajo nivel es otra representación del modelo de alto nivel en el que se definen 4 componentes: las tareas críticas, el kernel de tiempo real, el módulo de control y los KSs opcionales. Cada actividad del modelo de alto nivel se convierte en una tarea del modelo de bajo nivel y un grupo de KSs opcionales; el primer nivel de los agentes de percepción y cognición de la actividad compone la parte inicial (I) de la tarea, el resto de niveles de los agentes de cognición forman un grupo de KSs opcional y el primer nivel de los agentes de acción pasa a ser la parte final (F) de la tarea. El kernel de tiempo real permite asignar tiempo de ejecución a las tareas y calcular el tiempo libre disponible para los KSs opcionales.

El módulo de control se encarga de seleccionar los KS opcionales que obtendrán tiempo de ejecución mientras haya tiempo libre según el kernel.

2.7 Distributed Robot Architecture

En [44] se propone una arquitectura divida en 3 capas que están presentes en todos los agentes autónomos del colectivo. Esta arquitectura se orientó a dar soporte a colectivos de robots heterogéneos.

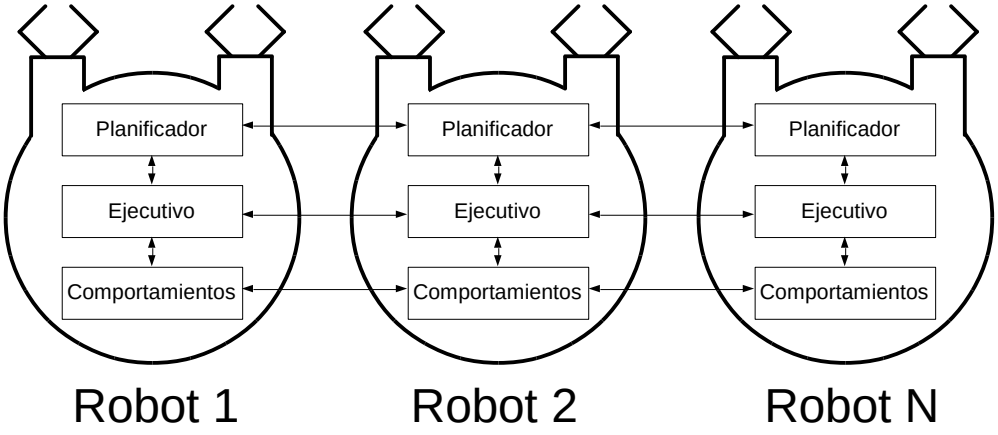


Ilustración 37: Arquitectura en capas de la DRA

Según esta definición, la capa "**planificador**" se ocupa de tomar las decisiones orientadas a conseguir objetivos de alto nivel; la capa "**ejecutivo**" permite la sincronización de los agentes, monitorizar el progreso de las tareas, secuenciar acciones, etc.; por último, la capa "**comportamientos**" sirve como interfaz entre la arquitectura y los sistemas de sensores y actuadores de cada robot. Siguiendo el patrón de diseño por capas, cada capa interactúa con la inmediatamente superior e inferior y, en este caso, además es capaz de interactuar con las capas del mismo nivel presentes en el resto de agentes del colectivo.

Esta pequeña modificación (la interacción entre capas del mismo nivel), permite que la capa de comportamientos de un robot pueda proporcionar la información de sus sensores a la misma capa de otro robot que podría usar con sus actuadores. Esta característica es de especial interés al estar orientada la arquitectura a colectivos de robots heterogéneos.

La interacción de la capa *ejecutivo* es casi imprescindible para la consecución de sus objetivos ya que solo gracias a una comunicación explícita se pueden coordinar acciones complejas entre los distintos robots.

En el caso de la capa *planificador,* esta interacción permite que el colectivo optimice los recursos disponibles. Para ello, los autores proponen varias opciones: por un lado, la negociación distribuida entre agentes (decidiendo qué agentes desempeñarán un cierto rol, cómo se cooperará,…) y por otro, la oferta (posiblemente competitiva) de agentes para convertirse en "capataces" en lo que a una tarea concreta se refiere. Estos capataces negocian la creación de equipos con otros agentes, les asignan tareas (sin restarles la autonomía para negociar entre ellos, siempre en la dirección del objetivo común) y monitorizan el progreso de dichas tareas.

DRA (Distributed Robot Architecture) fue una de las primeras arquitecturas que incluyó el patrón de diseño por capas, tendencia en otras ramas de la robótica (por ejemplo [45]). La aplicación de este patrón permite el desarrollo de nuevas implementaciones de una capa reutilizando las otras, lo que aumenta a su vez la portabilidad al poder desarrollar una nueva capa de *comportamientos* cuando se quieren incluir nuevos modelos de robots en un colectivo que emplee esta arquitectura.

2.8 CHARON

CHARON [46] es un lenguaje que permite la especificación modular de sistemas híbridos interactivos. Este lenguaje permite la definición de agentes (para la jerarquía arquitectónica), que se comunican con el entorno usando variables compartidas y modos (para la jerarquía de comportamiento), que básicamente son una máquina de estados jerárquica en donde cada modo es un estado con, posiblemente, subestados y transiciones entre ellos. La actualización de las variables puede ser de dos tipos: actualizaciones discretas y actualizaciones continuas.

La arquitectura propuesta en [47] permite (mediante el uso de CHARON) desarrollar comportamientos complejos usando composiciones jerárquicas y secuenciales, *modos* estimados y paralelización entre los agentes.

Dos agentes componen el nivel jerárquico más alto:

- Agente de *coordinación*: contiene la especificación de las comunicaciones entre los agentes robot.
- Agente del *grupo-robot*: representa el conjunto de robots.

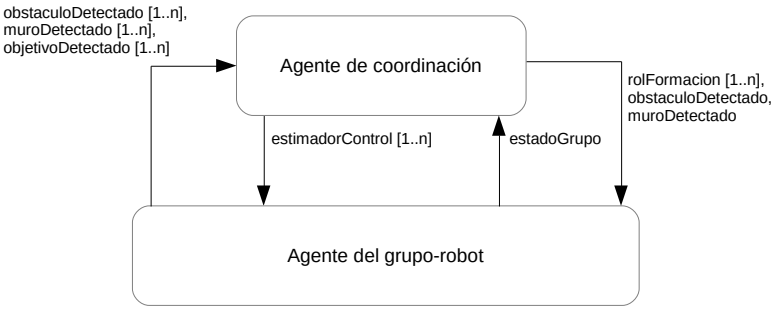


Ilustración 38: Jerarquía de agentes

El agente *grupo-robot* está compuesto por agentes robot, cada uno de los cuales puede recibir información con estimaciones de los obstáculos desde otros robots e instrucciones desde un operador humano. De la misma forma puede enviar su propia información a otros robots o al operador humano.

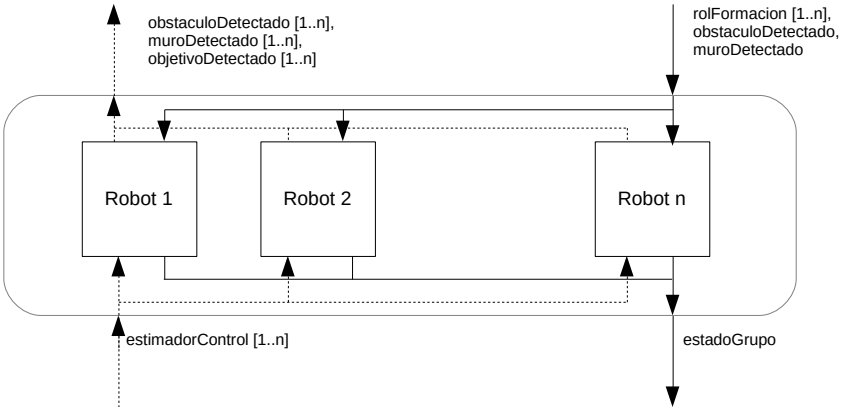


Ilustración 39: Agente grupo-robot

Cada agente robot se compone del agente estimador (representa un conjunto de sensores lógicos), el agente de control (que transita entre modos para definir el comportamiento) y dos agentes dependientes del hardware: agente de sensores (recogida de datos) y agente de control-motor (controla los actuadores).

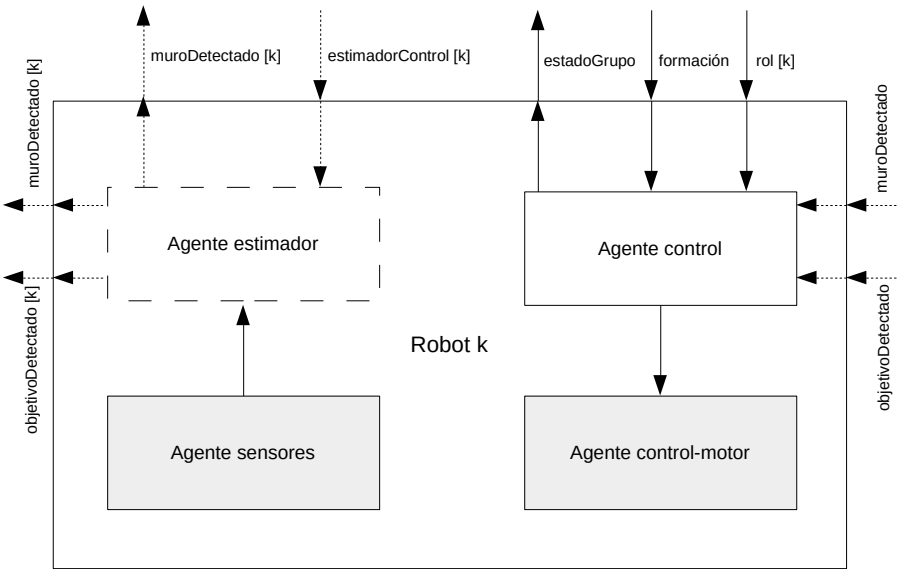


Ilustración 40: Agentes que componen el agente robot

El agente de control usa los modos con sub-modos que se activan de forma jerárquica según las actualizaciones que se produzcan. El ejemplo presente en [47] define un modo raíz (controller-top) dividido en dos modos: modo líder y modo seguidor.

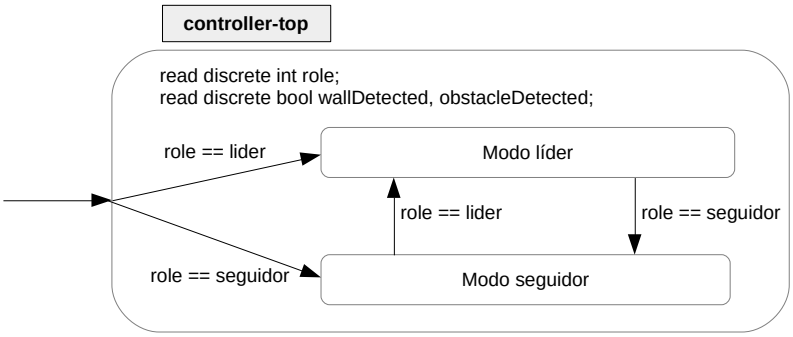


Ilustración 41: Ejemplo del modo coordinator-top

El modo líder está a su vez compuesto por tres sub-modos: ir al objetivo, seguir el muro y evitar obstáculo.

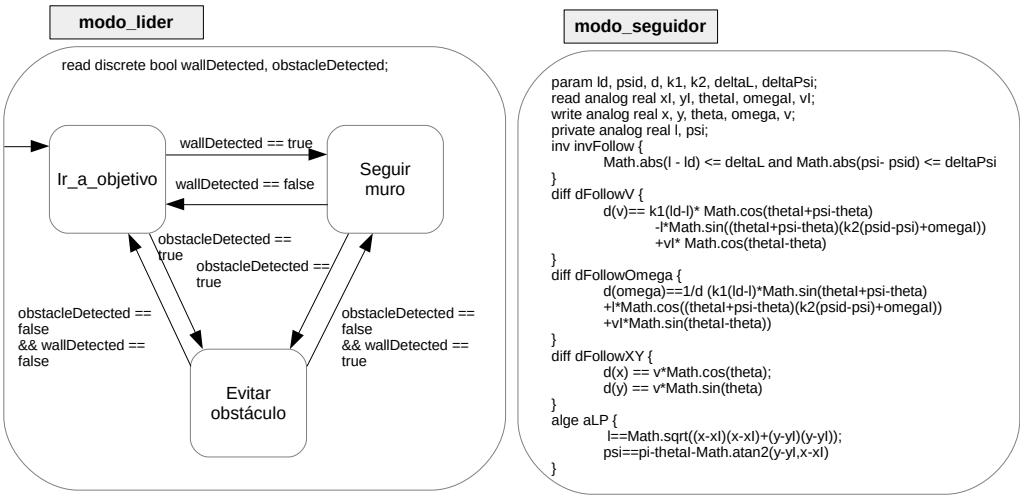


Ilustración 42: Ejemplo con el modo líder y el modo seguidor

2.9 OROCOS

El proyecto Open RObot COntrol Software (OROCOS) [48] nace para el desarrollo de un software de control para robots, de código abierto, modular, extensible y no orientado a un hardware concreto.

OROCOS [48] se basa extensivamente en un gran número de módulos, lo que permite cambiar comportamientos o ejecutar el mismo sistema sobre diferentes actuadores o sensores. Estos módulos se dividen (según [48]) grosso modo en tres tipos:

- **Módulos de soporte**: software sin componentes orientados a robótica tales como IPC (Inter Process Communication), Kernel en tiempo real, ...
- **Módulos de robótica**: aquellos módulos orientados específicamente a aplicaciones de robótica tales como un controlador de servos, etc.
- **Componentes**: Objetos CORBA [49] (Common Object Request Broker Architecture) incluyendo su descripción IDL (Interface Definition Language) y compuestos por los dos tipos de módulos anteriores. El uso de estos componentes garantiza la reutilización, la posibilidad de una ejecución distribuida y la implementación de sistemas de forma rápida.

Desde el punto de vista funcional, OROCOS se puede dividir [50] en 4 librerías:

- **OROCOS Real-Time Toolkit (RTT):** interfaces con una abstracción del sistema operativo y las primitivas de comunicación.

- **OROCOS Components Library (OCL)**: librería con todos los componentes (desde drivers para hardware específico hasta herramientas software para la depuración.
- **OROCOS Kinematics and Dynamics Library (KDL)**: tiene el objetivo de proporcionar modelos cinemáticos en tiempo real.
- **OROCOS Bayesian Filtering Library (BFL)**: contiene algoritmos basados en reglas bayesianas.

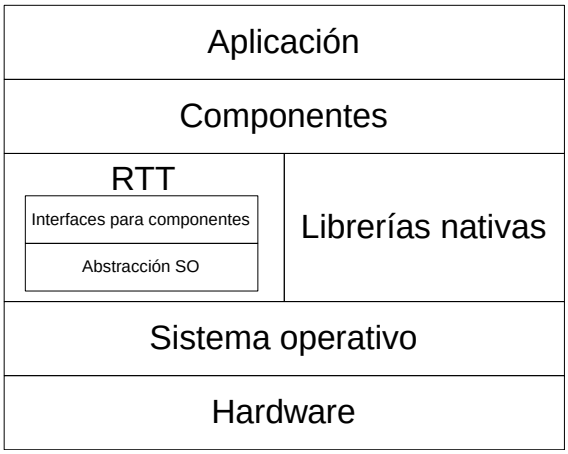


Ilustración 43: Capas en un sistema OROCOS

OROCOS es un framework y librería de componentes que permite implementar distintas arquitecturas y modelos, no solo con fines robóticos.

2.10 CAMPOUT

En [51] se presenta la Control Architecture for Multirobot Planetary OUTposts (CAMPOUT), una arquitectura hardware y software orientada a la exploración de terreno inaccesible y construcción en otros planetas.

Bajo esta arquitectura, los comportamientos básicos se componen para producir los mecanismos de movilidad y control (comportamientos grupales) gracias a un planificador multiagente. CAMPOUT incluye la definición de varios elementos arquitectónicos: la representación del comportamiento, composición de comportamientos y la coordinación de comportamientos y grupo, así como los interfaces entre estos. Esta definición mejora las posibilidades de extensión y escalado del sistema.

Como hemos dicho, dentro de esta arquitectura los *comportamientos primitivos* se combinan para formar *comportamientos compuestos* en múltiples robots. Unos comportamientos denominados "*comportamientos sombra*" se ocupan de la comunicación implícita.

Los *comportamientos de comunicación* permiten a los *comportamientos grupales* coordinar sus actividades.

Cada robot posee una instancia completa de la arquitectura.

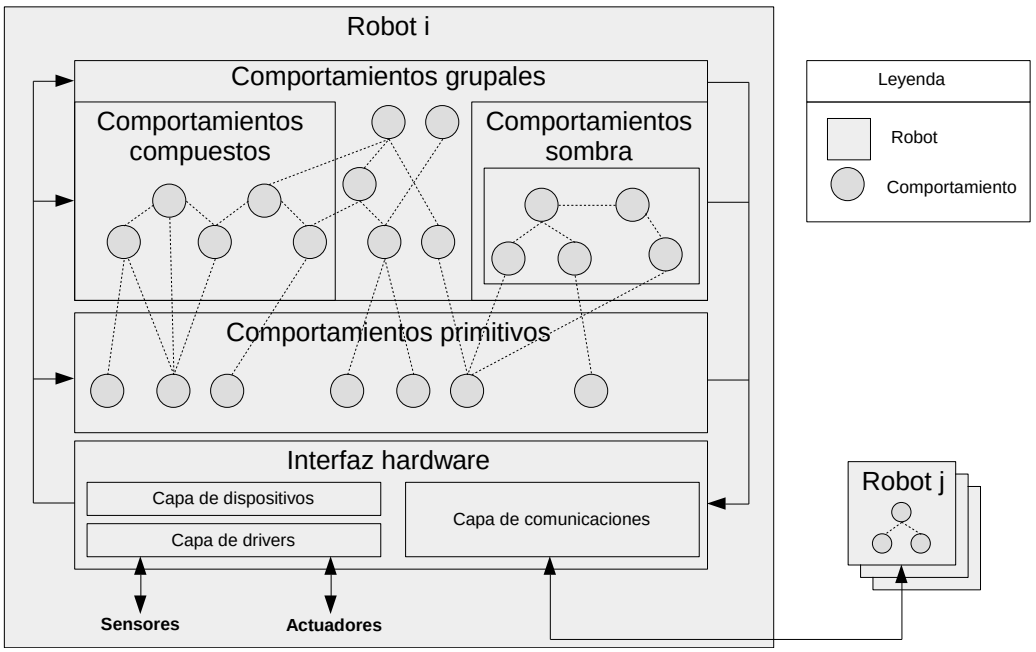


Ilustración 44: Descripción esquemática de CAMPOUT y su organización jerárquica

2.10.1 Representación de un comportamiento

CAMPOUT define un comportamiento *b* como una aplicación de *b: P* x X -> [0;1]* que enlaza cada par de secuencia de percepción ($p \in P^*$) y acción ($x \in X$) con un valor entre 0 y 1. Una percepción es cualquier dato (procesado o no) que proviene de los sensores. Una acción es cualquier actividad a desarrollar.

CAMPOUT implementa esta aplicación como una máquina de estados finitos.

2.10.2 Composición de comportamientos y mecanismos de coordinación de comportamientos

Dado que los comportamientos de alto nivel se componen usando los comportamientos de bajo nivel, CAMPOUT define una serie de mecanismos para la coordinación de comportamientos que controlan las actividades de los comportamientos de bajo nivel en el marco de los comportamientos de alto nivel.

Estos mecanismos se pueden dividir en dos clases: *arbitraje* y *fusión de comandos*.

Los mecanismos de *arbitraje* seleccionan un comportamiento de entre un subconjunto de ellos que asume el control hasta la próxima selección.

Los mecanismos de *fusión de comandos* combinan las salidas de un subconjunto de comportamientos para generar una acción a partir de ellas. CAMPOUT usa varios mecanismos de *fusión de comandos*:

- Sistema basado en votos, en el que la salida de cada comportamiento se entiende como un voto por unas determinadas acciones, seleccionando la acción (o acciones) con la máxima suma ponderada de votos.
- Fusión de comandos *fuzzy*: que usa lógica difusa.
- Sistema basado en la selección para múltiples objetivos en el que se seleccionan las acciones que poseen mejor relación con los objetivos que satisfacen.

2.10.3 Coordinación grupal

CAMPOUT extiende los mecanismos de coordinación de comportamientos para permitir la orquestación de múltiples comportamientos distribuidos en el colectivo de robots teniendo en cuenta que cada robot toma decisiones. Esta extensión está basada en [52].

2.10.4 Comportamientos de comunicación

A pesar de que la comunicación no tiene por qué ser explicita y que CAMPOUT define una serie de comportamientos (los denominados comportamientos sombra) para facilitar la comunicación implícita, también define como

comunicación tres funciones base implementadas en esta arquitectura sobre sockets UNIX:

- **Sincronización**: Para esta función define dos señales: esperar y continuar. Con estas dos señales CAMPOUT pretende sincronizar las actividades entre robots.
- **Intercambio de datos**: Para esta función se definen dos mensajes *sendEvent* (destino, evento) y getEvent (origen, evento) que se usan para intercambiar datos de eventos entre los robots. Un evento es cualquier tipo de dato (por ejemplo la imagen de una cámara).
- **Intercambio de comportamientos**: Esta función incluye dos mensajes sendObjective (destino, objetivo) y getObjective (origen, objetivo) que permiten el envío y recepción de objetivos. Un objetivo es un conjunto de valores con salidas de comportamientos.

2.11 IDEA

Sus autores definen Intelligent Distributed Execution Architecture (IDEA) [53] como un "framework unificado de planificación y ejecución". IDEA divide el sistema en capas, con cada capa siendo un agente. En este framework cada agente posee un modelo, una base de datos de planes, un ejecutor de planes y una variedad de planificadores deliberativos y reactivos. Permite garantizar un tiempo de reacción a cada agente así como definir comportamientos de forma flexible.

Dentro de IDEA, los *procedimientos* (abstracciones que representan procesos con entradas, salidas y un estado definidos -las salidas no tienen por qué obtenerse todas en el mismo momento-) se ejecutan durante un intervalo de tiempo denominado *token* (unidad de ejecución fundamental; el *token* finaliza en cuando se obtiene el estado del *procedimiento* y otro u otros *tokens* comienzan). La ejecución de un procedimiento dentro de un token puede detenerse también por otras múltiples razones (timeout, …).

IDEA usa un wrapper para las comunicaciones del agente (rotulado en la figura como "relé") que se encarga de ordenar la ejecución de procedimientos y recibir sus resultados (incluyendo las marcas de tiempo de inicio y fin de ejecución). IDEA

no define ninguna restricción en la topología de comunicaciones. Las comunicaciones están restringidas por el *modelo* que define qué procedimientos se pueden ejecutar, datos de entrada y salida, etc.

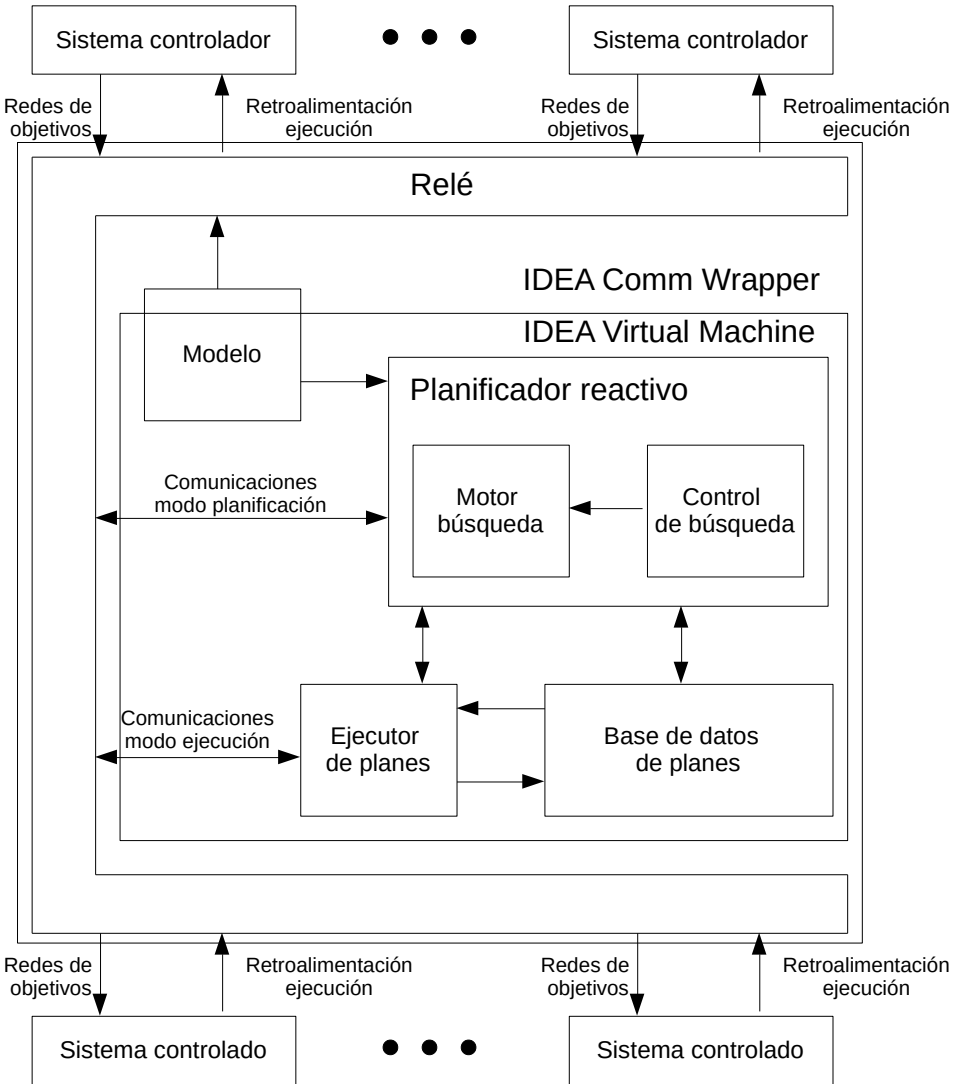


Ilustración 45: Estructura de un agente IDEA

Este framework usa una máquina virtual incorporando distintos componentes:

- **La base de datos de planes** contiene planes que permiten la ejecución de tokens, un agente solo podrá ejecutar tokens que aparezcan en estos planes. La base de datos se puede modificar al recibir nuevos objetivos desde otro agente (que controle el receptor) o porque alguna planificación interna ha generado esos nuevos objetivos.
- **El ejecutor de planes** permite la ejecución de un token interpretando el procedimiento paso a paso comprobando las posibles restricciones

temporales. El ejecutor de planes puede recibir un mensaje para ejecutar un determinado plan desde otro agente.

- **El planificador reactivo** es capaz de seleccionar un plan que se pueda ejecutar en el agente (comprueba las restricciones para su ejecución con respecto al modelo). IDEA no limita el número de módulos de planificación en un agente pudiendo existir varios según las necesidades.

2.12 Swarm-bot

En [54] se presenta (mayormente el hardware de) una plataforma de robótica distribuida esta plataforma se demuestra (entre otros) en [55]. El Swarm-bot / s-bot es interesante al estar especialmente diseñado para permitir el auto-ensamblado [56] sin limitaciones de orientación de un robot con otro y con un gran rango de ángulos en el plano vertical.

[56] describe el controlador usado como distribuido y basado en comportamientos, con una instancia autónoma en ejecución en cada robot. Los robots se comunican iluminando y cambiando de color un conjunto de LEDs.

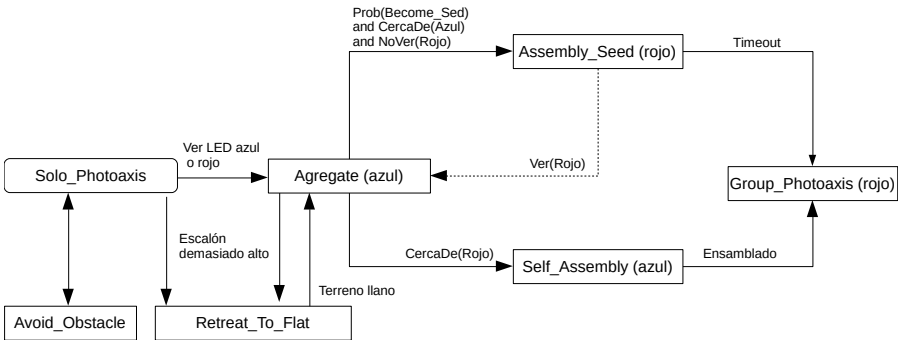


Ilustración 46: Diagrama de estados con las transiciones entre comportamientos en un ejemplo del controlador de SWARM-BOT

En este ejemplo cada SWARM-BOT dispone de varios comportamientos entre los que va transitando dependiendo de su estado y de las entradas de sus sensores (detección de la luz de los LEDs de otros robots, entre otros).

En otros artículos (por ejemplo [57, 58]) los autores proponen un sistema de control basado en Redes de Neuronas Artificiales e incluso [59] este sistema de control se genera usando técnicas de computación evolutiva.

En [60] el sistema descrito es similar a [56] con un agente de control distribuido entre los distintos individuos.

2.13 MAS2CAR

La arquitectura propuesta en [61] y denominada MultiAgents System to Control and Coordinate teAmworking Robots (MAS2CAR), se dispone en 3 capas: capa física, capa de control y capa de coordinación. Los componentes software están diseñados usando UTOPIA [62] y el modelo organizacional Moise^{Int} [63].

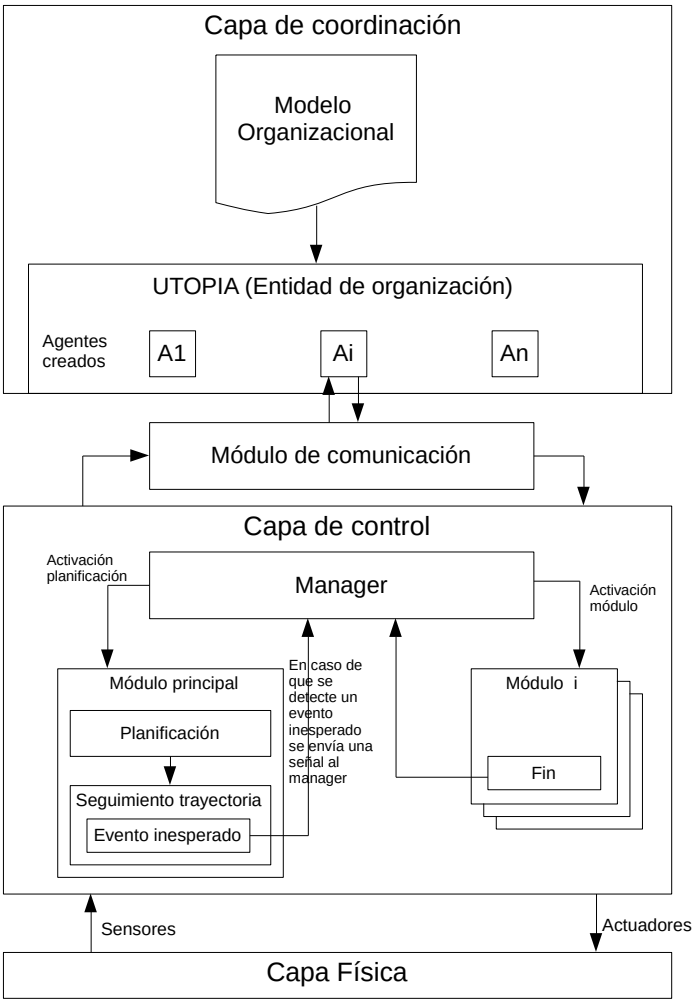


Ilustración 47: Capas MAS2CAR

La capa física es una agregado de múltiples sensores y / o actuadores. La capa de control proporciona los módulos básicos (planificar, re-planificar, modo reactivo…), la gestión de los sensores y actuadores, e interactúa con la capa de coordinación y el módulo de comunicación. La capa de coordinación está orientada a tareas más complejas e implementada como un agente.

Los autores consideran la comunicación o interacción uno de los principales objetos de estudio de los sistemas multi-robot, también considerando que lo más apropiado para un entorno MAS es la interacción a través de la comunicación (comunicación explícita). El diseño de MAS2CAR responde a esta creencia usando un sistema capaz de recibir comunicaciones de todos los robots del colectivo, lo que permite que un agente de la capa de coordinación pueda conectarse con la capa de control asociada de forma local o con la capa de control de otro individuo del colectivo (usando para ello el módulo de comunicaciones).

La existencia de una capa de control por robot permite que esta sea adaptada de forma específica a las capacidades del mismo, dando soporte a colectivos heterogéneos.

2.13.1 Modelo organizacional de MAS2CAR

MAS2CAR usa Moise^{Inst} para dar soporte a su modelo organizacional y, por lo tanto, a su capa de coordinación. Moise^{Inst} permite especificar este modelo en 4 dimensiones:

- Especificación funcional (FS por sus siglas en inglés): Una vez definidos los objetivos y conjuntos de objetivos (misiones) para los robots y supervisor, estos se pueden ejecutar/combinar de 3 formas:
    - Secuencia: define que un objetivo debe completarse antes de poder continuar con otro ($g_x \rightarrow g_y$ = g_x debe ejecutarse antes de g_y).
    - Paralelismo: define que varios objetivos deben ser concurrentes ($\|\{g_x, g_y\}$ = g_x y g_y deben ejecutarse en paralelo).
    - Opción: uno (y solo uno) de los objetivos debe completarse ($\Uparrow\{g_x, g_y\}$ = g_x o g_y deben ejecutarse, pero no ambos).

- Especificación Estructural (SS por sus siglas en inglés): En esta especificación se define un rol para cada uno de los robots. Además, debe definirse un rol de supervisor que ha de aportar un punto de vista global en la interacción del resto de agentes.
- Especificación Contextual (CS por sus siglas en inglés): Donde se definen contextos (estados), escenas (conjuntos de estados) y las transiciones entre contextos. Debe definirse, al menos, una escena para el supervisor y otra para cada robot.
- Especificación Normativa (NS por sus siglas en inglés): En esta especificación se relacionan las otras tres: cada norma forzará a un rol X (definido en la SS), cuando el contexto es el contexto Y (definido en la CS), ejecutar la misión Z (definida en la FS). Estas normas se definen tanto para los roles asignados a cada robot como para el supervisor.

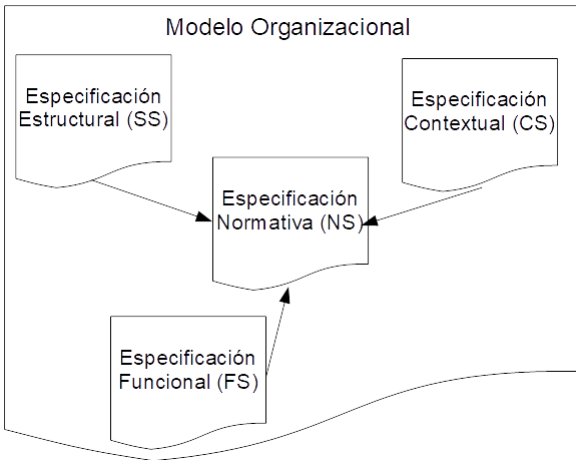


Ilustración 48: Moise^{Inst}, modelo organizacional

2.14 Conclusiones

Hemos revisado las distintas arquitecturas que han contribuido al desarrollo de la robótica colectiva: algunas puramente orientadas a la misma (como Matarić), otras orientadas a la robótica en general (como OROCOS) y otras con una orientación más general (como IDEA).

Las arquitecturas vistas resuelven problemas como la coordinación, la ejecución de tareas (tanto a nivel de grupo como a nivel de individuo) o la comunicación entre individuos de forma satisfactoria. Incluso alguna de ellas (ALLIANCE, CAMPOUT,…) es aplicable a conjuntos heterogéneos de robots, y otras poseen ciertas capacidades de tolerancia a fallos (ACTRESS,…). Más adelante trataremos de enunciar las principales características de las distintas arquitecturas.

Las arquitecturas del estado del arte, sin embargo, no resuelven otros problemas como la incorporación dinámica de individuos con características y capacidades distintas a los ya existentes. Las arquitecturas vistas tampoco dan un soporte adecuado para la migración de la lógica de control a otro conjunto de robots, para ampliar el número de comportamientos básicos o para cambiar la lógica que los selecciona, mucho menos dotan al colectivo de estas habilidades en caliente. Más adelante detallaremos en un listado todas las capacidades o problemas a los que las arquitecturas vistas no aportan una solución o consideramos que esta no es satisfactoria.

2.14.1 Comparativa de las principales características

En la siguiente tabla se reflejan los atributos considerados más relevantes de cada una de las arquitecturas descritas anteriormente. Cada casilla de la tabla puede tener varios valores o estar en blanco dado que no todos los atributos son aplicables a todas las arquitecturas (al ser atributos de sistemas multi-robot, no de la arquitectura usada para su implementación) y que hay arquitecturas que permiten distintas implementaciones, lo que implica varios valores de algunos atributos.

	CEB0T	ACTRESS	GOFER	Matarić	ALLIANCE	ARTIS	Distributed Robot Architecture	CHARON	OROCOS	CAMPOUT	IDEA	Swarm-bot	MAS2CAR
Propiedades del colectivo													
Multifuncional / Monofuncional	Multifuncional	Multifuncional	Multifuncional	Multifuncional	Multifuncional	Multifuncional	Multifuncional	Multifuncional	Multifuncional	Multifuncional aunque con una fuerte orientación concreta	Multifuncional	Monofuncional	Multifuncional
Flexibilidad	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí
Tamaño del colectivo	INF	INF	Limitado	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
Escalabilidad	Sí	Sí	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Robustez	Redundancia, Coordinación descentralizada, simplicidad de los individuos	Redundancia, Coordinación descentralizada, simplicidad de los individuos, Multiplicidad de detección			Redundancia, Coordinación descentralizada, Multiplicidad de detección		Redundancia, Coordinación descentralizada, Multiplicidad de detección	Redundancia, Multiplicidad de detección					
Coordinación	Descentralizada Jerárquica, Débil	Descentralizada, Fuerte	Centralizada	Descentralizada	Descentralizada	Centralizada	Descentralizada	Centralizada		Descentralizada	Jerárquica	Descentralizada	

Tipos de conocimiento colectivo	Incompleto	Relacionado con el individuo	Incompleto	Incompleto	Incompleto	Eventual y continuo	Eventual y continuo	Incompleto			Incompleto	Incompleto	Incompleto
Cooperatividad	Fuertemente cooperativos	Fuertemente cooperativos	Débilmente cooperativos	Fuertemente cooperativos			Fuertemente cooperativos	Fuertemente cooperativos		Fuertemente cooperativos	Débilmente cooperativos	Débilmente cooperativos	
Heterogeneidad	Levemente Heterogéneo	Heterogéneo	Homogéneo	Homogéneo	Heterogéneo	Heterogéneo	Heterogéneo	Heterogéneo	Heterogéneo	Heterogéneo	Heterogéneo	Homogéneo	
Estructura de comunicación	Comunicaciones	Comunicaciones	Comunicaciones	Comunicaciones	Comunicaciones	Comunicaciones	Comunicaciones	Comunicaciones		Comunicaciones	Comunicaciones	Comunicaciones muy limitado	Comunicaciones
Auto-regeneración o auto-reparación	Posible	Posible	No	No	Posible		No				No		
Auto-reconfiguración	Posible	Posible	No	No	Posible		No			No	No		
Auto-replicación	No	No	No	No	No		No	No		No	No	No	No
Canibalización	No	No			No		No	No		No	No	NO	No
Propiedades de los individuos tipo													
Capacidad de procesamiento	Limitada	No limitada	Limitada	Limitada	No limitada		No limitada	No limitada		No limitada	No limitada	Limitada	No limitada

Rango de comunicación	Limitado	Infinito	Limitado	Limitado	No limitado		No limitado	No limitado		No limitado	No limitado	Limitado	No limitada
Ancho de banda de comunicación	Limitado (desconectado) / prácticamente infinito (conectado)	Infinito	Limitado	Limitado	No limitado		No limitado	No limitado		No limitado	No limitado	Muy limitado	
Topología de comunicación	Totalmente conexa sin direccionamiento (desconectado) / Uno a uno (conectado)	Dependiente de la aplicación	Totalmente conexa con direccionamiento									Totalmente conexa sin direccionamiento	
Conciencia de la existencia de otros	Sí	Sí	Sí	Sí	Sí		Sí			Sí	Sí	Sí	
Auto-ensamblado	Sí	Posible	No	No	No		No	No		No	No	Sí	No
Auto-modificación, auto-adaptación o auto-reconfiguración	Podría argumentarse que cada CEBOT es una parte	Posible	No	No	No		No	No		No	No	No	No

Integración de comportamientos de organización especial													
Agregación	Sí	No parte arquitectura	Cierto soporte gracias a planificación de movimiento	Sí	No parte arquitectura	No parte arquitectura			No parte arquitectura		No parte arquitectura		
Dispersión		No parte arquitectura	Cierto soporte gracias a planificación de movimiento	Sí	No parte arquitectura	No parte arquitectura			No parte arquitectura		No parte arquitectura		
Formación de un patrón	Sí	No parte arquitectura	Cierto soporte gracias a planificación de movimiento	Sí	No parte arquitectura	No parte arquitectura		Sí	No parte arquitectura	Sí	No parte arquitectura	Sí	
Cadena	Sí	No parte arquitectura	Cierto soporte gracias a planificación de movimiento	Sí	No parte arquitectura	No parte arquitectura		Sí	No parte arquitectura		No parte arquitectura	Sí	
Auto-ensamblado y morfogénesis	Sí	No parte arquitectura	No	No	No parte arquitectura	No parte arquitectura		No	No parte arquitectura	No	No parte arquitectura	Sí	
Agrupación y ensamblado de objetos		No parte arquitectura			No parte arquitectura	No parte arquitectura			No parte arquitectura	Sí	No parte arquitectura	Sí	
Homing		No parte arquitectura	Cierto soporte gracias a planificación de movimiento	Sí	No parte arquitectura	No parte arquitectura			No parte arquitectura		No parte arquitectura		

Integración de comportamientos de navegación													
Exploración colectiva		No parte arquitectura	Cierto soporte gracias a planificación de movimiento		No parte arquitectura	No parte arquitectura			No parte arquitectura		No parte arquitectura	Sí	
Cobertura de un área		No parte arquitectura	Cierto soporte gracias a planificación de movimiento		No parte arquitectura	No parte arquitectura			No parte arquitectura		No parte arquitectura		
Navegación guiada por el colectivo		No parte arquitectura	Cierto soporte gracias a planificación de movimiento	Sí	No parte arquitectura	No parte arquitectura			No parte arquitectura	Sí	No parte arquitectura	Sí	
Movimiento coordinado		No parte arquitectura	Cierto soporte gracias a planificación de movimiento	Sí	No parte arquitectura	No parte arquitectura		Sí	No parte arquitectura	Sí	No parte arquitectura	Sí	
Transporte colectivo		No parte arquitectura	Cierto soporte gracias a planificación de movimiento		No parte arquitectura	No parte arquitectura		Sí	No parte arquitectura	Sí	No parte arquitectura	Sí	
Manipulación de un objeto		No parte arquitectura	Cierto soporte gracias a planificación de movimiento		No parte arquitectura	No parte arquitectura		Sí	No parte arquitectura	Sí	No parte arquitectura	Sí	
Integración de comportamientos de toma de decisiones colectiva													
Consenso													
Asignación de tareas		Sí	Sí									Sí	

Integración de otros comportamientos													
Regulación del tamaño de grupo			No										
Interacción con humanos		Posible	Sí					Sí					
Resolución de conflictos de recursos			Cierto soporte										

Tabla 1: Comparativa de las principales características de las arquitecturas

2.14.2 Características no observadas en las arquitecturas identificadas

Tal como mencionábamos anteriormente existen varios problemas que no quedan resueltos así como varias características o habilidades que no proporcionan las arquitecturas identificadas.

Por una parte está la regulación del colectivo en sí, entendida como gestión de sus componentes. Entre las habilidades que se podrían incluir en este grupo están:

- Expulsión de miembros dañados, innecesarios, malintencionados o para los que no se dispone de recursos.
- Adopción de nuevos miembros, ya sea de forma voluntaria (el individuo desea integrarse) o forzada (el individuo desea no integrarse). Este proceso podría iniciarse desde el individuo a ser integrado o desde el colectivo.
- Escisión o partición del colectivo: el colectivo se dividiría en varios colectivos que actuarían de forma independiente. Este proceso podría incluso ser parte de una función reproductiva a nivel de colectivo.

Por otra parte las arquitecturas se centran en el colectivo en sí dejando de lado la posibilidad de la existencia de varios colectivos que podrían relacionarse entre sí o incluso compartir individuos. Podríamos destacar las siguientes capacidades como ejemplos de esta categoría:

- Integración de otros colectivos (adopción de todos sus miembros). Esto da lugar a distintos escenarios: fusión de varios colectivos, asimilación de los miembros de otro colectivo descartando el colectivo en sí, etc.
- Comunicaciones con otros colectivos.

La adopción o asimilación de miembros como técnica para la formación o ampliación de un colectivo requiere de otras habilidades tales como:

- Autoconocimiento de los sensores, actuadores y otras capacidades disponibles de forma que cuando un individuo se integra, el colectivo dispone de la información de las características del nuevo individuo. Esta información podría no estar disponible desde el primer momento e irse descubriendo por experimentación.

- De la misma forma que debe disponerse de información sobre las capacidades aportadas por un individuo, podría disponerse de información sobre comportamientos básicos (una especie de memoria procedimental) que sería útil a otros individuos.
- Estos comportamientos deberían ser extrapolados entre individuos ya que lo más probable es que no se puedan usar directamente y se requiera un proceso de conversión.

Otras características que se echan en falta son:

- Buenas habilidades de tolerancia/recuperación ante fallos (cuando sea necesario) que en sentido último deben permitir la continuidad del colectivo incluso cuando todas las plataformas menos una dejen de funcionar. Entre otras:
    - Soporte a las propias habilidades de tolerancia a fallos de los individuos.
    - Reparación/rescate de individuos.
    - Restauración del colectivo.
- Capacidades de integración con otros sistemas (no solo técnicos) para beneficiar o beneficiarse de otras fuentes de información, interacción con seres humanos (u otros animales), etc.
- Maximizar las posibilidades de construcción de sistemas con orientación a distintos sub-campos de los sistemas multi-robot mediante diferentes niveles de comunicación, coordinación, etc. Para ello el sistema debe permitir un overhead variable gracias a la modularidad de los distintos componentes esenciales.

2.14.3 Principales problemas que debe resolver una arquitectura

Tal como hemos visto existe un gran número de características, comportamientos, metodologías, etc., muchos de los cuales solo son aplicables a un subconjunto de los sistemas de robótica colectiva. Por otra parte, las herramientas de desarrollo de este tipo de sistemas están igualmente fragmentadas.

El análisis de los atributos y taxonomías junto con el del estado del arte de las arquitecturas nos permite definir aquellos problemas a los que una arquitectura debe dar respuesta proporcionando una o varias soluciones. Siendo de la misma importancia dar solución al problema como permitir seleccionar entre distintas soluciones según sea más adecuado para la construcción de un sistema concreto.

2.14.3.1 Adaptación

Dependiendo de la tarea objetivo y de los individuos que conforman el colectivo, este tiene que adaptarse para alcanzar de forma eficiente el objetivo. Esta adaptación o flexibilidad requiere por su parte una cierta coordinación y los diferentes métodos de coordinación tienen implicaciones sobre la capacidad de adaptación de la arquitectura o sistema que da soporte al colectivo.

La adaptación tiene múltiples facetas y afecta tanto a capacidades para la construcción de sistemas (seleccionar las habilidades del colectivo antes del inicio del sistema) como a los sistemas resultantes (adaptación del sistema "en caliente").

2.14.3.2 Cooperación

La cooperación entre distintos individuos para conseguir un objetivo y conformar una entidad de mayor tamaño (grupo, sociedad, enjambre…) es el elemento principal que debe proporcionar cualquier sistema. Si los individuos no cooperan de alguna forma (aunque esta sea competitiva) mejorando la suma de resultados individuales, el sistema carecería de sentido.

La naturaleza es una infinita fuente de inspiración en lo que se refiere a entidades (incluso en gran número) que cooperan para conseguir un fin, siendo los ejemplos más típicos las abejas, que son capaces de construir nidos y comunicarse con movimientos; las hormigas, capaces de trasportar de forma colectiva un objeto; o las termitas, también capaces de construir un nido de forma colaborativa. Pero hay otros muchos como las asociaciones simbióticas o las sociedades humanas. Estos ejemplos dan idea de lo amplio del espectro de la cooperación.

2.14.3.3 Coordinación

La coordinación es la habilidad que permite a varios individuos unir sus acciones para realizar una acción que le sería imposible de realizar a uno solo. La

coordinación es uno de los elementos típicos para la cooperación, pero no es imprescindible ya que existen tareas en donde se puede cooperar de forma no coordinada. Sin embargo existen tareas que requieren una coordinación muy fina como puede ser el ensamblado de un objeto o el transporte colectivo.

Existen distintos niveles y formas de coordinación que una arquitectura debería soportar, desde una coordinación basada en comunicación hasta en la detección del estado de las acciones de otros individuos.

2.14.3.4 Comunicación

Otro de los problemas a los que se debe dar una solución es el de la comunicación y, de igual forma que en los puntos anteriores, hay que permitir la mayor flexibilidad posible, tanto durante el diseño del sistema como durante su operación.

La definición de un protocolo que permita cambiar la estructura, el medio, etc. puede ser de utilidad en el soporte durante la ejecución.

La comunicación no solo es un problema dentro del colectivo, también puede ser necesaria para relacionarse con usuarios del sistema o con otros colectivos.

2.14.3.5 Autoconocimiento

Para poder ejecutar ciertas acciones, como es la adopción de otros miembros en un entorno de robots heterogéneo, puede ser necesario proporcionar soporte de acceso a la información de las capacidades de los individuos del colectivo, de forma que el colectivo tenga acceso a qué puede hacer cada individuo.

Esta característica no es imprescindible en todos los casos, especialmente en sistemas de robots homogéneos, sin embargo una arquitectura que quiera dar soporte a todos los escenarios posibles debe aportar una solución a este problema.

2.14.3.6 Colectivos

Dependiendo del sistema que se quiera implementar para dar solución a un problema, puede ser necesario disponer da capacidades para adoptar nuevos miembros, expulsar otros, etc.

Además pueden ser necesarias funciones para fusionarse o absorber otros colectivos.

2.14.3.7 Interoperabilidad

Una arquitectura que pretenda dar un soporte global debe ser capaz de adaptarse a usar otros sistemas equivalentes de forma que se pudieran integrar individuos que implementan otras arquitecturas (o la misma pero con distinta implementación), dar soporte a otras plataformas, etc.

2.14.4 Implementaciones disponibles

No todas las arquitecturas identificadas disponen de una implementación disponible y mucho menos de una implementación de código abierto (que puede ser necesaria para aplicarle el sistema de evaluación).

En la siguiente tabla podemos ver la disponibilidad de implementación, sus características y la referencia a la misma.

Arquitectura	**Lenguaje de programación/ plataforma**	**Código abierto**	**Disponibilidad**
CEB0T	-	-	-
ACTRESS	-	-	-
GOFER	-	-	-
Matarić	-	-	-
ALLIANCE	-	-	-
L-ALLIANCE	-	-	-
ARTIS	-	-	-
Distributed Robot Architecture	-	-	-
CHARON	CHARON/Java 1.3/1.4	No	[64]
OROCOS	C++	Sí	[65]
CAMPOUT	-	-	-
IDEA	-	-	-
Swarm-bot	-	-	-
MAS2CAR	-	-	-

Tabla 2: Implementaciones disponibles de las arquitecturas

Como puede deducirse de la Tabla 2 es difícil encontrar implementaciones disponibles de las arquitecturas mencionadas y no hay garantía de poder usarlas con el sistema de evaluación aunque se encuentren.

Capítulo 3

Estado del arte en evaluación de arquitecturas y sistemas multi-robot

Para poder realizar una comparación cuantitativa de las distintas arquitecturas necesitamos disponer de métricas aplicables a los aspectos más relevantes de las mismas. Al pretender aplicarse sobre la arquitectura, y no sobre el sistema final, debemos fijarnos en aquellas que permiten medir tanto las capacidades que se pueden desarrollar como el rendimiento de los elementos de la arquitectura, tratando de abstraernos de aplicaciones concretas de las mismas.

Además, si queremos poder aplicar cualquier tipo de métrica sin necesidad de construir los sistemas robóticos, podemos necesitar disponer de simuladores que nos permitan ejecutarlas. Estos simuladores también permitirán mejorar el diseño del sistema final mediante la posibilidad de comparar diferentes opciones.

3.1 Métricas de rendimiento

Una métrica de rendimiento es según [66] “una medida cuantitativa de un comportamiento observable” siendo “normalmente medible de forma directa; sin embargo, en ciertos casos, debe calcularse”.

3.1.1 Métricas para el colectivo

En este conjunto agrupamos aquellas métricas que se aplican al conjunto del sistema o arquitectura.

3.1.1.1 Diversidad del grupo

En [67] el autor define medidas cuantitativas para la diversidad frente a la clasificación tradicional en heterogénea y homogénea. Es evidente que esta

clasificación (heterogénea/homogénea) no indica nada sobre el nivel de diversidad en el grupo (por ejemplo: la proporción de individuos diferentes, el nivel de la diferencia entre los individuos etc.

Para realizar estas medidas de la diversidad [67] propone la medición continua de la diferencia de comportamiento y la entropía social jerárquica.

La diferencia de comportamientos se refiere a la selección de distintos comportamientos a partir del mismo estado de percepción, por ejemplo, si dos individuos están en el mismo estado interno y reciben los mismos estímulos pero deciden aplicar comportamientos diferentes.

Para definir la métrica, el autor, hace uso de agrupaciones jerárquicas dentro del colectivo basadas en la diferencia de comportamiento de los individuos.

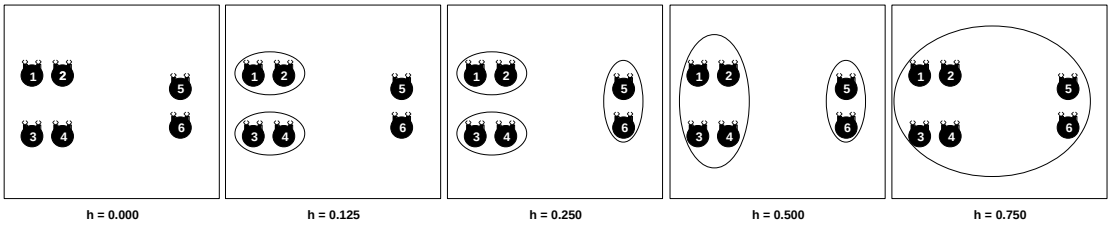


Ilustración 49: Agrupaciones de comportamientos

Estas agrupaciones se parametrizan con un valor de *h* (nivel taxonómico) que nos permite generar un dendrograma.

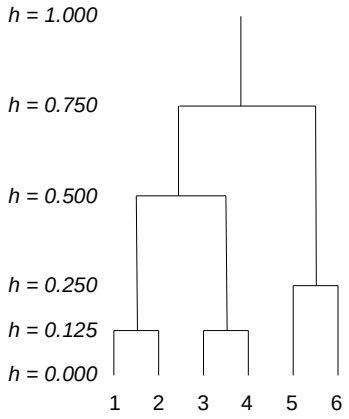


Ilustración 50: Dendrograma de las agrupaciones

En este dendrograma las diferencias de altura reflejan también diferencias en el comportamiento.

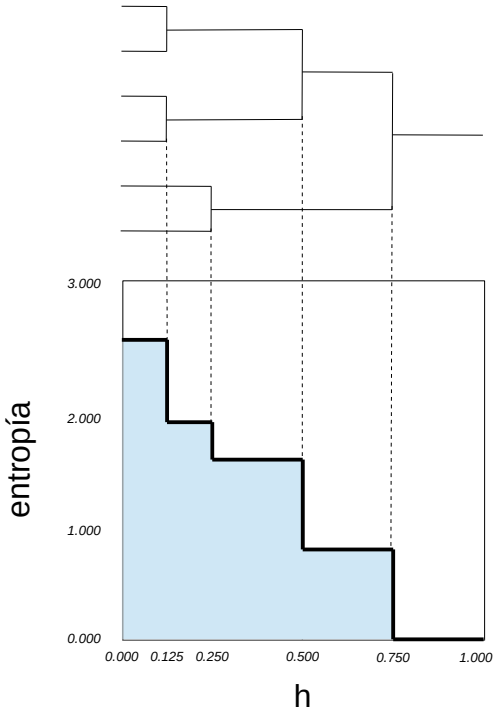


Ilustración 51: Dependencia entre la entropía y h

La entropía H es una función de R (sociedad de N agentes donde R = {r_1, …r_N}) y h (parámetro del algoritmo de agrupación). La entropía social jerárquica se define como:

$$S(R) = \int_0^{\infty} H(R,h)dh$$

Es importante recalcar que dado que el máximo de la medida de la diferencia de comportamiento está fijado en 1.0, la entropía social jerárquica también está limitada a este valor.

3.1.1.2 Comportamientos emergentes

Para medir los comportamientos emergentes surgidos de comportamientos individuales e información enteramente local, [68] propone una métrica basada en *stigmergía* [69] (que "ocurre cuando el comportamiento de los individuos modifica el entorno y, a su vez, es regulado por el estado de dicho entorno").

Esta propuesta [68] de métrica es más una clasificación pero es fácilmente transformable en valores numéricos:

- Sistemas no adaptativos (estimergía = 0): representa sistemas sin capacidad de adaptación al entorno.

- Sistemas adaptativos: Se adaptan a las condiciones del entorno regulando diversos parámetros.
- Sistemas auto-organizativos: Se adaptan a las condiciones del entorno y se reorganizan según sea necesario.
- Estimergia total: Modifican el entorno para cumplir sus fines.

3.1.1.3 Basado en aplicaciones concretas

Típicamente cada autor ha usado sus propias métricas aplicables solo al dominio y sistema que desarrolla. Algunos autores como [68] tratan de ir más allá y analizan las características a medir y los objetivos que deberían satisfacer las métricas orientadas a este campo. A continuación analizamos algunas de las métricas para aplicaciones concretas.

3.1.1.3.1 Tiempo empleado en la eliminación de residuos peligrosos

En [71], los autores definen un problema en el que es necesario desplazar elementos peligrosos de dos vertidos localizados en puntos desconocidos de una habitación cerrada mediante el uso de un equipo de tres robots. Estos vertidos deben trasladarse al punto definido dentro de la habitación y el equipo debe informar de sus progresos a un supervisor humano de forma periódica.

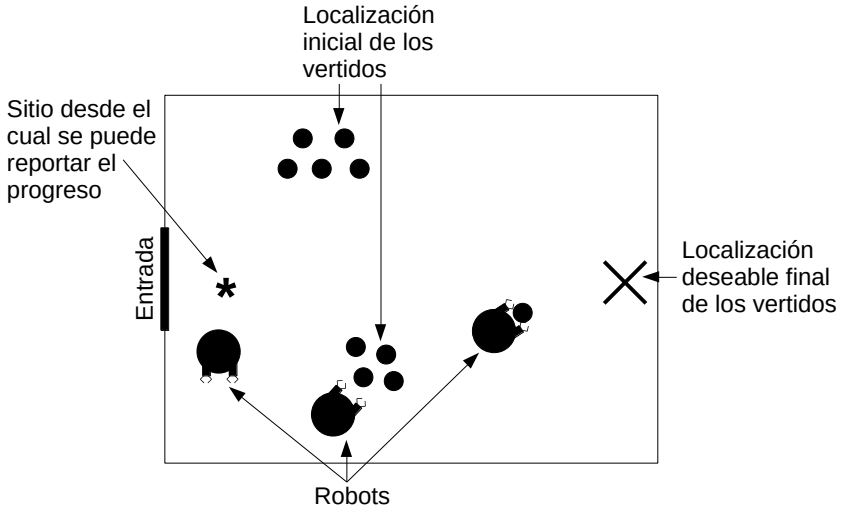


Ilustración 52: Limpieza de vertidos peligrosos

En este escenario, la primera métrica utilizada es el tiempo empleado en la tarea que se define como:

$$\text{tiempo\_tarea}_i\ (k, j, t)$$

$$= \left(\text{tiempo medio de las últimas } \mu \text{ pruebas del rendimiento de } r_k \text{ de la tarea } h_i(a_{ij})\right)$$

$$+ (1 \text{ desviación estándar de la } \mu \text{ pruebas, tal como se midieron para } r_i)$$

Además se usa el tiempo total empleado en la tarea denotado por t_{max}.

4.1.1.2.2 Energía empleada en la eliminación de residuos peligrosos

En el escenario definido en el punto anterior de limpieza de un vertido peligroso, [71] propone otra métrica más como el total de energía empleada por los robots para realizar la tarea. Esta métrica queda definida por:

$$\sum_{t=1}^{tmax} \sum_{i=1}^{m} e_i(t)$$

Donde $e_i(t)$ es la energía usada por el robot i en el tiempo t y m el número de robots.

3.1.1.3.3 Distancia perpendicular por unidad de tiempo en el transporte coordinado de objeto

En [72] se propone el desplazamiento de un objeto entre dos robots idénticos, con uno solo (cuando falla uno de ellos) y por último con dos robots diferentes para observar la adaptación a la heterogenia (los dos robots tienen una capacidad de empuje diferente).

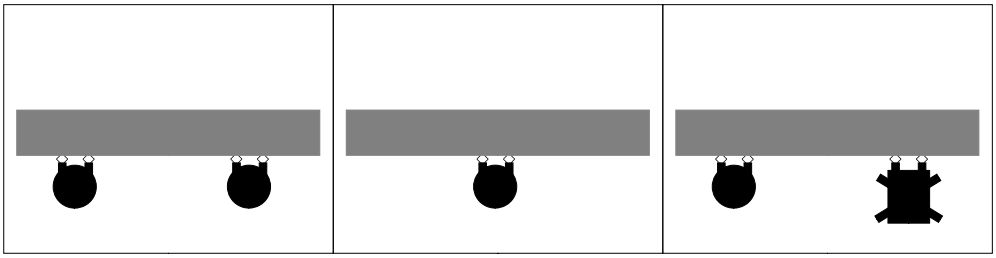


Ilustración 53: Desplazamiento de un objeto rectangular

Para este escenario [70] propone una métrica basada en la distancia que se consigue desplazar la pieza, definida como:

$$d_{\perp}(t)/t$$

Donde $d_{\perp}(t)$ es la distancia ⊥ (perpendicular) que se ha desplazado el objeto en el tiempo t.

3.1.1.2.4 Tiempo de limpieza de una estancia desconocida

En [73] se define una tarea de limpieza de una habitación en principio desconocida.

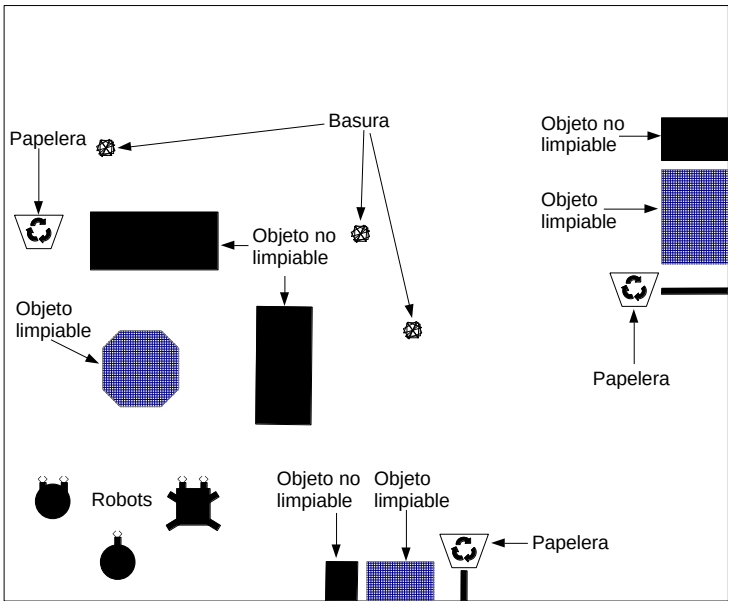


Ilustración 54: Servicio de limpieza

La tarea de limpieza se divide, a su vez, en tres subtareas: el vaciado de las papeleras, la limpieza de los muebles y la limpieza de los suelos (recogiendo basura del suelo). En este ejemplo, el equipo de robots desconoce las capacidades del resto de miembros del equipo y estas últimas pueden variar con el tiempo debido a degradación del rendimiento o un fallo. Además un robot aislado no dispone de las capacidades para llevar a cabo la tarea.

En [70] se definen las mismas métricas que en el dominio de la limpieza de un vertido peligroso, siendo la primera basada en el tiempo usado (t_{max}).

3.1.1.2.5 Energía empleada en la limpieza de una estancia desconocida

En el mismo escenario definido en el punto anterior, [70] propone otra métrica basada en la energía total consumida tras realizar la tarea:

$$\sum_{t=1}^{tmax} \sum_{i=1}^{m} e_i(t)$$

Donde $e_i(t)$ es la energía usada por el robot i en el tiempo t; y m el número de robots.

Esta métrica es idéntica a la usada en la limpieza de un vertido potencialmente peligroso.

3.1.1.3.6 Distancia por unidad de tiempo en desplazamiento por grupos

Otro escenario [73] es el desplazamiento de un conjunto de varias unidades de dos tipos de robots que deben:

1. Dividirse en dos grupos conteniendo una distribución por tipo igual para ambos grupos.
2. Concentrar cada grupo en un punto inicial.
3. Cada grupo elige un líder.
4. Uno de los grupos inicia el desplazamiento al siguiente punto mientras el otro grupo vigila.
5. Una vez alcanzado el destino, los roles entre los equipos se intercambian.

La métrica definida en [70] para este escenario es la de la distancia por unidad de tiempo:

$$d(t)/t$$

Donde d(t) es la distancia que han conseguido moverse en el tiempo t.

3.1.1.3.7 Error acumulativo por en el mantenimiento de una formación

En [74] se define uno de los escenarios más típicos de la literatura: el mantenimiento de una formación mientras el colectivo se desplaza por el terreno en el menor tiempo posible.

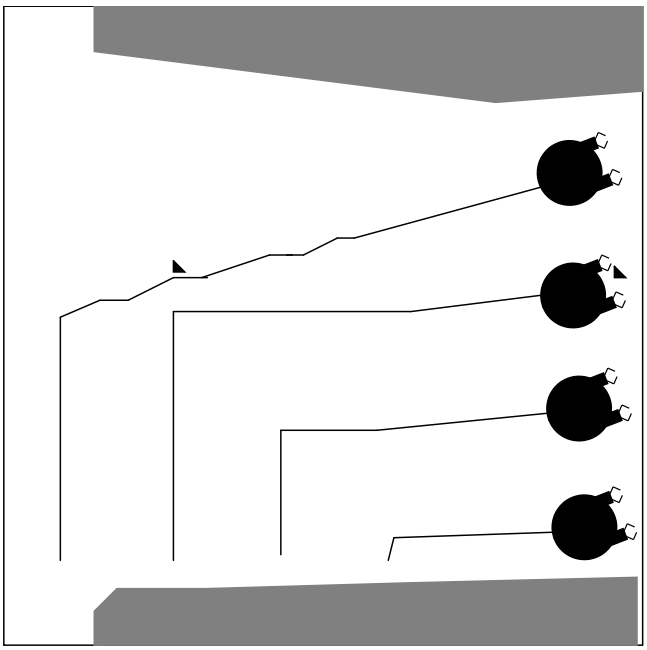


Ilustración 55: Desplazamiento de un equipo siguiendo una de las estrategias de mantenimiento de la formación

La métrica definida por la autora es:

$$\sum_{t=0}^{t_{max}} \sum_{i \neq leader} d_i(t)$$

Donde $d_i(t)$ es la distancia que el robot i está fuera de alineación en el momento t.

3.1.1.3.8 Número de objetos manipulados por unidad de tiempo

En [75] la autora define otro problema clásico: el paso de un testigo. En esta versión los robots tienen que llevar un testigo desde el punto inicial al final y, para cambiar dicho testigo de bahía (en la que solo hay un robot) deben colocarse en una zona específica donde pueden pasarlo a otra concreta, para lo que tiene que haber otro robot en el área equivalente de la bahía de destino. Esta operación puede repetirse pasando varios testigos.

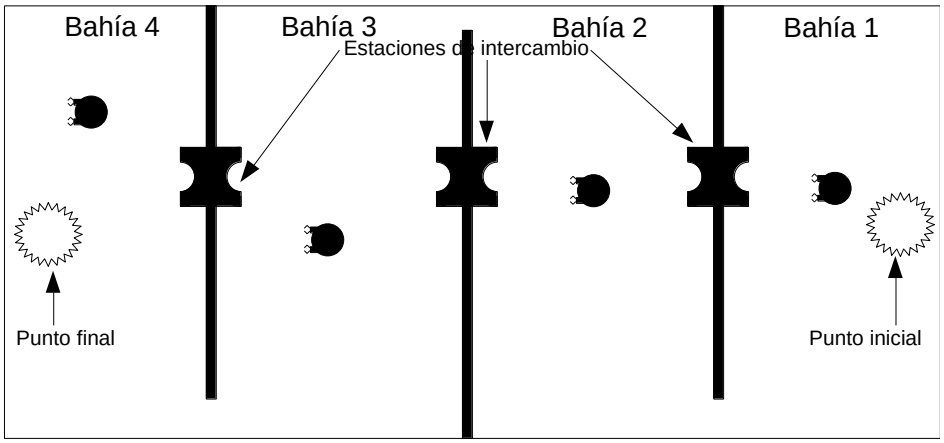


Ilustración 56: Diagrama del paso de testigo

Para este escenario la métrica propuesta por la autora es el número de objetos transferidos del punto inicial al final por unidad de tiempo, que se define como:

$$j(t)/t$$

Donde j(t) es el número de objetos movidos hasta el destino en el momento t.

3.1.1.3.9 Número medio de objetivos observados en seguimiento cooperativo

Otro de los problemas clásicos, definido en [76], es la observación y seguimiento (entendido como la continuación de dicha observación en el tiempo) de varios objetos móviles por un conjunto de robots. Los robots disponen de sensores omnidireccionales con un cierto rango y de un sistema de comunicación cuyo rango es mayor que el del sensor.

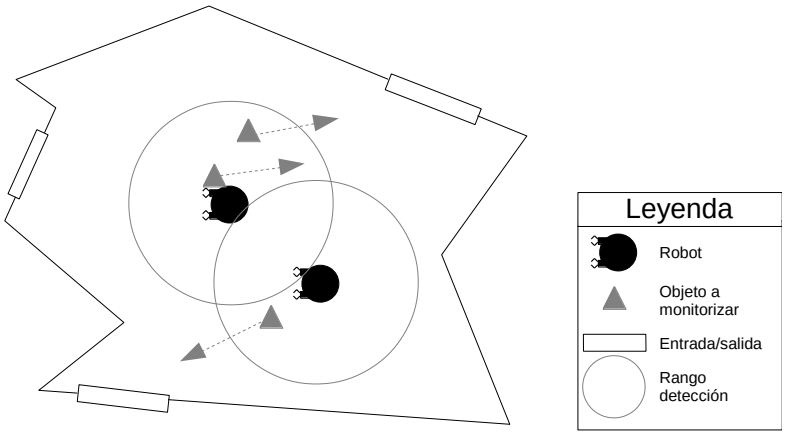


Ilustración 57: Detección de objetos

La métrica propuesta por [70] para este escenario representa el número medio de objetos observados, definida como:

$$A = \sum_{t=0}^{t_{max}} \sum_{j=1}^{n} \frac{g(B(t), j)}{t_{max}}$$

Donde B(t) = $[b_{ij}(t)]_{mxn}$ (m robots y n objetivos); $b_{ij}(t)=1$ quiere decir que el robot i ha detectado el objeto j en el momento t siendo 0 en otro caso; y $g(B(t), j) = 1$ si existe i tal que $b_{ij}(t)=1$ y 0 en otro caso.

3.1.1.3.10 Volumen de material desplazado por unidad de tiempo en excavación colectiva

En este escenario [70], el equipo de robots debe desplazar de forma cooperativa un gran volumen de material (tierra, áridos…).

Para este escenario se propone la métrica de volumen de material por unidad de tiempo que queda definida de la forma:

$$q(t)/t$$

Donde q(t) es el volumen de material desplazado hasta el momento t.

3.1.1.3.11 Tiempo empleado en exploración multi-robot

Otro de los problemas clásicos (al estar incluido en un gran número de actividades) es la exploración colectiva (construcción de un mapa) de un área desconocida por parte de un equipo de robots.

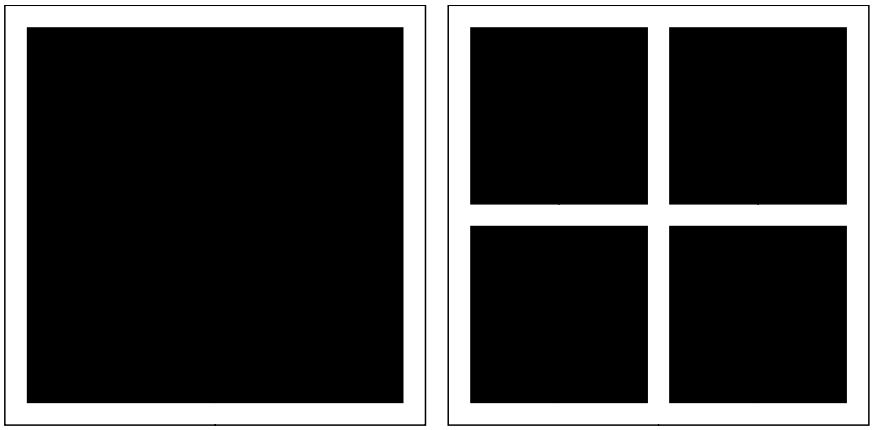


Ilustración 58: Escenarios típicos de exploración bucle y cruce

Para este escenario (que está parcialmente incluido en otros ya descritos) [77] propone varias métricas, siendo la primera propuesta el tiempo empleado en realizar la exploración. Esta métrica se define como: "el tiempo total requerido para completar la misión de exploración" comenzando a medir "el tiempo cuando el equipo comienza la exploración y terminando cuando se alcanza un porcentaje de terreno explorado (por ejemplo: 99%)". Esta métrica podría expresarse también como:

$$t_{max}$$

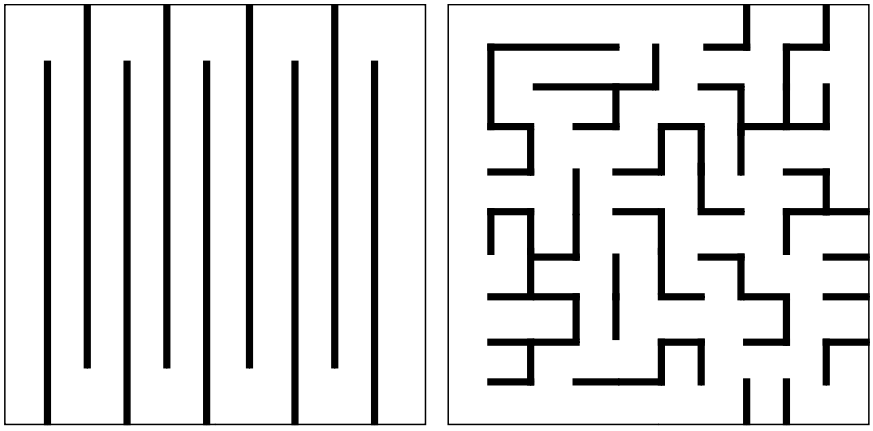


Ilustración 59: Escenarios típicos de exploración zig-zag y laberinto

En [78], donde se realizan varias iteraciones por elemento a comparar, se propone una métrica prácticamente idéntica: tiempo medio de exploración.

3.1.1.3.12 Coste en exploración multi-robot

En el mismo escenario definido en el punto anterior, [77] propone otra métrica: el coste. Esta propuesta se basa en realidad no en una evaluación de coste total de la exploración sino en el consumo de energía y más concretamente lo que los

autores entienden como aquel consumo de energía más impactado por el proceso de exploración: el consumo de energía provocado por el desplazamiento usando como estimador del mismo la distancia recorrida. De esta forma la métrica queda definida por:

$$CosteExploración\ (n) = \sum_{i=1}^{n} d_i$$

Siendo n el número de robots del equipo y di la distancia que ha recorrido el robot i durante la exploración.

3.1.1.3.13 Eficiencia en exploración multi-robot

Otra de las métricas definidas en [77] para el escenario de exploración colectiva es la de la eficiencia entendida como el área explorada por unidad de coste. La formulación de esta métrica quedaría:

$$EficienciaExploración(n) = \frac{M}{CosteExploración(n)}$$

Donde M es el área explorada en metros cuadrados, n es el número de robots y CosteExploración(n) es la métrica de coste anteriormente definida.

Dado que previamente se usa como unidad de coste un estimador de la energía consumida, que es el sumatorio de todas las distancias que se desplazan los robots, esta métrica queda como el área explorada partido de dicho sumatorio:

$$EficienciaExploración(n) = \frac{M}{\sum_{i=1}^{n} d_i}$$

3.1.1.3.14 Completitud del mapa en exploración multi-robot

[77] define la completitud del mapa como la proporción del área explorada:

$$CompletitudMapa = \frac{M}{P}$$

Siendo M la superficie explorada y P la superficie total del mapa.

[78] también propone una métrica similar a la que se refiere como “Maximizar el área total explorada del mapa”.

3.1.1.3.15 Calidad del mapa en exploración multi-robot

La exactitud del mapa con respecto a la realidad del escenario puede ser un problema al existir múltiples causas para los errores: precisión de los sensores, precisión del posicionamiento del robot, diferencias en los datos provenientes de distintos robots…

Como métrica para la calidad, [77] propone el número de aciertos del mismo partido del área total a explorar. Para definir acierto [77] usa una matriz de ocupación para el mapa explorado y el real, siendo un acierto cuando estas dos matrices coinciden y un error cuando son diferentes. La métrica queda definida como:

$$CalidadMapa = \frac{M - A(errorMapa)}{P}$$

Siendo M el área del mapa explorado, P el total del área del mapa y A(errorMapa) el área representada por las celdas de la matriz que contienen errores.

3.1.1.3.16 Número de saltos de comunicaciones en exploración multi-robot

En [78] se propone otra métrica más para la exploración de un área por parte de un equipo multi-robot que mide el coste de las comunicaciones usando como estimador el número de saltos de las mismas (para conectar dos robots puede ser necesario más de un salto, al haber robots intermedios). La métrica se propone como: para todos los pares conectados, el número total de saltos (enlaces entre dos robots) para conectar dos robots concretos en un momento concreto.

Es importante destacar que el número de saltos entre un robot i y otro j no tiene por qué ser el mismo en ambas direcciones.

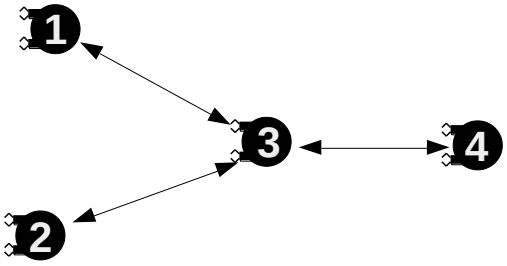


Ilustración 60: Red multi-salto

La definición matemática quedaría como:

$$NúmeroSaltos = \sum_{i=1}^{m}\sum_{j=1}^{m} s(i,j)$$

Donde m es el número de robots y s(i,j) es el número de saltos para una conexión desde el robot i al j.

3.1.1.3.17 Interferencia en recolección de objetos

En otro de los problemas clásicos, la recolección de objetos (foraging) de un entorno desconocido.

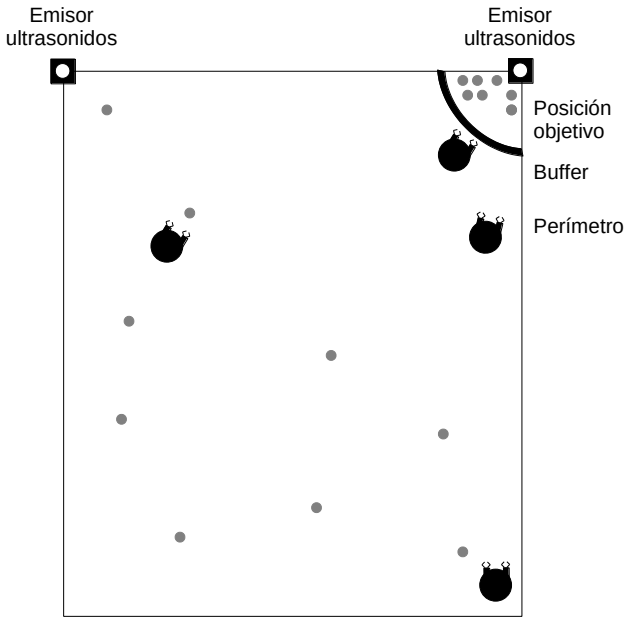


Ilustración 61: Actividad de recolección

[79] propone la interferencia entre robots como métrica para comparar distintos sistemas de control. En concreto, se centra en lo que denomina interferencia física (por ejemplo: la competición por el uso de un espacio determinado). Los autores definen la interferencia característica de un punto concreto en el espacio como la suma, a lo largo de un periodo de tiempo, de todas las interferencias que han ocurrido en ese punto.

Para medir con mayor facilidad la interferencia, los autores, proponen el uso del tiempo empleado por los robots evitando a los otros robots:

$$Interferencia = \sum_{i=1}^{m} \sum_{j=1}^{int(i)} t(i,j)$$

Donde m es el número de robots, int(i) es el número de veces que el robot i ha tenido que esquivar a otro robot y t(i,j) el tiempo empleado por el robot i en la ocasión j.

3.1.1.4 Métricas para tolerancia a fallos

Existen múltiples métricas para medir la posibilidad de que un sistema deje de funcionar correctamente, como el tiempo medio entre fallos, nos centraremos en algunas de las métricas específicas para sistemas multi-robot.

[80, 81] define el rendimiento de un sistema basándose en los siguientes términos que luego se usan para definir métricas de cada uno:

- **Eficiencia**: "Habilidad del sistema para utilizar los recursos disponibles de la mejor forma posible".
- **Robustez**: "Habilidad del sistema para identificar y recuperarse de un fallo".
- **Aprendizaje**: "Habilidad para adaptarse a cambios en el entorno extrayendo e integrando información útil para el sistema durante el curso de la ejecución de la tarea encomendada"

3.1.1.4.1 Métrica de la eficiencia con respecto a la tolerancia a fallos

[82] propone una métrica para la eficiencia basándose en la definición del tiempo empleado:

$$t_j = t_{normal_j} + t_{fallo_j} + t_{recuperación_j}$$

Donde:

- t_{normal_j} es el tiempo que el robot j emplea en estado normal durante la tarea.
- t_{fallo_j} es el tiempo que el robot j emplea en estado de fallo durante la tarea.
- $t_{recuperación_j}$ es el tiempo que el robot j emplea en estado de recuperación durante la tarea.

La métrica de eficiencia (Є) queda definida como:

$$Є = \sum_{j:T_j \in X} \frac{t_{normal_j}}{t_j}$$

Siendo T_j una tarea determinada y t_j y t_{normal_j}los términos anteriormente definidos.

3.1.1.4.2 Métrica de robustez

Con la definición de los posibles fallos durante el intento i de una tarea j (T_j) como la suma de los fallos conocidos ($f^i_{conocidos_j}$), desconocidos ($f^i_{desconocidos_j}$) y no diagnosticables ($f^i_{no\_diagnsticables_j}$):

$$F^i_j = f^i_{conocidos_j} + f^i_{desconocidos_j} + f^i_{no\_diagnsticables_j}$$

La métrica para la robustez de una tarea j (T_j) en el intento i queda definida por [82] como:

$$p^i_j = \frac{f^i_{conocidos_j} + f^i_{desconocidos_j}}{F^i_j}$$

Quedando la métrica para todas las tareas:

$$p_s = \sum_{j:T_j \in Y} \sum_{i=1}^{q_j} p^i_j$$

Donde qj es el número de total de intentos fallidos de la tarea Tj.

3.1.1.4.3 Métrica de aprendizaje

[82] define una métrica para el aprendizaje basada en la ganancia de rendimiento producida por la aplicación del mismo:

$$\delta = P - P'$$

Donde P representa el rendimiento del sistema con aprendizaje y P' el del sistema sin aprendizaje. Los autores definen el rendimiento como:

$$P = \sum_{j:T_j \in X} u_j - \sum_{j:T_j \in y} (c_j * q_j) * (q_j - \sum_{i=1}^{q_j} p^i_j)$$

En esta fórmula:

- Y = {Tj | Task Tj ∈ T ha fallado}.
- X = {Tj | Task Tj ∈ T se ha completado satisfactoriamente}.
- q_j = el número de total de intentos fallidos de la tarea Tj.
- u_j = componente de utilidad.
- p^i_j = robustez de la iteración i de la tarea j.
- c_j = componente de coste del fallo de la tarea j.

3.1.2 Métricas para los individuos

Además de las métricas aplicables al colectivo, que en muchas ocasiones se pueden aplicar a los individuos (como un colectivo formado por un solo miembro), es interesante revisar algunos ejemplos de aquellas que son de aplicación a los individuos en sí o a colectivos con un solo miembro por su posible generalización o aplicación a todos los miembros de un colectivo heterogéneo como comparativa de sus capacidades individuales o la suma de las mismas.

3.1.2.1 Métricas para evaluación estática

Las técnicas para la evaluación estática de los robots, según [83], "se emplean cuando el robot está en una posición estacionaria o se mueve de manera estáticamente estable". Este tipo de evaluación es de especial importancia para los robots dotados de patas.

Para entender por qué unas métricas de estabilidad de robots es de interés para un colectivo de robots debemos pensar en varias de las tareas típicas y características que unen varios robots físicamente como son el auto-ensamblado o el transporte de un objeto de forma colectiva. Estos escenarios puede decirse que hacen equivaler varios de los individuos del sistema multi-robot a un robot en términos de estabilidad.

3.1.2.1.1 Polígono de estabilidad para robots equipados con patas

En [84] analiza las propiedades para la estabilidad de un cuadrúpedo, en lo que es uno de los primeros trabajos aplicables sobre la estabilidad de los robots. Aunque el trabajo está orientado a cuatro patas es fácilmente generalizable. Primeramente, el autor, define una *máquina de movimiento equipada con patas ideal* como "un cuerpo rígido al cual están conectadas un número n de patas sin masa".

Sobre la anterior definición, [84] propone el *polígono (o patrón) de soporte*, "asociado con cualquier fase de un paso de una *máquina de movimiento equipada con patas ideal",* como "el área mínima convexa que contiene todas los puntos de contacto de las patas con el plano de soporte. "

[84] teoriza (y aporta una prueba) que "una *máquina de movimiento equipada con patas ideal* soportada por un plano horizontal es estáticamente estable en un

momento t sí y solo sí la proyección vertical del centro de gravedad está en el interior del *polígono de soporte*".

3.1.2.1.2 Energía necesaria para la desestabilización

[85] define el *margen de estabilidad energético* como "el mínimo de los *niveles de estabilidad energéticos* asociados con cada borde del *polígono de soporte*". Y, a su vez, define un *nivel de estabilidad energético* asociado con un borde del *polígono de soporte* como "el trabajo necesario para rotar el centro de gravedad del cuerpo usando el borde como eje a la posición donde la proyección vertical del centro de gravedad del cuerpo descanse a lo largo de ese borde". Los autores se quedan con un estimador (h) de esta energía al considerar constantes la gravedad y la masa del robot.

$$h = |\overline{\boldsymbol{R}}|\,(1-\cos\theta)\cos\psi$$

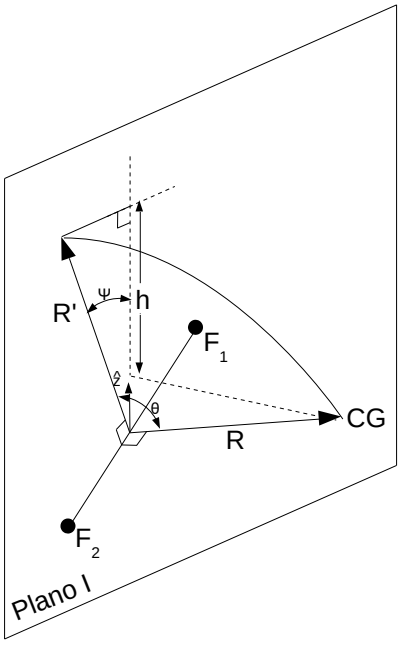


Ilustración 62: Nivel de estabilidad energético

Donde:

- F_1-F_2: Representa uno de las líneas del polígono de estabilidad.
- CG: Centro de gravedad.
- $\overline{\boldsymbol{R}}$: Vector desde F_1-F_2 a CG.
- $\overline{\boldsymbol{R'}}$: Rotación del vector $\overline{\boldsymbol{R}}$ sobre la línea F_1-F_2 hasta estar sobre el Plano I.
- θ: Ángulo entre $\overline{\boldsymbol{R}}$ y $\overline{\boldsymbol{R'}}$.

- ψ: Ángulo entre $\overline{\boldsymbol{R}'}$ y la vertical.

3.1.2.2 Métricas para evaluación dinámica

La evaluación dinámica se encarga de evaluar las capacidades del robot para desplazarse (por ejemplo para superar obstáculos).

De la misma forma que con las métricas de evaluación estática, algunas de las métricas de movilidad son aplicables al colectivo, por ejemplo para medir el incremento de movilidad proporcionado por el auto-ensamblado.

3.1.2.2.1 Curva Característica de movilidad

La MCC (Movility Characteristic Curve por sus iniciales en inglés) [86], es una métrica no dimensional para la superación de obstáculos en una pendiente. El obstáculo se define como un cilindro de longitud infinita y enterrado perpendicularmente a su diámetro un tercio de este. La MCC queda definida por el plano inclinado en el eje horizontal y el diámetro más grande del cilindro que el robot es capaz de superar en dicho plano. La métrica queda definida por el área debajo de la curva.

3.1.2.2.2 Contorno isocrónico / isoenergético

A partir de un punto de origen [87] define varios contornos, siendo cada contorno alcanzable en la misma cantidad de tiempo (para los contornos isocrónicos) o consumiendo la misma cantidad de energía (para contornos isoenergéticos).

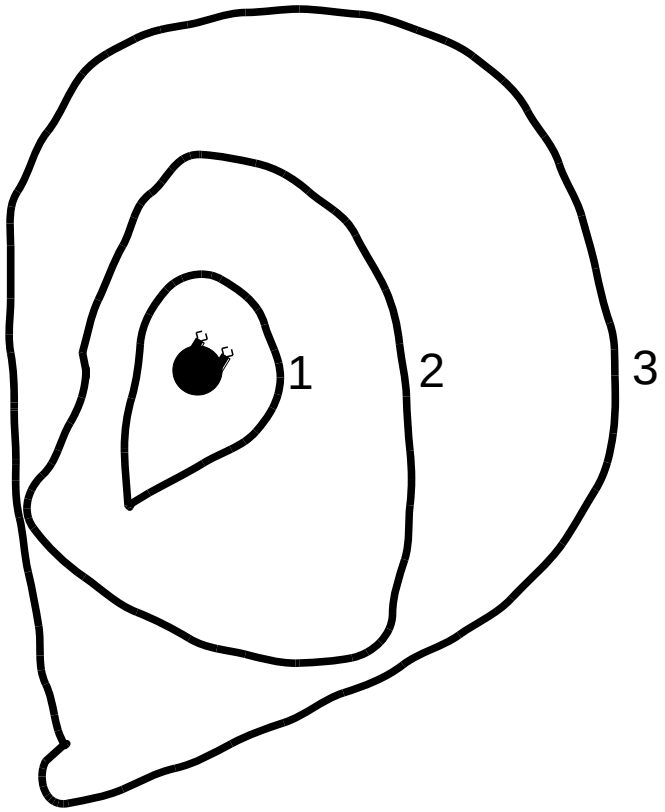


Ilustración 63: Ejemplo de contornos isocrónicos

Estos contornos son de gran interés cuando el colectivo/robot está supervisado por un ser humano al proveer información de las posibilidades alcanzables, lo que permite al usuario decidir qué objetivos proponer al sistema robótico.

3.2 Simuladores

En la actualidad existe un gran número de simuladores que permite la visualización en 2D o 3D de entornos y robots en los que probar una lógica de control. La tendencia destacada por [88] del uso de motores para juegos (que ya incluyen motores de física, de renderización 3D, editores de escenarios y otras herramientas) podría hacer pensar que las simulaciones no son realistas; sin embargo, veremos que los simuladores actuales permiten varios niveles de realismo en la simulación y nos aportan una retroalimentación imposible de obtener en el mundo real.

3.2.1 USARSim

USASim [89, 90, 91] es un simulador basado en el -ya no disponible- Unreal Development Kit (UDK). El UDK se encarga de cargar el mapa y los modelos y, para añadir la lógica de los robots, creamos controladores que se comunican con el motor de Unreal para ejecutar sus acciones y recibir los datos visuales.

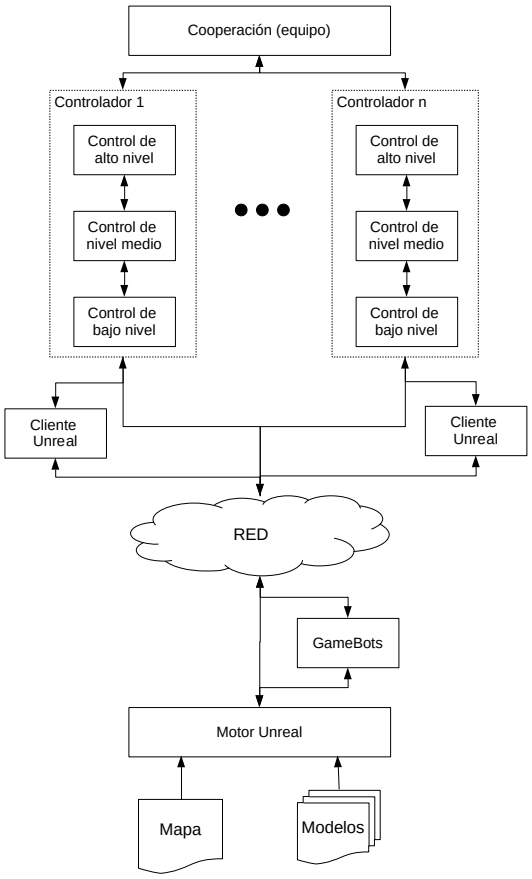


Ilustración 64: Diagrama de bloques de la arquitectura de USARSim

Como vemos en el diagrama de bloques de la arquitectura, USARSim requiere una serie de componentes para una simulación completa de un equipo de robots:

- Motor de Unreal: motor 3D sobre el que se basa el sistema para la renderización del entorno. Unreal dispone de un editor específico para la creación de los escenarios (Unreal Editor).
- Gamebots: una extensión de Unreal [92] modificada por los desarrolladores de USARSim para permitir la comunicación (usando un protocolo de USARSim sobre TCP/IP) con el motor de Unreal (cuyo protocolo es propietario).

- Controlador: es el software que actúa como controlador del robot. Debe conectarse al motor de Unreal, crear el robot y luego recibir los datos de los sensores y enviar ordenes al robot simulado. Este software puede ser implementado usando otros sistemas como MOAST, que veremos más adelante.
- Coordinación: Software que permite la coordinación entre los distintos robots simulados.

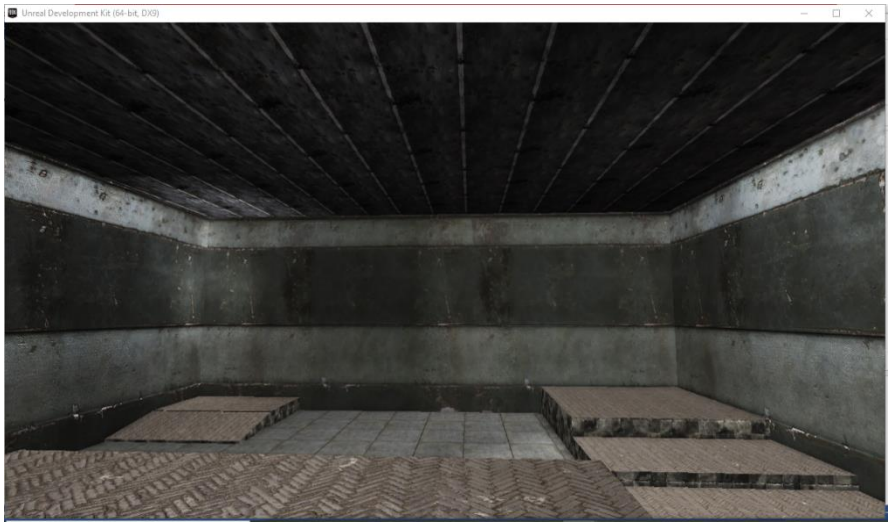


Ilustración 65: Simulación de ejemplo de USARSim

USARSim se centra en la simulación en sí e incluye varios elementos para la misma:

- Entorno: permite la simulación de la escena donde se desarrollará la acción. Incluye: los modelos 3D del propio escenario y los objetos inmutables; los obstáculos (objetos que se pueden mover); luz; efectos especiales (espejos, cristales…); y víctimas que pueden mover los brazos, emitir sonidos, etc. (no hay que olvidar que es un simulador de rescate en entonos urbanos).
- Sensores:
    - Propioceptivos (nivel de batería, estado…).
    - Estimación de posición (coordenadas, orientación, velocidad, dirección).
    - Percepción (sonar, laser, cámaras fijas y orientables).
- Robots: usando el motor Karma incluido en Unreal, los desarrolladores de USARSim, permiten la simulación de robots compuestos de varios

elementos interconectados entre sí por uniones móviles sujetas a la acción de la física.

USARSim es fácilmente conectable con otros sistemas y componentes de simulación pero no puede considerarse un sistema de simulación completo al requerir otros componentes.

3.2.2 MOAST

Mobility Open Architecture Simulation and Tools (MOAST) [93] cuya última versión (3.9) que data de 2012 es un conjunto de herramientas para asistir en simulaciones. MOAST descansa sobre USARSim y tiene una arquitectura altamente modular.

Ilustración 66: Descomposición en módulos de MOAST

Los módulos se agrupan en niveles (echelons) y la mayoría son controlables desde línea de comandos y, además, usando la herramienta RCS Diagnostics (desarrollada en Java por NIST). Estos módulos se intercomunican usando NML (Neutral Message Language), definido en clases C++, y pueden requerir la ayuda de un NMLServer (un proceso que facilita el intercambio de buffers entre módulos).

Los niveles en los que se reparten estos módulos son:

- Nivel de servos: Conexión con el mundo real o con las API's de simulación.
- Nivel de primitivas: Enlace con el sistema nativo de entrada y salida del agente autónomo.

- Nivel de movilidad autónoma: Proporciona la habilidad de seguir un camino o calcular mapas a partir de los datos de los sensores.
- Nivel de vehículo: Comportamientos de alto nivel, interrelación entre las coordenadas locales y globales y otras tareas a nivel de un robot.
- Nivel de sección (equipo): Este es el nivel más elevado y se ocupa de la coordinación de varios robots.

Gracias a este alto nivel de modularidad y al sistema de comunicación entre módulos, MOAST permite la creación de un sistema de evaluación del rendimiento del sistema.

3.2.3 Player/Stage/Gazebo

Stage [94] es una librería en C++ que permite la simulación de múltiples robots móviles en 2D como ha quedado demostrado en [96, 97]. Stage es especialmente interesante para los fines de la presente tesis debido a su uso en simulaciones con una gran cantidad de robots [97].

Player [98] es una capa de abstracción del hardware (HAL por sus siglas en inglés) que permite el desarrollo de sistemas de control aislándonos de un hardware específico. Además, Player proporciona funciones de comunicaciones entre distintos programas de control.

Gazebo [99] es un simulador de multi-robot para interiores y exteriores que se diferencia de Stage en que los entornos son 3D.

Los tres han sido desarrollados como parte del mismo proyecto y se usan en pares (Stage + Player o Gazebo + Player).

3.2.3.1 Stage + Player

Para poder realizar una simulación con Player + Stage es necesaria la creación de ficheros que definan la escena: uno con los elementos usados en la simulación (fichero del mundo, con extensión world); también es posible crear uno o varios archivos incluibles en el archivo del mundo que tendrán la misma sintaxis que este último pero permiten su reutilización (tendrán extensión inc); y por último otro con la información referente al robot (fichero de configuración, con extensión cfg).

Un ejemplo de mapa (archivo de mundo) en donde incluimos la definición de un escenario y una estación de carga:

```
include "map.inc"
include "sick.inc"

interval_sim 20
speedup 100
paused 0

quit_time 1800 # 30 mins of simulated time

resolution 0.02

threads 2

# configure the GUI window
window
(
 size [ 683.000 713.000 ]

 center [ -0.061 -0.975 ]
 rotate [ 0 0 ]
 scale 33.591

 show_data 0
 show_flags 1

 # interval 50
)

# load an environment bitmap
floorplan
(
 name "cave"
 pose [0 0 0 0]
 size [16.000 16.000 0.600]
 bitmap "bitmaps/floorplantexture.png"
)

zone
(
 size [ 2.000 2.000 0.01 ]
 color "green"
 pose [ -7.000 -7.000 0 0 ]
 name "source"

 ctrl "source 1
 update_interval 10000
)

define charge_station model
(
 size [ 0.100 0.300 0.100 ]
 color "purple"

 # side blocks to restrict view angle
 model( color "purple" size [0.100 0.050 0.250] pose [ 0 0.100 0 0 ] )
 model( color "purple" size [0.100 0.050 0.250] pose [ 0 -0.100 0 0 ] )
)

charge_station(  pose [ 7.908 -2.510 0 0 ] )
```

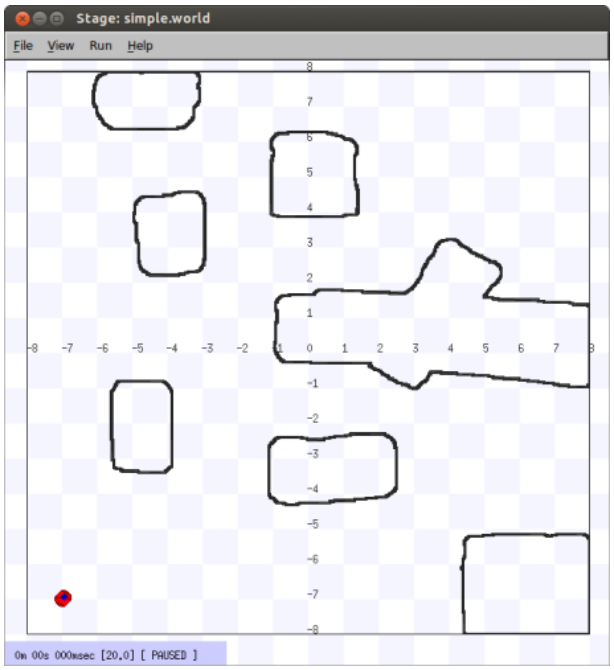


Ilustración 67: Resultado de un mapa simple (imagen extraída de [95])

Ejemplo de archivo de configuración:

```
driver
(
  name "stage"
  plugin "stageplugin"

  provides ["simulation:0" ]
  worldfile "mundo.world"
)
driver
(
    name "stage"
    provides ["6665:position2d:0"
        "6665:ranger:0"
        "6665:blobfinder:0"
        "6665:ranger:1"]
    model "bob1"
)
```

Los robots pueden definirse de la misma forma que los archivos de mundo, por ejemplo:

```
define bob1 position
(
    block
    (
        points 6
        point[0] [0.75 0]
        point[1] [1 0.25]
        point[2] [1 0.75]
        point[3] [0.75 1]
        point[4] [0 1]
        point[5] [0 0]
        z [0 1]
    )
)
```

En este ejemplo se pueden añadir sensores, dispositivos, etc. Para referenciar un dispositivo se usa una dirección en texto.

Para añadir un controlador debemos escribir el código en C, C++ o Python importando las cabeceras del sistema y conectarnos al nombre de máquina y puerto donde está ejecutándose.

3.2.3.2 Player + Gazebo

Esta combinación funciona de forma muy similar a la anterior, requiere la creación de modelos y un fichero de definición del mundo (en este caso son ficheros en XML) para poder ejecutar Gazebo, luego necesitaremos ejecutar player con su fichero de configuración cuya diferencia con el del caso anterior es la inclusión del plugin de gazebo y por último ejecutaremos el software de control del robot en un programa independiente (de la misma forma que en el caso anterior.

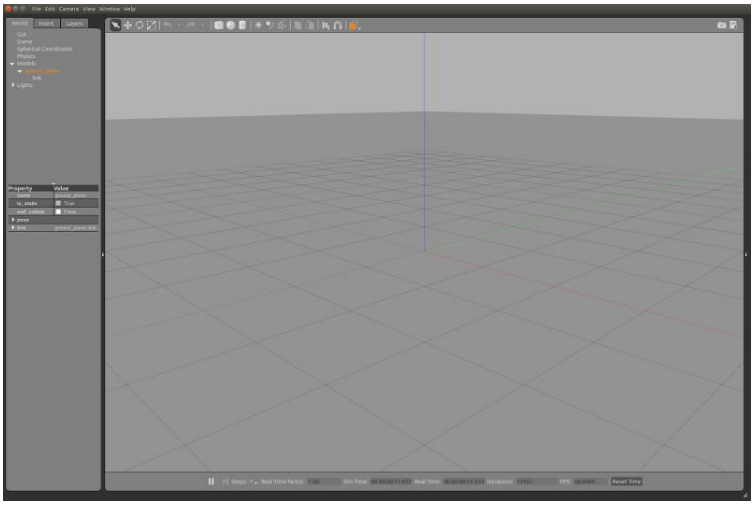


Ilustración 68: Captura de Gazebo (extraída de [99])

3.2.4 Webots

Webots [100] es un simulador de código abierto mantenido por Cyberbotics Ltd cuya última versión data de 2020.

Una simulación usando Webots requiere el diseño del escenario usando un derivado de VRML97 en un archivo con extensión wbt. Este escenario (o mundo como lo denomina Webots) puede crearse usando el interfaz gráfico de Webots o escribiendo directamente el archivo. Un ejemplo de diseño sería:

```
#VRML_SIM R2020b utf8
WorldInfo {
 coordinateSystem "NUE"
}
Viewpoint {
 orientation -0.4756014181398883 0.8130841177483876 0.33570449584218154 1.411501624877876
```

```
  position 1.9535292732913023 2.157853138320272 0.9230463423261032
}
TexturedBackground {
}
TexturedBackgroundLight {
}
RectangleArena {
  floorTileSize 0.25 0.25
  wallHeight 0.05
}
WoodenBox {
  translation -0.193743 0.05 -0.242029
  size 0.1 0.1 0.1
}
WoodenBox {
  translation -0.32997 0.05 0.337663
  name "wooden box(1)"
  size 0.1 0.1 0.1
}
```

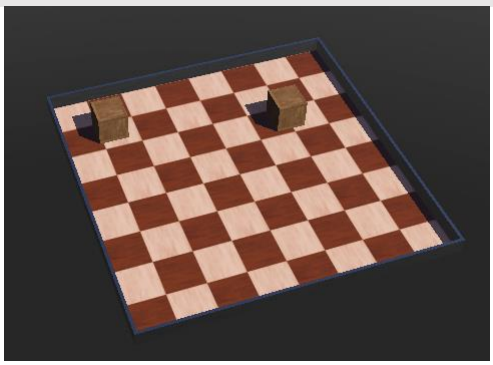


Ilustración 69: Resultado del diseño anteriormente mostrado

Webots también permite simular el controlador del robot escribiéndolo en uno de los lenguajes disponibles (C, C++, Java, Python o Matlab). Básicamente este controlador requiere una función / método que define lo básico de comportamiento:

```
import com.cyberbotics.webots.controller.Robot;
import com.cyberbotics.webots.controller.Motor;

public class epuck_go_forward {

  public static void main(String[] args) {

   int TIME_STEP = 64;

   Robot robot = new Robot();

   // get the motor devices
   Motor leftMotor = robot.getMotor("left wheel motor");
   Motor rightMotor = robot.getMotor("right wheel motor");
   // set the target position of the motors
   leftMotor.setPosition(10.0);
   rightMotor.setPosition(10.0);

   while (robot.step(TIME_STEP) != -1);
  }
}
```

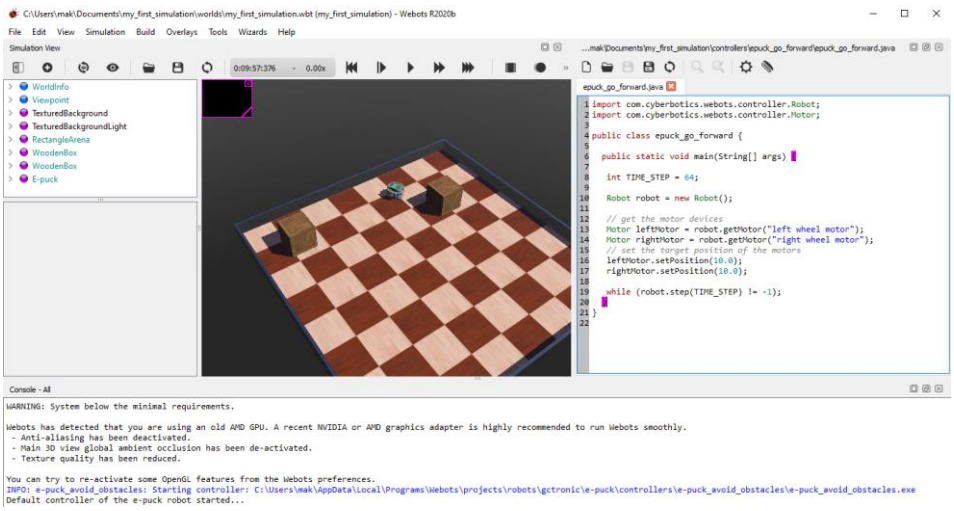


Ilustración 70: Interfaz gráfico de Webots incluyendo el controlador, el escenario y el resultado de la simulación

Los objetos (incluyendo los robots) del escenario son fácilmente extensibles.

Webots es una excelente plataforma de simulación que incorpora diversos robots y dispone de múltiples elementos para el escenario ya creados, todo ello en un entorno de desarrollo integrado. Webots está también integrado con ROS [114].

Es posible añadir métricas al propio interfaz gráfico de Webots mediante la adición de campos en el objeto o robot creado. Además Webots dispone de un sitio web para el aprendizaje (robot benchmark [101]).

3.2.5 Simbad

Simbad [102] es un simulador simple de código abierto desarrollado en Java y Java 3D pensado para entornos educativos y de investigación. Su última versión (1.4) data de 2007.

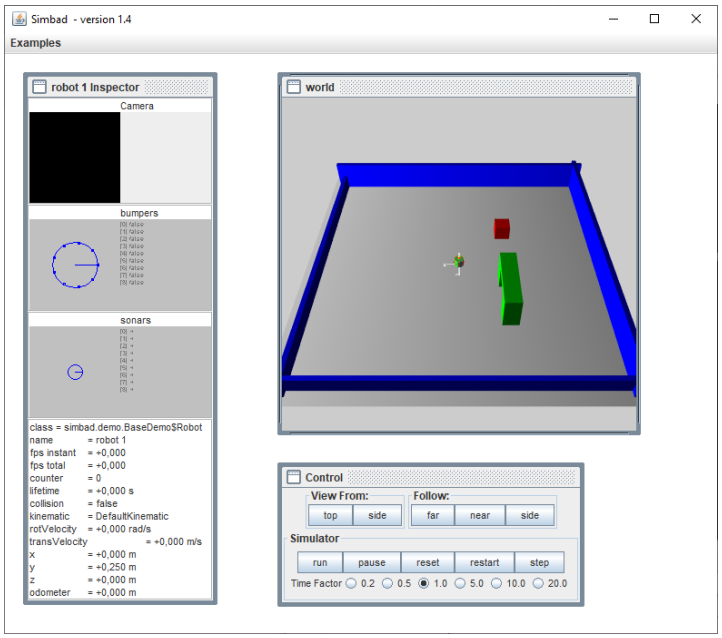


Ilustración 71: Captura de un ejemplo ejecutado con Simbad

Para usar simular un determinado escenario con Simbad es necesario escribir una clase java (o en Python usando Jython) que implemente el controlador del robot y defina el entorno en el que este operará.

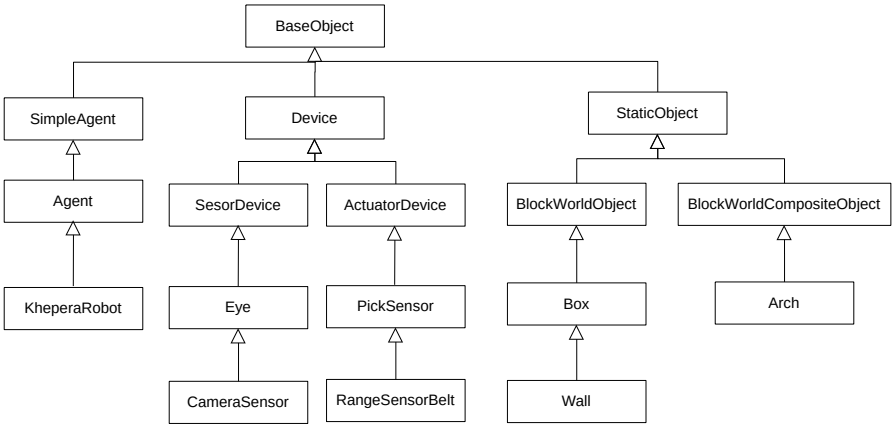


Ilustración 72: Diagrama de clases de parte de los elementos de Simbad

También se pueden desarrollar nuevos sensores y actuadores así como elementos del escenario.

Simbad es un simulador muy básico en 3D que, gracias a su sencillez, permite reescribir cualquier comportamiento o característica pero dispone de una base de elementos implementados bastante reducida.

3.2.6 Eyewyre Simulation Studio

El eyewire Simulation Studio [103] fue un programa de pago que permitía la simulación de la ejecución de código sobre un robot en un entorno 3D. Proporcionaba un motor de física completo para simular las interacciones y posibilitaba la escritura del código del robot en un lenguaje similar a basic, así como la expansión del sistema.

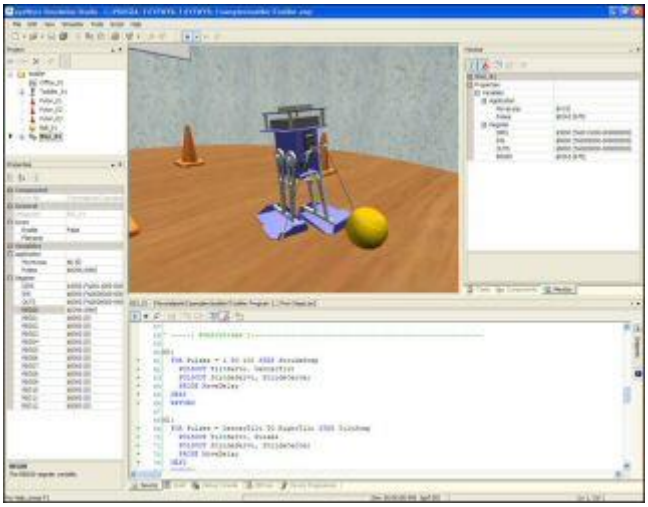


Ilustración 73: Captura de eyewire Simulation Stutio extraída de [102]

Este software ya no está disponible.

3.2.7 Microsoft Robotics Developer Studio

El Robotics Developer Studio de Microsoft [104] es un entorno de desarrollo para Windows cuya última versión (4) es de 2012. El Microsoft Robotics Developer Studio se apoya en el Microsoft XNA (conjunto de herramientas para el desarrollo de juegos) y en el motor de física PhysX de Nvidea.

Este entorno de desarrollo consta de varios componentes [105]: un editor visual para definir los comportamientos de los robots, un simulador y un software de soporte en tiempo de ejecución.

El editor visual está basado en un VPL (Visual Programming Language por sus siglas en inglés) que nos permite desarrollar pequeños programas. Este entorno permite la ejecución del programa y del simulador. También se puede desarrollar usando el Microsoft Visual Studio disponiendo el MRDS de ejemplos en C#.

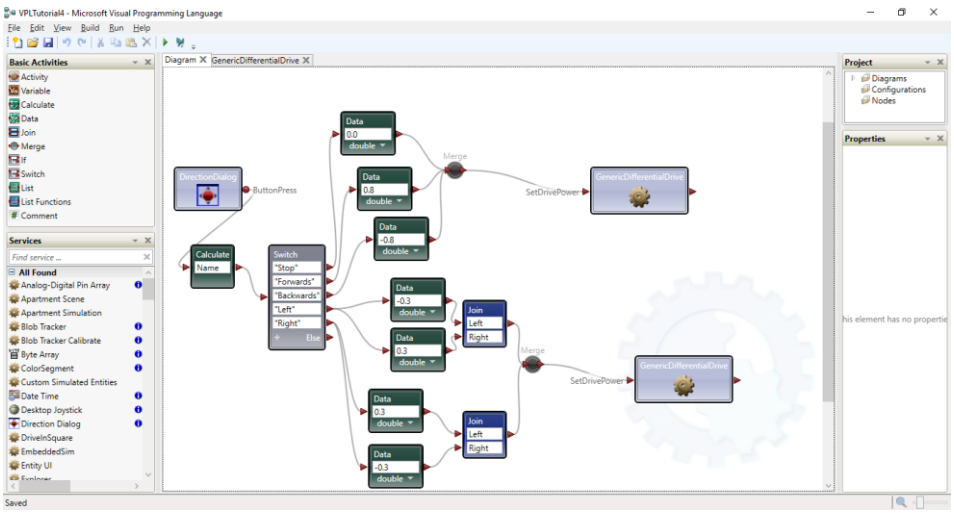


Ilustración 74: Captura del Microsoft Visual Programing Language

El runtime MRDS (por las siglas en inglés de Microsoft Robotics Developer Studio) está compuesto a su vez de dos componentes: el CCR (Concurrency and Coordination Runtime por sus siglas en inglés) que se ocupa de la ejecución concurrente abstrayéndonos de las técnicas de bajo nivel empleadas (hilos, semáforos, …) y el DSS (Decentralized Software Services por sus siglas en inglés) que usa HTTP y se basa en una versión de SOAP con soporte de estado además de orientación a eventos para la comunicación. Los desarrollos en MRDS se basan en servicios que se intercomunican ejecutados sobre los mencionados componentes.

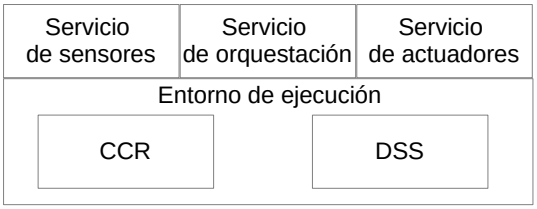


Ilustración 75: Componentes y servicios en un sistema desarrollado en MRDS

El simulador permite la renderización del mundo de varias formas para visualizar la información más importante (visual, física…) así como visualizar en otras ventanas los datos de los sensores. Está integrado con robots basados en LEGO MINDSTORMS NXT (NXT), Creater de iRobot, Mobile Robots de Pioner 3DX y con el brazo LBR3 de Kuka.

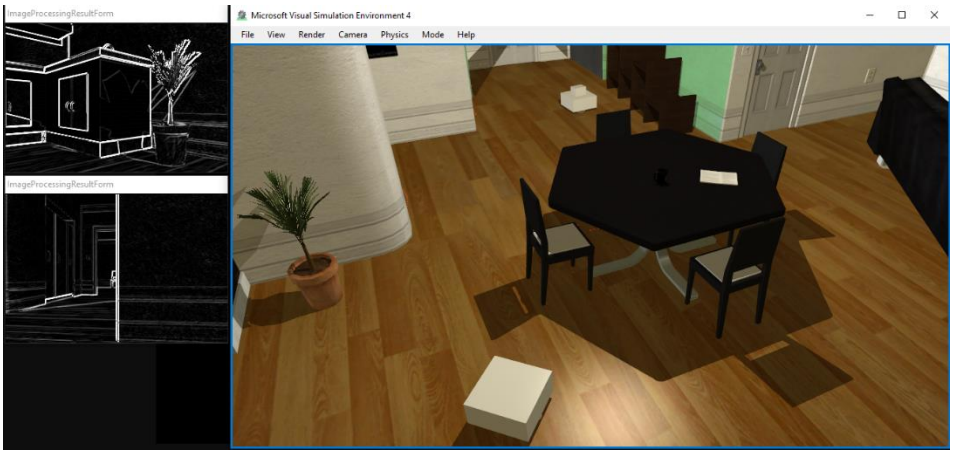


Ilustración 76: Captura del simulador del MRDS

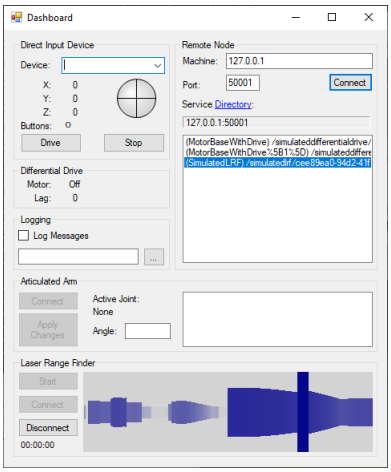


Ilustración 77: Captura del dashboard del simulador de MRDS

MRDS hace muy simple el simular robots ya integrados en escenarios ya definidos. También permite la extracción de información y / o métricas fácilmente mediante la integración de diálogos o la conexión al servicio que las genere.

3.2.8 MissionLab

MissionLab [106] es un conjunto de herramientas cuya última versión (7.0.0) data de 2006. Está integrado con varios robots comerciales: ATRV-Jr y Urban Robot de iRobot, AmigoBot y Pioneer AT de ActivMedia, y Nomad 150 / 200 de Nomadics Technologies.

La principal herramienta incluida en este paquete es *mlab* que interactúa con el propio hardware de los robots o simula sus actuadores y sensores (según sea necesario) además de con el propio programa del robot (el software de control).

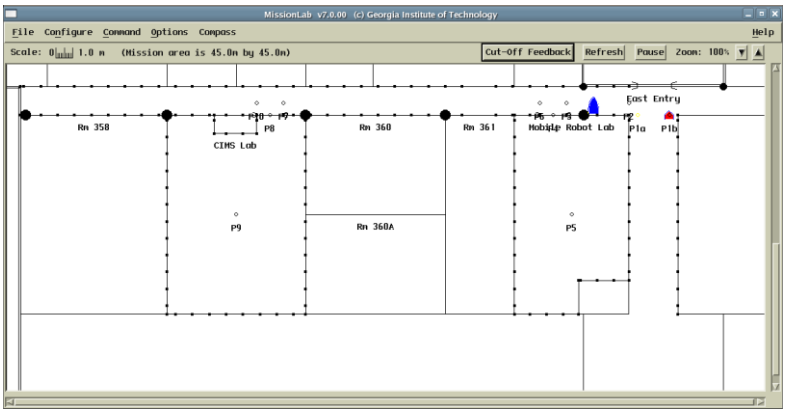


Ilustración 78: Captura de un ejemplo ejecutado con la MissionLab User Interface Console (mlab)

La siguiente herramienta en importancia es *CfgEdit*, el editor de configuraciones, que permite construir comportamientos complejos usando su interfaz gráfico. Los elementos generados pueden ser luego compilados para controlar un robot real o simulado.

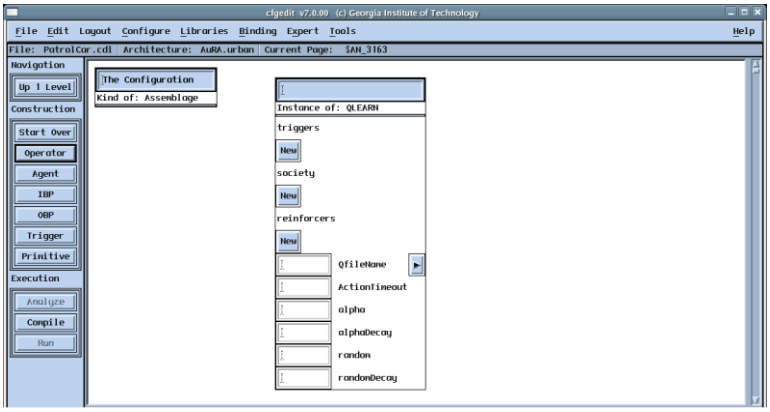


Ilustración 79: Captura de CfgEdit

Otro de los componentes es HServer (Hardware Server) que permite controlar los robots reales accediendo a sus sensores y actuadores.

La última herramienta, el CBRServer (Case Based Reasoning Server) permite generar planes a partir de una base de datos con un histórico de planes exitosos.

Aparte de las herramientas, MissionLab incluye la definición de varios lenguajes:

- CDL: *Configuration Definition Language,* el lenguaje que genera el editor gráfico (CfgEdit) y que puede ser traducido a CNL usando una herramienta auxiliar (llamada cdl).

- CNL: *Configuration Network Language,* un lenguaje similar a C que puede ser usado directamente y que, una vez acabado el programa, puede ser traducido a C++ usando otra herramienta auxiliar (llamada cnl).
- CMDL: *Command Description Language* permite describir planes simples que interactúan con los comportamiento de bajo nivel desarrollados en CNL.
- ODL: *Overlay Description Language* es un lenguaje que permite definir el entorno.

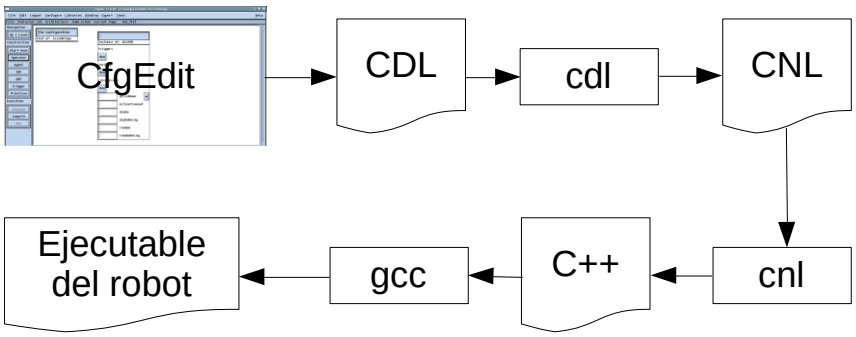


Ilustración 80: Proceso desde la edición con CfgEdit hasta la generación del ejecutable del robot con MissionLab

MissionLab posee un ciclo de vida único en cuanto varias de sus herramientas permiten la construcción de comportamientos complejos para los robots que pueden integrarse directamente en los mismos.

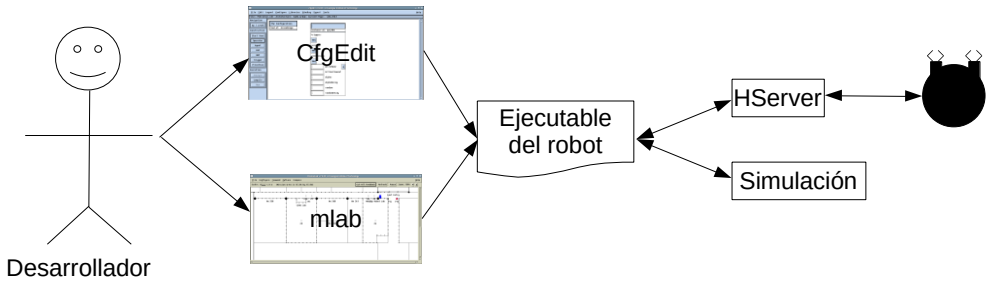


Ilustración 81: Uso típico de MissionLab en donde un desarrollador crea un ejecutable de robot (gracias a las herramientas) y luego lo ejecuta sobre el propio robot o lo simula

Como hemos visto, MissionLab posee un gran potencial para diseñar los comportamientos de los propios robots pero es poco adaptable a otras funciones.

3.2.9 SimRobot

SimRobot [107] es un simulador de código abierto y multiplataforma desarrollado en C/C++ cuya última versión data de 2015. Está orientado a la RoboCup [108]. Las simulaciones descansan sobre ODE (Open Dynamic Engine) que permite la simulación dinámica de cuerpos rígidos.

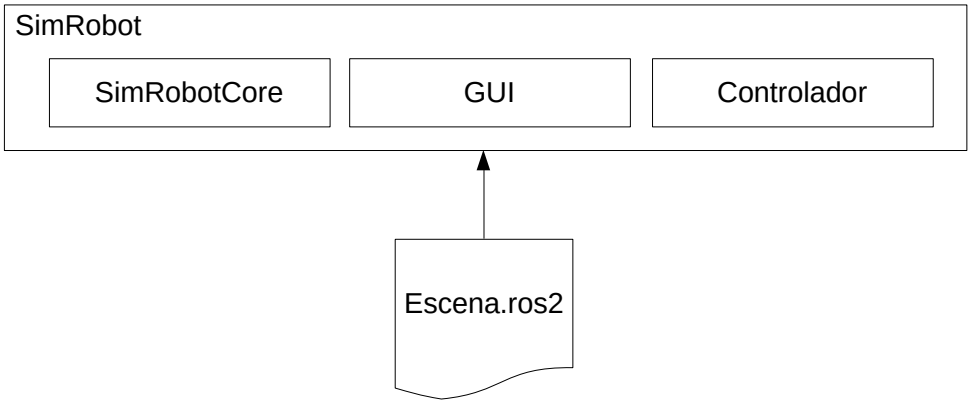


Ilustración 82: Componentes principales de una simulación con SimRobot

Al igual que la mayoría de los simuladores, SimRobot define sus simulaciones en archivos (con extensión ros2) de texto, en este caso, definidos por Robot Simulation Markup Language (RoSiML) [109] una aplicación de XML. Este lenguaje está pensado para permitir el intercambio entre simuladores.

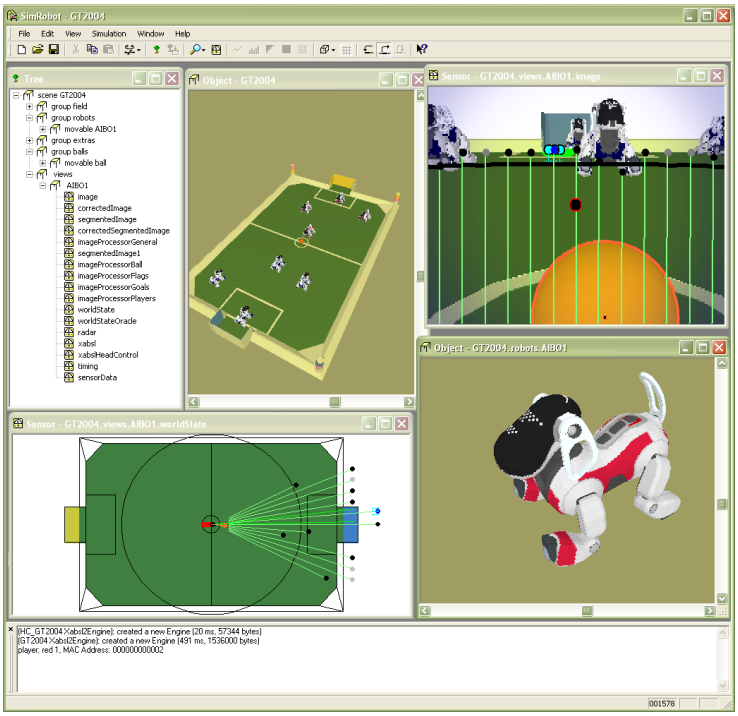


Ilustración 83: Captura del entorno gráfico de SimRobot extraida de [110]

Estos archivos permiten definir superficies, figuras geométricas, los propios robots, etc.

SimRobotCore representa el núcleo del sistema de simulación y es independiente del entorno gráfico que muestra la misma.

El controlador es el elemento que implementa el control de los robots siendo seleccionado en los archivos de descripción de las escenas.

Por último el GUI es el interfaz gráfico dependiente de la plataforma.

SimRobot permite además guardar un archivo de "log" con el resultado de la simulación, lo que permite reproducirla cuando se desee.

Este simulador al proveer todo su código fuente se puede extender para incluir las métricas necesarias en el desarrollo de una determinada simulación, incluso permite la modificación del entorno gráfico. El código descargable [106] no incluye una librería completa de componentes que permita realizar una simulación de un robot complejo pero existen ejemplos de uso con equipos de robots en Internet.

3.2.10 SubSim

SubSim [111] es un simulador orientado a AUVs (Autonomous Underwater Vehicles) de código cerrado desarrollado en C/C++ cuya última versión (2.4A) data de 2009. Está integrado con RoBIOS [112], el sistema operativo / librería para EyeBot. A pesar de que el código parece no estar disponible, SubSim permite su extensión usando plugins en librerías independientes.

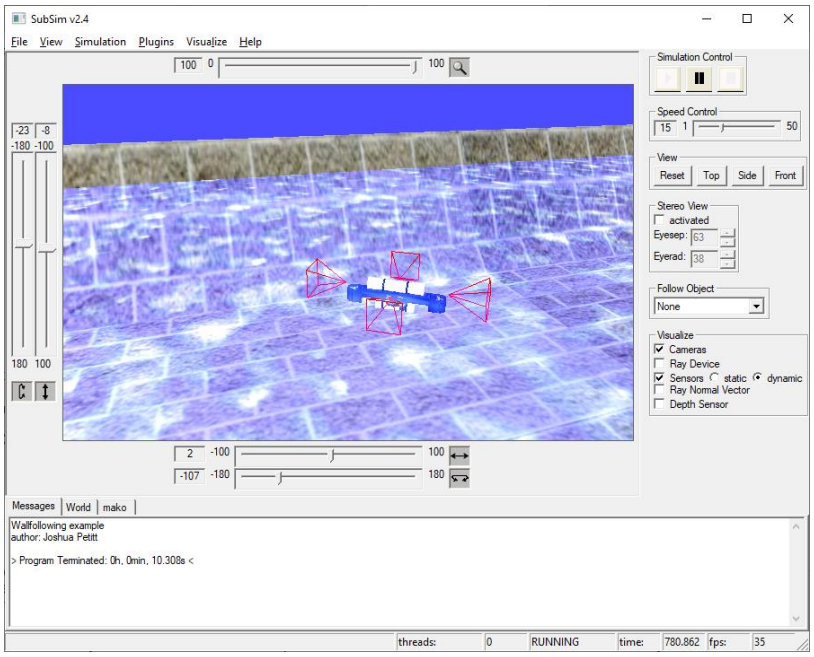


Ilustración 84: Captura de la pantalla principal de SubSim ejecutando uno de los ejemplos (FollowWall)

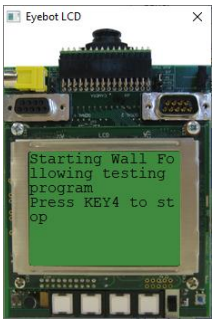


Ilustración 85: Captura de pantalla extra de SubSim ejecutando uno de los ejemplos (FollowWall)

Al igual que el resto de simuladores, SubSim usa un archivo para definir el escenario que divide en tres partes: mundo, submarino (el AUV) y objetos. Todos ellos son archivos XML separados, a pesar de que el archivo principal tenga extensión "sub".

El "mundo" define el escenario global en donde se desarrolla la simulación y puede tener 3 elementos:

- **water**: Define el tamaño y la apariencia (textura) del agua.
- **pool**: Define el tamaño y la apariencia (textura) de la "piscina" (muros) que rodea al agua.

- **terrain**: Permite definir, usando un BMP en donde el blanco representa la máxima altura y el negro la máxima profundidad, la altura del suelo marino. Además se puede incluir la textura.

El (o los) submarino(s) permite(n) definir la posición inicial, la librería de lincado dinámico que incluye el control, un archivo con la definición 3D y otro archivo XML que define al submarino incluyendo:

- Formas genéricas (todas ellas con con sus dimensiones y peso): **Box**, **Sphere** y **Cylinder**
- Sensores:
  - **Inclinometer**: Ángulo de orientación.
  - **Gyroscope**: Velocidad angular del objeto.
  - **PSD**: (Position Sensitive Device por sus siglas en inglés) devuelve la distancia al objeto más próximo en la línea de visión del sensor.
  - **Contact**: Devuelve verdadero si el submarino ha entrado en contacto con otro objeto.
  - **Velocimeter**: Velocidad lineal del submarino en la dirección especificada.
  - **Camera**: Cámara que se desplaza con el submarino.
- Actuadores:
  - **Propeller**: Sistema de propulsión del submarino que incluye su posición, orientación y el factor que multiplica al voltaje suministrado para indicar el empuje obtenido.
  - **Hydrofoil**: Simula el empuje y arrastre de las superficies de control.
  - **Liquid Drag**: Simula el efecto de arrastre aplicado al cuerpo por su desplazamiento en el agua.
  - **Buoyancy**: simula la flotación del cuerpo.

Por último se permite definir objetos del entorno con su modelo y posición.

SubSim no permite modificar todos los comportamientos del sistema y cualquier métrica que se quiera aplicar debe realizarse como un plugin externo que no siempre tiene que tener acceso a toda la información del simulador.

3.2.11 TeamBots

TeamBots [113] es un conjunto de componentes y herramientas de código abierto, desarrollado en Java y orientado a la investigación en robótica. Su última versión (2.0e) data de abril de 2000. Sus principales funcionalidades son el uso con robots reales (en concreto Nomad 150 y llamado TBHard) y la simulación (denominado TBSim) de estos mismos robots en un entorno.

TBSim permite simular equipos de robots operando en un entorno bidimensional. Para ello hay que definir el escenario, el equipo de robots, los objetos que se sitúan en el escenario y los parámetros de tiempo, tamaño del escenario etc. Todos estos elementos deben definirse en un archivo (con extensión dsc) que es el que usa como entrada TBSim para iniciar su ejecución. Este archivo permite definir las siguientes entidades:

- Elementos de la simulación:
    - **robot** robottype controlsystem x y theta forecolor backcolor visionclass:
        - robottype: Full Quallified Name (FQN) de la clase que representa el robot.
        - controlsystem: Full Quallified Name (FQN) de la clase que representa el control del robot.
        - x, y, theta: las coordenadas donde se sitúa el robot.
        - forecolor backcolor visionclass: características que definen cómo se muestra el robot tanto en el simulador como a los otros robots.
    - **object** objecttype x y theta radius forecolor backcolor visionclass: con parámetros similares al robot, representa un objeto sin sistema de control (obstáculos, elementos a recolectar…).
- Parámetros de control del tiempo:
    - **maxtimestep** milliseconds: tiempo máximo entre puntos de la simulación.
    - **time** accel_rate: velocidad de paso del tiempo con respecto al tiempo real.

- Parámetros de visualización / entorno:
  - **background** color: define el color de fondo durante la simulación.
  - **windowsize** width height: tamaño inicial de la ventana.
  - **bounds** left right bottom top: límites del entorno simulado.

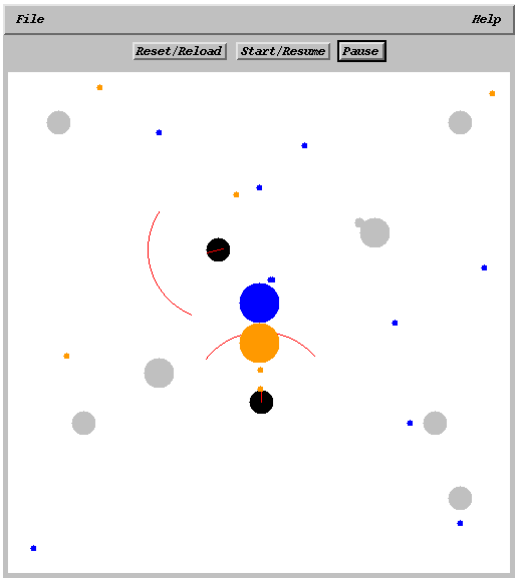


Ilustración 86: Tarea de recolección en donde los dos robots (representados por círculos negros) deben colocar los elementos naranja en la zona naranja en el centro del escenario y los puntos azules en el área azul (imagen extraída de la documentación de TeamBots)

TeamBots no dispone de componentes para una evaluación en tiempo real de métricas o eventos del propio escenario. Sin embargo, es un sistema muy simple para realizar simulaciones rápidas y puede ser fácilmente extendido con los comportamientos que se desee.

3.2.12 Rossum's Playhouse

Rossum's Playhouse (RP1) [114] es un simulador muy básico en dos dimensiones pensado para ayudar a los desarrolladores de software de control robótico. Los últimos desarrollos datan de 2013 y la última versión (0.61) de 2009. Está desarrollado en Java aunque dispone de un API para C/C++.

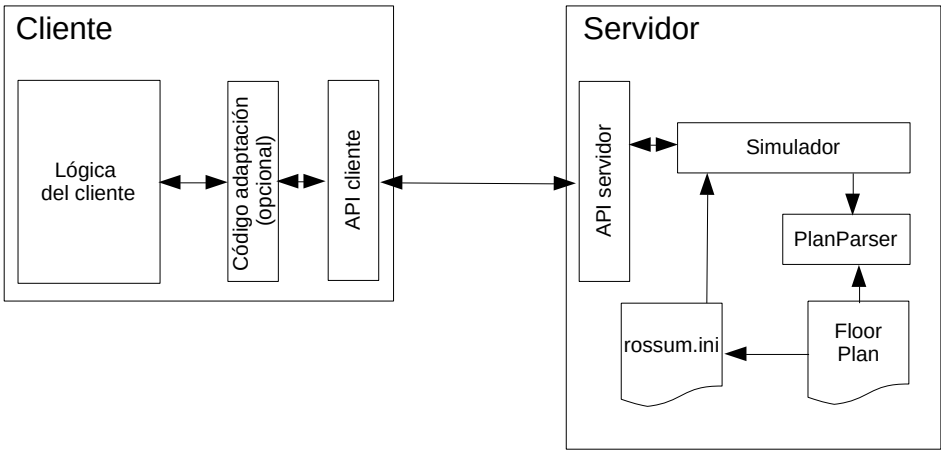


Ilustración 87: Diagrama de bloques de Rossum's playhouse

Rossum's playhouse tiene una arquitectura cliente-servidor ejecutándose la simulación en lo que los autores definen como servidor y la lógica del robot en lo que definen como cliente. En el servidor se cargan una serie de propiedades entre las que se encuentra el nombre del fichero que sirve como plano bidimensional. Este plano permite definir:

- Muros: con 2 coordenadas y su grosor.
- Obstáculos: con su geometría, alcance y orientación.
- Objetivos: con sus coordenadas y tamaño.
- Localización: coordenadas de posición del robot.
- Elementos de navegación:
  - Nodos: Con sus coordenadas y nombre.
  - Enlaces: Entre los nodos

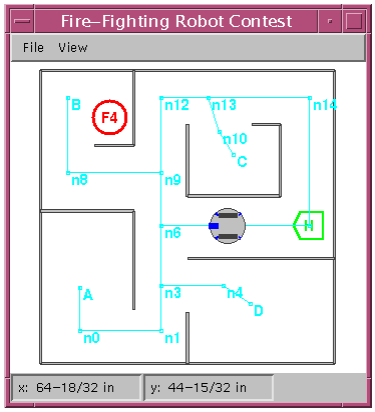


Ilustración 88: Representación de uno de los ejemplos de Rossum's playhouse (extraída del manual de la aplicación)

Cuando creamos el cliente podemos definir además la forma con la que se renderiza el robot. Esta forma debe enviarse al servidor así como registrarse los manejadores de los distintos eventos.

Rossum's Playhouse no ha tenido mantenimiento desde 2009 y varias de las herramientas (como el MapViewer) ya no están disponibles en los enlaces que provee. La arquitectura permite establecer métricas en tiempo real aunque no dispone de un soporte específico para ello.

3.2.13 MORSE

Los creadores de MORSE (Modular OpenRobots Simulation Engine por sus siglas en inglés) [115, 116] lo definen como un simulador genérico para robótica académica basado en el motor de Blender.

MORSE proporciona una serie de componentes que están en su mayor parte escritos en Python (con excepción de los computacionalmente extensivos como los que se ocupan de la renderización 3D). Además se integra (a diferentes niveles) con varios middlewares: ROS [117], YARP [118], Pocolibs [119], MOOS [120], HLA, Mavlink [121] y PPRZLink [122]. MORSE proporciona integración con ellos mediante la definición de dos tipos de elementos:

- Overlay de componentes: permiten adaptar servicios de la arquitectura concreta del robot (sensores y actuadores) a MORSE.
- Gestores de datastreams: representan streams de datos que tienen origen o destino en la arquitectura robótica.

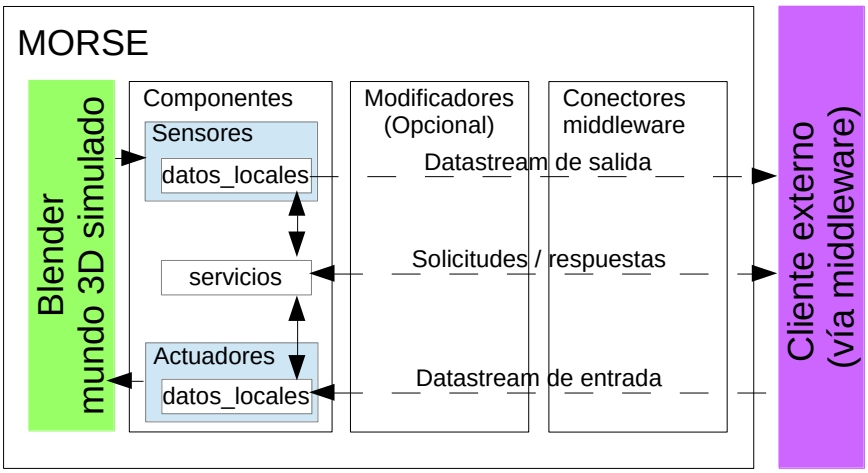


Ilustración 89: Diagrama de bloques de MORSE

La librería de componentes contiene:

- La implementación de distintas bases robóticas.

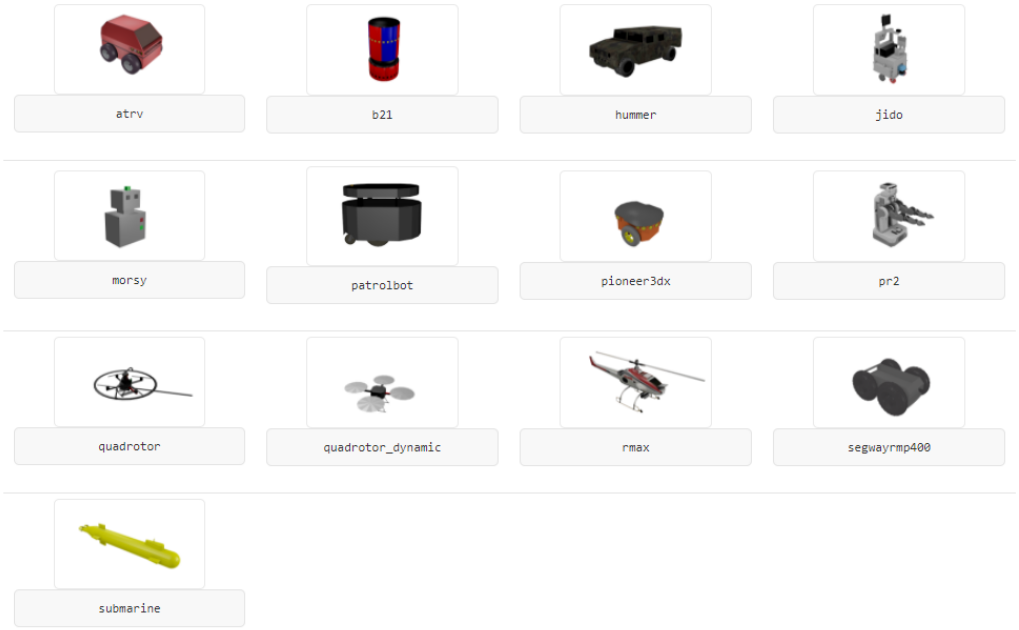


Ilustración 90: Robots y bases robóticas disponibles (imagen capturada del sitio oficial de MORSE)

- Distintos tipos de sensores: Acelerómetro, medidor de la velocidad del aire, sensor de posición, sensor de altura, barómetro, nivel de batería, , cámara genérica, reloj, detector de colisión, sensor compuesto, cámaras con eje de profundidad (XYZ), GPS, giróscopo, sensor Hokuyo, postura humana, unidad de medición inercial, sensor de proximidad por infrarrojos, Kinect, escáner láser, magnetómetro, postura humana (usando kinect), odometría, postura, sensor de proximidad, sensor de postura PTU (Pan Tilt Unit), altímetro por radar, sensor de búsqueda y rescate, cámara semántica, SICK (escáner láser), SICK LD-MRS (escáner láser), cámara estereoscópica, termómetro, velocímetro y cámara de vídeo.
- Distintos tipos de actuadores: actuador combinación de rotación y desplazamiento, ArUco Marker (postura de la cámara), destino, arrastrar, torque externo, controlador de movimiento basado en torque, pinza, actuador Joystick, actuador teclado, KUKA LWR, luz, controlador mocap, actuador de orientación, Mitsubishi PA-10, unidad de giro

horizontal y vertical, controlador dinámico para quadcóptero, control de movimiento de postura de rotor, control de movimiento de velocidad de rotor, control de movimiento de punto de ruta para rotor, sonido, estabilizador de vuelo para quadcópteros, actuador de fuerza/giro, teletransporte, actuador de velocidad angular y lineal, motor diferencial, actuador de velocidad angular, lineal y punto de ruta.

Para comunicarse con elementos de la simulación, MORSE hace uso intensivo de sockets, lo que permite controlar la simulación desde cualquier programa o sistema externo.

Para construir una simulación solo hay que crear un pequeño archivo en Python definiendo el escenario, los robots que participan y otras características (cámara, sockets,…). El ejemplo más simple (construido automáticamente) crea el siguiente archivo:

```
#! /usr/bin/env morseexec

""" Basic MORSE simulation scene for <phd_example> environment

Feel free to edit this template as you like!
"""

from morse.builder import *

# Add the MORSE mascott, MORSY.
# Out-the-box available robots are listed here:
# http://www.openrobots.org/morse/doc/stable/components_library.html
#
# 'morse add robot <name> phd_example' can help you to build custom robots.
robot = Morsy()

# The list of the main methods to manipulate your components
# is here: http://www.openrobots.org/morse/doc/stable/user/builder_overview.html
robot.translate(1.0, 0.0, 0.0)
robot.rotate(0.0, 0.0, 3.5)

# Add a motion controller
# Check here the other available actuators:
# http://www.openrobots.org/morse/doc/stable/components_library.html#actuators
#
# 'morse add actuator <name> phd_example' can help you with the creation of a custom
# actuator.
motion = MotionVW()
robot.append(motion)


# Add a keyboard controller to move the robot with arrow keys.
keyboard = Keyboard()
robot.append(keyboard)
keyboard.properties(ControlType = 'Position')

# Add a pose sensor that exports the current location and orientation
# of the robot in the world frame
# Check here the other available actuators:
# http://www.openrobots.org/morse/doc/stable/components_library.html#sensors
#
# 'morse add sensor <name> phd_example' can help you with the creation of a custom
# sensor.
pose = Pose()
```

```
robot.append(pose)

# To ease development and debugging, we add a socket interface to our robot.
#
# Check here: http://www.openrobots.org/morse/doc/stable/user/integration.html
# the other available interfaces (like ROS, YARP...)
robot.add_default_interface('socket')


# set 'fastmode' to True to switch to wireframe mode
env = Environment('sandbox', fastmode = False)
env.set_camera_location([-18.0, -6.7, 10.8])
env.set_camera_rotation([1.09, 0, -1.14])
```

Este archivo a su vez permite ejecutar la siguiente simulación:

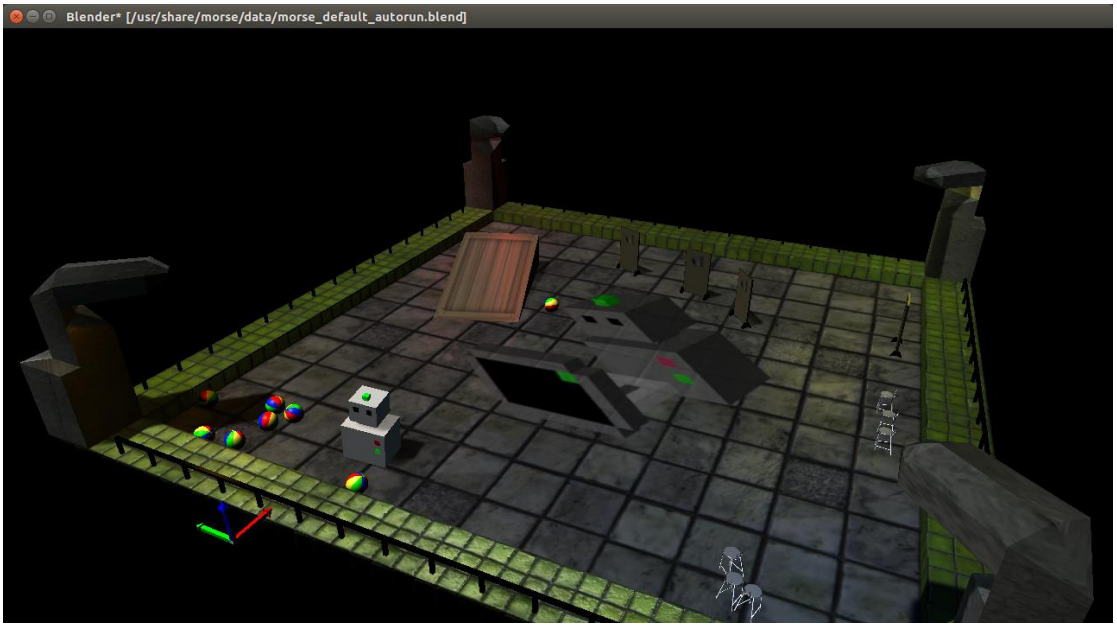


Ilustración 91: Captura de MORSE ejecutando el ejemplo generado automáticamente

MORSE no posee componentes específicos para la evaluación, pero son fácilmente integrables gracias al diseño del simulador. La capacidad de simular escenarios complejos queda demostrada en [123]. La última versión disponible (1.4) de MORSE data de 2016.

3.2.14 LogReader

LogReader [124] pretende dar solución a la problemática de revisar los resultados de simulaciones o escenarios desarrollados con robots reales desarrollando "una herramienta que permite renderizar los logs guardados por un robot (o una secuencia de ordenes generada por un algoritmo o un usuario) en tres dimensiones". LogReader está enteramente desarrollado en Java y, a pesar de su objetivo, puede operar con varios individuos. Su última versión data de 2013.

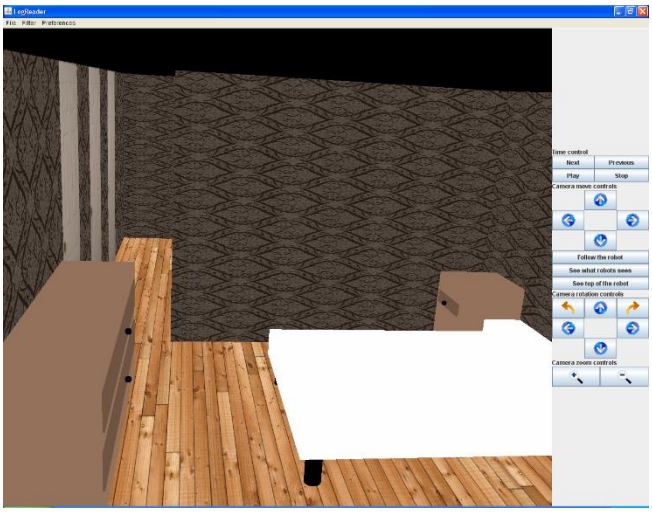


Ilustración 92: Captura de uno de los escenarios de ejemplo de LogReader

Al igual que otros ejemplos discutidos en este capítulo, LogReader permite la carga de un fichero con la definición del escenario que además puede contener la secuencia de acciones. La operativa de trabajo para la simulación implica la definición del escenario, la ejecución de la simulación y la carga de los resultados de esta última y de la definición del escenario en LogReader.

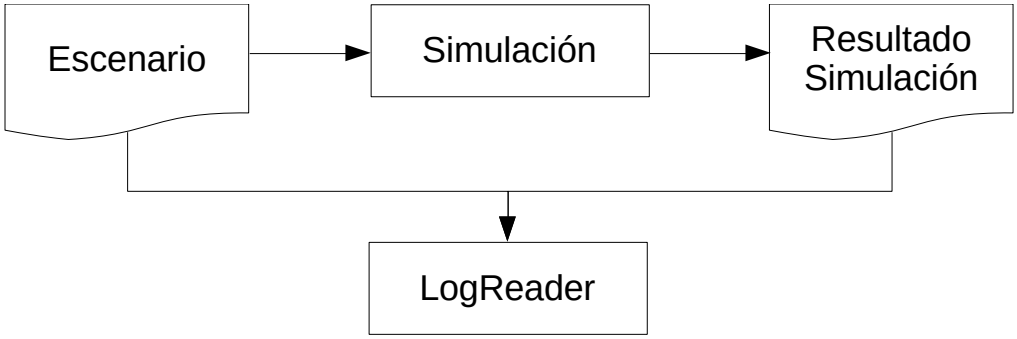


Ilustración 93: Flujo de trabajo con LogRead

El archivo de intercambio permite la definición de los siguientes elementos:

- RealBeacon: Una baliza real con su ubicación, orientación y tamaño reales.
- DetectedBeacon: Una detección de una baliza con su ubicación, orientación y tamaño detectadas y el momento en el que se detecta.
- Robot: Un robot con su ubicación, orientación y tamaño y el momento en el que ocupa esta posición.
- Plane: Superficie en 3D con su ubicación, orientación, tamaño y propiedades visuales.

- Line: Línea en 3D con su inicio, fin, grosor y propiedades visuales.
- Cone: Cono en 3D con su ubicación, orientación, tamaño y propiedades visuales.
- Cylinder: Cilíndro en 3D con su ubicación, orientación, tamaño y propiedades visuales.
- Ellipsoid: Elipsoide en 3D con su ubicación, orientación, tamaño y propiedades visuales.
- Prism: Prisma rectangular en 3D con su ubicación, orientación, tamaño y propiedades visuales.
- Tetrahedron: Tetraedro en 3D con su ubicación, orientación, tamaño y propiedades visuales.

LogReader representa una herramienta de supervisión útil tanto en tiempo real de la simulación como tras la misma para revisar su resultado. Es fácilmente extendible para definir tanto nuevos elementos gráficos como redefinir el sistema de lectura o conectarse con otros sistemas. También se puede modificar para incluir ciertas métricas de forma automatizada. Su principal defecto es la ausencia de capacidades de simulación física.

3.3 Conclusiones

Como hemos visto, existe un gran número de métricas para problemas específicos que podrían aplicarse a sistemas finales resolviendo dichos problemas y de la misma forma existen métricas para dichos sistemas sin importar el problema que traten de resolver. Sin embargo, apenas hemos encontrado métricas aplicables a la arquitectura en sí, por lo que aplicar estas métricas para tratar de comparar arquitecturas requeriría usar una implementación equivalente de todo el resto de elementos implicado y esto no siempre es posible.

En cuanto a los simuladores, existen múltiples opciones con diversos niveles de realismo visual y físico. Siguiendo el sistema de clasificación de [88] para los simuladores y basándonos en la valoración ya realizada en dicho artículo obtenemos la siguiente tabla:

Simulador	Última versión	Open Source	Fidelidad física	Fidelidad funcional	Facilidad de desarrollo
USARSim	1.4 (2013)	Sí	Alta	Media	Media
MOAST	3.9 (2012)	Sí	-	Media	Baja
Player + Stage	3.1.1 + 4.3.0	Sí	Media	Baja	Alta
Player + Gazebo	3.1.1 + 11.0.0	Sí	Alta	Baja	Alta
Webots	R2020b (2020)	Sí	Alta	Media	Alta
Simbad	1.4 (2009)	Sí	Media	Baja	Media
Eyewire Simulation Studio	-	-	-	-	-
MRDS	4 (2012)	No	Alta	Alta	Media
MissionLab	7.0.0 (2006)	Sí	Media	Baja	Media
SimBot	2015	Sí	Media	Baja	Baja
SubSim	2.4a (2009)	No	Media	Baja	Media
TeamBots	2.0e (2000)	Sí	Baja	Baja	Alta
Rossum's Playhouse	0.61 (2009)	Si	Baja	Baja	Alta
MORSE	1.4 (2016)	Sí	Alta	Media	Alta
LogReader	2013	Sí	Baja	Baja	Alta

Tabla 3: Comparativa de simuladores (parcialmente basada en [88])

A pesar de la existencia de varios simuladores cuyo nivel de fidelidad a la realidad es muy alto, la naturaleza de la ejecución de partes del software que debería realizarse en el hardware embarcado en el propio robot, en vez de en un equipo de simulación, resta realismo a los resultados.

Por otra parte, es sorprendente la ausencia de un sistema de evaluación para la comparación ya integrado en los simuladores.

Como podemos ver en la tabla anterior (Tabla 3), de las quince opciones valoradas solo tres simuladores que gozan de la valoración más alta presente (2 elementos valorados como alta y uno en media): Webots, MRDS y MORSE. De estas tres opciones seleccionamos MORSE como herramienta para las simulaciones en la presente tesis por tres razones:

- Integración con un mayor número de plataformas robóticas. Lo que permite aplicar aquello construido sobre una variedad mayor de robots.

- Facilidad de desarrollo tanto del escenario como de la integración de las arquitecturas. Esto reduce el coste de desarrollo centrando el esfuerzo en lo esencial.
- Está integrada como paquete en varias distribuciones Linux. Permite una instalación rápida y sencilla de nuevo centrando el esfuerzo en lo esencial.

Parte II

Metodología y evaluación

Capítulo 4

Sistema de evaluación

Como hemos visto en el capítulo anterior existe una gran cantidad de métricas orientada a soluciones concretas y problemas clásicos así como una gran cantidad de simuladores disponibles. Pero la aplicación directa para poder cuantificar las bondades de una arquitectura no es posible debido a que una arquitectura no es un sistema directamente ejecutable.

4.1 Metodología de evaluación

4.1.1 Objetivos del sistema de evaluación

El objetivo del sistema de evaluación es permitir seleccionar la arquitectura que mejor se adapte a las necesidades específicas del proyecto. Este objetivo general se convierte en los siguientes objetivos específicos:

- Valoración cuantitativa de las principales características de una arquitectura.
- Valoración cuantitativa de las capacidades de implementación de sistemas con la arquitectura.
- Esquema de implementación integrado en el sistema para las arquitecturas de modo que:
    - Sea posible implementar una arquitectura de forma simple.
    - La implementación quede integrada para su evaluación o ejecución del sistema.
- Disponer de un conjunto de valores cuantitativos como resultado que permitan:
    - La comparación entre arquitecturas.

  - La evaluación de la aplicabilidad de una arquitectura a un problema.
- Identificar las debilidades de una arquitectura.
- Automatización de todas las tareas que lo permitan.

4.1.2 Diseño del sistema de evaluación

Una tentación para definir el sistema de evaluación es dividir las arquitecturas en sus elementos principales, medir de forma independiente cada uno de estos elementos y por último asignar un conjunto arbitrario de pesos agregando el resultado. Otra podría ser la aplicación de una serie de métricas para soluciones finales. Ninguna de estas opciones es la ideal ya que la primera desprecia la característica multidimensional de la arquitectura y la segunda es altamente dependiente del problema e implementación concretos.

Para crear una evaluación que permita realmente comparar las arquitecturas seguiremos un doble camino: por un lado, definiremos las dimensiones más importantes y sus respectivas métricas pero sin aplanar estos resultados en un solo valor (lo que nos permite evaluar las arquitecturas sin tener en cuenta particularidades de la implementación) y, por otro lado, definiremos métricas sobre sistemas que resuelvan problemas concretos que nos permitan evaluar la aplicación de la arquitectura.

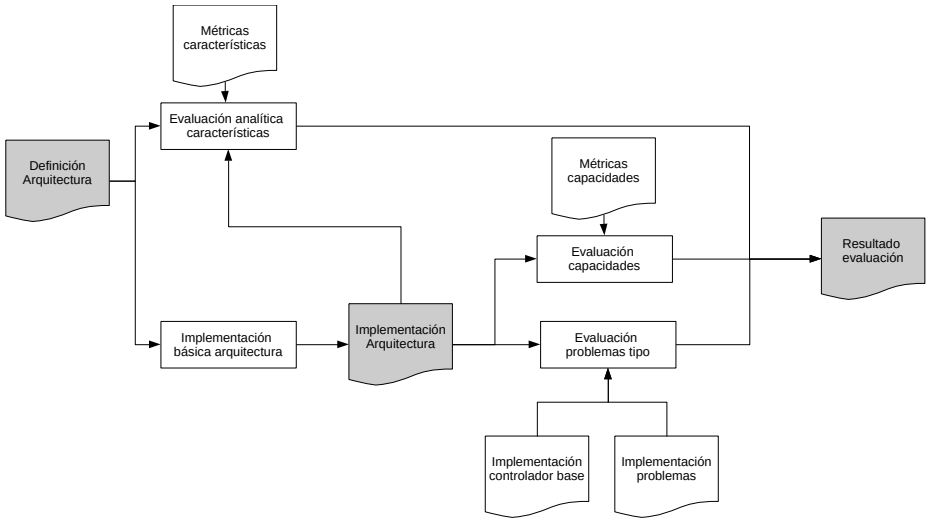


Ilustración 94: Estructura del sistema de evaluación con los elementos propios del sistema con fondo en blanco y aquellos específicos de cada arquitectura en sombreado gris

El resultado de la evaluación será un conjunto de valores numéricos, cada uno de los cuales define la capacidad de la arquitectura evaluada en algún aspecto.

Dada la naturaleza de las arquitecturas, no todas las métricas pueden ser evaluadas de forma computacional durante la ejecución de la propia arquitectura, sino que serán el resultado de la aplicación de un procedimiento manual.

4.2 Evaluación analítica de las características

El primer paso será la valoración cuantitativa de las características de mayor importancia de la arquitectura sujeto de la evaluación.

4.2.1 Dimensiones de la evaluación

Debemos definir las dimensiones de importancia para una arquitectura que luego trataremos de medir con métricas para cada una de ellas. Para definirlas nos apoyamos en las características ya identificadas en el análisis del estado de las arquitecturas:

- Adaptación: capacidad de implementar sistemas con diferentes características. En este caso vamos a centrarnos en esta faceta de la adaptación y dejaremos la adaptación “en caliente” como parte de la evaluación del sistema implementado.
- Cooperación: permite que varios individuos (robots) colaboren para conseguir un objetivo común.
- Coordinación: característica necesaria para que varios individuos ejecuten una acción que sería imposible de realizar por separado.
- Comunicación: Intercambio de información entre los individuos (o incluso entre colectivos).
- Autoconocimiento: capacidad de conocer las capacidades de los individuos del colectivo así como el conocimiento que poseen.
- Colectivos: habilidades de adopción, expulsión, etc.
- Interoperabilidad: capacidad de relacionarse o integrar diferentes individuos o sistemas.

4.2.2 Métricas de la evaluación para las arquitecturas

Varias de las dimensiones anteriormente definidas son difícilmente medibles de forma cuantitativa pero siempre se puede definir un estimador que aproxime el valor de la característica.

4.2.2.1 Adaptación

Para caracterizar la capacidad de desarrollar sistemas con diferentes objetivos nos centraremos en el número de estos objetivos al que es capaz de dar soporte la arquitectura y usaremos la taxonomía definida en [2], en su eje de aplicaciones (ver Ilustración 20: Esquema de la taxonomía propuesta por Serge Kernbach en 2013), como la colección de objetivos posible. Estas aplicaciones son: búsqueda y rescate (SR), aérea (AIR), terrestre (TR), submarina (SUB), espacial (SPC), industrial (IND), servicios (SVC) y competición por equipos (TEAM).

$$Adp(a) = \frac{SR + AIR + TR + SUB + SPC + IND + SVC + TEAM}{8}$$

En esta fórmula los diferentes elementos toman un valor entre 0 y 1, para simplificar 0 si no es posible implementar una aplicación de este tipo con la arquitectura a y 1 si es posible. Dado que siempre es posible la existencia de una cierta subjetividad en esta evaluación y para reducir esta, en caso de duda se puede dar el valor de 0,5 a la aplicación sobre la que se dude.

4.2.2.2 Cooperación

La medición de la cooperación, la contribución común a la consecución de un fin concreto, no puede realizarse a nivel de la arquitectura, solo puede realizarse a nivel del sistema. Por esta razón se aplican métricas en un problema tipo.

4.2.2.3 Coordinación

La coordinación en sí tiene múltiples facetas, nos centraremos en la más fácilmente evaluable, la sincronización, como un estimador de la capacidad de coordinación. La métrica diseñada para la coordinación se basa en la diferencia de tiempo mínima que añade la arquitectura a la ejecución de las acciones parciales de dos individuos que deban componerse en una sola acción (como podrían ser dos individuos desplazando un objeto).

$$Crd_{sinc}(a) = \frac{\sum_{i=1}^{n} \min(t_a^i \dots t_z^i)}{\sum_{i=1}^{n} (\min(t_a^i \dots t_z^i) + \max(t_a^i \dots t_z^i) - \min(t_a^i \dots t_z^i))}$$

Donde t_a^i es el retardo en tiempo de la ejecución de la acción i entre la decisión de ejecutarla por el sistema y su ejecución en el robot a, $\min(t_a^i \dots t_z^i)$ es el mínimo entre todos los retardos para la acción i y $\max(t_a^i \dots t_z^i)$ es el máximo.

De forma práctica aplicaremos esta métrica sobre un Colectivo de dos Individuos siendo la tarea conjunta empujar una pieza de forma que se mantenga perpendicular al eje de avance y cada una de las acciones parciales serán las activaciones de los sistemas de propulsión. En este escenario el tiempo mínimo es el requerido para desplazar la pieza la distancia indicada y el sobrecoste es el añadido por la complejidad computacional de los algoritmos de coordinación de la propia arquitectura y de las comunicaciones (en caso de ser necesarias). Asumiremos siempre el mejor caso de implementación.

4.2.2.4 Comunicación

En este caso podemos aprovechar las distintas métricas de comunicación ya existentes pero centrándonos en aquellos aspectos directamente relacionados con la arquitectura, en caso de definir esta un protocolo: eficiencia del uso del ancho de banda (sobrecoste del ancho de banda provocado por el uso protocolo), eficiencia del procesado (sobrecoste en tiempo por el proceso de envío y recepción). Es importante destacar que no estamos evaluando un único enlace sino una red entera.

La métrica propuesta sería entonces la suma de dos:

- Eficiencia del ancho de banda: ancho de banda disponible (b_t) descontando el ancho de banda consumido (b_u) por el propio protocolo en relación con el total.

$$\frac{b_t - b_u}{b_t}$$

- Eficiencia del procesado: usando como métrica el tiempo medio desde que se inicia el proceso de envío hasta que se procesa la recepción (t_r) con relación a un protocolo que no realice ningún procesado y posea un ancho de banda infinito (t_i).

$$\frac{t_i}{t_i + t_r}$$

La combinación de estas métricas nos da un estimador de las capacidades de comunicación de la arquitectura (a) que llamaremos Com(a):

$$Com(a) = \frac{\frac{b_t - b_u}{b_t} + \frac{t_i}{t_i + t_r}}{2}$$

Definiremos Com(a) = 0 para aquellas arquitecturas sin capacidad de comunicación explícita.

4.2.2.5 Autoconocimiento

Las capacidades de listar los sensores y actuadores o listar y transferir conocimiento entre individuos están interrelacionadas entre sí al ser el conocimiento sobre el conocimiento y el conocimiento sobre sensores y actuadores prácticamente lo mismo y por lo tanto el intercambio de este conocimiento entre individuos se producirá de la misma forma.

Definiremos una métrica que valore, por un lado, el nivel conocimiento que tiene el propio individuo de sí mismo y por otro, el de poder transferirlo a otro individuo.

$$Aut(a) = \frac{\frac{s_c + w_c}{s_t + w_t} + \frac{k_{trans}}{k_{total}}}{2}$$

Donde s_c y w_c son respectivamente el número de los sensores y actuadores de los que los individuos saben de su existencia, s_t y w_t son el número de los sensores y actuadores que los individuos poseen, k_{trans} es el volumen de conocimiento que la arquitectura puede transferir de un individuo a otro y k_{total} es el volumen de conocimiento total que poseen los individuos del colectivo.

4.2.2.6 Colectivos

Las habilidades de adoptar miembros y expulsarlos del colectivo se pueden valorar por la eficiencia combinada de dichas operaciones, midiendo la adopción seguida de expulsión media. En concreto usaremos como estimador el coste de las comunicaciones de la operación de adopción y expulsión, tomando como figura de mérito el inverso del volumen de datos (en bytes) necesario para adoptar y expulsar un individuo.

$$Col(a) = \frac{\frac{1}{\sum_{i=1}^{n} b(pa_i)} + \frac{1}{\sum_{j=1}^{m} b(pe_j)}}{2}$$

Donde pa_i es el paquete de datos número i necesario para la adopción de un individuo, $b(pa_i)$ los bytes totales del paquete de datos pa_i, n el número total de

paquetes necesario para la adopción, m número total de paquetes necesario para la expulsión y pe_j el paquete de expulsión número j.

Definiremos Col(a) = 0 para aquellas arquitecturas sin capacidad de adopción o expulsión explícitas (lo que es equivalente a decir que requieren un número de bytes infinito para estas operaciones).

Para aquellas arquitecturas donde el sumatorio de bytes para una de las actividades sea cero (es decir, no lo comuniquen al individuo adoptado o expulsado) consideraremos que el valor de la suma es 1.

4.2.2.7 Interoperabilidad

La interoperabilidad entre cualesquiera sistemas no es directamente medible ya que no podemos evaluar el conjunto de sistemas posibles. Como estimador de la interoperabilidad, podemos evaluar el sobrecoste en tiempo medio que añadiría una adaptación en las comunicaciones entre un individuo del sistema con otro de un sistema tipo.

$$Int(a,b) = \frac{t_{com}}{t_{com} + t_{sb}}$$

Donde t_{com} es el tiempo de las comunicaciones en sí y t_{sp} representa el tiempo extra que ha sido necesario para realizar la comunicación con un individuo de otro sistema (b).

4.3 Implementación básica de la arquitectura

Para poder aplicar varias de las métricas es necesario disponer de una implementación de la arquitectura. Dicha implementación no tiene por qué ser completa, pero debe cumplir unos mínimos que permitan realizar las mediciones correspondientes de las métricas, así como ejecutar pequeñas simulaciones integrándose con el simulador correspondiente. Dado que no se dispone de implementaciones de la mayor parte de las arquitecturas (ver Tabla 2) debemos crearlas con el menor coste posible que permita su correcta evaluación.

4.3.1 Robot base

Debemos poseer la estructura mínima de un robot sobre la cual poder implementar la arquitectura básica correspondiente. Esta estructura debe ser lo más simple y extensible posible para permitir evaluar las arquitecturas de forma eficiente.

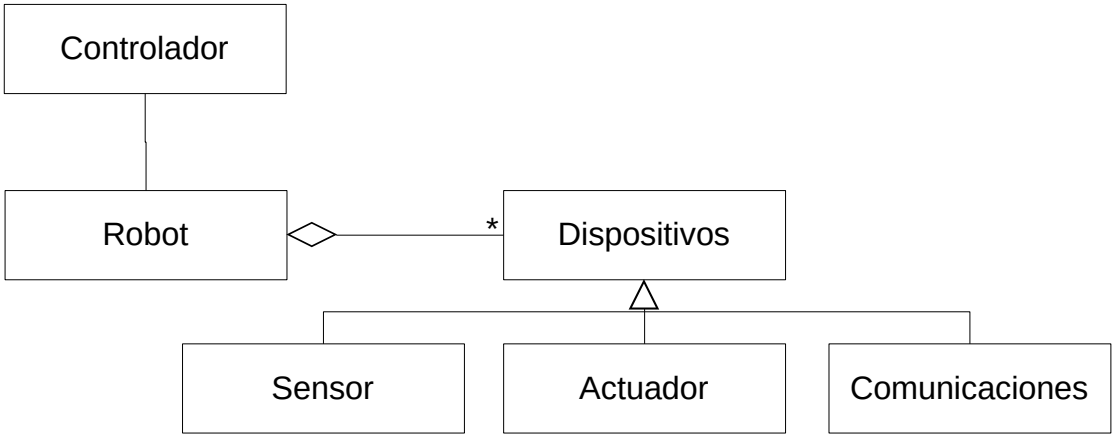


Ilustración 95: Diagrama de clases de un robot simplificado

Dicho robot base constaría de una colección de dispositivos que podrían ser sensores (dispositivos que proporcionan datos), actuadores (dispositivos que permiten modificar alguna variable en el mundo físico) y comunicaciones (dispositivos que permiten el envío de información a otro robot). Además tendría un controlador que se ocuparía de leer los datos de los sensores y usar los actuadores y dispositivos de comunicaciones para cumplir las tareas encomendadas. Este controlador puede estar limitado en memoria o capacidad computacional.

4.3.1.1 Definición del robot para las pruebas

Dado que vamos a comparar varias arquitecturas, el conjunto de dispositivos disponibles será el mismo para los individuos del colectivo usado durante la evaluación.

El robot dispondrá de los siguientes dispositivos:

- Actuador 1: Motor sobre tren derecho. Velocidad máxima 50ud/s, proporcional al voltaje aplicado.
- Actuador 2: Motor sobre tren izquierdo. Velocidad máxima 50ud/s, proporcional al voltaje aplicado.
- Sensor 1: Contacto frontal.

- Sensor 2: Medidor de distancia láser de 270 grados centrado en dirección frontal.
- Sensor 3: Detector de posición y postura que permite obtener las coordenadas y orientación del robot con un cierto margen de error.
- Comunicaciones 1: Sistema direccionable totalmente conexo de distancia infinita (para los propósitos de las pruebas).

Se considera que la energía disponible en el robot no se agotará durante el transcurso de la prueba.

Los actuadores 1 y 2 se combinan en uno solo que permite definir la velocidad lineal (v) y angular (w) (además de definir el tamaño de las ruedas como R y e como la mitad de la distancia entre los dos trenes; para facilitar el cálculo supondremos que son la unidad) de forma que:

$$TrenIzquierdo = \frac{(v - e\,w)}{R}$$

$$TrenDerecho = \frac{(v + e\,w)}{R}$$

4.3.2 Componentes para las arquitecturas

El software de la arquitectura se ejecutará de forma distribuida en cada controlador de robot simulado, siendo este el único elemento que se modificará para implementar la arquitectura.

Dispondremos de una serie de componentes que se podrán reutilizar para la implementación de las arquitecturas como varios comportamientos básicos, tipos de comunicación, etc.

4.4 Evaluación de capacidades de la arquitectura

Evaluaremos de forma empírica la capacidad de implementación de sistemas sobre una arquitectura definiendo varios pequeños sistemas con sus objetivos y enfrentándolos a la implementación de la arquitectura.

4.4.1 Número de individuos

Incluso cuando se trata de individuos sin comunicación entre ellos ni dependencias de otro tipo, siempre existe un máximo de individuos dependiendo de las condiciones del ambiente (aunque solo sea por el máximo nivel de ocupación del espacio) siendo diferente en condiciones diferentes.

Para estimar este número, simularemos las comunicaciones necesarias para que la arquitectura opere de forma normal hasta alcanzar el máximo número de individuos en un escenario de 10 robots de ancho por 10 robots de alto. Las unidades de la métrica serán la densidad de robots por unidad de espacio ocupado, siendo esta unidad el espacio que ocupa un robot.

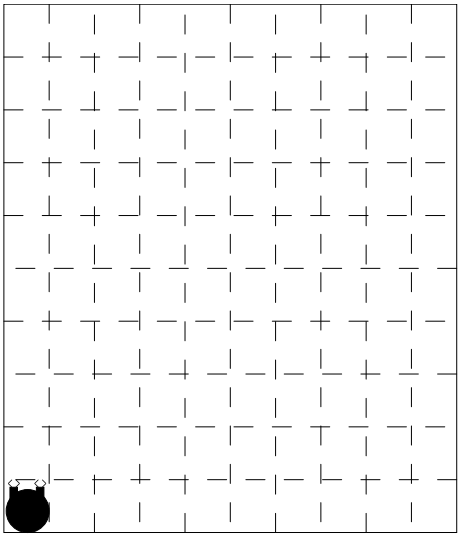


Ilustración 96: Área de ocupación de un robot

4.4.2 Intercambio de información simultáneo

Una medida de la capacidad de comunicación y registro de datos de la arquitectura es el número de elementos de información que es capaz de generar e intercambiar por unidad de tiempo, entendiendo un elemento de información, tanto lecturas de sensores como estados internos o resultados de evaluaciones.

Para estimarla usaremos el máximo número de elementos de los sensores incluidos en nuestro robot tipo que la arquitectura es capaz de intercambiar entre dos individuos por unidad de tiempo.

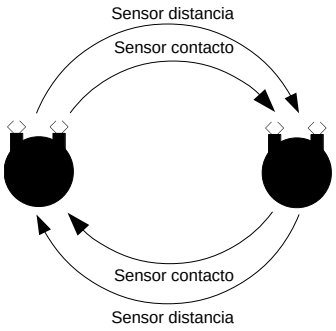


Ilustración 97: Transferencia de información entre dos individuos

Dado que las comunicaciones pueden ser bidireccionales, el total es la suma de lo enviado en ambas direcciones. Hay que destacar que no solo se está midiendo la capacidad a nivel de comunicaciones, ya que las transferencias de información deben ser diferentes por lo que, en este caso, implican una nueva lectura, procesado y envío. Para establecer el número por unidad de tiempo se realiza una prueba enviando el máximo número de datos durante 100 segundos entre dos unidades y se divide por esa cantidad.

4.4.3 Acciones con origen diferente al robot

La otra capacidad de interés es el número de acciones sobre un actuador por unidad de tiempo que se pueden realizar y que hayan sido ordenadas por un controlador residente en otro robot. Esto nos da un estimador de la capacidad que tiene la arquitectura de distribuir el control.

Se considera que esta acción comienza cuando el controlador recibe el estímulo por el cual decide emitir la orden al actuador y termina cuando el actuador la ha completado (y, de ser el caso, después de actualizar la información referente a esta acción). Para establecer el número por unidad de tiempo se realiza una prueba enviando el máximo número de acciones durante 100 segundos entre dos individuos y se divide por esa cantidad. Al igual que en el caso anterior, el número de acciones es la suma de las acciones enviadas en ambas direcciones.

4.5 Evaluación problema tipo

Como problema tipo nos quedaremos con uno en el que se desarrollen el mayor número de operaciones incluidas en problemas clásicos posibles, pero manteniendo un nivel de complejidad reducido para que una métrica simple sobre dicho problema sea comparable. Consideramos que el problema ideal es el ensamblado de dos piezas (A y B), cada una de las cuales debe ser transportada (o empujada) por varios robots, estando todos los robots implicados dentro del mismo colectivo. Este problema añade características de otros problemas clásicos como son: mantener una formación, empujar un objeto por parte de un equipo y ensamblado de objetos.

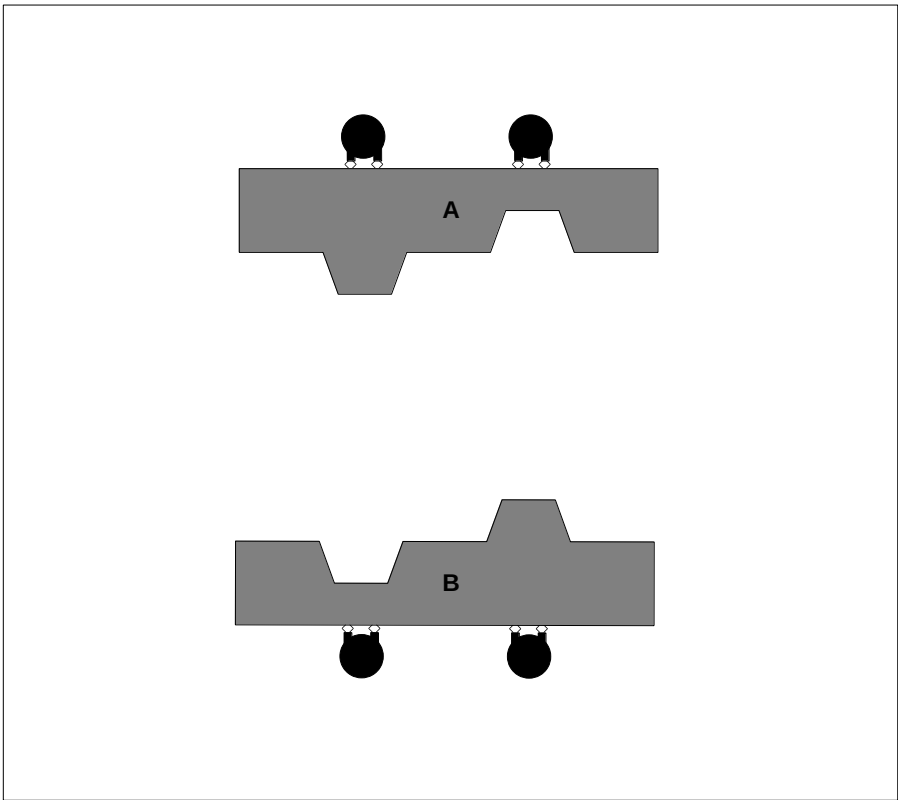


Ilustración 98: Posición inicial del ensamblado de dos piezas por dos grupos de robots

La posición inicial de las piezas no las sitúa centradas entre sí para evitar que se pueda resolver simplemente empujándolas hacia adelante. De esta forma los robots deben desplazarlas lateralmente (al menos una) antes de acercarlas para ensamblarlas. La distancia de descentrado es superior a la de la holgura existente entre el saliente y la oquedad.

El problema queda completado cuando la distancia entre los bordes horizontales de la pieza A y de la pieza B es cero. La aplicación exacta de esta condición puede variar según el sistema de detección de colisiones del simulador, ya que no todas las versiones y programas soportan correctamente la colisión de varios elementos en donde uno encaje en el otro. Teniendo en cuenta esto y para permitir la comparativa entre varios sistemas, la condición de finalización antes descrita se refiere a los bordes exteriores de ambas piezas.

Este problema admite múltiples variaciones: con piezas que no tienen por qué presentar igual resistencia a ser empujadas o igual peso al ser alzadas en ambos extremos; con diferentes robots y con diferentes posiciones de inicio.

4.5.1 Métricas para el problema tipo

Sobre este problema aplicaremos tres métricas parciales, dos inspiradas por las ya revisadas para problemas clásicos (energía y tiempo) y otra (precisión) que tratará de evaluar la calidad del trabajo final.

4.5.1.1 Tiempo consumido

La métrica más evidente es el tiempo total hasta conseguir el objetivo de la tarea:

$$tmax$$

Si el objetivo no se consiguiese se supone un tmax = ∞.

4.5.1.2 Energía consumida

Proponemos como métrica para medir la eficiencia, el total de energía empleada por los robots para realizar la tarea. Esta métrica queda definida por:

$$\sum_{t=1}^{tmax} \sum_{i=1}^{m} e_i(t)$$

Donde $e_i(t)$ es la energía usada por el robot i en el tiempo t; m, el número de robots.

4.5.1.3 Distancia horizontal

Los huecos y salientes de ambas piezas no son del mismo tamaño, siendo los huecos ligeramente más grandes, y las piezas no están centradas entre sí. Como métrica para la precisión con la que se ha ejecutado la tarea usaremos la distancia

horizontal (d_h) que puede quedar entre el centro de ambas piezas (que será la misma que quedará si una sobresale con respecto a la otra).

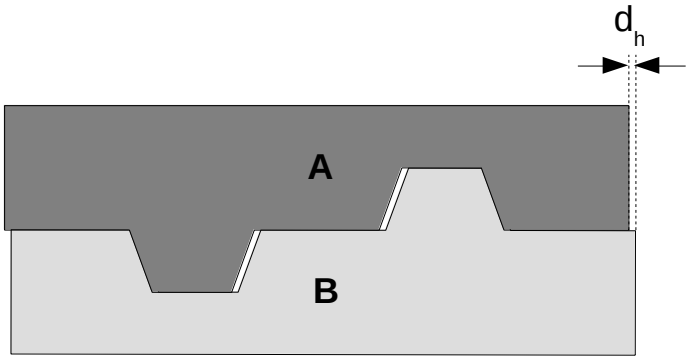


Ilustración 99: Distancia horizontal en el ensamblado de ambas piezas

Dada la existencia de un máximo (que es igual a la holgura que existe entre los salientes y huecos) debemos considerar la posibilidad de que este se supere como un fallo.

Esta métrica tiene un especial impacto debido a la posición inicial de las piezas que deben moverse lateralmente antes de intentar ensamblarse.

4.5.2 Implementación del escenario del problema tipo

4.5.2.1 Diseño de las piezas

Las dos piezas a ensamblar son idénticas entre sí, simplemente colocadas de forma que una de ellas está rotada sobre el eje vertical 180 grados.

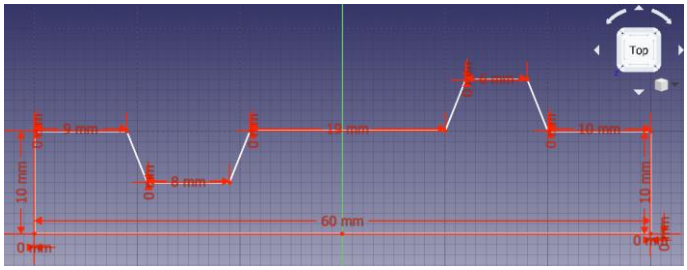


Ilustración 100: Dimensiones de la pieza a desplazar

La altura será dos tercios de la longitud máxima de la pieza. El saliente es de 10 mm de ancho mientras que la oquedad dispone de 12 mm de ancho. La longitud del saliente y la de la oquedad son iguales (5 mm).

Con este diseño se permite ensamblar ambas piezas de forma que no queden totalmente centradas para permitir la aplicación de la métrica diseñada a tal fin.

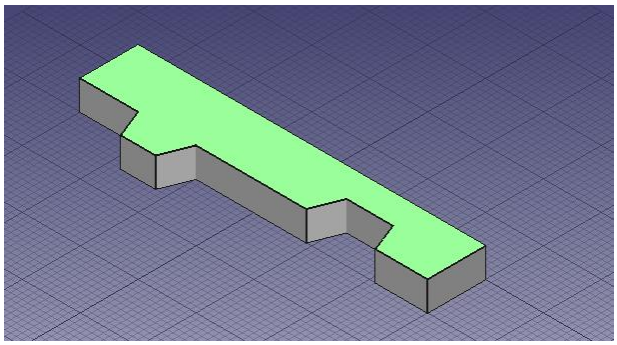


Ilustración 101: Diseño de la pieza

El peso de la pieza y la fuerza de rozamiento de la misma son reducidos en un suelo liso pero proporcional a su volumen, lo que resulta en un centro de gravedad que no coincide con la línea media.

4.5.2.2 Diseño del robot

Tomando como base las características definidas para el robot teórico a emplear durante las simulaciones se le crea un aspecto físico que encaje en ellas: con 4 ruedas para simular el sistema de propulsión diferencial, un bumper para simular el sensor de contacto, una elevación superior para el detector de distancias laser y una antena para denotar la capacidad de comunicación. Además se le añaden otros complementos visuales para mejorar su aspecto.

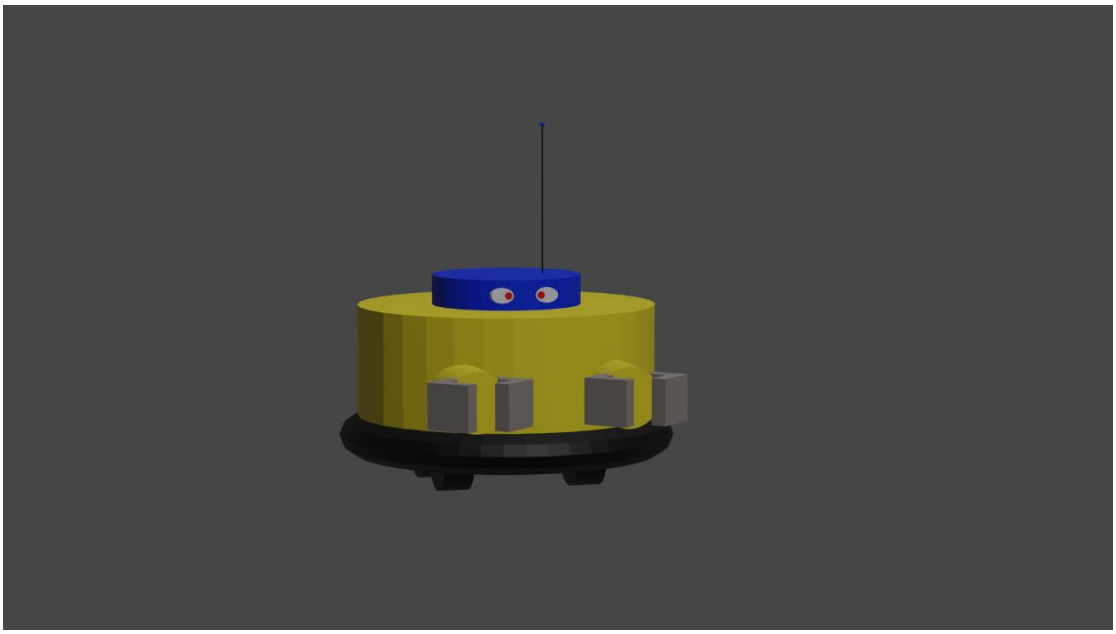


Ilustración 102: Aspecto del robot para las simulaciones

Las dimensiones del robot son muy reducidas (22 mm de diámetro).

4.5.2.3 Escenario

El escenario está compuesto por un suelo de 200 mm de ancho por 200 mm de alto rodeado de un muro no superable por ningún objeto. Los únicos objetos en el escenario son los cuatro robots del equipo y las dos piezas a ensamblar.

El suelo es rugoso y está compuesto por una matriz de 12 * 12 pequeñas baldosas, estas características aumentan el rozamiento y pueden producir pequeñas desviaciones en la dirección.

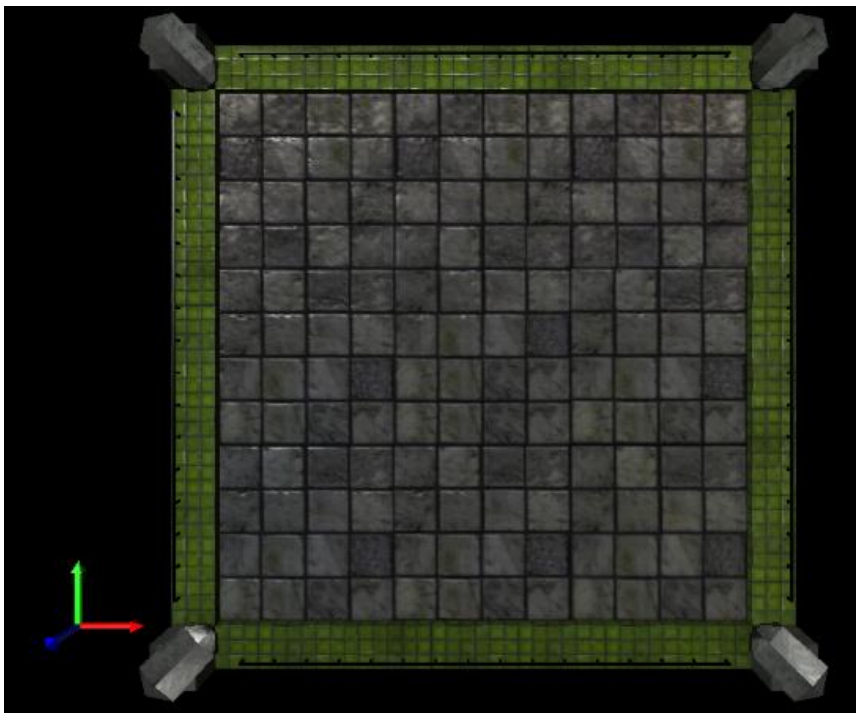


Ilustración 103: Vista superior del escenario donde se desarrollan las acciones

4.5.2.4 Simulación

La simulación consta de los cuatro robots colocados en grupos de dos en los extremos del escenario y separados el espacio que ocuparía un tercero (20 mm), las dos piezas separadas 90 mm y cada una situada a 5 mm de distancia de su grupo de robots y descentradas entre sí 5 mm.

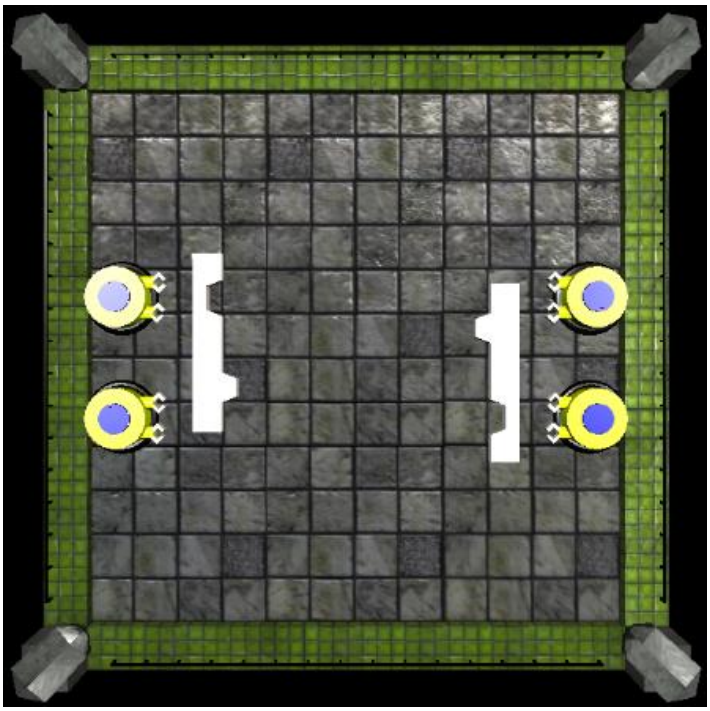


Ilustración 104: Posición inicial de la simulación

4.5.3 Ejecución y recogida de datos de las simulaciones

Con cada arquitectura evaluada se realizarán 50 repeticiones teniendo en cuenta las posibles variaciones en las métricas causadas por la acumulación de pequeñas variaciones en la posición inicial de los elementos (el motor de física del simulador genera pequeños movimientos en el inicio cuyo resultado no es idéntico entre repeticiones).

Para cada repetición se capturan los siguientes valores:

- Tiempo inicial
- Tiempo final
- Energía consumida por cada robot
- Distancia entre los ejes verticales de las dos piezas al final de la simulación

Con estos datos se calculan los resultados de las tres métricas, siendo el tiempo total la resta entre el final y el inicial; la energía total consumida, el sumatorio de la energía de todos los robots y la distancia horizontal, la distancia obtenida.

4.6 Resultado e interpretación

4.6.1 Resultado

Tal como hemos descrito a lo largo de este capítulo el resultado del sistema de evaluación será un conjunto de valores numérico que caracterizará la arquitectura sobre la que se aplique.

Métrica	**Peor caso**	**Mejor caso**
Características: Adaptación	0	1
Características: Coordinación	0	1
Características: Comunicación	0	1
Características: Autoconocimiento	0	1
Características: Colectivos	0	1
Características: Interoperabilidad	0	1
Capacidades: Número de individuos	0	1
Capacidades: Intercambio de información simultáneo	0	∞
Capacidades: Acciones con origen diferente al robot	0	∞
Problema tipo: Tiempo	∞	0
Problema tipo: Energía consumida	∞	0
Problema tipo: Distancia horizontal	0	1

Tabla 4: Resumen de métricas para las arquitecturas

4.6.2 Interpretación y comparación

Es importante, para la interpretación de los resultados, tener en cuenta que estas métricas son indicadores, no valoraciones absolutas de las capacidades de una arquitectura. Un valor superior en las capacidades que mejoran proporcionalmente a dicho valor o inferior en las que lo hacen inversamente quiere decir que la arquitectura debería comportarse mejor en este aspecto, pero eso no sucederá con todos los posibles sistemas implementados.

Para facilitar la comparación dividiremos las métricas en dos gráficos de radar: por un lado los resultados de las métricas de las características y por otro el resultado de las métricas de los ensayos de capacidades.

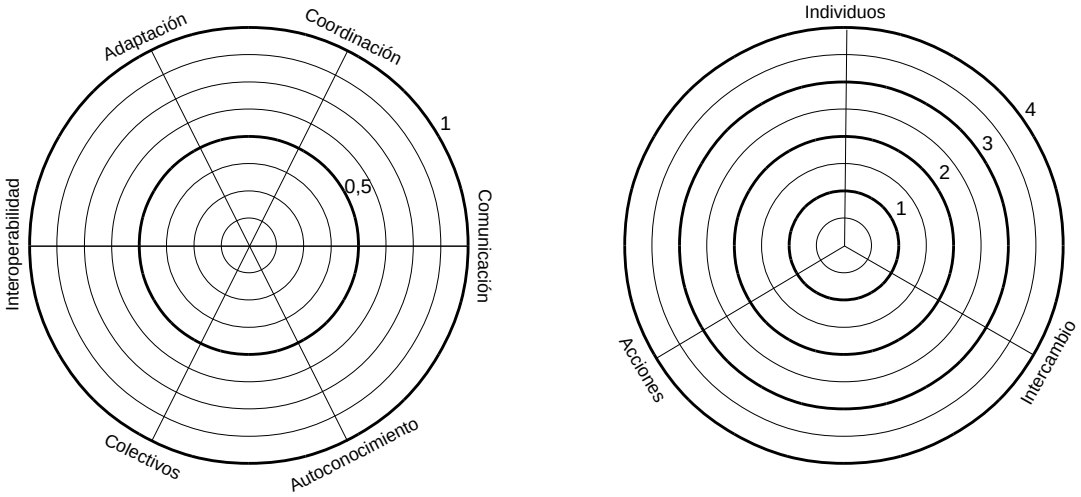


Ilustración 105: Representación para facilitar la comparativa

Prestaremos especial atención a la evaluación de los resultados del problema tipo, como visión general de las arquitecturas, presentando los resultados de cada una de las métricas en diagramas de cajas y bigotes, y diagramas de violín para facilitar la comparación. Una vez aplicado este sistema de evaluación a la arquitectura propuesta, esta será comparada de la misma forma con el resto de arquitecturas.

Además, sobre los resultados de la simulación del problema, obtenidos tal como se ha descrito anteriormente, se demostrará que la nueva arquitectura es estadísticamente mejor que las otras evaluadas, siguiendo el procedimiento descrito en [125], según el cual debemos realizar una prueba de hipótesis nula y, antes de seleccionar y aplicar un test paramétrico o no paramétrico, debemos comprobar ciertas condiciones previas: independencia, normalidad y heterocedasticidad. La condición de normalidad (distribución gaussiana) se comprobará usando el test de Shapiro-Wilk [126]. Si esta se cumple, la condición de igualdad de varianzas se comprobará usando el test de Bartlett [127].

Nº de técnicas	Paramétricas	No paramétricas
Medidas repetidas, n=2	T-test	Wilcoxon
Medidas repetidas, n>2	ANOVA	Friedman, Quade

Tabla 5: Pruebas paramétricas y no paramétricas más comunes según [125]

Siguiendo los resultados de las condiciones anteriores, dado que el número de técnicas es superior a 2, seleccionaremos la técnica más apropiada usando la tabla de pruebas (Tabla 5 extraída de [125]).

Capítulo 5

Análisis de los resultados y comparativa entre arquitecturas previas

No todas las arquitecturas son evaluables por el sistema definido, por diversos motivos: no todas las arquitecturas identificadas son implementables tal como han sido definidas, sus implementaciones originales no están disponibles o no son integrables en un simulador. Trataremos de hacer adaptaciones según el caso y remarcaremos las causas y posibles consecuencias sobre los resultados.

5.1 Análisis previo a la evaluación

Antes de la ejecución de las métricas descritas sobre cada arquitectura, las analizaremos para comprobar la aplicabilidad del sistema de evaluación así como las posibles ventajas o inconvenientes cualitativos del diseño particular.

5.1.1 CEB0T

A pesar de que no hay una implementación de referencia de esta arquitectura disponible, es factible realizarla usando el esquema definido para el sistema de evaluación.

CEBOT es una arquitectura basada en la composición de comportamientos, jerárquica y pensada para robots homogéneos. Su diseño está muy unido a la forma de procesado de los comportamientos, por lo que no es aplicable a sistemas que no posean esta orientación.

5.1.2 ACTRESS

No se ha localizado una implementación que se pueda usar de esta arquitectura a pesar del interés de los conceptos que aplica, definiendo, en esencia, una de las de las arquitecturas evaluables más modificables. Su mayor ventaja es el disponer de una pila de protocolos que denomina *protocolo de mensajes* basada en el estándar OSI que permite su integración con otros sistemas.

5.1.3 GOFER

GOFER fue diseñada para un robot concreto siendo difícil la extensión de su implementación, que tampoco está disponible, para otros robots. En su mayor parte es implementable, pero no de forma completa al no disponerse de una especificación de varios elementos esenciales del sistema.

Esta arquitectura aporta el concepto de objetivo frente al de tarea. Por otro lado, el CTPS (el sistema central que realiza las planificaciones) limita las posibilidades de uso al definir un método de generación de planes (frente a comportamientos emergentes) y estar centralizado.

5.1.4 Matarić

Esta es otra de las arquitecturas orientadas a comportamiento de interés por la simplicidad de su definición. Su implementación original no está disponible, pero es posible reproducirla para el objetivo de evaluación.

5.1.5 ALLIANCE

ALLIANCE y L-ALLIANCE son otro ejemplo más de arquitecturas orientadas a comportamientos. No se dispone de la implementación original, pero parte de la arquitectura es implementable para nuestros objetivos teniendo varios elementos en común con otras arquitecturas.

5.1.6 ARTIS

Esta arquitectura tiene un objetivo mucho más amplio que los sistemas multi-robot siendo una arquitectura orientada a agentes.

No se cuenta con la implementación original de sus autores y la información disponible no permite su evaluación completa.

5.1.7 Distributed Robot Architecture

No se dispone de la implementación original, orientada a un conjunto muy concreto de robots, y la información sobre su diseño no permite realizar una implementación que pueda cumplir los objetivos de evaluación.

DRA añade la distribución en capas como patrón de diseño.

5.1.8 CHARON

Este lenguaje es de interés por su orientación a agentes y las posibilidades que brinda para la implementación de controladores en sistemas multi-robot. A pesar de que sí se dispone de una implementación, no es directamente evaluable por la imposibilidad de integración con el sistema de simulación. Como consecuencia de lo anterior, es imposible aplicar muchas de las métricas sobre CHARON. Requiere especial atención la modelización del colectivo como un agente más.

5.1.9 OROCOS

OROCOS es un framework con una fuerte orientación a componentes reutilizables en aplicaciones robóticas, siendo este su mayor interés. No puede decirse que sea una arquitectura para sistemas multi-robot, aunque dispone de muchas funcionalidades que pueden ayudar en su construcción.

No es posible evaluar OROCOS con nuestro sistema de evaluación al tener más de una forma de usarse.

5.1.10 CAMPOUT

Es otra de las arquitecturas orientadas a comportamientos. Su propuesta de comportamientos de comunicación, en especial los que permiten transferir comportamientos y su estructuración jerárquica de los mismos, es de gran interés por las posibilidades de distribución de habilidades en el colectivo.

No hay una implementación disponible de esta arquitectura, pero es posible realizar una para su evaluación.

5.1.11 IDEA

IDEA es una arquitectura flexible gracias a la aplicación del patrón de diseño de capas y su orientación a agentes. Además posee un componente para el almacenamiento de conocimiento que puede ser de gran utilidad para dotar de nuevas capacidades a un sistema basado en ella.

La inclusión en su definición de elementos con técnicas concretas destinados a crear los planes limita su uso de forma universal.

No hay una implementación disponible pero se puede realizar una suficientemente completa para proceder a su evaluación.

5.1.12 Swarm-bot

Este sistema está basado en un conjunto específico de robots cuya aportación más notable es el auto-ensamblado para la consecución de fines específicos (superación de obstáculos en el terreno).

No se dispone de una implementación del sistema software que se ejecuta en estos robots ni es posible realizarla.

5.1.13 MAS2CAR

Esta arquitectura es diferente a las otras al usar un modelo para definir el colectivo (objetivos…) separado de la implementación del resto del sistema. Esta misma faceta de su diseño la hace compleja y difícil de integrar con otros sistemas.

No se dispone de una implementación y no es posible realizarla por la dificultad de integración con el sistema de simulación así como el esfuerzo necesario para el desarrollo de sus componentes.

5.2 Aplicación de las métricas de evaluación para las características

5.2.1 Aplicación de la métrica para la adaptación

La métrica de la adaptación se basa en la evaluación de ocho elementos cuyos resultados por arquitectura podemos ver en la siguiente tabla:

	Búsqueda y rescate	Aérea	Terrestre	Submarina	Espacial	Industrial	Servicios	Competición por equipos
CEB0T	1	1	1	1	1	0	1	1
ACTRESS	1	1	1	1	1	1	1	1
GOFER	0	0	1	0	0	0	0	1
Matarić	0	1	1	1	1	1	0	1
ALLIANCE	1	1	1	1	1	1	1	1
ARTIS	1	1	1	1	1	1	1	1
Distributed Robot Architecture	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2
CHARON	1	1	1	1	1	1	1	1
OROCOS	1	1	1	1	1	1	1	0
CAMPOUT	0	1	1	1	1	1	1	1
IDEA	1	1	1	1	1	1	1	0
Swarm-bot	1	0	1	0	0	0	1	1
MAS2CAR	1	1	1	1	1	1	1	1

Tabla 6: Resultados evaluación adaptación

5.2.2 Aplicación de la métrica para la coordinación

Dado que todas las arquitecturas se ejecutan de la misma forma sobre el sistema, el coste mínimo es el mismo:

$$\sum_{i=1}^{n} \min(t_a^i \dots t_z^i) = 5{,}283$$

	Medición de tiempos
CEB0T	5,483
ACTRESS	5,383
GOFER	5,383
Matarić	5,283
ALLIANCE	6,280
ARTIS	6,172
Distributed Robot Architecture	6,106
CHARON	-
OROCOS	-
CAMPOUT	5,552
IDEA	6,258
Swarm-bot	5,682
MAS2CAR	-

Tabla 7: Tiempos máximos resultado de aplicar las acciones

No es posible aplicar a CHARON esta métrica al no poder integrarse con otros sistemas. Así mismo no es posible aplicarla a OROCOS de la misma forma que con el resto de arquitecturas al no poder integrarse directamente.

MAS2CAR no se puede evaluar con esta métrica al no disponer de una implementación completa.

5.2.3 Aplicación de la métrica para el autoconocimiento

La métrica de autoconocimiento evalúa dos elementos: la capacidad de transferir el autoconocimiento de los dispositivos y la de transferir el conocimiento.

	Autoconocimiento de los sensores y actuadores	Autoconocimiento sobre conocimiento
CEB0T	0	1
ACTRESS	0	1
GOFER	0	1
Matarić	0	0
ALLIANCE	0	1
ARTIS	0	1
Distributed Robot Architecture	1	1
CHARON	0	1
OROCOS	1	1
CAMPOUT	0	1
IDEA	0	0
Swarm-bot	0	0
MAS2CAR	0	1

Tabla 8: Resultados de la aplicación de la métrica de autoconocimiento

5.2.4 Aplicación de la métrica para las comunicaciones

La métrica de comunicaciones evalúa dos elementos: la eficiencia del ancho de banda y la eficiencia del proceso de envío.

	Eficiencia con respeto al ancho de banda	Eficiencia del proceso
CEB0T	1	1

ACTRESS	0,779	0,999
GOFER	1	1
Matarić	-	-
ALLIANCE	1	1
ARTIS	0,788	0,973
Distributed Robot Architecture	0,761	0.615
CHARON	-	-
OROCOS	-	-
CAMPOUT	0,576	0,329
IDEA	1	0,935
Swarm-bot	-	-
MAS2CAR	-	-

Tabla 9: Resultados de la aplicación de la métrica de comunicaciones

Las comunicaciones usando ARTIS consideran el uso del blackboard como punto intermedio.

La especificación de CHARON incluye interesantes capacidades de comunicación entre agentes cuando estos residen en la misma máquina pero no dispone de implementación para comunicaciones fuera de esta.

Al no estar disponible una implementación o una descripción completa de su funcionamiento interno de MAS2CAR, no se puede evaluar con esta métrica.

Por su parte OROCOS, siendo una arquitectura basada en componentes, no dispone de un protocolo específico, por lo que evaluar sus comunicaciones no tiene objeto.

IDEA posee un sistema de comunicaciones capaz de enviar objetivos y recibir resultados (retroalimentación de algún tipo) sobre canales dedicados.

Consideramos Swarm-bot como incapaz de comunicarse de forma explícita ya que, a pesar del LED bicolor que usa, la señalización de este no permite el envío de información y más bien denota un estado interno. El sistema de comunicaciones descrito es equivalente a observar la postura o acciones desde otro robot.

5.2.5 Aplicación de la métrica para las funciones de gestión del colectivo

Consideramos que la posible habilidad de formar equipos de DRA no es un sistema de adopción o expulsión explícita de miembros del colectivo, ya que esto no los aísla siendo un sistema jerárquico dentro del colectivo y no una división del mismo.

También se considera que las arquitecturas OROCOS, CHARON y MAS2CAR que lo permiten podrían añadir métodos para este tipo de comportamientos ya que, al no proveerlos desde la propia arquitectura, no poseen capacidades de adopción y expulsión explícitas.

Por último, la señalización de errores de los robots o los comportamientos de impaciencia de ALLIANCE por los que un robot pasa a desarrollar la tarea de otro, no representa un sistema de autoexclusión / expulsión sino una reacción interna al propio Colectivo.

5.2.6 Aplicación de la métrica para la comunicación

Tal como se define la métrica, Matarić y Swarmbot se evalúan con un resultado de cero al no contar con sistemas de comunicación explícita.

5.2.7 Aplicación de la métrica para la interoperabilidad

Las arquitecturas Matarić, Swarmbot y CHARON, al no disponer de un sistema de comunicación genérico, obtienen una métrica de cero.

IDEA posee un sistema de comunicaciones sobre canales dedicados que impide su interoperabilidad sin redefinir el sistema.

OROCOS es fácilmente interoperable con cualquier sistema. Al ser una arquitectura basada en componentes, se puede crear un componente o adaptación para cada sistema con el que se tenga que comunicar.

MAS2CAR, al no poder disponer de una implementación o una descripción completa de su funcionamiento interno, no se puede evaluar con esta métrica.

	Tiempo de una comunicación (ms)	Tiempo con adaptación (ms)
CEB0T	99,694	299,083
ACTRESS	99,695	199,385
GOFER	99,694	199,386
Matarić	-	-
ALLIANCE	99,685	299,057
ARTIS	13,151	175,843
Distributed Robot Architecture	0,543	1,268
CHARON	-	-
OROCOS	-	-
CAMPOUT	1,565	6,931
IDEA	-	-
Swarm-bot	-	-
MAS2CAR	-	-

Tabla 10: Resultados parciales para la métrica de interoperabilidad

5.2.8 Totales

	Adaptación	Coordinación	Comunicación	Autoconocimiento	Colectivos	Interoperabilidad
CEB0T	6/8	0,963	1	0,5	0	0,333
ACTRESS	1	0,981	0,889	0,5	0	0,5
GOFER	2/8	0,981	1	0,5	0	0,5

Matarić	6/8	0,999	0	0	0	0
ALLIANCE	1	0,841	1	0,5	0	0,333
ARTIS	1	0,856	0,881	0,5	0	0,074
Distributed Robot Architecture	4/8	0,865	0,688	1	0	0,348
CHARON	1	-	0	0,5	0	0
OROCOS	7/8	-	-	1	0	-
CAMPOUT	7/8	0,951	0,453	0,5	0	0,225
IDEA	7/8	0,844	0,968	0	0	0
Swarm-bot	4/8	0,929	0	0	0	0
MAS2CAR	1	-	-	0,5	0	-

Tabla 11: Resultados de las métricas para las características de las arquitecturas

5.3 Aplicación de las métricas de evaluación para las capacidades

	Número de individuos	Intercambio de información simultáneo	Acciones
CEB0T	0,19	19,31	186,5
ACTRESS	0,10	6,66	5,1
GOFER	-	-	-
Matarić	1	0	0
ALLIANCE	0,20	19,16	0
ARTIS	-	-	-
Distributed Robot Architecture	-	-	-

CHARON	-	-	-
OROCOS	-	-	-
CAMPOUT	0,15	19,68	64,75
IDEA	0,29	0	0
Swarm-bot	-	-	-
MAS2CAR	-	-	-

Tabla 12: Resultados para las métricas de las capacidades

ACTRESS está limitada por el direccionamiento de su protocolo a 1 byte (255) aunque el espacio de direccionamiento podría aumentarse.

La arquitectura propuesta por Matarić, al no depender de un sistema de comunicaciones, no está limitada en número, pero tampoco puede realizar acciones remotas o intercambiar información de forma explícita.

IDEA, al enviar objetivos y no instrucciones concretas, no puede controlar los actuadores o hacer un intercambio de información genérico. Para la prueba de número de individuos con esta arquitectura se enlazan los agentes de cada robot en cadena.

5.4 Simulación de problema

En la Tabla 13 podemos encontrar las medias y desviaciones típicas resultado de las 50 ejecuciones de cada una de las arquitecturas con el problema simulado. En la sección de conclusiones (5.5.2.3 Comparativa de los resultados de la simulación del problema) se realiza una comparativa de los resultados de las tres métricas usando las técnicas descritas en el capítulo anterior (4.6.2 Interpretación y comparación).

	Tiempo		Energía consumida		Distancia horizontal (m)	
	Media	Desviación típica	Media	Desviación típica	Media	Desviación típica
CEB0T	5,5392	0,2775	6,4245	0,3062	0,0342	0,0198
ACTRESS	7,0379	0,3785	7,6148	0,5075	0,0443	0,0237
GOFER	-		-		-	
Matarić	8,5816	2,0850	7,7653	1,9717	0,0737	0,0479
ALLIANCE	7,9713	0.5248	7,3406	0,7832	0,0769	0,0577
ARTIS	-		-		-	
Distributed Robot Architecture	-		-		-	
CHARON	-		-		-	
OROCOS	-		-		-	
CAMPOUT	7,9442	1,4868	7,1722	1,3569	0,0977	0,0689
IDEA	8,4227	1,7044	7,6317	1,4710	0,0902	0,0656
Swarm-bot	-		-		-	
MAS2CAR	-		-		-	

Tabla 13: Media y desviación típica de los resultados para las métricas de la simulación del problema

5.5 Comparativa y conclusiones

Como hemos visto en los resultados de la evaluación, las distintas arquitecturas proporcionan ciertas ventajas e inconvenientes. Antes de comentar los resultados numéricos trataremos los cuantitativos.

5.5.1 Conclusiones sobre las características y atributos

Entre las arquitecturas descritas hay varios detalles que deben considerarse patrones de diseño a reutilizar en una nueva arquitectura:

- La elegante simplicidad de la composición de comportamientos de Matarić.
- El concepto de robotor definido en ACTRESS (una abstracción que debe cumplir dos características: Ser capaz de tomar decisiones y ejecutar acciones de forma autónoma y ser capaz de comunicarse con otros componentes) define el individuo base.
- La definición de un protocolo de mensajes (tal como lo denomina ACTRESS) que incluyan las funciones básicas permite la implementación rápida de colectivos con altos niveles de coordinación. Este protocolo debe incluir la capacidad de transferir comportamientos (como CAMPOUT).
- El concepto de asignación de objetivos en vez de tareas presente en GOFER y llevado al extremo de una definición completa en Mas2Car permite cambiar los propósitos de un sistema ya implementado con gran facilidad.
- La división en capas de DRA e IDEA.
- El repositorio de información presente en IDEA que permite almacenar planes.

Por otra parte existen características que es necesario evitar:

- La orientación exclusiva a comportamientos (por ejemplo CEBOT), agentes (por ejemplo ARTIS) o componentes/servicios (por ejemplo OROCOS) representa una limitación a la adaptabilidad e interoperabilidad de la arquitectura resultante. Una nueva arquitectura debería dar soporte a colectivos basados en cualquiera de estas orientaciones.
- Implementaciones que solo funcionan en una plataforma o son poco reutilizables cuando se cambian los modelos de robot.

5.5.2 Comparativa de los resultados

5.5.2.1 Comparativa de los resultados de las características

Los resultados de la evaluación de las características refuerzan lo ya observado sobre la ausencia de capacidades:

- Una reducida capacidad en las arquitecturas evaluadas para distribuir el autoconocimiento entre los miembros del colectivo con excepción de DRA.
- Una prácticamente nula habilidad para gestionar el colectivo.
- Una capacidad para integrarse con otros sistemas que podría mejorarse.

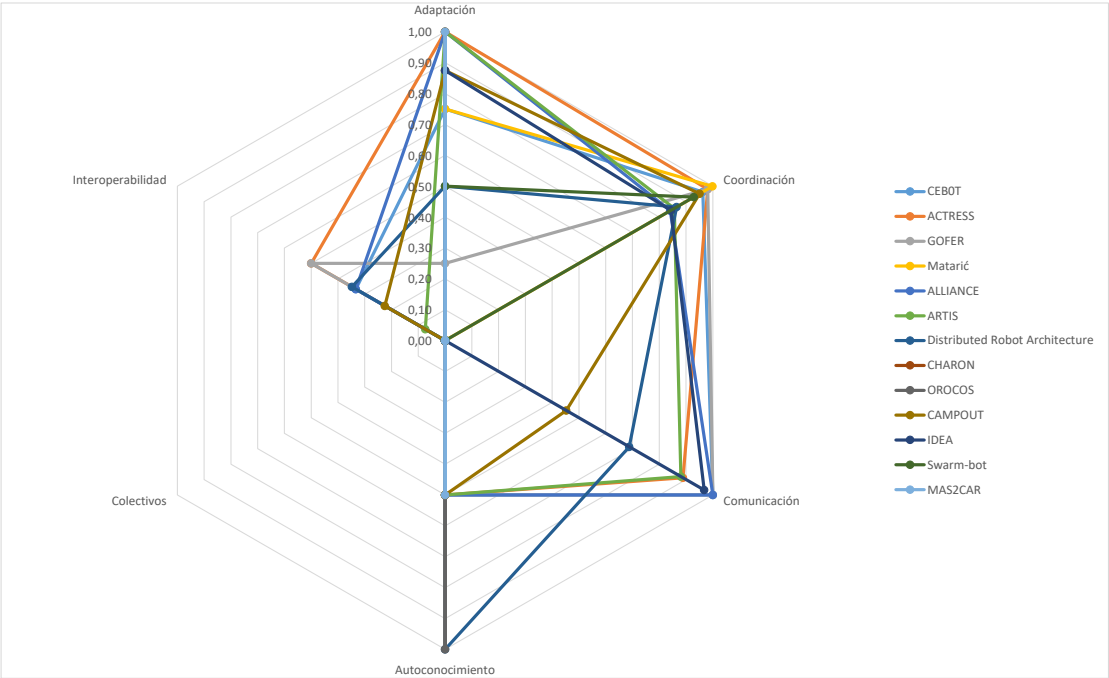


Ilustración 106: Comparativa de resultados para las características

Por otra parte, la adaptación a distintos escenarios y la eficiencia de las comunicaciones son variables.

En la dimensión de coordinación la diferencia es reducida, existiendo dos grupos: aquellas arquitecturas que tienen protocolos de coordinación más pesados tienen un comportamiento con respecto a la eficiencia peor que aquellas arquitecturas con protocolos menos robustos o sin ningún protocolo. Esto se debe a la simplicidad de la tarea usada para la métrica que permite su resolución sin un uso de procedimientos de coordinación complejos.

5.5.2.2 Comparativa de los resultados de las capacidades

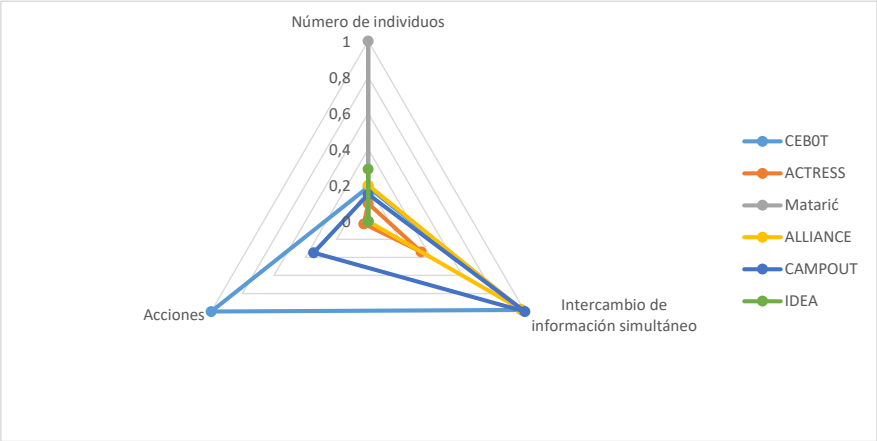


Ilustración 107: Comparativa de los resultados de las métricas de capacidades

En la Ilustración 107 podemos ver una comparativa gráfica de las capacidades frente al máximo obtenido en cada métrica. Aquellas arquitecturas con una menor (o nula) dependencia de las comunicaciones (como la propuesta por Matarić o IDEA) tienen una menor limitación en el número de individuos y, al mismo tiempo, están más limitadas en el intercambio de información y en la capacidad de ejecutar acciones de forma directa sobre individuos diferentes a aquellos donde se ejecuta el proceso.

De igual forma que lo visto en cuanto a las características, aquellas arquitecturas con protocolos más complejos tienen una eficiencia menor en la ejecución de acciones concretas que requieren la comunicación con otros individuos.

5.5.2.3 Comparativa de los resultados de la simulación del problema

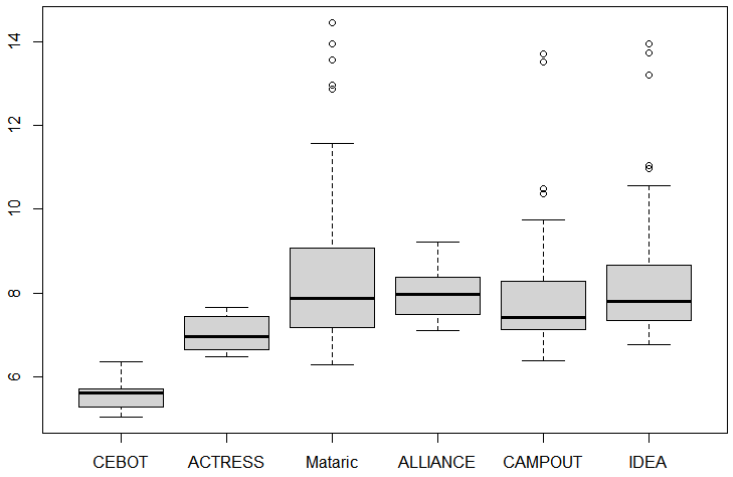


Ilustración 108: Diagrama de cajas y bigotes para los resultados de la métrica de tiempo (en segundos). ACTRESS y, sobre todo, CEBOT obtienen unos resultados mejores en comparación con el resto.

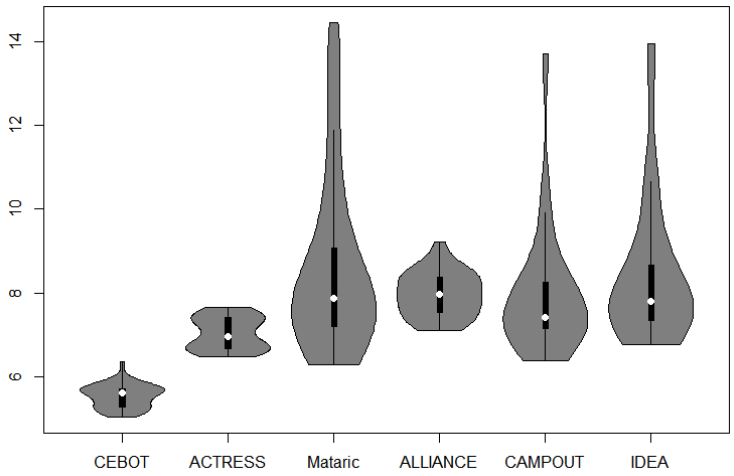


Ilustración 109: Diagrama de violín con la distribución de los resultados de la métrica de tiempo. ACTRESS y, sobre todo, CEBOT obtienen unos resultados mejores en comparación con el resto.

En la Ilustración 108 y la Ilustración 109 podemos ver una comparativa de los resultados de la métrica de tiempo consumido en la resolución del problema. Los resultados en tiempo son similares en Matarić, ALLIANCE, CAMPOUT e IDEA, mientras que son ligeramente mejores en ACTRESS y notablemente mejores en CEBOT.

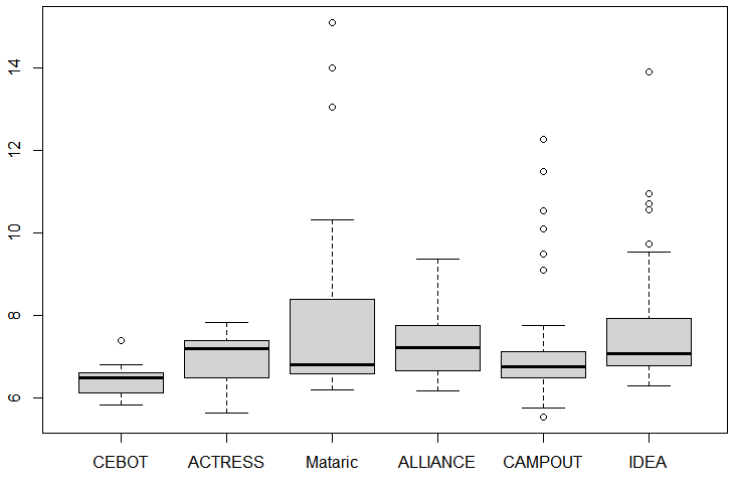


Ilustración 110: Diagrama de cajas y bigotes para los resultados de la métrica de consumo total de energía.

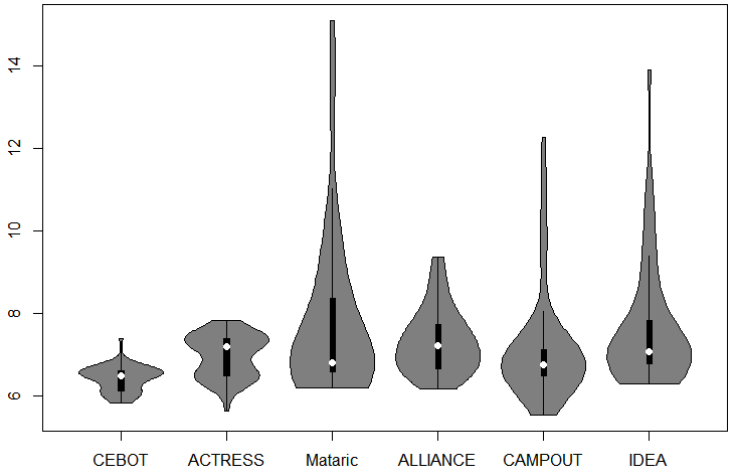


Ilustración 111: Diagrama de violín con la distribución de los resultados de la métrica de consumo total de energía.

Los resultados para la métrica de consumo de energía (ver Ilustración 110 e Ilustración 111) son similares a los de tiempo empleado, teniendo CEBOT, de nuevo, mejores resultados que el resto de arquitecturas, siendo las diferencias entre el resto de arquitecturas menores que en la métrica de tiempo.

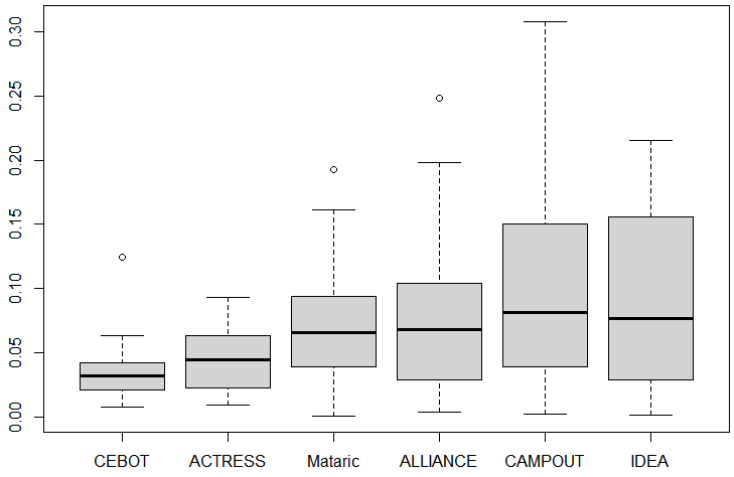


Ilustración 112: Diagrama de cajas y bigotes para los resultados para la métrica de distancia (precisión). ACTRES y, sobre todo, CEBOT obtienen mejores resultados siendo el resto de arquitecturas más variables.

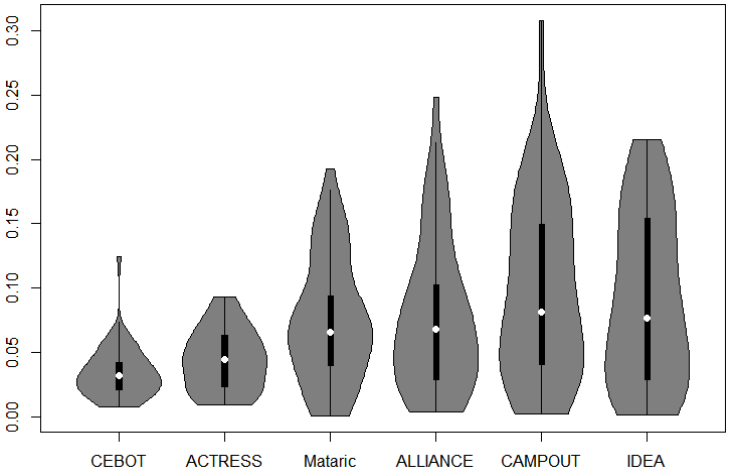


Ilustración 113: Diagrama de violín para la distribución de los resultados para la métrica de distancia (precisión). ACTRES y, sobre todo, CEBOT obtienen mejores resultados siendo el resto de arquitecturas más variables.

Al igual que en el resto de arquitecturas, CEBOT obtiene mejores resultados de precisión que el resto de arquitecturas (ver Ilustración 112 e Ilustración 113).

5.5.3 Conclusiones

Todas las arquitecturas evaluadas carecen de capacidades de gestión explícita del colectivo que son deseables para mejorar la tolerancia a fallos (expulsión de

individuos que causan problemas) y la adaptación a tareas (por ejemplo, la división del colectivo).

La correcta sincronización entre las acciones de distintos individuos es altamente dependiente de las capacidades de comunicación de la arquitectura. Este efecto es opuesto en la métrica para el número de individuos, siendo las comunicaciones una de las fuentes de limitación del colectivo. Como resultado de esto, las arquitecturas sin capacidades de comunicación explícita o aquellas con limitaciones producen mejores resultados en la capacidad de número de individuos pero peores resultados en la distancia horizontal del problema propuesto.

Para que una arquitectura pudiera enfrentarse de forma eficiente a una gran variedad de problemas, debe poder operar de las dos formas anteriormente descritas: en primer lugar, con un sistema de comunicaciones que le permita una sincronización entre sus acciones para maximizar la precisión del resultado y, en segundo lugar, sin dependencia de las comunicaciones para poder operar en mayor número de individuos y con menos requisitos técnicos sobre los mismos.

La capacidad de controlar actuadores de otros individuos de forma directa o disparar comportamientos locales simples a partir de las comunicaciones (como hacen arquitecturas como ALLIANCE) que realicen estas acciones son las opciones mayoritarias para sincronizar acciones; pero CAMPOUT posee mensajes específicos (parar y continuar) para la mejor sincronización de las acciones.

Los resultados de las simulaciones del problema muestran una ventaja para las arquitecturas simples con capacidad de control remoto de los actuadores en otros individuos, en especial para CEBOT.

Parte III

Nueva arquitectura

Capítulo 6

Una nueva propuesta de arquitectura

Tras analizar las distintas arquitecturas, los sistemas de evaluación, evaluar las arquitecturas y analizar los resultados, hemos identificado las necesidades (no todas ellas resueltas en las soluciones encontradas) a las que debe dar solución una nueva arquitectura. Además, gracias al sistema de evaluación, podemos comparar la arquitectura diseñada con las anteriores.

Proponemos una nueva arquitectura capaz de resolver los problemas de coordinación y control de un grupo de robots heterogéneo para integrarlos al nivel más alto posible pero dejando a elección del diseñador del sistema final (el usuario de la arquitectura) todas aquellas características que admitan varias opciones.

6.1 Objetivos para una nueva arquitectura

El objetivo principal es disponer de una arquitectura que permita implementar cualquier tipo de sistema multi-robot disponiendo de una variedad de componentes intercambiables y combinables que así lo permitan. Dividiremos los objetivos concretos que debe cumplir una nueva arquitectura en grupos, asignándolos a subsistemas, para facilitar su comprensión.

6.1.1 Objetivos específicos del subsistema de procesado

Existe un conjunto de requisitos para la transformación de estímulos en acciones (llamaremos genéricamente a esta transformación *procesado*):

- Ejecución paralela de procesos de forma distribuida.

- Las decisiones y coordinación sobre las acciones deben poder distribuirse sobre un subgrupo o sobre el conjunto del colectivo para evitar puntos críticos.

6.1.2 Objetivos específicos del subsistema de comunicaciones

El subsistema de comunicaciones es el que se ocupa del intercambio de información entre individuos/robots/nodos. Este subsistema puede tomar muchas formas y descansa sobre el hardware correspondiente. Sus requisitos específicos son:

- Subsistema de comunicaciones transparente capaz de crear redes de malla si la tecnología subyacente no lo permite directamente.
- Debe permitir cambios en los protocolos de comunicación y de mensajes sin afectar al funcionamiento.
- Acceso a los dispositivos situados en otras plataformas incluyendo estrategias de resolución de conflictos y sincronización:
    - Recibir la información de los sensores.
    - Activar los actuadores.
    - Relé de comunicaciones.

6.1.3 Objetivos de interoperabilidad

Como ya se ha destacado, la interoperabilidad de la arquitectura con otros sistemas y su adaptación a ellos es de gran importancia para maximizar su utilidad, requiriéndose:

- Independencia de la plataforma computacional sobre la que se ejecute.
- Integración con otros sistemas:
    - Diferentes sistemas robóticos (como ROS [117], YARP [118]…).
    - Entornos de simulación (MORSE [115, 116], Webots [100]…).

6.1.4 Objetivos sobre el colectivo

Se engloban aquí los objetivos relacionados con la dimensión de colectivos que se describió para las arquitecturas:

- Funciones de gestión del colectivo:

    - Adopción de nuevos miembros.
    - Expulsión de miembros.
    - División del colectivo.
    - Asimilación o fusión con otros colectivos.
- Superposición de colectivos estableciendo diferentes grupos jerárquicos o no de colectivos dentro del conjunto de individuos.

6.1.5 Objetivos aplicables a todo el sistema

- Distintos niveles de tolerancia a fallos según sea necesario para cada sistema, desde no aplicar ninguna técnica hasta que el fallo de todos los individuos menos uno permita intentar continuar con la misión.
- Estrategias para la resolución de conflictos (espacio, recursos externos e internos…).
- Alta modularidad y disponibilidad de varios módulos con diferentes implementaciones y, cuando sea posible, reemplazables en caliente.

6.1.6 Otros objetivos

- Autoconocimiento de los dispositivos disponibles tanto a nivel individuo como a nivel colectivo (sensores, actuadores y sistemas de comunicación disponibles).

6.2 Diseño

6.2.1 Visión general de la arquitectura

De igual forma que en Distributed Robot Architecture [44], la arquitectura propuesta se divide en varias capas, repartiendo entre ellas funciones específicas. Pero a diferencia de la arquitectura mencionada no supone una orientación concreta.

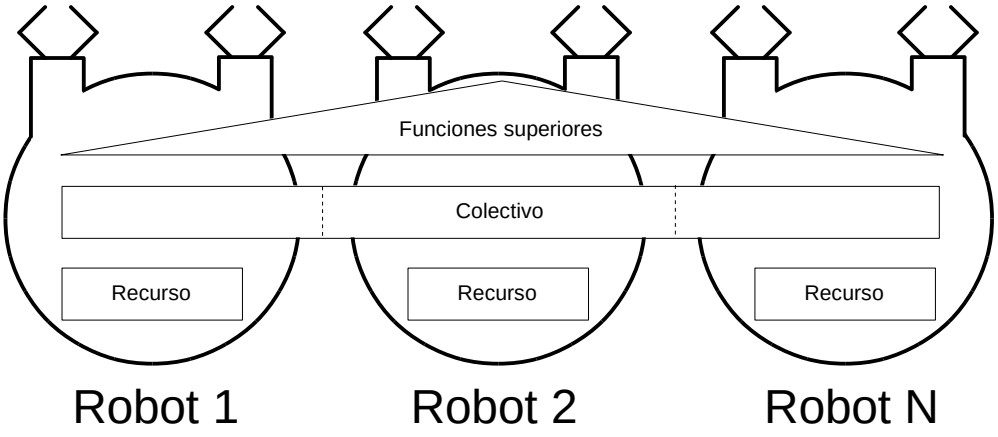


Ilustración 114: División en capas de la nueva arquitectura

6.2.1.1 Capa recursos

La capa inferior (recursos) se encarga de encapsular las funcionalidades propias de cada individuo/robot/nodo. Entre ellas están:

- Información sobre los dispositivos disponibles (sensores, actuadores, comunicaciones…).
- Ejecución de acciones de forma transparente.
- Composición de acciones simples.
- Lectura de los datos de los sensores.

En esta capa reside la conexión con los dispositivos del individuo, conformando una capa de abstracción del hardware.

6.2.1.2 Capa colectivo

La capa intermedia será la encargada de las funciones de coordinación entre los individuos que forman parte del colectivo. Sus responsabilidades concretas son:

- Servicios de soporte básico: individuos disponibles, información de los dispositivos disponibles en cada individuo, datos de los sensores, resultados de operaciones parciales... todo ello en forma de un repositorio de información.
- Resolución de conflictos y coordinación entre nodos para la ejecución de acciones por parte de la capa inferior.
- Control del estado de los individuos, lo que incluye la detección de individuos en estado de funcionamiento incorrecto.
- Dar soporte a la reasignación de procesos a otros individuos cuando un individuo deja de estar disponible por alguna razón.

- Proceso de adopción de nuevos individuos (incluyendo transferencia de la "herencia cultural", es decir, el repositorio).
- Distribución de los cambios en el estado global del colectivo a los individuos que deban conocerlos.
- Soporte a la generación de acciones agregadas (aprendizaje de acciones coordinadas en un grupo).

6.2.1.4 Capa funciones superiores

Esta capa se encarga de proporcionar la generación de acciones a partir de los estímulos de forma que el comportamiento resultante sea cooperativo y posiblemente inteligente.

No hay restricciones sobre esta capa, que puede llegar a contener varias arquitecturas, permitiendo así implementar la idea de varios colectivos (con el grafo que se desee) conviviendo en un individuo.

6.2.1.5 Componentes

Cada plataforma (robot o equipo con alguna capacidad para ser incluido en la arquitectura) ejecuta varios componentes de la arquitectura pero no tiene por qué ejecutarlos todos, dependiendo de las decisiones de diseño del sistema multi-robot y sus objetivos.

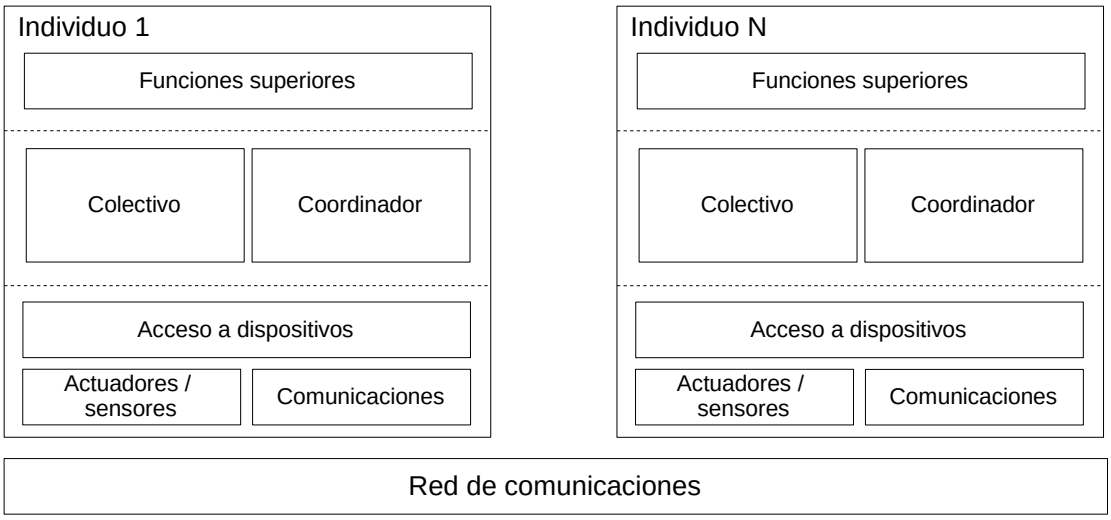


Ilustración 115: Visión general de los elementos de la arquitectura

Las abstracciones de la visión general se concretan en componentes con responsabilidades concretas y aisladas.

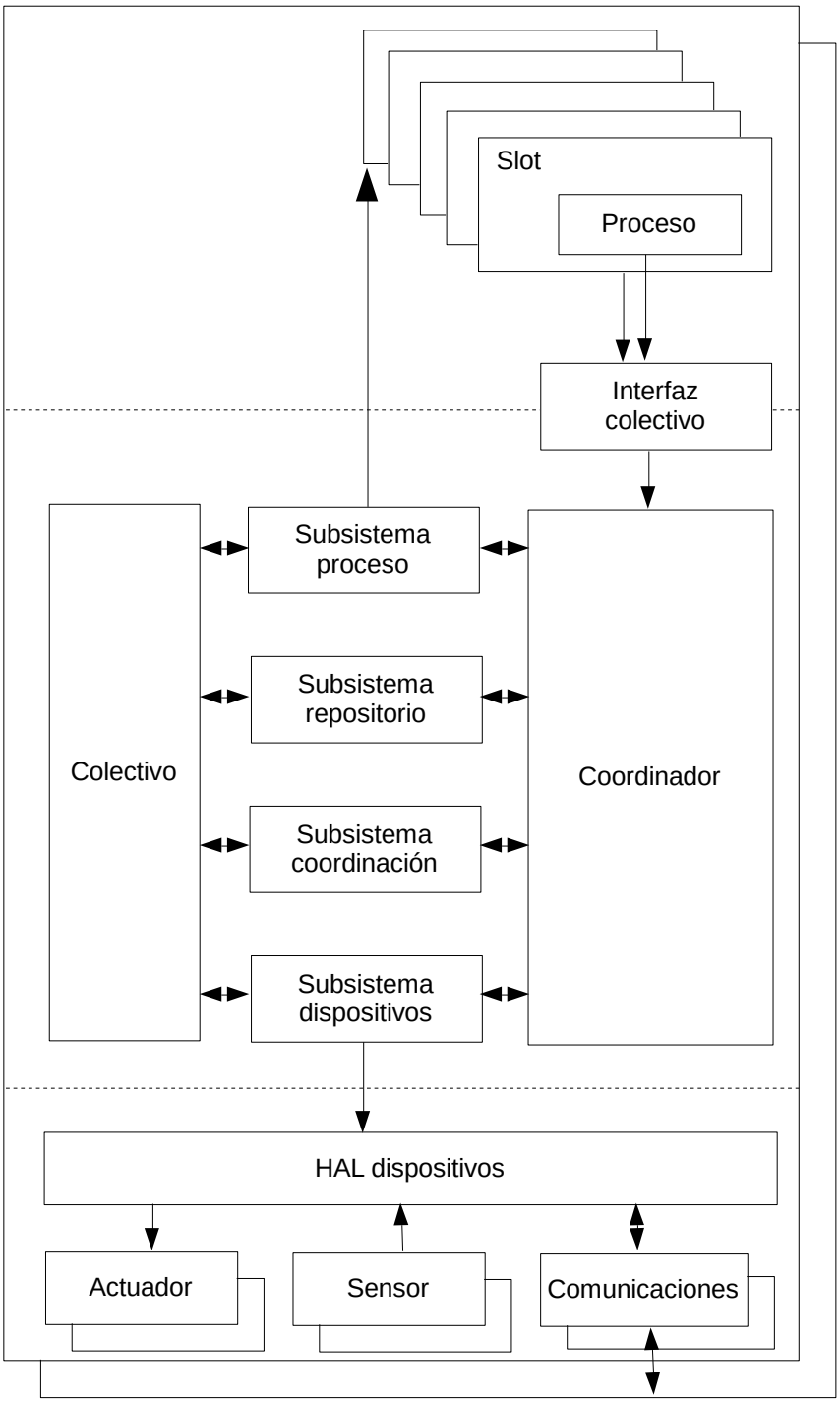


Ilustración 116: Estructura de bloques para un individuo del colectivo

En el diagrama de bloques de la Ilustración 116 podemos ver los principales componentes de la arquitectura; vemos en ella que la capa de colectivo se divide en varios subsistemas, cada uno de los cuales soporta una responsabilidad concreta. Dichos subsistemas están pensados para disponer de varias implementaciones intercambiables de forma transparente para el resto del sistema (lógicamente respetando su interfaz).

En la capa de funciones superiores se ejecutan procesos (pueden ser comportamientos, agentes…) dentro de slots de proceso estancos. Estos slots mantienen informada del estado del proceso a la capa del colectivo y de igual forma el propio proceso puede solicitar acciones sobre la misma. Ambos usan un interfaz intermedio que sigue el patrón Mediador ocultando la posible complejidad entre las dos capas.

El Coordinador recibe y procesa de forma ordenada las solicitudes provenientes de todo el sistema priorizando, sincronizando y arbitrando, según sea necesario, el acceso a los recursos.

El subsistema de proceso gestiona los procesos que se ejecutan en la capa de funciones superiores seleccionado su slot y (si es un slot local) iniciando su ejecución.

El subsistema de repositorio se ocupa del almacenamiento de los datos relevantes para todo el colectivo (comportamientos, ubicaciones…). Estos datos suelen ser el resultado de un proceso y no variables internas del mismo.

El subsistema de coordinación se ocupa de sincronizar las acciones usando bloqueos sobre partes del colectivo (por ejemplo un actuador) cuando es necesario; de esta forma solo el proceso que posee ese bloqueo puede interactuar con él. No se requiere un bloqueo sobre un elemento para poder usarlo, es una decisión de diseño del sistema multi-robot que corresponda.

El subsistema de dispositivos descansa directamente sobre la capa de recursos y es el único que interactúa con la misma. Este subsistema es el responsable de recabar el listado de dispositivos (actuadores, sensores…) que posee el individuo de las capas inferiores.

El componente de abstracción del hardware permite una independencia de las implementaciones concretas de los drivers de acceso a los dispositivos.

Los componentes de tipo actuador se comunican directamente con el dispositivo hardware (o simulado) que permite realizar acciones que modifican el entorno.

Por su parte los componentes de tipo sensor reciben datos directamente del dispositivo hardware (o simulado) que permite conocer algún tipo de estado del entorno.

De la misma manera los componentes de comunicaciones interactúan directamente con el dispositivo hardware (o simulado) que permite enviar mensajes a otro individuo.

6.2.2 Diseño detallado

Una vez vista la asignación de las responsabilidades a partes del sistema, procederemos a exponer sus interfaces y estructuras internas.

6.2.2.1 Interfaz del colectivo

Comenzaremos por definir el interfaz del colectivo, siendo el punto de entrada de todas las solicitudes con origen en el propio individuo y por lo tanto el que mejor puede definir el funcionamiento del sistema.

Ilustración 117: Definición del interfaz al colectivo

Modelizamos este interfaz como una clase (InterfazColectivo) no instanciable cuyos métodos son estáticos y con el conocimiento necesario del propio sistema para delegar las solicitudes a la parte del mismo que le corresponda. De esta forma, cualquier proceso o sistema externo puede hacer uso de funcionalidades de la arquitectura sin necesidad de conocer sus detalles o la ubicación física de los dispositivos a los que se trata de acceder.

Las funciones que proporciona este interfaz son:

- executeAction: lanza un nuevo proceso dentro del colectivo. Esta función inicia el proceso de selección del slot, transferencia de los parámetros (de existir), registro de la existencia del proceso en el colectivo, etc. que culmina con el proceso ejecutándose en alguno de los individuos del colectivo.
- endExecuteAction: finaliza la ejecución de un proceso (se entiende que porque ya no es necesario o hay otro con más prioridad).
- registerData: añade o reemplaza datos del repositorio del colectivo. Esto supone que la nueva clave de conocimiento que se guarde puede enviarse al resto de individuos del colectivo según sea la estrategia definida.
- removeData: elimina una clave del repositorio del colectivo, por lo que, dependiendo de la estrategia de replicación de los datos definida, puede requerir que se borre en varios o en todos los individuos.
- requestFullData: permite la transferencia completa del contenido del repositorio del colectivo desde otros individuos a los que se les solicita dicho contenido.
- observe: subscribe un proceso para recibir notificaciones cada vez que el estado de un dispositivo o el contenido de un elemento del repositorio cambien de forma relevante. Si el dispositivo es un sensor recibiremos una notificación cuando cambie el valor de su detección, si es un actuador, cuando el mismo reciba una instrucción para ejecutar una acción; en el caso de un dispositivo de comunicaciones, la notificación se generará con los cambios en el estado de conectividad y en las estadísticas de transmisión. Coordinador es responsable de la agrupación y eliminación de estos registros si el proceso que lo solicita deja de estar disponible.
- stopObserve: elimina el registro de cambios anterior.
- move: ejecuta una acción sobre un actuador. Esto supone la coordinación de la misma con el individuo que posee ese actuador y con cualquier otro individuo que desee usarlo.

- newIndividual: añade información referente a un nuevo individuo que no pertenece al colectivo.
- removeIndividual: expulsa a un individuo del colectivo modificando el repositorio.
- adopt: inicia el proceso de inclusión de un individuo con una instancia de la arquitectura en un colectivo. Puede que el individuo sea el que inicia el proceso.
- asimilate: inicia el proceso de inclusión de un individuo cuyo software es desconocido en un colectivo, forzando en él la ejecución de una nueva instancia de la arquitectura.
- fusión: inicia el proceso de fusión con otro colectivo, lo que resultará en uno solo.
- divide: inicia el proceso de división binaria del colectivo que concluirá con la creación de dos colectivos a partir del original.
- getData: recupera el valor actual de una clave del repositorio. Dependiendo de la estrategia de distribución, esto puede requerir enviar una solicitud a otro individuo.
- getDeviceList: permite el acceso a las funciones de autoconocimiento del colectivo devolviendo la lista completa de dispositivos disponibles, pudiendo restringir dicha lista por individuos o tipos (sensor, actuador y comunicaciones).
- getIndividuals: es otra de las características de autoconocimiento que permite acceder al listado de individuos que conforma el colectivo.
- getDataIds: recupera un listado completo de todas las claves almacenadas en el repositorio.
- getColectiveName: devuelve el identificador arbitrario del colectivo.
- getSensorData: permite acceder a los datos más recientes de un sensor de forma transparente a su ubicación y tiempo de registro.
- getLock: permite obtener una restricción de acceso a un recurso concreto respetando las condiciones solicitadas.
- removeLock: elimina la restricción de acceso a un recurso concreto; solo puede ejecutarlo el proceso que lo ha solicitado previamente.

- faultDetected: informa al Coordinador que se ha detectado un fallo en alguno de los dispositivos o individuos del sistema indicando, en la medida de lo posible, el alcance del mismo.
- modifySlots: cambia el número de slots disponibles en un individuo (no necesariamente el individuo donde se está ejecutando el método). Puede incluir una estrategia para decidir qué hacer con los procesos en ejecución dentro de los slots en caso de reducción.

6.2.2.2 Información interna del colectivo

Dependiendo del sistema que se desea implementar, cada individuo debe poseer un nivel de información sobre el colectivo diferente, desde una copia completa de toda la información a solo la información del propio individuo (o incluso solo algunos elementos del propio individuo).

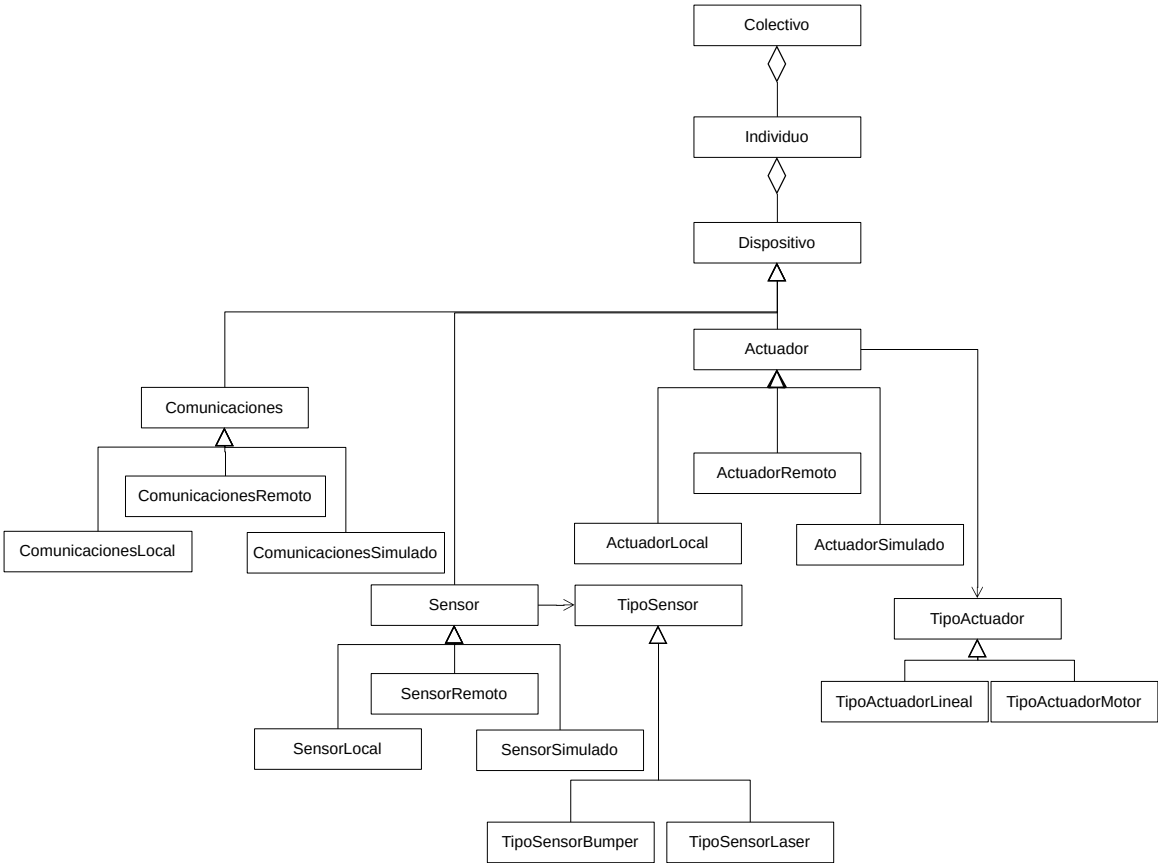


Ilustración 118: Estructura de información del colectivo

Como podemos ver en el diagrama de clases de la imagen anterior, hay varios elementos de importancia:

- Colectivo: representa un colectivo encapsulando su autoconocimiento. Puede haber varios colectivos dado que la arquitectura no se limita a un solo colectivo.
- Individuo: encapsula los datos de un individuo, ya sea el local o uno remoto.
- Dispositivo: clase de la que heredan todos los dispositivos de un nodo permitiendo el tratamiento unificado de todos los dispositivos más allá de su tipo, ubicación, etc.
- Sensor: clase abstracta padre de todos los sensores.
- TipoSensor: tipo de sensor, indicando su funcionalidad, rendimiento, etc. A pesar de que solo aparecen dos ejemplos en el diagrama, deben existir múltiples opciones, siendo otros ejemplos: cámara, sensor de distancia por infrarrojos, acelerómetro, etc.
- Actuador: clase abstracta padre de todos los actuadores.
- TipoActuador: tipo del actuador que permite comprender la funcionalidad que aporta el mismo, con posibles datos de consumo energético, rendimiento, etc.
- Comunicaciones: cualquier dispositivo que permita el intercambio de información sin modificar el entorno pertenece a esta clase abstracta.

Todos los dispositivos poseen un homomorfismo en su herencia para indicar su ubicación o existencia real:

- Local: Representa un dispositivo cuyo sistema de control se ubica en el propio individuo (el dispositivo en sí puede no estar físicamente conectado al individuo).
- Remoto: Representa un dispositivo cuyo sistema de control se ubica en otro individuo, pero el individuo de la estructura tiene acceso a dicho dispositivo.
- Simulado: representa un dispositivo que no existe en la realidad pero cuyo sistema de control se ubica en el propio individuo.

6.2.2.3 Coordinador

El coordinador es el componente central del sistema, interconectado con el resto de componentes y responsable de coordinar la ejecución de las peticiones de servicio que le llegan desde componentes locales y remotos.

La implementación de este componente es altamente dependiente de múltiples factores, por lo que deben existir varias implementaciones diferentes. Entre los factores de los que depende la implementación se encuentran:

- Las comunicaciones: capacidades del sistema de comunicaciones y el protocolo de mensajes.
- La estrategia de coordinación: por ejemplo, priorización de las peticiones.
- La información inicial del colectivo: cómo se forma inicialmente el colectivo.
- Las capacidades y cómo se implementan: por ejemplo la gestión del colectivo (si se dispone de las opciones para adoptar miembros, detección de individuos ajenos al colectivo…).

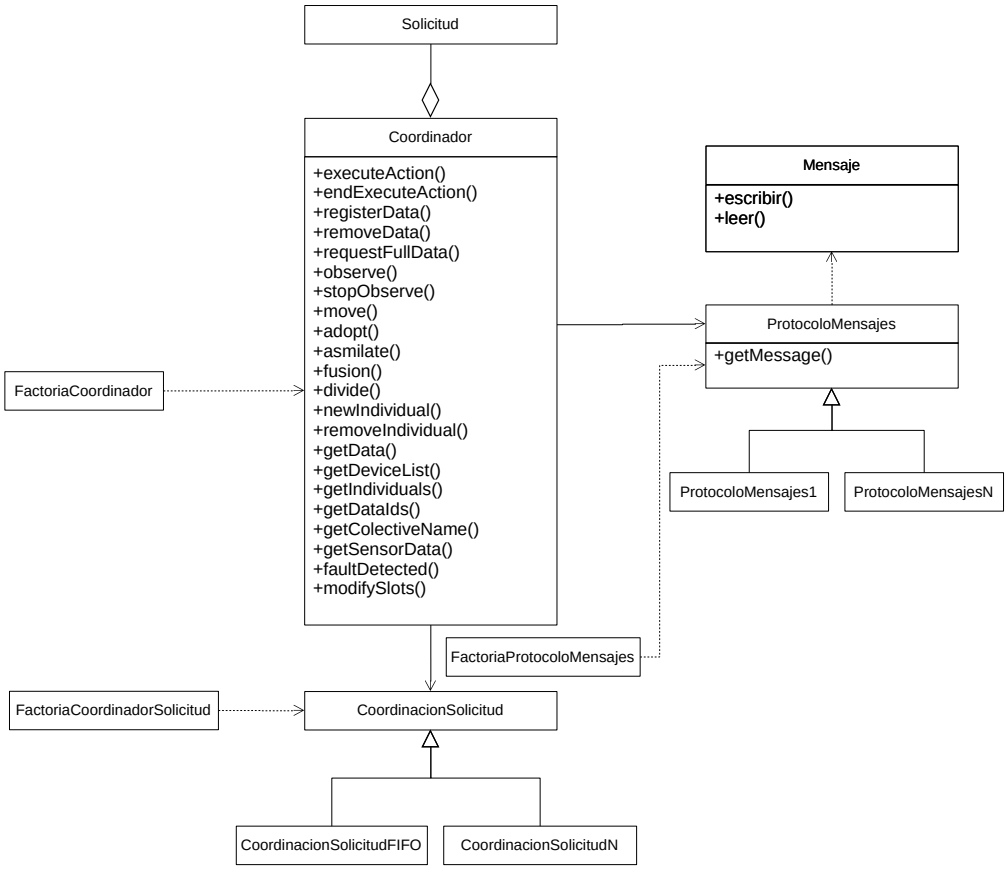


Ilustración 119: Diagrama de clases para el coordinador del colectivo

Para permitir que este componente disponga de suficiente versatilidad y se pueda modificar con facilidad, la implementación concreta del controlador se crea usando una factoría que identifica cuál debe instanciar gracias a parámetros de configuración.

La clase Coordinador de la imagen es abstracta y de ella se heredarán implementaciones concretas. Para facilitar la adaptación, parte de la responsabilidad del coordinador recae en componentes externos que implementan el patrón estrategia. De entre estos componentes, los de mayor importancia son el responsable del protocolo de mensajes (que se puede modificar según sea necesario) y el que se ocupa de la coordinación de las solicitudes. Estas jerarquías también disponen de sus propias factorías alimentadas desde la configuración.

Las llamadas a métodos de la clase Coordinador se traducen en la creación de instancias de la clase Solicitud que se incluyen en una lista. Estas instancias se priorizan usando la clase de CoordinacionSolicitud, ejecutando la que corresponda.

Cuando una solicitud debe generar una petición remota, Coordinador solicita un mensaje a la instancia concreta de ProtocoloMensajes que creará una instancia de la clase Mensaje conteniendo la información necesaria. La instancia de Coordinador usará la instancia de Mensaje y el subsistema de dispositivos para enviarlo al individuo que corresponda.

6.2.2.4 Subsistema de procesos

Cada individuo puede tener disponible una cierta capacidad de procesado útil para la capa de funciones superiores. Esta capacidad de proceso se traduce en slots donde poder ejecutar procesos.

En este contexto entendemos por proceso una función que la arquitectura considerará atómica. El proceso recibe unos parámetros y hará solicitudes sobre el InterfazColectivo para conseguir su propósito. Las instrucciones a ejecutar por el proceso (es decir: su código) son parte del conocimiento del colectivo y se almacenan en el repositorio, pero este subsistema no interactúa directamente con este último.

La estructura interna del subsistema sigue las mismas líneas que las empleadas en el resto de componentes: la instanciación de la implementación concreta de

FachadaProcesos (una clase abstracta) la realiza una factoría que obtiene la información de la clase a instanciar gracias a parámetros de configuración. El componente responsable de la selección del Slot (local o remoto) a usar para ejecutar el proceso implementa el patrón estrategia, con los diferentes métodos para dicha selección siendo clases independientes. Las estrategias de asignación que aparecen en el diagrama de clases son ejemplos de algunas de las implementaciones de selección del slot que deberían estar disponibles.

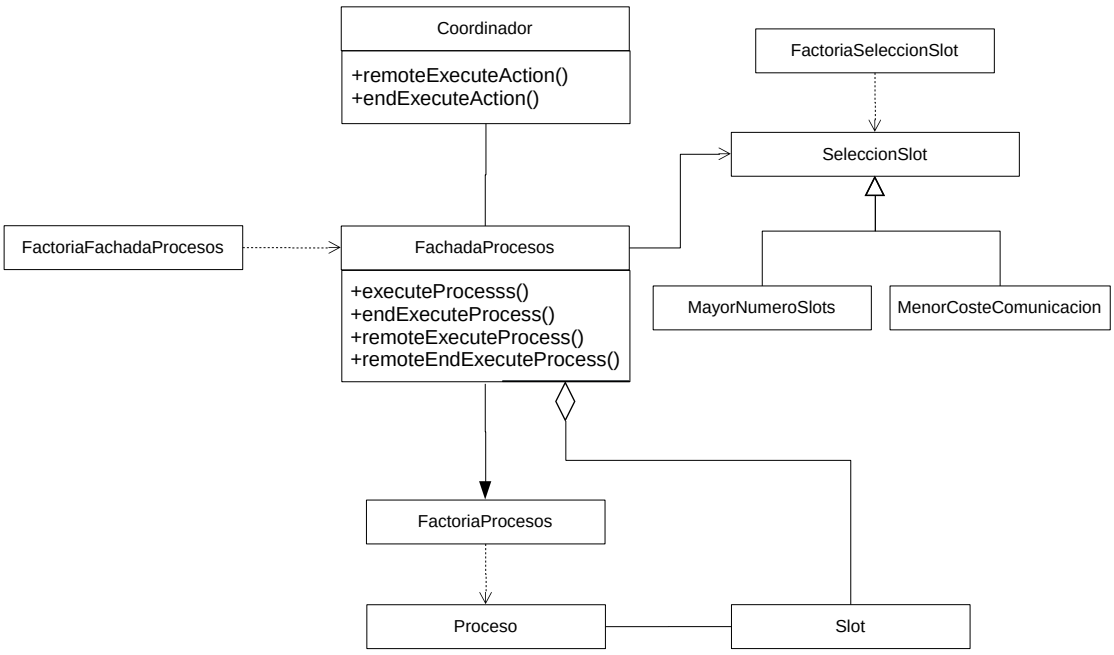


Ilustración 120: Diagrama de clases del subsistema de procesos

Para ejecutar un proceso, el coordinador solicita a la implementación de la FachadaProcesos que ha obtenido de la FactoriaFachadaProcesos que ejecute un proceso; la FachadaProcesos solicita a la implementación de la SeleccionSlot que ha obtenido de su factoría el Slot en el que se ejecutará. Si este Slot es local FacahadaProcesos le asignará el proceso y solicitará que inicie la ejecución del mismo. Si el Slot se encuentra en otro individuo, FachadaProcesos solicitará al Coordinador local que envíe el proceso al Coordinador remoto.

El subsistema FachadaProcesos también puede crear procesos cuando así se necesita (por ejemplo cuando se produce una situación excepcional en uno y debe reiniciarse).

La necesidad de disponer de varias implementaciones de FachadaProcesos viene dada entre otros factores por:

- Control del proceso: en la implementación más básica FachadaProcesos deja de interactuar con el proceso una vez asignado al Slot pero puede implementar el control de las excepciones del mismo.
- Gestión de los Slots: las necesidades energéticas pueden llevar a un individuo a reducir su capacidad de procesamiento (por ejemplo mediante la desactivación de núcleos de su CPU o bajando la frecuencia máxima de la misma). Esto debe traducirse en cambios en el número de Slots disponibles (la situación opuesta también puede darse con el aumento de slots).
- Tolerancia a fallos a nivel colectivo: se requiere una estrategia para volver a lanzar los procesos necesarios cuando un individuo deja de estar disponible.

6.2.2.5 Subsistema de repositorio

Este subsistema es el responsable del almacenamiento de conocimiento a nivel del colectivo garantizando que todos los procesos puedan acceder a la información que requieren más allá de la ubicación de ambos. El conocimiento se distribuirá entre los individuos según la estrategia seleccionada.

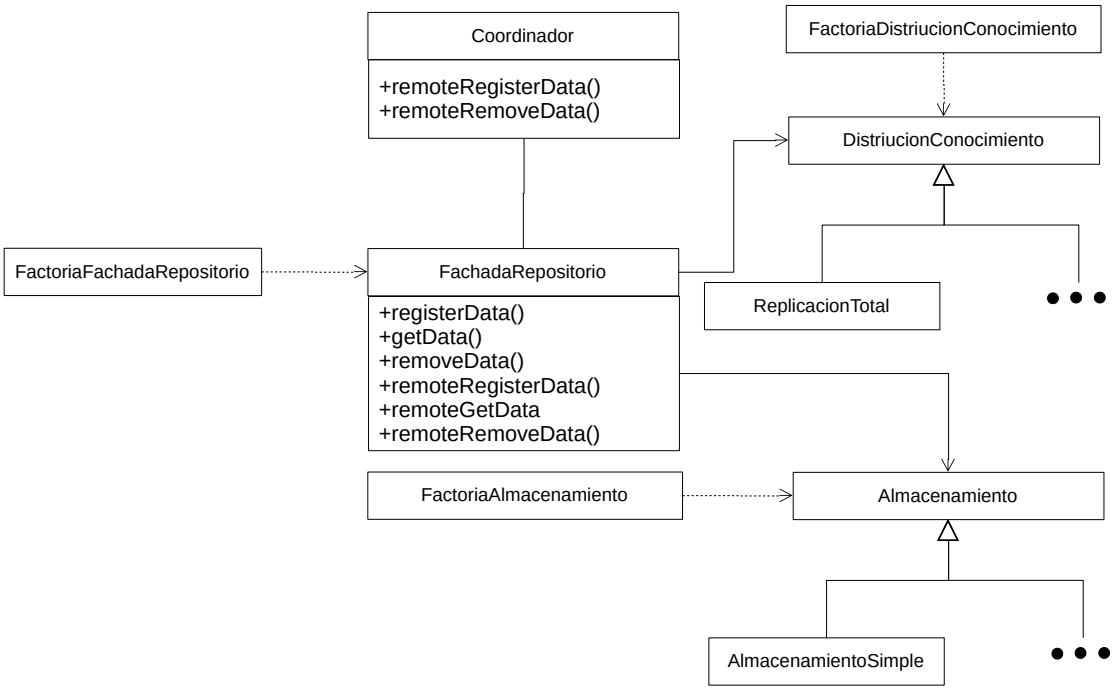


Ilustración 121: Diagrama de clases del subsistema de repositorio

De igual forma que el resto de subsistemas, las funciones principales del mismo son accesibles usando una fachada (FachadaRepositorio) cuya implementación concreta especializará los comportamientos tal como sea necesario para el sistema concreto.

En el caso de este subsistema solo hemos identificado dos elementos por los que sería necesario cambiar el comportamiento del subsistema: la distribución y el almacenamiento local. En ambos casos estas responsabilidades se aíslan de la fachada para maximizar la reutilización y facilitar la implementación de distintos sistemas.

La estrategia de distribución del conocimiento entre individuos (DistribucionConocimiento) se ocupa de, tras recibir una solicitud para almacenar un cierto conocimiento, identificar los individuos a los que se les remitirá dicho conocimiento para que se registre en sus respectivos repositorios.

Por su parte, el modo de almacenar los elementos de forma local tiene su propia jerarquía de clases (Almacenamiento) que define cómo se almacena (selección de dispositivos de almacenamiento, replicación…) y cómo se recupera (caches…).

6.2.2.6 Subsistema de coordinación

Ciertas acciones requieren una sincronización o una exclusividad en el uso de ciertos recursos. Este subsistema se ocupa de prevenir problemas por el acceso concurrente y de arbitrar cuando se produce algún tipo de conflicto de acceso a recursos entre dos procesos. El problema del acceso concurrente a los recursos se ve agravado en el escenario de procesos que accedan a los mismos de forma distribuida debido a los retrasos provocados por las comunicaciones además del propio paralelismo.

La estructura de las clases es similar a la del resto de subsistemas con una fachada (FachadaCoordinación) cuya instancia se obtiene empleando una factoría que identifica la clase a instanciar usando un parámetro de configuración.

Las restricciones de acceso por parte de otros procesos se pueden revocar en caso de que el Coordinador decida hacerlo (por ejemplo, si el proceso muere o hay otro proceso de mayor prioridad que solicita el acceso de forma urgente).

La solicitud de uso de un recurso (cualquier cosa del sistema susceptible de modificarse o usarse de forma exclusiva: comunicaciones, actuadores, sensores, elementos del repositorio…) incluye un cierto nivel de aislamiento (serializable,

simultáneo…), modelo de concurrencia (optimista/pesimista) y la sincronicidad de la misma (síncrona, asíncrona). Si esta solicitud tiene éxito, el subsistema genera una instancia de la clase Lock que le entrega al proceso para gestionar la referencia. La instancia de la clase Lock mantiene una referencia al proceso por si es necesario informarle de algún cambio. El resto de individuos es informado (o no, dependiendo de la configuración) de la creación de nuevas restricciones de acceso.

De la misma forma, cuando un proceso ya no necesita el acceso al recurso, puede eliminar las restricciones sobre el mismo y dicha eliminación se propagará al resto de individuos.

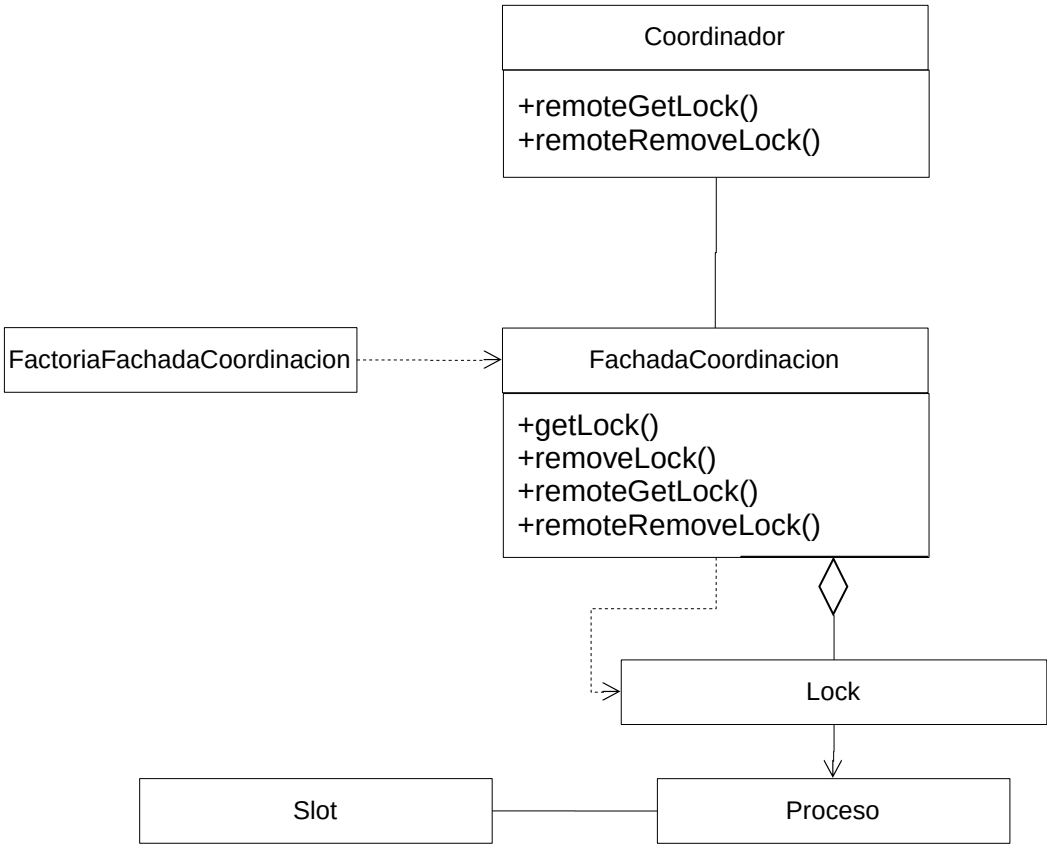


Ilustración 122: Diagrama de clases del subsistema de coordinación

6.2.2.7 Subsistema de dispositivos

Este subsistema encapsula todo el acceso a los distintos tipos de dispositivos (sensores, actuadores y comunicaciones) existentes en el colectivo.

En cuanto a los actuadores se encarga, haciendo uso de la capa HAL, de proceder a su activación y desactivación según las instrucciones, siendo capaz de

usar marcas de tiempo para permitir la sincronización de varias acciones a lo largo del colectivo.

Con respecto a los sensores, se ocupa de gestionarlos y de gestionar también la información que producen, siendo capaz de obtener la información de dos formas diferentes: la presente en el momento que se solicita o notificaciones con la nueva información según se vayan generando. La responsabilidad de saber cuándo se genera la nueva información recae en capas inferiores.

Los dispositivos de comunicación permiten enviar mensajes al resto de individuos del colectivo y este subsistema debe gestionar las colas de mensajes priorizándolos según corresponda (las comunicaciones pueden tener un coste: en energía, tiempo…). Es importante destacar que las habilidades para modificar la configuración de la red, ya sea física o lógica, (por ejemplo, el sistema de direccionamiento o la orientación de antenas) se consideran actuadores.

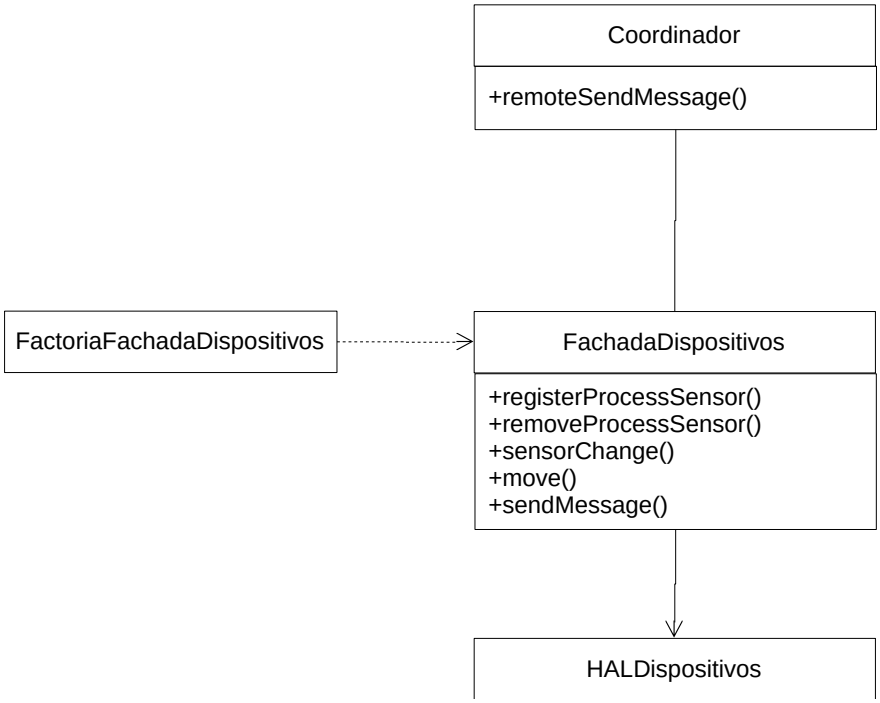


Ilustración 123: Diagrama de clases para el subsistema de dispositivos

La estructura del subsistema sigue el mismo esquema que en el resto de los mismos con una fachada (FachadaDispositivos) cuya implementación concreta es instanciada por una factoría que identifica la clase concreta gracias a un parámetro de configuración.

La necesidad de disponer de varias implementaciones de FachadaDispositivos viene dada, entre otros factores, por:

- Tratamiento (priorización, decisiones de enrutado…) de las colas de mensajes entre el Coordinador y los dispositivos de comunicaciones.
- Presencia de cachés en la información de los sensores.
- Gestión de la energía (el medidor de carga de la batería sería un sensor y la activación de otras fuentes de energía se modelizarían como actuadores, pero la gestión central del consumo de energía recaería en este subsistema).
- Nuevos tipos de dispositivos (fuera de actuadores, comunicaciones y sensores).

6.2.2.8 Colectivo

Este componente es una abstracción del conocimiento que los individuos poseen sobre el colectivo.

6.2.2.9 HAL dispositivos

La capa de abstracción del hardware para los dispositivos representa un conjunto de elementos software que presentan un interfaz común para las capas superiores siendo capaz de interactuar con los dispositivos conectados al individuo. Esta capa debe ser capaz de integrarse con otros sistemas, ya sean simulados o reales, permitiendo la ejecución de la arquitectura en múltiples plataformas robóticas. Además de permitir la integración con otros sistemas/plataformas robóticas, esta capa debe interactuar con el sistema operativo aislando al resto del sistema de las particularidades del mismo.

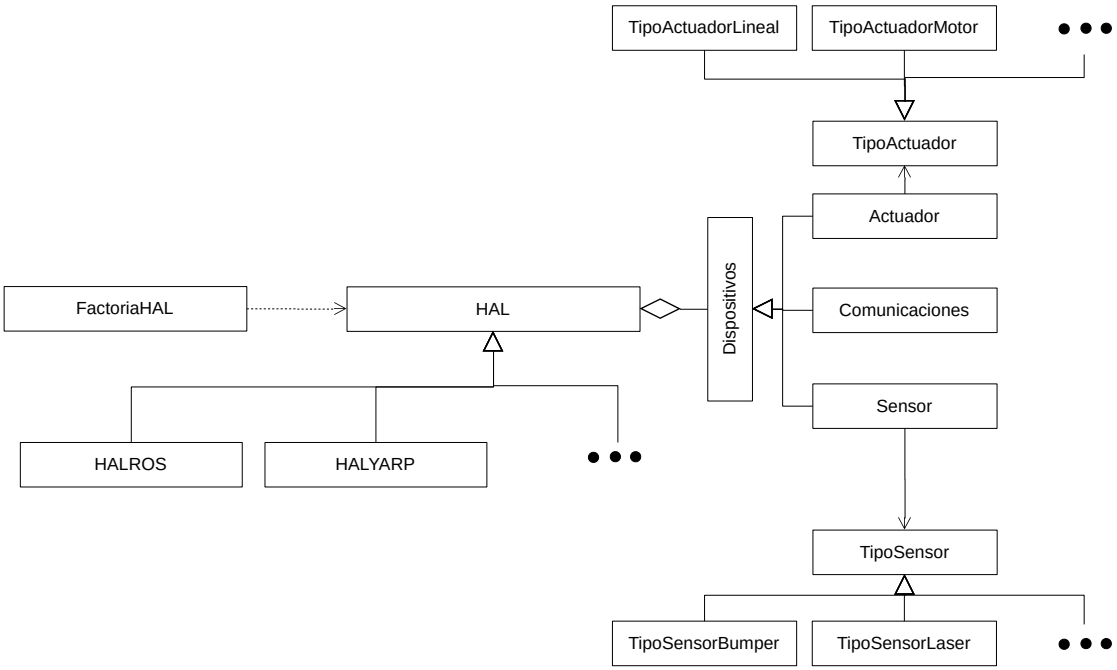


Ilustración 124: Diagrama de clases para la capa HAL

Además de abstraer las plataformas completas, esta capa debe abstraer las particularidades de acceso a los dispositivos propias de los mismos proveyendo un interfaz general para cada tipo de dispositivo (por ejemplo un actuador lineal conectado por una UART o por USB serán el mismo tipo para las capas superiores).

6.2.3 Funciones específicas del colectivo

Las funciones de la arquitectura definidas en los puntos anteriores son equivalentes a las disponibles en otras arquitecturas. Estas otras arquitecturas carecen en su mayor parte de funciones explícitas para la gestión del colectivo. A continuación definiremos aquellas consideradas más relevantes.

6.2.3.1 Detección de individuos no pertenecientes al colectivo pero con instancia de la arquitectura

Previamente a poder adoptar un nuevo miembro en el colectivo, este último debe disponer de la habilidad de detectar otros individuos que no formen parte del colectivo.

Para que esto sea posible, es necesario que los dispositivos de comunicaciones o los sensores sean capaces de detectar otros individuos. La detección puede ser simple (de la información obtenida se infiere directamente la existencia de un

nuevo individuo) o requerir el procesado de diferente información que puede tener su origen en diferentes dispositivos.

Para facilitar la detección de individuos definimos las siguientes técnicas básicas para los sistemas de comunicación:

- Responder con mensaje de presencia a mensajes de escaneo.
    - En caso de que el sistema permita la emisión por broadcast, cada cierto tiempo (intervalo entre escaneos), un individuo de un colectivo que esté buscando otros individuos emitirá varios mensajes separados una determinada cantidad de tiempo (ventana del intervalo) en la que esperará respuestas (dicha respuesta debe incluir algún identificador del nuevo individuo) y continuará emitiendo estos mensajes hasta que acabe el tiempo de escaneo. Los individuos que hacen estas búsquedas son seleccionados por el colectivo, pudiendo ser todos los individuos del mismo.
    - En caso de que el sistema de comunicaciones no permita hacer broadcast y el sistema disponga de direccionamiento, la técnica anteriormente descrita se usará recorriendo el espacio de direccionamiento completo.
- Escucha de todo el tráfico: si el dispositivo de comunicaciones lo permite, se puede escuchar el tráfico en busca de mensajes de individuos desconocidos. Incluso puede que todos los individuos emitan una cierta información (incluyendo su identificador) de forma periódica para facilitar su detección.

Los individuos detectados son automáticamente añadidos al repositorio de conocimiento estando sujeto, como todo conocimiento del sistema, a la estrategia de distribución del mismo por lo que todo el colectivo podría acceder a él. Al ser añadido por una observación de un dispositivo de comunicaciones, el colectivo dispondrá de todo lo necesario para poder contactar con el nuevo individuo. Existe la posibilidad de que un mismo individuo sea detectado de formas (desde individuos o dispositivos de comunicación diferentes) o en momentos distintos. El repositorio actualizará la información del individuo según corresponda.

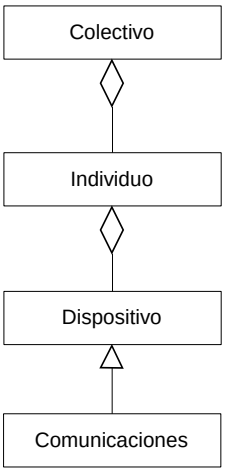


Ilustración 125: Información añadida al repositorio gracias a la detección usando el sistema de comunicaciones

Todas las estrategias de detección a nivel del sistema de comunicaciones se implementan como parte del Subsistema de Dispositivos.

Estas técnicas son de especial importancia para la formación inicial del colectivo, ya que de esta forma el colectivo se formaría sin necesidad de establecer un listado de miembros en la configuración de todos los individuos incrementando notablemente la flexibilidad.

Además de las técnicas para la detección de nuevos individuos basadas en los dispositivos de comunicación, para posibilitar la existencia de detecciones complejas con información proveniente de distintas fuentes, existe un método en el InterfazColectivo (newIndividual) que permite añadir el conocimiento de la existencia de un nuevo individuo al sistema. Dado que, a priori, se desconoce a qué colectivo pertenece el nuevo individuo, el colectivo al que se asigna internamente es “Desconocido”.

6.2.3.2 Detección de otros colectivos

Otra capacidad relacionada con la anterior es la detección de colectivos diferentes al actual. En realidad, todos los individuos que ejecutan la arquitectura forman siempre parte de algún colectivo, aunque esté formado por un solo individuo.

Es difícil identificar nuevos colectivos si ellos no se identifican como tales, por lo que vamos a extender las técnicas descritas anteriormente, en especial la de escaneo, para incluir en la respuesta a dicho escaneo un identificador del colectivo

además del identificador del individuo. De esta forma podremos identificar nuevos colectivos mediante las técnicas descritas anteriormente. Como hemos mencionado, estas técnicas se implementan como parte del Subsistema de Dispositivos.

Un colectivo puede disponer de procesos que implementen fusión de datos para detectar nuevos individuos (procesos que usan información proveniente de diferentes fuentes). De existir, dichos procesos son los responsables de registrar en el colectivo los individuos identificados. Si el proceso en cuestión es capaz de identificar el colectivo, entonces podría indicarlo. En caso contrario, el individuo se agregará al colectivo "Desconocido".

6.2.3.3 Adopción de un individuo

El objetivo de este proceso es integrar un nuevo individuo en el colectivo de modo que su nivel de funcionalidad sea aprovechado de igual forma que cualquier otro. En este punto solo cubriremos la adopción de miembros que deseen incluirse en el colectivo y posean una implementación mínima de la arquitectura. Estos objetivos implican que, como mínimo:

- Deben extraerse las características técnicas del individuo (dispositivos disponibles, capacidad computacional…).
- La información sobre dichas características debe distribuirse al resto del colectivo.
- El contenido del repositorio del colectivo debe transferirse al nuevo individuo siguiendo la estrategia de distribución correspondiente.

Hay dos posibilidades para una adopción de este tipo dependiendo del punto de origen de la acción: iniciada por el nuevo individuo o iniciada por el colectivo. La detección basada en escaneo activo que describíamos anteriormente produce una detección recíproca (el individuo que escanea y el que es escaneado adquieren conocimiento mutuo) lo que permite que cualquiera de los dos pueda iniciar el proceso de integración en el colectivo. La decisión de adoptar un miembro o de ser adoptado por un colectivo de forma automática es tomada por el Coordinador, existiendo una función en InterfazColectivo para iniciar dicha acción.

El proceso de adopción, lo inicie quien lo inicie, posee los mismos pasos:

1. El Coordinador en el individuo del colectivo que adopta genera un Lock sobre el conocimiento del nuevo individuo.

2. El mismo Coordinador solicita al nuevo individuo el conocimiento correspondiente a sus dispositivos y capacidades.
3. El nuevo individuo envía la información solicitada.
4. El Coordinador actualiza en el repositorio colectivo la información, (lo que dependiendo de la estrategia de distribución de la información puede hacer que estos datos sean enviados a otros individuos dentro del colectivo.
5. El Coordinador construye un mensaje con el conocimiento completo del Colectivo y se lo envía al nuevo individuo. Es importante destacar dos detalles: el conocimiento incluye los procesos y el bloqueo sobre el nuevo individuo.
6. El nuevo individuo confirma el fin del proceso de incorporación del conocimiento del colectivo.
7. El Coordinador actualiza el repositorio indicando que el nuevo individuo pertenece al colectivo.
8. El Coordinador elimina el Lock sobre el individuo (que será propagado a todos los individuos, incluyendo el nuevo en función de la estrategia de distribución).

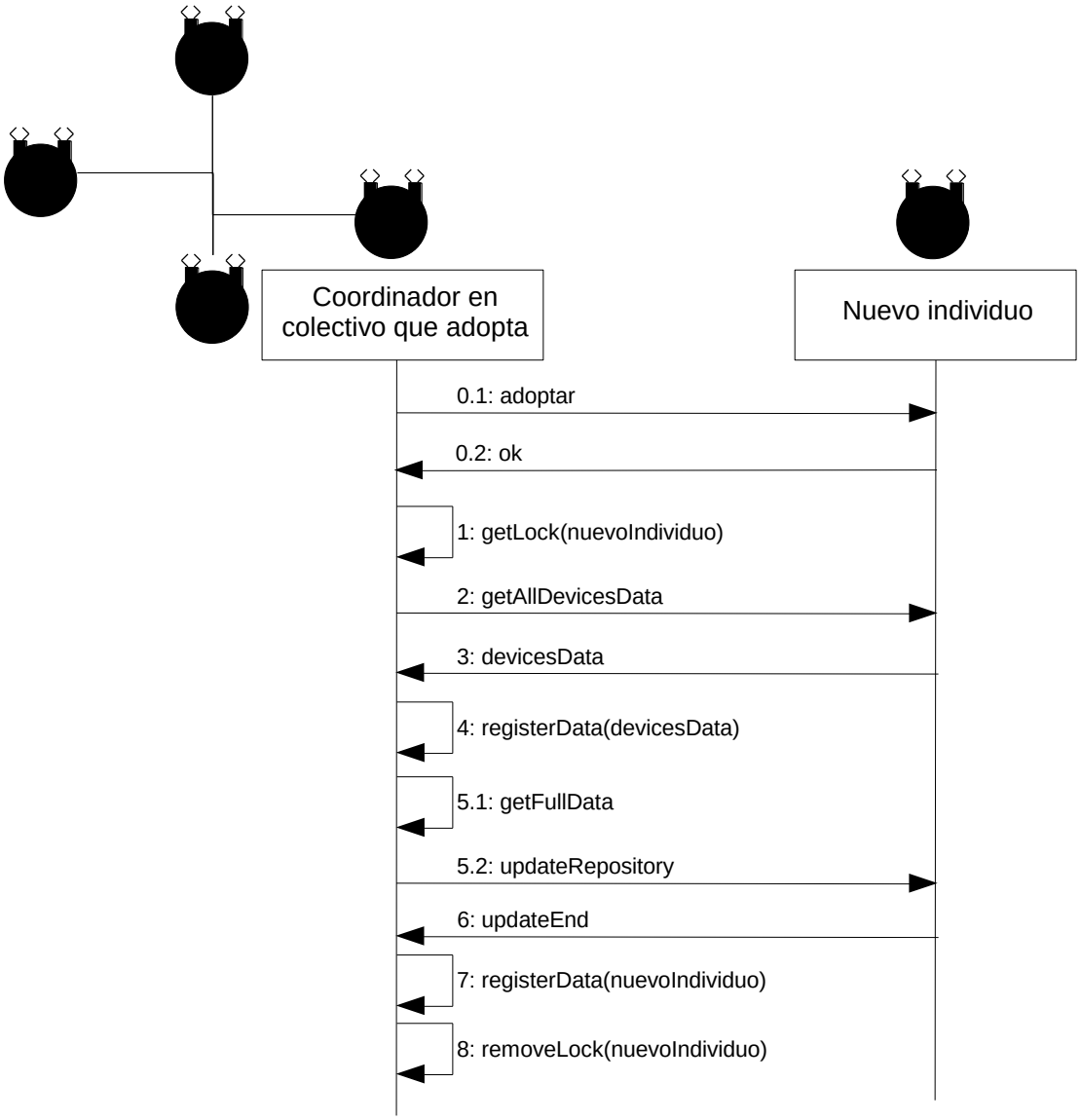


Ilustración 126: Procedimiento de adopción iniciado por el colectivo

Una vez que el procedimiento de adopción termina, el individuo forma parte del colectivo igual modo que el resto de individuos del mismo, siendo la única posible diferencia que todos sus Slots carecen de procesos asociados.

6.2.3.4 Autodescubrimiento

La arquitectura está diseñada para disponer inicialmente de una capa HAL y un autoconocimiento de los dispositivos existentes. Sin embargo, no en todas las situaciones se cumplirá esto, siendo el ejemplo más destacable la asimilación de un individuo.

Para este caso es necesaria la existencia de un conjunto de elementos en todas las capas, siendo los más relevantes:

- Una capa HAL adaptada a la arquitectura, plataforma y sistema operativo del individuo (esto supone disponer de una biblioteca de estas capas HAL, no solo un sistema de compilación cruzada). Dicha capa debe ser capaz de identificar los dispositivos disponibles (por ejemplo reconociendo los identificadores de vendedor y dispositivo de los distintos buses del sistema).
- Una estrategia en la implementación del Coordinador que sea capaz de estimar la capacidad computacional del Individuo para establecer el número de Slots de los que dispondrá (por ejemplo evaluando cuántos hilos paralelos puede ejecutar y el tiempo de ciertas operaciones).
- Posiblemente varios Procesos que activen actuadores y reciban los datos de los sensores distribuidos en varios individuos para poder identificar las capacidades de los dispositivos (por ejemplo, la orientación de un sensor o el movimiento que produce en actuador).

El proceso de autodescubrimiento no tiene por qué ejecutarse en un momento concreto, podría ser continuo.

6.2.3.5 Asimilación de un individuo

Este proceso difiere en sus objetivos del anterior en que no requiere que el individuo a incluir en el colectivo posea una instancia de la arquitectura en ejecución o colabore motu proprio. Es decir, el proceso de asimilación representa la imposición del colectivo sobre el nuevo individuo, esto hace que el proceso esté muy relacionado con el de los virus de ordenador y las técnicas de hacking.

Para poder ejecutar un determinado programa o conjunto de programas, como es la arquitectura, el sistema que desea asimilar al nuevo individuo, necesita ejecutar código (compatible con la arquitectura de la plataforma) que inicie la ejecución de la arquitectura sobre la nueva plataforma con permisos suficientes y luego inicie el proceso de adopción tal como se ha indicado en la descripción del mismo. Como ya hemos dicho, la forma de conseguir dicha ejecución es la de emplear técnicas de hacking de forma automatizada desde el colectivo que desea asimilar el nuevo individuo, siendo el esquema clásico adaptado a estos fines:

1. Escanear para obtener listado de posibles vulnerabilidades a las que el individuo está expuesto. Esto requiere acceso a una base de datos de vulnerabilidades.

2. Ejecutar secuencialmente los exploits para acceder a la plataforma. También requiere acceso a una base de datos con los exploits y posiblemente a un compilador (para compilación cruzada) de la arquitectura atacada.
3. Escalar los privilegios para disponer de los necesarios para los fines de la arquitectura.
4. Inyectar el programa mínimo de la arquitectura.
5. Iniciar la arquitectura.

Todos estos pasos requieren de una infraestructura que proporcione información (vulnerabilidades, exploits…), herramientas de compilación, librerías de escaneo de puertos, etc. Además requerirá que estas habilidades estén disponibles para el Coordinador usando el Subsistema de Dispositivos (que se apoyará en la capa HAL) para los procesos que requieran comunicaciones y el Subsistema de Procesos que debe incluir funcionalidades de compilación cruzada.

Cuando la arquitectura inicie su ejecución deberá iniciar un procedimiento para detectar las capacidades y dispositivos que posee el individuo tal como se describe anteriormente.

6.2.3.6 Expulsión de un individuo

La expulsión de un individuo del colectivo supone el corte de las relaciones con el mismo (lo que posiblemente resulte en una pérdida de información o capacidades) y, si es necesario, la ejecución de los procesos del individuo expulsado en otro.

A pesar del nombre, la expulsión no tiene por qué estar iniciada por el colectivo, pudiendo iniciarse a solicitud del individuo a ser expulsado (ante la detección de un fallo, por ejemplo).

El proceso de expulsión en sí es sencillo, y solo se requiere actualizar el repositorio asignando en el mismo al colectivo “Excluido” el individuo en cuestión y borrando el enlace con el colectivo. El conocimiento se expandirá según la estrategia definida, lo que puede incluir informar al individuo expulsado una última vez.

Tanto el Coordinador como un proceso dentro del sistema pueden solicitar el inicio de este proceso.

6.2.3.7 Fusión con otro colectivo

Este proceso podría entenderse como una adopción de varios individuos de forma ”simultánea” pero difiere de este en que el conocimiento de ambos colectivos sobrevive completamente a la fusión.

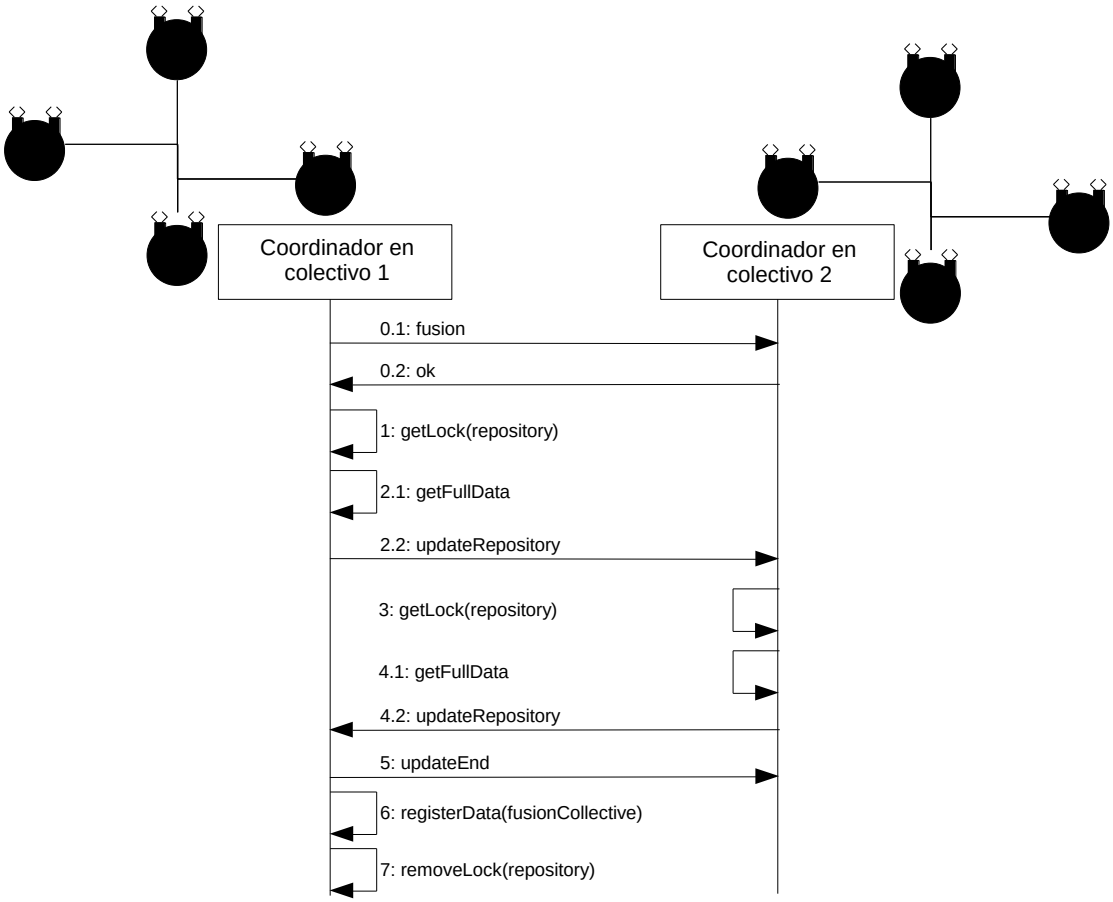


Ilustración 127: Proceso de fusión de dos colectivos

El proceso de adopción, lo inicie quien lo inicie, posee los mismos pasos:

0. Uno de los colectivos que ha detectado al otro le envía un mensaje para iniciar la fusión al que el otro responde aceptando.
1. El Coordinador que se ejecuta en el individuo que origina la acción establece un Lock sobre el repositorio.
2. El mismo Coordinador recupera una copia completa de su repositorio al nuevo colectivo.
3. El Coordinador que se ejecuta en el individuo que recibió la solicitud establece un Lock sobre el repositorio.
4. El mismo Coordinador recupera una copia completa de su repositorio al colectivo original.

5. El Coordinador que se ejecuta en el individuo que origina la acción reporta el fin de la acción, lo que provoca la asignación de todos los individuos al mismo colectivo en ambos colectivos originales.
6. El Coordinador que se ejecuta en el individuo que origina la acción registra los datos necesarios en el repositorio, lo que se expande por todos individuos del nuevo colectivo.
7. Se elimina el Lock sobre el repositorio (que ya es uno solo).

El proceso de fusión puede iniciarlo el propio Coordinador o un proceso en ejecución en el Colectivo.

6.2.3.8 División del colectivo

El proceso de división del colectivo requiere que se definan previamente los individuos y procesos en ejecución que quedarán en cada una de las dos partes, existiendo la posibilidad de que se lancen los procesos que no estén en ejecución en los dos colectivos resultantes.

Una vez que una de las Funciones Superiores decide dividir el colectivo (se supone que con el objetivo de mejorar la eficiencia o eficacia en la consecución de un objetivo), el Coordinador que recibe la solicitud original emprenderá el proceso consistente en los siguientes pasos:

0. Un Proceso de la capa de Funciones Superiores envía la solicitud de división del colectivo. Esta solicitud incluirá la forma de seleccionar los individuos de uno y otro grupo y la de seleccionar los procesos que se ejecutarán en los grupos resultantes.
1. El Coordinador que se ejecuta en el individuo que origina la acción establece un Lock sobre el repositorio.
2. El mismo Coordinador actualiza el repositorio (de forma local) creando un nuevo colectivo y asignando al mismo los individuos según la estrategia establecida.
3. Envía instrucciones a los distintos Coordinadores de los Individuos para iniciar o parar los procesos que correspondan, teniendo en cuenta la estrategia de asignación de Slots y la existencia de dos colectivos.
4. Se actualiza el repositorio de todos los individuos con la existencia de los dos colectivos, que denominaremos “colectivo_1” y “colectivo_2”,

y la asignación de individuos a los colectivos. El individuo mencionado en los pasos anteriores pertenece a "colectivo_1", por lo que necesitaremos un miembro de "colectivo_2" para eliminar el Lock de este otro colectivo. Para ello, se inicia un proceso en un individuo concreto de "colectivo_2" que ejecutará la acción que elimine el Lock del repositorio.

5. El Coordinador del individuo indicado en el paso anterior elimina el Lock sobre el repositorio del colectivo donde reside ("colectivo_2").
6. Se elimina el Lock sobre el repositorio del colectivo donde reside el individuo que origina la acción ("colectivo_1").

6.2.3.9 Detección de individuos defectuosos

La detección de individuos con disfunciones en alguno de sus sistemas requiere una aproximación distribuida en todas las capas y componentes del sistema. Cada uno de dichos componentes deberá disponer de funciones internas que permitan detectar estas disfunciones. Una vez que una disfunción es detectada, el Coordinador debe ser informado, pudiendo disparar acciones o procesos específicos para la mitigación del fallo, amén de incluir en el repositorio esta información.

Además de la detección dentro del propio individuo (autochequeo/autodiagnóstico), el sistema puede detectar disfunciones desde otros individuos, por ejemplo: errores en las tramas de datos de alguno de los sistemas de comunicación o movimientos que no se corresponden con las instrucciones sobre los actuadores. Los procesos (o cualquier otra parte del sistema) pueden notificar al Coordinador la existencia de la disfunción indicando el individuo, posiblemente el dispositivo y alcance del fallo (si es conocido).

La implementación concreta del Coordinador puede disponer de una estrategia para la resolución de fallos dentro de un individuo y/o dentro del colectivo. Dicha estrategia tiene varias opciones en la arquitectura para reaccionar (aparte de las que le puedan proveer las implementaciones específicas de la capa de Recursos y los Subsistemas de Dispositivos y de Procesos):

- Eliminar el actuador, sensor o dispositivo de comunicaciones.
- Reducir el número de Slots disponibles.
- Expulsar al individuo defectuoso.

Uno de los objetivos de aplicación de la arquitectura son los colectivos de robots autónomos. En este tipo de robots una de las principales razones para iniciar esta clase de comportamientos es que se alcance un determinado nivel de batería y, aparte de que las distintas capas inicien procedimientos para reducir el consumo de energía o las Funciones Superiores envíen al individuo a una estación de recarga, se puede emplear el mecanismo de fallo para indicar al colectivo un estado que impacta el nivel funcional del individuo.

6.2.3.9.1 Autodiagnóstico

Por su parte las implementaciones concretas de la capa de Recursos y de los Subsistemas de Procesos y Dispositivos pueden implementar (dependiendo del objetivo del individuo y sistema concretos) funcionalidades para la detección de errores internos, por ejemplo la detección de errores en tramas de comunicaciones, el accionamiento de un actuador que no llega a su posición en el tiempo esperado, etc. Puede incluso que estos componentes provean funciones de autodiagnóstico hardware y software que el Coordinador pueda ejecutar cuando se detecta un error o cada cierto tiempo.

De la misma forma las Funciones Superiores pueden incluir procesos cuyo objetivo sea el de evaluar el estado (presencia y alcance de disfunciones) de un individuo mediante la ejecución de acciones y la comprobación de resultados en el mismo. Estos procesos podrían ser lanzados como respuesta a la detección de un fallo por otros componentes y pueden requerir varios individuos.

6.2.4 Protocolo de mensajes

Usamos la terminología de [35] que usa el término *protocolo de mensajes* para referirse a la definición de la sintaxis de la información intercambiada.

Como se ha descrito anteriormente, los distintos individuos pueden hacer uso de comunicaciones explícitas para cumplir con su objetivo y, además, los propios procesos de funcionamiento del colectivo precisan del intercambio de información. Esto requiere de un protocolo pero, dado que esta arquitectura pretende proveer de la posibilidad de implementación de cualquier sistema multi-robot, solo diseñaremos las guías básicas que un protocolo completo deberá

cumplir, siendo responsabilidad de las implementaciones finales el diseño concreto del mismo.

Este (o estos) protocolo (s) se enmarcarán normalmente en la capa superior del modelo OSI (Aplicación).

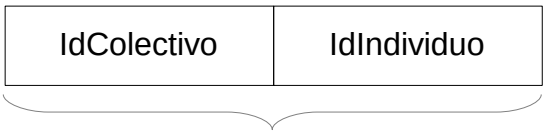
6.2.4.1 Estructura de un mensaje

La estructura del mensaje define la cantidad mínima de información que debe enviarse y la organización interna para que el mensaje sea válido.

Campo	**Descripción**
Origen	Identificador del individuo que envía el mensaje
Destino	Identificador del individuo que debe recibir el mensaje
Control	Identificador de secuencia y prioridad
Tipo	Tipo del mensaje
Mensaje	Contenido del mensaje

Tabla 14: Campos mínimos de un mensaje

Los campos Origen y Destino de un mensaje representan identificadores del Individuo, Individuos o Colectivo que ha enviado o debe recibir el mensaje. Estos identificadores se conforman incluyendo el identificador del colectivo y el identificador del individuo de manera que la ausencia del identificador del individuo representa que el destino es todo un colectivo.



Identificador de Origen o Destino

Ilustración 128: Composición del identificador

La necesidad de este tipo de identificadores viene dada por la existencia de múltiples sistemas de comunicaciones cada uno de los cuales puede tener (o no) sistemas de direccionamiento distintos. Pudiendo generar duplicidad en los mensajes y/o la imposibilidad de enviarlos (la interconexión entre individuos no tiene por qué permitir la comunicación directa entre ambos).

El campo de control incluye el número de secuencia del mensaje (por si este se retransmite, etc.) y la prioridad del mismo si el protocolo de mensajes incluye la capacidad de priorización en la gestión de las colas de mensajes.

El tipo de mensaje representa la función solicitada de entre las que ambos interlocutores entienden.

La última parte del mensaje incluye la parte específica del tipo. Describiremos en detalle el contenido de cada uno de estos tipos de mensaje.

Seq	**Descripción**
1	Ejecutar un proceso
2	Finalizar la ejecución de un proceso
3	Actualizar datos en el repositorio
4	Eliminar datos en el repositorio
5	Registrarse en los cambios de un dispositivo / conocimiento
6	Eliminar la actualización automática de datos
7	Ejecutar una acción sobre un actuador
8	Iniciar el proceso de adopción sobre un individuo
9	Solicitar la fusión con otro colectivo
10	Aceptar o rechazar la fusión con un colectivo o una adopción
11	Solicitar información de un dispositivo o del repositorio
12	Establecer una sincronización
13	Eliminar una sincronización
14	Informar de un fallo
15	Modificar los slots disponibles
16	Ack

Tabla 15: Tipos de mensajes

No hay un tamaño máximo o mínimo de mensaje por lo que capas inferiores de los dispositivos de comunicaciones deberán encargarse del fraccionamiento y relleno según corresponda en el lado del emisor y de la operación inversa en el lado del receptor.

6.2.4.2 Mensaje para ejecutar un proceso

Este mensaje instruye al destinatario (el Coordinador residente en un Individuo) para que ejecute un determinado Proceso en un determinado Slot de aquellos que pertenecen al Individuo del receptor.

El identificador de la clase de Proceso referencia a una entrada del repositorio que contendrá el código a ejecutar. Si el repositorio de este Individuo no contiene esa entrada (depende de la estrategia de distribución), deberá enviar una solicitud para recuperarla.

Campo	**Descripción**
Proceso	Identificador de la clase de Proceso
Slot	Identificador del Slot
ArgV	Secuencia de valores de los parámetros

Tabla 16: Campos mínimos para el mensaje de ejecución de un proceso

El identificador del Slot asigna ya el proceso a un Slot dentro de los disponibles en el Individuo.

Los parámetros que requiera el proceso (y que no sean parte del repositorio) pueden enviarse incluyendo el mensaje sus valores de forma secuencial.

6.2.4.3 Mensaje para finalizar la ejecución de un proceso

Cuando un Coordinador en otro Individuo decide finalizar un proceso en ejecución en el receptor envía este mensaje.

Campo	**Descripción**
Slot	Identificador del Slot
Proceso	Identificador de la clase de Proceso

Tabla 17: Campos mínimos para el mensaje de finalización de un proceso

Para finalizar el proceso se debe enviar, como mínimo, el identificador del Slot que está ejecutando dicho proceso. Además, se puede enviar el identificador del proceso para confirmar que es el que está en ejecución.

6.2.4.4 Mensaje para actualizar datos en el repositorio

La actualización remota del repositorio representa el envío de un número de identificadores de conocimiento y sus valores, lo que permite el envío del repositorio completo en un solo mensaje.

Campo	Descripción
Id	Identificador del conocimiento
Longitud	Longitud del contenido
Contenido	Contenido del conocimiento

Tabla 18: Campos mínimos para el mensaje de actualización del repositorio

Un mensaje de este tipo es una secuencia de una o varias ternas con el identificador, longitud y contenido, tal como se ven en la tabla anterior. De esta forma es posible enviar un número ilimitado de elementos de conocimiento que se pueden leer de forma secuencial.

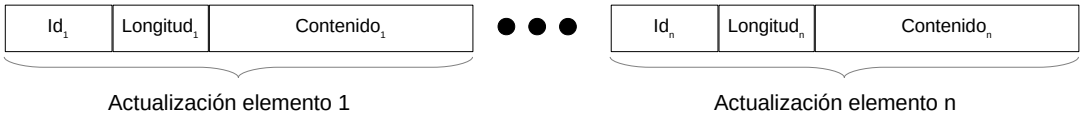


Ilustración 129: Secuencia de valores en un mensaje de actualización del repositorio

El identificador representa un valor único para todo el Colectivo de un determinado conocimiento (por ejemplo, un proceso o la posición de un robot). Estos identificadores son un esquema general que incluye el contenido del repositorio, los valores de los sensores, etc.

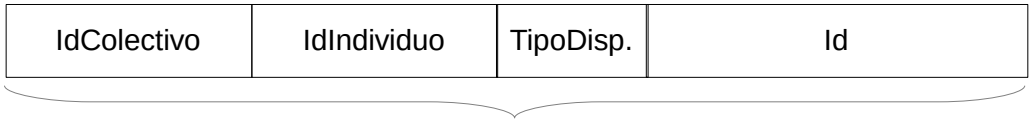


Ilustración 130: Composición de un identificador completo

Seq	Tipo
1	Elemento repositorio
2	Sensor
3	Actuador
4	Comunicaciones

Tabla 19: Tipos de elementos para los identificadores

La longitud representa el número de bytes de la carga útil tras las transformaciones necesarias para su envío y es seguido del contenido en sí.

Entre la información que se puede actualizar en el repositorio están los individuos detectados.

6.2.4.5 Mensaje para eliminar datos en el repositorio

Este mensaje permite eliminar un determinado conocimiento cuando este deja de ser útil.

Campo	**Descripción**
Id	Identificador del conocimiento

Tabla 20: Campos mínimos para el mensaje de eliminación de un elemento del repositorio

Al igual que el anterior, un mensaje de este tipo es una secuencia de uno o varios identificadores tal como se ven en la tabla anterior. De esta forma es posible enviar un número ilimitado de órdenes de borrado de elementos de conocimiento que se pueden leer de forma secuencial.

6.2.4.6 Mensaje para registrarse en los cambios de un dispositivo o conocimiento

Es posible registrarse para recibir los datos de un determinado dispositivo de manera que no sea necesario ejecutar lecturas sobre él de forma continua.

Campo	**Descripción**
Id	Identificador del elemento
Intervalo	Tiempo máximo y mínimo de actualización

Tabla 21: Campos mínimos para un registro de cambio de información de un dispositivo o conocimiento

Además del identificador del elemento (sigue el mismo esquema definido para los mensajes de recuperación y actualización de datos) se requiere el intervalo (tiempo máximo y mínimo) de actualización (el tiempo que pasa entre el envío de los datos actualizados). Si el valor es cero solo se enviarán datos cuando cambien el estado (por ejemplo, cuando un detector booleano pase de valor falso a cierto).

Puede que la implementación de estos mensajes incluya otros parámetros como:

- Nivel de cambio: Diferencia en los valores del estado que generarán una notificación de actualización.

- Identificadores de Slot/Proceso: Identificadores del Slot y Proceso que generan el registro (el de Colectivo e Individuo ya están en la cabecera de todos los mensajes).
- Timeout de las notificaciones: Tiempo máximo durante el cual se mantendrá el registro de actualizaciones.
- Identificador de la solicitud: Si se permiten varios registros de notificaciones (esto depende de la implementación del Coordinador) se necesitará identificar la solicitud concreta para poder eliminarla si es necesario.

6.2.4.7 Mensaje para eliminar la actualización automática de datos

Este mensaje elimina el registro de notificaciones creado por el mensaje anterior.

Campo	**Descripción**
Id	Identificador del elemento

Tabla 22: Campos mínimos para la eliminación de un registro para la actualización de datos

Si se permiten varios registros de notificaciones (esto depende de la implementación del Coordinador) se necesitará un identificador de la solicitud para poder eliminar la que corresponda.

6.2.4.8 Mensaje para ejecutar una acción sobre un actuador

Este mensaje permite solicitar la ejecución de una acción sobre un actuador, por ejemplo, iniciar el movimiento de un motor o encender un LED.

Campo	**Descripción**
Id	Identificador del elemento
Parámetros	Secuencia de parámetros

Tabla 23: Campos mínimos para la aplicación de una acción sobre un actuador

El identificador del elemento representa un actuador único en el Colectivo.

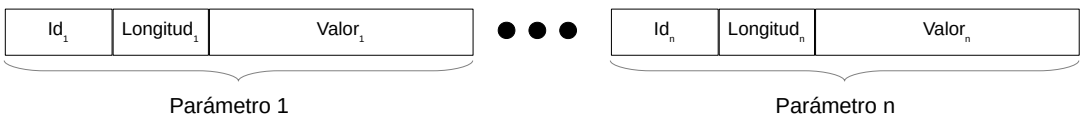


Ilustración 131: Secuencia de parámetros

Los parámetros son una secuencia de ternas incluyendo un identificador del parámetro, su longitud y su valor. Estos identificadores son solo relevantes para el tipo de actuador concreto. Los valores pueden incluir múltiples parámetros para modificar la acción del actuador, así como tiempos, etc.

6.2.4.9 Mensaje para iniciar el proceso de adopción sobre un individuo

Este mensaje permite solicitar el inicio de la adopción sobre un individuo desde un colectivo.

La única información necesaria para el inicio de esta operación es el identificador del colectivo que ya está presente en la cabecera de cualquier mensaje. Sin embargo, para permitir la respuesta a este mensaje se puede añadir un identificador de la solicitud.

6.2.4.10 Mensaje para solicitar la fusión con otro colectivo

La fusión de colectivos requiere que uno de los dos le envíe al otro un mensaje solicitándolo.

La única información necesaria para el inicio de esta operación es el identificador de los colectivos a fusionar que ya está presente en la cabecera de cualquier mensaje. Pero para permitir la respuesta a este mensaje se puede añadir un identificador de la solicitud.

6.2.4.11 Mensaje para aceptar o rechazar la fusión con un colectivo o una adopción

Este mensaje se emite en respuesta a los dos anteriores para confirmar de forma explícita el deseo de fusión o adopción desde el receptor del mensaje original.

Campo	**Descripción**
Tipo	Tipo del mensaje original

Tabla 24: Campos mínimos para el mensaje de confirmación de una fusión o adopción

Si el mensaje original incluye un identificador de la solicitud, este debe añadirse también a los campos del mensaje.

El campo tipo representa el identificador (igual al usado por los tipos de mensajes) del mensaje que origina la respuesta (fusión o adopción).

6.2.4.12 Mensaje para solicitar información de un dispositivo o del repositorio

De la misma forma que es posible registrarse para recibir notificaciones de actualización de la información referida a un dispositivo o del repositorio, es posible solicitar la información actual.

Campo	**Descripción**
Id	Identificador del elemento

Tabla 25: Campos mínimos para el mensaje de solicitud del último estado de un dispositivo o de un elemento del repositorio

Esta solicitud solo requiere el identificador, el dispositivo o elemento del repositorio que se reclama. El esquema de identificadores es el anteriormente expuesto.

6.2.4.13 Mensaje para establecer una sincronización

Es posible establecer un proceso de sincronización entre varios procesos (serialización, ejecución simultánea...) que permita mejorar la coordinación en las acciones conjuntas del Colectivo. Para ello este mensaje debe enviarse a todos los Individuos con Procesos o Dispositivos implicados en la acción o conjunto de acciones que se desea o desean acometer.

Campo	**Descripción**
IdLock	Identificador del proceso de sincronización (no confundir con un Proceso de Funciones Superiores)
Tipo	Forma de la sincronización
Prioridad	Nivel de prioridad de la operación
Id(s)	Identificador o identificadores implicados

Tabla 26: Campos mínimos para la emisión de un mensaje que establezca una sincronización

El identificador del proceso de sincronización es un identificador que sigue el mismo esquema que el descrito para los dispositivos y elementos del repositorio con un nuevo tipo (lock).

El tipo de sincronización define la forma en la que se llevará a cabo la misma.

Seq	Tipo
1	Bloqueo: hasta que sea liberado, ningún Proceso puede usar estos elementos.
2	Simultaneidad: todas las solicitudes sobre este elemento se ejecutarán de forma simultánea.
3	No excluyente: no excluye que otro lo use
4	Excluir modificación: no permite que otro individuo modifique este elemento mientras exista este bloqueo

Tabla 27: Tipos de acciones de sincronización (pueden extenderse)

La prioridad de la operación permite que Procesos con mayor valor añadido puedan llegar a eliminar los bloqueos generados por otros de supuesto menor valor.

Los identificadores de los elementos son una secuencia de identificadores como los definidos previamente.

6.2.4.14 Mensaje para eliminar una sincronización

Un proceso de sincronización (un bloqueo) puede eliminarse en cualquier momento mediante este mensaje.

Campo	Descripción
IdLock	Identificador del proceso de sincronización (no confundir con un Proceso de Funciones Superiores)

Tabla 28: Campos mínimos para la eliminación de un proceso de sincronización

El único dato imprescindible es el identificador del mencionado proceso de sincronización. Es posible que se incluya un campo más con la estrategia para cambiar el estado del proceso o que esta modificación se haga con otro mensaje (por ejemplo, solicitando su fin).

6.2.4.15 Mensaje para informar de un fallo

Cuando se detecta un fallo puede involucrar a varios elementos del sistema, incluso distribuidos en varios individuos. Este mensaje permite informar de forma explícita de tales problemas; la semántica es diferente a la actualización del

repositorio ya que este mensaje puede requerir acciones por parte del Coordinador.

Campo	**Descripción**
Id	Identificador del elemento que ha provocado el fallo
FaultLevel	Nivel de funcionalidad afectado

Tabla 29: Campos mínimos para un registro dentro de un mensaje de fallo

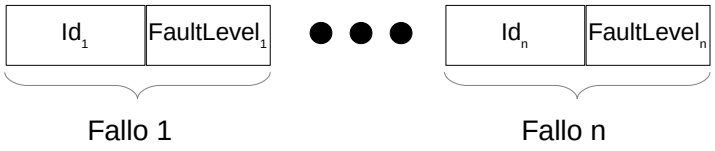


Ilustración 132: Lista de fallos en un mensaje de reporte de fallo

El campo Id es un identificador de un dispositivo, elemento de conocimiento (un fallo en un elemento de conocimiento puede representar que se haya perdido la información que contenía), individuo o colectivo.

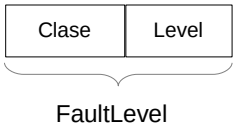


Ilustración 133: Composición del FaultLevel

El campo de nivel de fallo define el nivel de funcionalidad del elemento que ha sido afectada. El significado concreto de estos niveles es dependiente del elemento afectado, pero se organizan en clases para poder estimar su severidad sin necesidad de poseer ese nivel de conocimiento.

Seq	**Tipo**
1	Leve: La funcionalidad del elemento apenas se ve afectada (por ejemplo, es ligeramente más lento)
2	Medio
3	Alto
4	Crítico: El elemento ha dejado de estar disponible

Tabla 30: Clases de niveles de fallo

6.2.4.16 Mensaje para modificar los slots disponibles

La arquitectura permite modificar el número de Slots para procesos desde otro Individuo, lo que representa el envío de este mensaje.

Campo	**Descripción**
Slots	Número de Slots

Tabla 31: Campos mínimos para el mensaje de modificación del número de Slots disponibles

Además de los campos mínimos, es posible que sea necesario enviar la estrategia a seguir para los Procesos en ejecución dentro de los Slots que habrían de suprimirse. Dicha estrategia tendrá dos vertientes:

- Priorización de los procesos: o, lo que es lo mismo, selección de los Procesos que continuarían ejecutándose.
- Acciones sobre los procesos que quedan en los Slots que desaparecerán. Entre otras, estas podrían ser:
    - Dejar que los procesos acaben sin lanzar nuevos procesos en esos Slots.
    - Pausar los procesos hasta que haya un Slot libre en el mismo Individuo y reasignarlo a dicho Slot.
    - Parar los procesos y volver a lanzarlos siguiendo la estrategia de asignación como si el proceso se lanzase por primera vez.

6.2.4.17 Ack

Un mensaje de Ack representa una respuesta a la recepción de un mensaje. No es imprescindible para el funcionamiento del sistema (este protocolo descansa sobre capas que deben garantizar la recepción y acuse de recibo) pero puede añadir una comprobación extra extremo a extremo o un reporte de errores en el procesado del mensaje.

Campo	**Descripción**
Seq	Secuencia a la que se responde
Resultado	Respuesta de ok/error al procesado del mensaje original

Tabla 32: Campos mínimos para un mensaje de Ack

El campo de secuencia contiene la secuencia con la que va marcado el mensaje al que se responde.

El campo de resultado incluye si el fin último del mensaje ha tenido éxito o, por el contrario, no se ha podido realizar. Este comportamiento puede ser extendido incluyendo códigos de error que aporten información sobre la naturaleza y causa del fallo y/o sobre las condiciones resultantes en caso de que se haya conseguido el objetivo solicitado en el mensaje original.

6.2.5 Librería de componentes

Como hemos visto en la descripción de varios de los componentes, el propio diseño de la arquitectura implica disponer de varias implementaciones de cada uno de dichos componentes. Incluso para alguno de los procesos puede ser necesario el disponer de una librería que nos permita tener acceso a implementaciones de los componentes en tiempo de ejecución (por ejemplo durante el proceso de asimilación de un individuo).

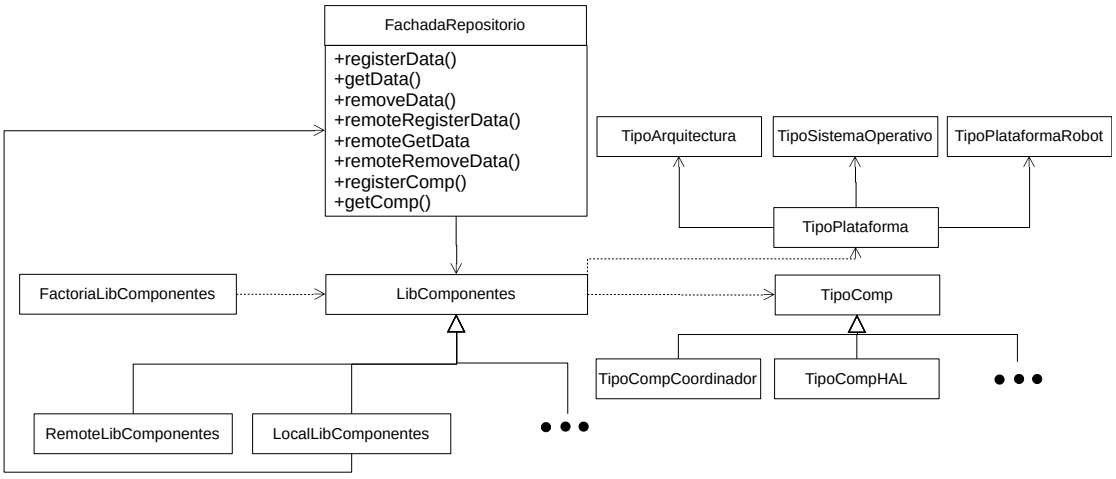


Ilustración 134: Extensión del Subsistema de Repositorio para permitir el procesado de componentes con distintas estrategias

Esta librería podría estar integrada en el repositorio del Colectivo como elementos de conocimiento del mismo (al igual que, por ejemplo, los procesos) pero también podría ser un servicio externo al que una implementación del Coordinador solicitase versiones concretas de los componentes.

Capítulo 7

Evaluación de la arquitectura propuesta

Aplicaremos el mismo sistema de evaluación (ver Capítulo 4 para una descripción detallada del mismo) con el que evaluamos las arquitecturas del estado del arte (ver Capítulo 5 para ver los resultados obtenidos con dichas arquitecturas) sobre esta nueva propuesta. Los resultados de la evaluación se compararán con los obtenidos en la evaluación de las arquitecturas identificadas aplicando, en el caso de la simulación del problema, un análisis estadístico exhaustivo para demostrar la mejora proporcionada por la nueva arquitectura.

A pesar de que la arquitectura proporciona múltiples opciones, todas las métricas se aplican sobre la misma implementación y configuración. El uso de diferentes parámetros de configuración según la métrica mejoraría los valores de las métricas, pero proporcionaría resultados no consistentes entre sí. La evaluación comparativa de los cambios en la configuración está fuera del alcance de este trabajo.

7.1 Aplicación de las métricas de evaluación para las características

Las características de la arquitectura son: adaptación (capacidad de implementar sistemas con diferentes características), coordinación (característica necesaria para que varios individuos ejecuten una acción que sería imposible de realizar por separado), comunicación (intercambio de información entre los individuos), autoconocimiento (capacidad de conocer las capacidades de los individuos del colectivo), colectivos (habilidades de adopción, expulsión, etc.) e interoperabilidad (capacidad de relacionarse o integrar diferentes individuos o sistemas).

	Valor métrica
Adaptación	1
Coordinación	0,905
Comunicación	0,701
Autoconocimiento	1
Colectivos	1,005
Interoperabilidad	0,874

Tabla 33: Resultados de las métricas para las características de la arquitectura (ver descripción de las métricas en "4.2.2 Métricas de la evaluación para las arquitecturas", Capítulo 4)

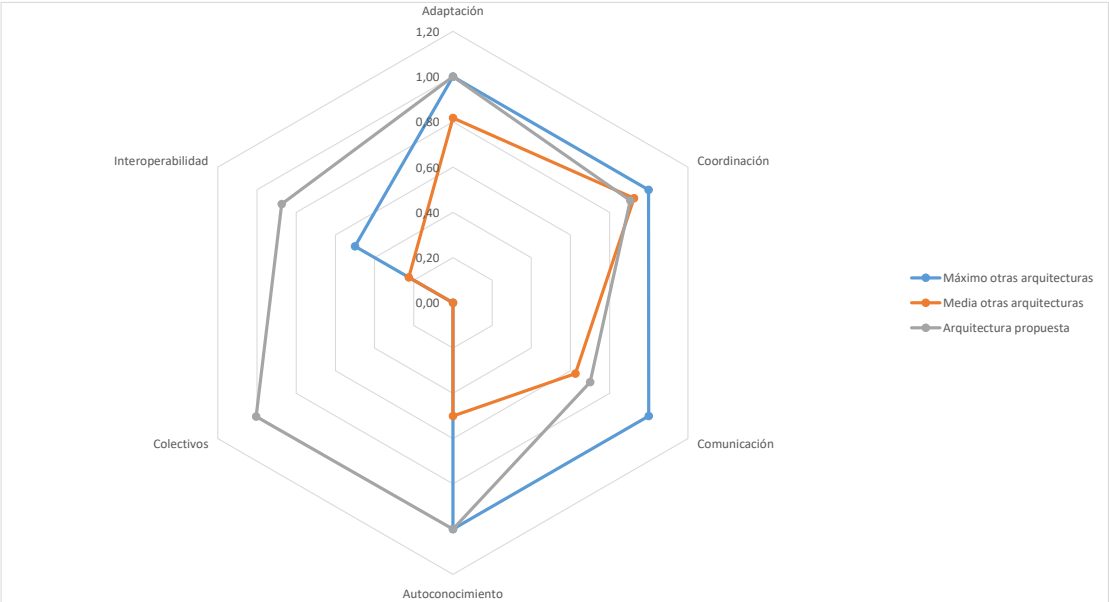


Ilustración 135: Resultados de la evaluación de las características para la arquitectura propuesta

En la Ilustración 135 podemos ver que los resultados de la evaluación de las características de la arquitectura propuesta son, en el peor de los casos, similares a la media del resto de arquitecturas. Al igual que con la comparativa de los resultados de las arquitecturas identificadas, la causa de que los valores de la métrica de Coordinación no mejore con respecto al valor de estas, es producto de un protocolo de coordinación más complejo (como resultado del incremento en sus habilidades).

También se puede ver que destacan por encima de los resultados de las otras arquitecturas los valores de las métricas de Interoperabilidad y Colectivos. En el caso de la métrica de Interoperabilidad, el resultado se ve beneficiado por la posibilidad de disponer de varios protocolos de mensajes en un solo robot (dentro del Coordinador). Y en el resultado de la métrica de Colectivos, este se obtiene gracias a las habilidades a tal efecto incluidas en la arquitectura (no disponibles en las otras arquitecturas evaluadas).

7.2 Aplicación de las métricas de evaluación para las capacidades

En este punto evaluamos los límites de las capacidades de la arquitectura en cuanto a número máximo de individuos que pueden pertenecer a un colectivo, cuánta información son capaces de intercambiar dos individuos por unidad de tiempo y cuántas acciones puede ejecutar el controlador de un individuo sobre los actuadores de otro individuo por unidad de tiempo.

	Valor métrica
Número de individuos	0,12
Intercambio de información simultáneo	20,01
Acciones	240

Tabla 34: Resultados para las métricas de las capacidades

La implementación de la arquitectura propuesta en estas pruebas de capacidad hace un uso intensivo de las comunicaciones. Esto genera una limitación que puede verse muy claramente en el resultado de la métrica para el número de individuos. La arquitectura puede usarse sin comunicaciones eliminando la limitación en el número de individuos o modulando el uso de las comunicaciones para adaptarse a las necesidades del problema. Este uso intensivo de las comunicaciones proporciona una mejora en las métricas que dependen

directamente de ella como son el intercambio de información o el control remoto de actuadores. Como resultado de estas características, esta implementación y configuración se adaptan muy bien a sistemas con control centralizado.

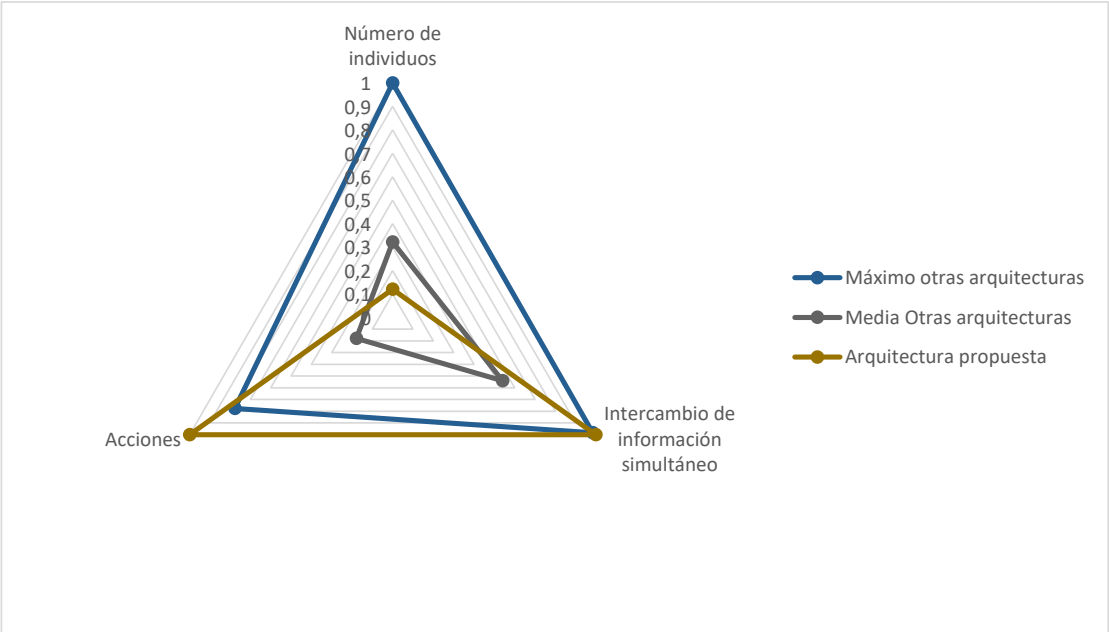


Ilustración 136: Comparativa de las capacidades del mejor resultado de entre las capacidades de las otras arquitecturas y de la arquitectura propuesta

7.3 Simulación del problema

Por último, simularemos un problema real comparando los resultados de tiempo, energía consumida y precisión del resultado. Dada la predecible variabilidad en estas métricas se ejecutan 50 repeticiones.

	Media	Desviación típica
Tiempo	5,5477	0,2507
Energía consumida	5,9075	0,3571
Distancia horizontal (dm)	0,0117	0,0086

Tabla 35: Resultados para las métricas de las simulaciones

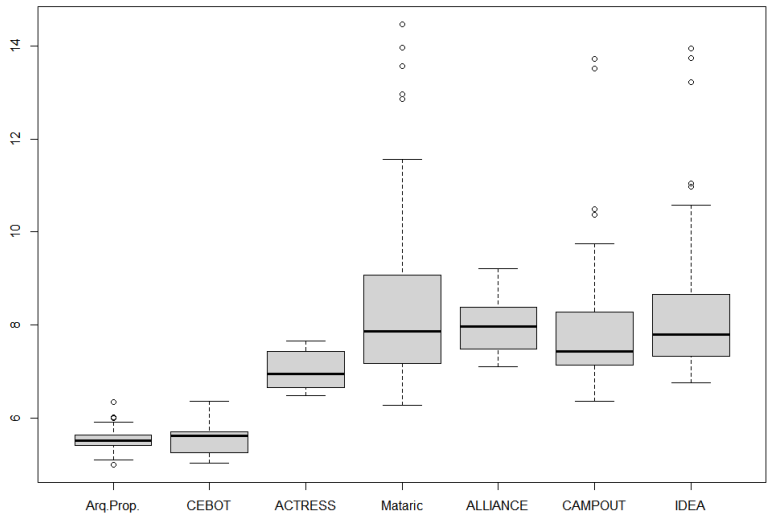


Ilustración 137: Diagrama de cajas y bigotes para la métrica de tiempo comparando la arquitectura propuesta (Arq. Prop.) con el resto de arquitecturas evaluadas. Los resultados obtenidos con la nueva arquitectura son estadísticamente mejores a los obtenidos con la mayoría de arquitecturas.

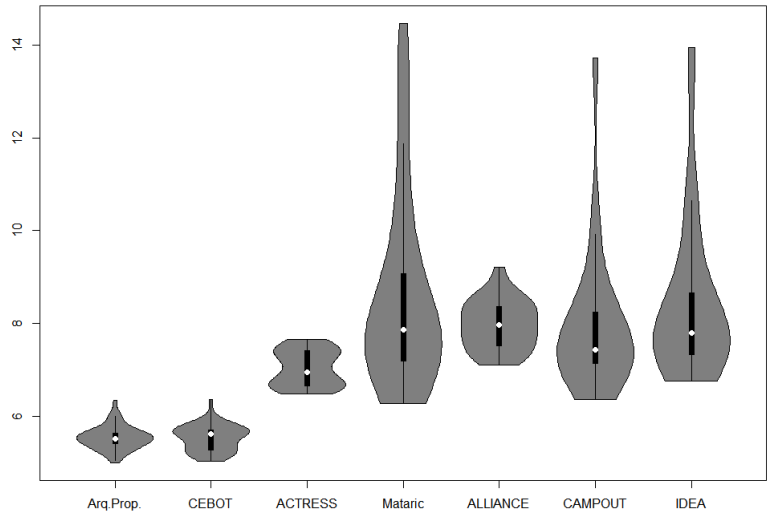


Ilustración 138: Diagrama de violín para la distribución de los resultados de la métrica de tiempo comparando la arquitectura propuesta (Arq. Prop.) con las otras arquitecturas evaluadas

En la Ilustración 137 y la Ilustración 138 podemos ver que los resultados para la métrica de tiempo son mejores para la arquitectura propuesta en comparación con la mayoría de arquitecturas. Los resultados cercanos de CEBOT se deben a que el problema permite una resolución eficiente usando arquitecturas que no

generen una alta sobrecarga derivada de la coordinación y capaces de centralizar la toma de decisiones (en especial, arquitecturas jerárquicas).

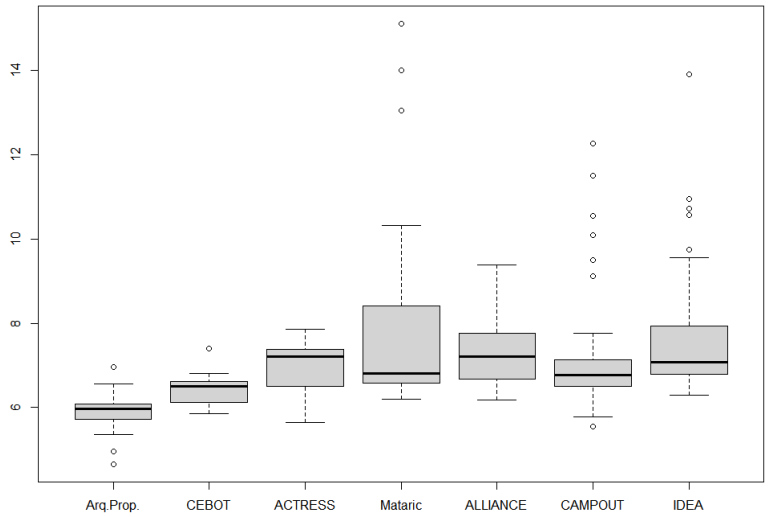


Ilustración 139: Diagrama de cajas y bigotes para la métrica de consumo total de energía comparando la arquitectura propuesta (Arq. Prop.) con el resto de arquitecturas evaluadas. Los resultados obtenidos con la nueva arquitectura son estadísticamente mejores a los obtenidos con el resto de arquitecturas.

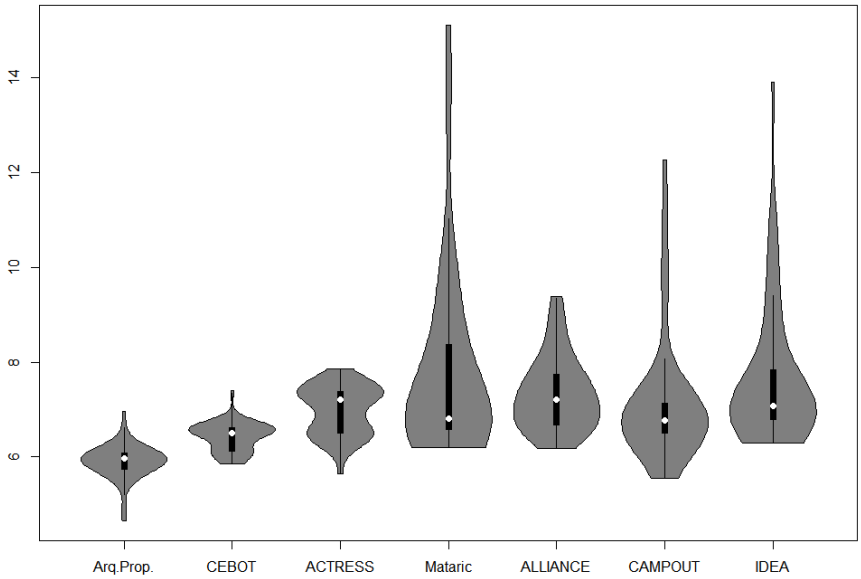


Ilustración 140: Diagrama de violín para la energía total consumida. La arquitectura propuesta (Arq.Prop.) muestra un menor consumo de energía.

Los resultados de la métrica de consumo total de energía son similares a los obtenidos en la métrica de tiempo. En la Ilustración 139 y la Ilustración 140

podemos ver que la arquitectura propuesta mejora los resultados de cualquier otra arquitectura.

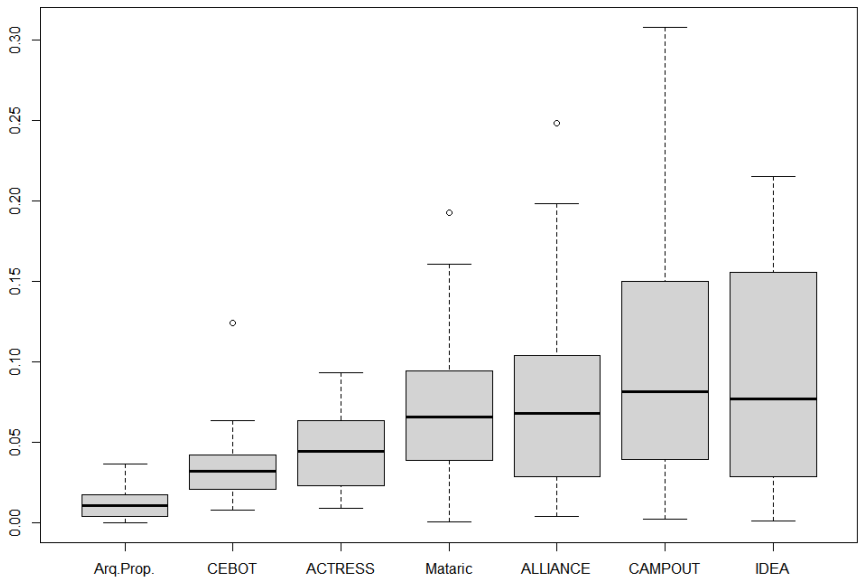


Ilustración 141: Diagrama de cajas y bigotes para la métrica de distancia (precisión). La arquitectura propuesta muestra estadísticamente mejores resultados siendo la única cuya distancia máxima se halla por debajo de 0.04.

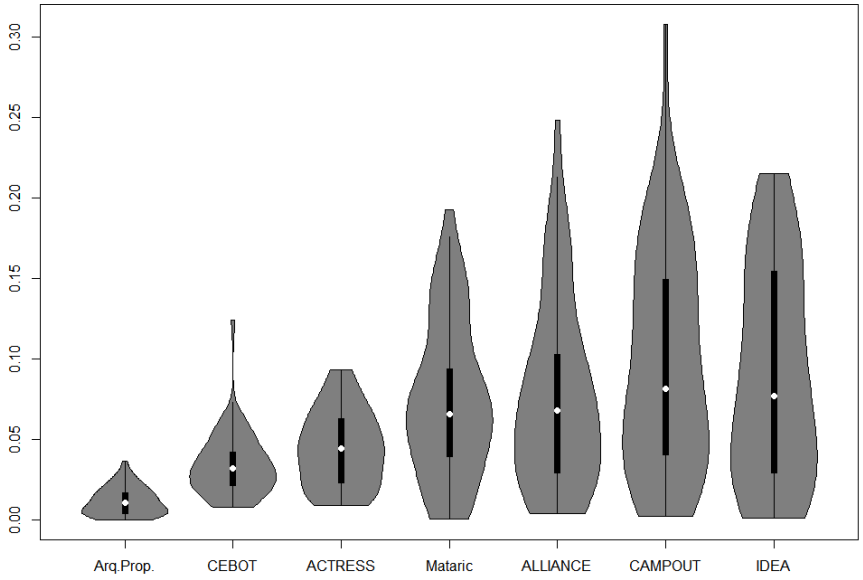


Ilustración 142: Diagrama de violín para la métrica de distancia (precisión). La arquitectura propuesta muestra estadísticamente mejores resultados siendo la única cuya distancia máxima se halla por debajo de 0.04.

Los resultados de la métrica de distancia (ver Ilustración 141 e Ilustración 142) muestran una ventaja para la arquitectura propuesta. Es importante destacar que

es la única cuyos resultados en los experimentos realizados son todos inferiores a 0,04 (4mm). Todos los resultados por encima de esta cifra representan un acoplamiento incorrecto de las piezas.

Según lo descrito en el punto "4.6.2 Interpretación y comparación" del Capítulo 4, aplicaremos un análisis estadístico de los resultados obtenidos en las métricas de la simulación del problema siguiendo las metodologías propuestas en [125] y [128].

7.3.1 Análisis estadístico para la métrica de tiempo

Aplicando el test Shapiro-Wilk, tomando como hipótesis nula (H_0) que los resultados siguen una distribución normal, obtenemos un p-valor=2,22045e-16 y un W= 0,865184. Dado que el p-valor es inferior a 0,05 rechazamos H_0 y asumimos que los datos no siguen una distribución normal. Con este resultado debemos aplicar una prueba no paramétrica para comparar los resultados de las arquitecturas.

Al disponer de 50 repeticiones de la métrica por cada arquitectura, debemos usar el test de Friedman [129] tomando como hipótesis nula la inexistencia de diferencias significativas entre los resultados. Obtenemos un p-valor= 5,047519e-43 (<0,05) que refuta la hipótesis nula de que todos los datos pertenecen a la misma distribución indicando que, al menos, uno de los grupos de datos es significativamente diferente, sin indicar cuál de ellos.

Además realizamos el test por pares para identificar cuáles poseen diferencias significativas. Se realizan dos de los posibles test: (1) Conover y (2) Nemenyi [130]. En el caso del test de (1) Conover, ajustaremos el p-valor con dos procedimientos: family-wide error rate (FWER) de Holm y false discovery rate (FDR) de Benjaminyi-Hochberg [130].

	Arq.Prop.	CEBOT	ACTRESS	Matarić	ALLIANCE	CAMPOUT
CEBOT	6,920269e-01					
ACTRESS	5,033230e-63	2,020497e-64				
Matarić	7,043551e-114	5,857291e-115	2,100218e-40			
ALLIANCE	7,715407e-121	7,086871e-122	6,170775e-50	0,041788		
CAMPOUT	4,830729e-100	3,245882e-101	3,022547e-23	0,000015	1,516380e-11	
IDEA	4,003736e-117	3,484835e-118	8,468832e-45	0,498805	4,988054e-01	4,631606e-08

Tabla 36: P-valores de Conover, ajustados con el método Holm FWER

	Arq.Prop.	CEBOT	ACTRESS	Matarić	ALLIANCE	CAMPOUT
CEBOT	6,920269e-01					
ACTRESS	8,808152e-64	3,626534e-65				
Matarić	1,540777e-114	1,447095e-115	3,769622e-41			
ALLIANCE	4,050589e-121	7,086871e-122	1,070961e-50	0,012188		
CAMPOUT	9,057616e-101	6,491764e-102	5,667275e-24	0,000004	3,032760e-12	
IDEA	1,167756e-117	1,283887e-118	1,482046e-45	0,246968	1,837704e-01	1,013164e-08

Tabla 37: P-valores de Conover, ajustados con el método Benjaminyi-Hochberg FDR

	Arq.Prop.	CEBOT	ACTRESS	Matarić	ALLIANCE	CAMPOUT
CEBOT	9,999999e-01					
ACTRESS	5,759553e-06	3,510779e-06				
Matarić	5,284662e-14	4,851675e-14	0,004004			
ALLIANCE	6,161738e-14	5,617729e-14	0,000335	0,996780		
CAMPOUT	2,665645e-13	1,476597e-13	0,128199	0,925015	0,607425	
IDEA	8,715251e-14	7,971401e-14	0,001339	0,999963	0,999908	0,808137

Tabla 38: P-valores de Nemenyi sin ajustes

De acuerdo con los p-valores de la Tabla 36, la Tabla 37 y la Tabla 38 en sus columnas para la arquitectura propuesta (Arq.Prop.), menores a 0,05, podemos

concluir que existen diferencias significativas entre todas las muestras excepto con CEBOT.

7.3.2 Análisis estadístico para la métrica de energía total consumida

Aplicando el test Shapiro-Wilk, tomando como hipótesis nula (H_0) que los resultados siguen una distribución normal, obtenemos un p-valor=5,55112e-16 y un W= 0,751383. Dado que el p-valor es inferior a 0,05 rechazamos H_0 y asumimos que los datos no siguen una distribución normal. Con este resultado debemos aplicar una prueba no paramétrica para comparar los resultados de las arquitecturas.

Al disponer de 50 repeticiones de la métrica por cada arquitectura, debemos usar el test de Friedman [129] tomando como hipótesis nula la inexistencia de diferencias significativas entre los resultados. Obtenemos un p-valor= 2,286793e-28 (<0,05) que refuta la hipótesis nula de que todos los datos pertenecen a la misma distribución indicando que, al menos, uno de los grupos de datos es significativamente diferente, sin indicar cuál de ellos.

Además realizamos el test por pares para identificar cuáles poseen diferencias significativas. Se realizan dos de los posibles test: (1) Conover y (2) Nemenyi [130]. En el caso del test de (1) Conover, ajustaremos el p-valor con dos procedimientos: family-wise error rate (FWER) de Holm y false discovery rate (FDR) de Benjaminyi-Hochberg [130].

	Arq.Prop.	**CEBOT**	**ACTRESS**	**Matarić**	**ALLIANCE**	**CAMPOUT**
CEBOT	1,296793e-24					
ACTRESS	2,462045e-76	2,249412e-34				
Matarić	4,612649e-87	2,813270e-46	4,450813e-03			
ALLIANCE	9,946685e-94	5,243978e-54	1,411817e-06	7,581245e-02		
CAMPOUT	6,343379e-64	6,687875e-22	1,936123e-03	5,956840e-10	9,154750e-16	
IDEA	3,496563e-99	1,840919e-60	9,756247e-11	7,255423e-04	7,869155e-02	2,109802e-21

Tabla 39: P-valores de Conover, ajustados con el método Holm FWER

	Arq.Prop.	CEBOT	ACTRESS	Matarić	ALLIANCE	CAMPOUT
CEBOT	2,269388e-25					
ACTRESS	7,180966e-77	4,037406e-35				
Matarić	1,699397e-87	5,274881e-47	1,639773e-03			
ALLIANCE	5,222009e-94	1,048796e-54	3,088349e-07	3,980154e-02		
CAMPOUT	1,567188e-64	1,160706e-22	5,647025e-04	1,191368e-10	1,643160e-16	
IDEA	3,496563e-99	4,027010e-61	1,829296e-11	1,792516e-04	7,869155e-02	3,692154e-22

Tabla 40: P-valores de Conover, ajustados con el método Benjaminyi-Hochberg FDR

	Arq.Prop.	CEBOT	ACTRESS	Matarić	ALLIANCE	CAMPOUT
CEBOT	1,505171e-02					
ACTRESS	1,963762e-12	7,498324e-04				
Matarić	7,283063e-14	9,367945e-06	0,968535			
ALLIANCE	4,674039e-14	3,404946e-07	0,728208	0,996780		
CAMPOUT	2,600441e-09	3,169011e-02	0,949984	0,450819	0,143081	
IDEA	5,961898e-14	1,557608e-08	0,391215	0,925015	0,998739	0,036487

Tabla 41: P-valores de Nemenyi sin ajustes

De acuerdo con los p-valores de la Tabla 39, la Tabla 40 y la Tabla 41 en sus columnas para la arquitectura propuesta (Arq.Prop.), todos ellos menores a 0,05, podemos concluir que existen diferencias significativas entre todas las muestras.

7.3.3 Análisis estadístico para la métrica de distancia horizontal (precisión)

Aplicando el test Shapiro-Wilk, tomando como hipótesis nula (H_0) que los resultados siguen una distribución normal, obtenemos un p-valor=0,00000 y un W= 0,409297. Dado que el p-valor es inferior a 0,05 rechazamos H_0 y asumimos que los datos no siguen una distribución normal. Con este resultado debemos aplicar una prueba no paramétrica para comparar los resultados de las arquitecturas.

Al disponer de 50 repeticiones de la métrica por cada arquitectura, debemos usar el test de Friedman [129] tomando como hipótesis nula la inexistencia de diferencias significativas entre los resultados. Obtenemos un p-valor= 1,500632e-18 (<0,05) que refuta la hipótesis nula de que todos los datos pertenecen a la misma distribución indicando que, al menos, uno de los grupos de datos es significativamente diferente, sin indicar cuál de ellos.

Además realizamos el test por pares para identificar cuáles poseen diferencias significativas. Se realizan dos de los posibles test: (1) Conover y (2) Nemenyi [130]. En el caso del test de (1) Conover, ajustaremos el p-valor con dos procedimientos: family-wide error rate (FWER) de Holm y false discovery rate (FDR) de Benjaminyi-Hochberg [130].

	Arq.Prop.	**CEBOT**	**ACTRESS**	**Matarić**	**ALLIANCE**	**CAMPOUT**
CEBOT	4,304115e-28					
ACTRESS	4,523813e-38	3,868864e-02				
Matarić	1,045344e-64	3,174140e-19	4,010184e-11			
ALLIANCE	1,504740e-60	1,231401e-15	3,237727e-08	0,497853		
CAMPOUT	2,350603e-74	1,397038e-28	3,174140e-19	0,038689	0,000838	
IDEA	2,862571e-67	1,369213e-21	4,242743e-13	0,497853	0,184410	0,176458

Tabla 42: P-valores de Conover, ajustados con el método Holm FWER

	Arq.Prop.	**CEBOT**	**ACTRESS**	**Matarić**	**ALLIANCE**	**CAMPOUT**
CEBOT	8,608230e-29					
ACTRESS	1,117648e-38	7,965308e-03				
Matarić	3,851267e-65	5,127456e-20	7,197767e-12			
ALLIANCE	4,388825e-61	2,137142e-16	6,070739e-09	0,261373		
CAMPOUT	2,350603e-74	3,056021e-29	5,127456e-20	0,007965	0,000168	
IDEA	1,502850e-67	2,567274e-22	7,424800e-14	0,470850	0,067940	0,051467

Tabla 43: P-valores de Conover, ajustados con el método Benjaminyi-Hochberg FDR

	Arq.Prop.	CEBOT	ACTRESS	Matarić	ALLIANCE	CAMPOUT
CEBOT	1,106108e-03					
ACTRESS	1,912067e-05	0,975665				
Matarić	1,085221e-11	0,023715	0,236812			
ALLIANCE	1,539359e-10	0,070854	0,450819	0,999798		
CAMPOUT	9,858780e-14	0,000912	0,023715	0,975665	0,874431	
IDEA	1,963762e-12	0,010974	0,143081	0,999987	0,996780	0,995156

Tabla 44: P-valores de Nemenyi sin ajustes

De acuerdo con los p-valores de la Tabla 42, la Tabla 43 y la Tabla 44 en sus columnas para la arquitectura propuesta (Arq.Prop.), todos ellos menores a 0.05, podemos concluir que existen diferencias significativas entre todas las muestras.

Capítulo 8

Conclusiones y trabajo futuro

En esta memoria hemos podido comprobar la consecución del objetivo marcado: la mejora en la comprensión del funcionamiento de las técnicas y arquitecturas de coordinación para avanzar en la dirección de un sistema más eficiente e integrado.

Los resultados de evaluación de la nueva arquitectura obtienen mejores valoraciones en las métricas de características en comparación con las otras arquitecturas, en especial en las funciones de gestión del colectivo.

La implementación usada para todas las pruebas de la arquitectura propuesta hace un uso intensivo de las comunicaciones para mejorar la sincronización entre las acciones y posibilitar el control eficiente de los actuadores y de la distribución del conocimiento. Este uso intensivo de las comunicaciones causa una limitación en el número de individuos a, en el peor de los casos, 12; pero una mejora en el número de acciones a ejecutar en otros individuos y de intercambio de la información. La limitación en el número de individuos no es tan diferente con respecto a las otras arquitecturas, en especial con aquellas cuyo número máximo de individuos no es infinito.

Los resultados de las simulaciones muestran una mejora de eficiencia energética y precisión de la nueva arquitectura, quedando estadísticamente demostrado que dicha diferencia es significativa. En la Ilustración 143, podemos ver los porcentajes de mejora para la media de estas métricas con respecto a los valores obtenidos por el resto de arquitecturas evaluadas. Estas mejoras son el resultado de las capacidades de coordinar acciones entre individuos (usando el subsistema de coordinación) y control de actuadores y sensores desde otros individuos.

Al comparar los resultados de la arquitectura propuesta con los de CEBOT (que tal como hemos visto en el Capítulo 5, es la que tiene mejores resultados en las métricas de simulación) podemos ver una mejora de un 65% en la métrica de distancia (precisión) y un 8% de mejora en el consumo de energía a favor de la nueva arquitectura.

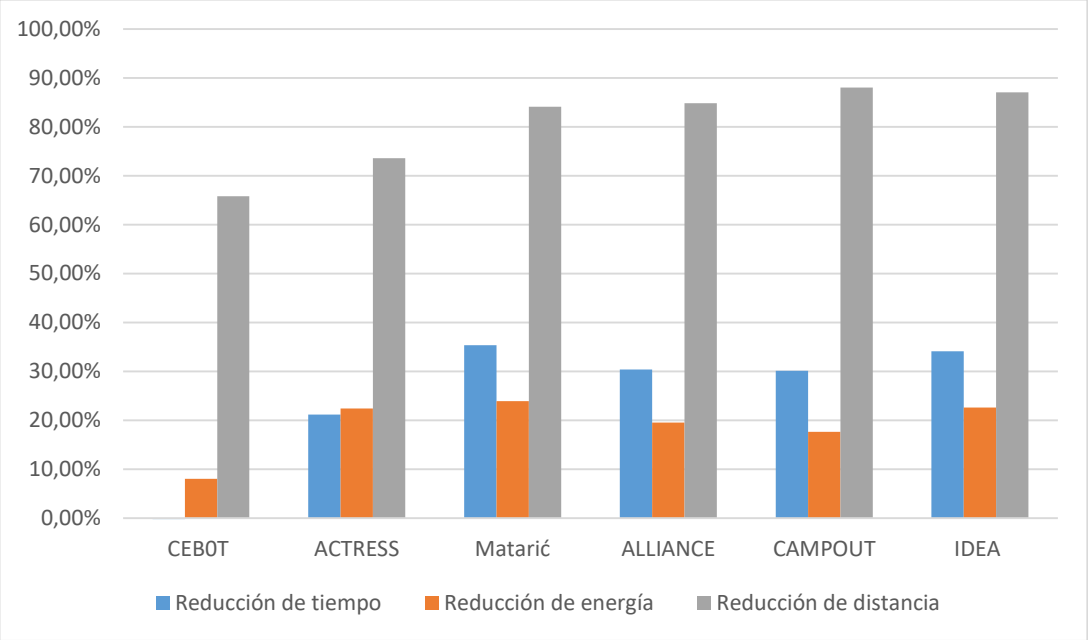


Ilustración 143: Porcentaje de reducción de cada una de la medias de las métricas de simulación en comparación con las otras arquitecturas

8.1 Contribuciones de esta tesis

Las principales contribuciones realizadas durante el desarrollo del presente trabajo han sido:

- La definición de los principales atributos que permiten tanto clasificar una aproximación/solución concreta, como describir sus capacidades. Esta definición incluye los atributos relacionados con el sistema/colectivo, aquellos relacionados con los individuos que pertenecen a dicho colectivo y las distintas clases que se establecen en cada atributo (cuando es aplicable).
- La identificación de las principales técnicas de coordinación y arquitecturas aplicables a grupos de robots que deben trabajar de forma

colaborativa para la obtención de un fin, así como su caracterización usando los atributos previamente definidos.

- La selección de metodologías, métricas y sistemas de evaluación para el desempeño de las arquitecturas del punto anterior que permiten identificar sus ventajas e inconvenientes, así como compararlas entre sí.
- La aplicación de la metodología de evaluación desarrollada a las diferentes arquitecturas.
- Diseñar y desarrollar una implementación de referencia de una nueva arquitectura que mejora los resultados de evaluación de las identificadas además de que reduce los inconvenientes presentes en muchas de ellas, demostrando con un análisis estadístico dicha mejora.

8.2 Trabajo futuro

A partir del trabajo realizado en la presente tesis se abren nuevas líneas de investigación aparte de las posibles continuaciones.

8.2.1 Líneas de continuación

El listado de atributos da lugar a una clasificación en la que cada conjunto de valores de dichos atributos se corresponde con una clase. Esta taxonomía podría formalizarse estableciendo una taxonomía completa para sistemas multi-robot.

La línea de continuación más evidente de este trabajo de investigación es proseguir el desarrollo de la arquitectura propuesta para otros escenarios incrementando las posibilidades de aplicación de la misma que se salen de los objetivos de esta tesis. Junto con dicha implementación y la evaluación de los nuevos componentes en distintos ámbitos, se podría crear un conjunto de patrones cada uno de los cuales representaría una recomendación para un tipo de escenario concreto; este conjunto de patrones se podría condensar en una guía práctica de aplicación. Una implementación concreta de elementos de la arquitectura que se ha evidenciado como necesaria es aquella que limite el uso de comunicaciones (o, incluso, lo elimine por completo) posibilitando así el uso con

individuos sin capacidades hardware de comunicaciones explícitas y reduciendo la limitación en el número de individuos.

Una mejora en alguna de las métricas del sistema de evaluación, en particular la que valora la coordinación para proporcionar más información sobre tareas más complejas, incrementaría el valor añadido de los resultados de dicho sistema. Cualquier modificación a las métricas propuestas debe tener en cuenta los posibles sesgos producidos por diferentes implementaciones de la arquitectura sobre la que se apliquen.

Otra de las posibilidades de continuación del presente trabajo es la aplicación del sistema de evaluación a nuevas arquitecturas que puedan surgir o a modificaciones/evoluciones de las ya analizadas. Así mismo, el sistema de evaluación ha sido diseñado con capacidad de ampliación futura, lo que permite añadir la evaluación de características no previstas en el presente trabajo y la posibilidad de adaptar el sistema a otros ámbitos.

8.2.2 Nuevas líneas abiertas

La nueva arquitectura implementa un conjunto de funciones aplicables sobre colectivos como la migración de conocimiento, la fusión, etc. que abren la posibilidad de aplicar las técnicas ya empleadas sobre los individuos a nivel de colectivos. De esta forma se pueden añadir un número indeterminado de niveles en un colectivo global creando varios colectivos que se interrelacionasen de distinta forma que los individuos que los conforman para la consecución del objetivo global.

Esta arquitectura también permite la investigación en las estrategias de distribución del conocimiento y de los procesos en el Colectivo para mejorar el rendimiento y/o la tolerancia a fallos. Estas estrategias pueden hacer uso de información, disponible en la arquitectura, de la latencia entre el proceso, las fuentes de conocimiento y actuadores que usa.

Otro de los campos abiertos es la aplicación de las técnicas, ya existentes en redes de comunicaciones, a la estrategia de envío de mensajes y su integración con el controlador de los individuos.

Bibliografía

[1] L. Iocchi, D. Nardi y M. Salerno, "Reactivity and Deliberation: A Survey on Multi-Robot Systems", en *Balancing Reactivity and Social Deliberation in MultiAgent Systems. BRSDMAS 2000. Lecture Notes in Computer Science*, vol. 2103, 2001, pp. 9–32.

[2] S. Kernbach, *Handbook of Collective Robotics: Fundamentals and Challenges*, 1ª ed., Singapur: Pan Stanford Publishing, 2013.

[3] M. Brambilla, E. Ferrante, M. Birattari, y M. Dorigo, "Swarm Robotics: A Review from the Swarm Engineering Perspective", *Swarm Intell.*, vol. 7, pp. 1–41, en. 2013.

[4] G. Dudek, M. Jenkin y D. Wilkes, "A Taxonomy for Multi-Agent Robotics"*, Auton. Robot.*, vol. 3, pp. 375–397, dic. 1996.

[5] A. Khamis, *Swarm Intelligence in Robotics*, 2015, doi: 10.13140/RG.2.1.2779.6006

[6] E. Şahin, "Swarm Robotics: From Sources of Inspiration to Domains of Application", en *Swarm robotics*, Springer, 2005, pp. 10–20.

[7] J. Ferber, *Multi-Agent Systems*, Amsterdam: Addison-Wesley, 1999.

[8] Y. Uny Cao, A. Fukunaga y A. Kahng, "Cooperative Mobile Robotics: Antecedents and Directions", en *Autonomous Robots*, 4, 1997, pp. 1–23.

[9] M. Yogeswaran y S.G. Ponnambalam, "Swarm Robotics: An Extensive Research Review", en *Advanced Knowledge Application in Practice*, 2010, pp. 259–278, doi: 10.5772/10361

[10] L. E. Parker, "Multiple Mobile Robot Systems", en *Springer Handbook of Robotics*, Springer, 2008, pp. 921–941.

[11] I. Jawhar, N. Mohamed, J. Wu y J. Al-Jaroodi, "Networking of Multi-Robot Systems: Architectures and Requirements", en *Journal of Sensor and Actuator Networks*, 7:52, nov. 2018, doi: 10.3390/jsan7040052

[12] B. Horling y V. Lesser, "A Survey of Multi-Agent Organizational Paradigms", *The Knowledge Engineering Review*, vol. 19/4. pp. 281–316, nov. 2005.

[13] C. Hewitt, "Toward an Open Systems Architecture", en *Information Processing 89. Proceedings of the IFIP 11th World Computer Congress*, 1993, pp. 389–392.

[14] M. J. Matarić, "Interaction and Intelligent Behavior", PhD thesis, MIT, EECS, Cambridge, MA, 1994.

[15] N. Gildert, A. G. Millard, A. Pomfret y J. Timmis, "The Need for Combining Implicit and Explicit Communication in Cooperative Robotic Systems", *Frontiers in Robotics and AI*, vol. 5, jun. 2018. Disponible: https://doi.org/10.3389/frobt.2018.00065

[16] M. Rubenstein, "Self-Assembly and Self-Healing for Robotic Collectives" Ph.D. Dissertation, University of Southern California, Los Angeles, CA, USA, 2010.

[17] A. R. Ismail y J. Timmis, "Towards Self-Healing Swarm Robotic Systems Inspired by Granuloma Formation", *15th IEEE International Conference on Engineering of Complex Computer Systems*, *ICECCS*, pp. 313-314, mar. 2010, doi: 10.1109/ICECCS.2010.63

[18] G. Bekey y J. Yuh, "Reviewing the Issues of Robotic Self-X", *IEEE Robotics and Automation Magazine*, vol.14, no.4, pp.6–7, dic. 2007, doi: 10.1109/M-RA.2007.905746

[19] C. F. Touzet, "Robot Awareness in Cooperative Mobile Robot Learning", *Autonomous Robots*, 8, pp. 87–97, en. 2000, doi: https://doi.org/10.1023/A:1008945119734

[20] J. Barca y Y. Sekercioglu, "Swarm Robotics Reviewed", *Robotica*, vol. 31, no.3, pp. 345-359, may. 2013, doi: 10.1017/S026357471200032X

[21] A. Christensen, R. Grady y M. Dorigo, "A Mechanism to Self-Assemble Patterns with Autonomous Robots", en: *Proceedings of the 9th European conference on Advances in Artificial Life*, Springer, Berlín, Alemania, 2007, pp. 716– 725.

[22] C. Möslinger, T. Schmickl y K. Crailsheim, "Emergent Flocking with Low-End Swarm Robots", en: *Swarm Intelligence: 7th International Conference, ANTS*, Bruselas, Bélgica, 2010, pp. 424–431.

[23] M. J. Matarić, "Designing Emergent Behaviors: From Local Interactions to Collective Intelligence" en: *Proc. of the Second Int. Conf. on From Animals to*

Animats 2: simulation of adaptive behavior, Cambridge, MA, EE.UU., 1993, pp. 432–441.

[24] J. Goldstein, "Emergence as a Construct: History and Issues", *Emergence*, vol. 1, no. 1, pp. 49–72, mar. 1999.

[25] V. Crespi, A. Galstyan y K. Lerman, "Top-down vs Bottom-up Methodologies in Multi-Agent System Design", *Autonomous Robots*, vol. 24, no. 3, pp.303–313, en. 2008.

[26] M. J. Matarić, "Reinforcement Learning in the Multi-Robot Domain" *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, mar. 1997.

[27] M. J. Matarić, "Using Communication to Reduce Locality in Multi-Robot Learning", en *AAAI/IAAI*, 1997, pp. 646–648.

[28] M. J. Matarić, "Using Communication to Reduce Locality in Distributed Multi-Agent Learning", en *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Learning in DAI Systems*, vol. 10, no. 3, 1998, pp. 357–369.

[29] E. H. Ostergaard, G. S. Sukhatme y M. J. Matarić, "Emergent Bucket Brigading: A simple mechanisms for improving performance in multi-robot constrained-space foraging tasks", en: *Proceedings of the fifth international conference on autonomous agents*, *ACM*, Montreal, Quebec, Canadá, 2001, pp. 29–30.

[30] T. Balch, "Taxonomies of Multirobot Task and Reward", en: *Robot Teams: From Diversity to Polymorphism*, Natick, MA, EE.UU., 2002, pp. 23–35.

[31] G. Dudek, M. Jenkin y E. Milios, "A Taxonomy of Multirobot Systems", en: *Robot Teams: From Diversity to Polymorphism*, Natick, MA, EE.UU., 2002, pp. 3–22.

[32] K. Van Lehn, editor, *Architectures for Intelligence: The 22nd Carnegie Mellon Symposium on Cognition*, Hillsdale, NJ, EE.UU.: Lawrence Erlbaum Associates, 1991**.**

[33] T. Fukuda, S. Nakagawa, Y. Kawauchi y M. Buss, "Self Organizing Robots Based on Cell Structures – CEBOT", en *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS*, Tokio, Japón: IEEE Computer Society Press, 1988, pp. 145–150.

[34] T. Ueyama., T. Fukuda, A. Sakai y F. Arai, "Hierarchical Control Architecture with Learning and Adaptation Ability for Cellular Robotic System", en *Distributed Autonomous Robotic Systems*, Springer, Tokyo, 1994, pp. 17–28.

[35] H. Asama, A. Matsumoto y Y. Ishida, "Design Of An Autonomous And Distributed Robot System: Actress", en *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, IROS*, *The Autonomous Mobile Robots and Its Applications,* Tsukuba, Japón, 1989, pp. 283-290, doi: 10.1109/IROS.1989.637920

[36] C. Hewitt, P. B. Bishop y R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence", en *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, *IJCAI*, 1973, pp. 235–245.

[37] P. Caloud, W. Choi, J-C. Latombe, L. C. Pape y M. Yin, "Indoor automation with many mobile robots", en *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 1990, pp. 67–72, doi: 10.1109/IROS.1990.262370

[38] L. E. Parker, "ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation", en *IEEE Trans. Robot. Autom.*, vol. 14, no. 2, 1998, pp. 220–240, doi: 10.1109/70.681242

[39] L. E. Parker, "Heterogeneous Multi-Robot Cooperation", tesis doctoral, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1994.

[40] A. Garcia-Fornes, A. Crespo y V. Botti, "Adding hard real-time tasks to artificial intelligence environments", en *Proceedings of the 20th 1FAC/IFIP Workshop on Real-Time Programming*, Fort Lauderdale, FL, EE.UU., 1995.

[41] A. García-Fornes, A. Terrasa, V. Botti y A. Crespo, "Analyzing the Schedulability of Hard Real-Time Artificial Intelligence Systems", *Engineering Applications of Artificial Intelligence*, Pregamon Press Ltd., vol.10, no. 4, pp. 369-377, ag. 1997.

[42] V. Botti, C. Carrascosa, V. Julian y J. Soler, "Modelling Agents in Hard Real-Time Environments", en *Lecture Notes in Computer Science,* vol. 1847/1999, 1999, pp. 83–78.

[43] H. P. Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", *The AI Magazine*, vol. 7, no. 2, pp. 38-53, ag. 1986.

[44] R. Simmons, S. Singh, D. Hershberger, J. Ramos y T. Smith, "First Results in the Coordination of Heterogeneous Robots for Large-Scale Assembly", en *Proc. ISER 7th Int. Symp. Exp. Robot.*, Springer, Nueva York, vol.271, 2000, pp. 323-332.

[45] R. Brooks, "A robust layered control system for a mobile robot", en *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, 1986, pp. 14-23, doi: 10.1109/JRA.1986.1087032

[46] R. Alur, R. Grosu, Y. Hur, V. Kumar y I. Lee, "Modular Specification of Hybrid Systems in Charon", en *Hybrid Systems: Computation and Control. HSCC 2000*, Lecture Notes in Computer Science, Springer, Berlín, Heidelberg, vol. 1790, 2000, pp. 6-19.

[47] R. Fierro *et al*., "A Framework and Architecture for Multi-Robot Coordination", *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 977–995, oct. 2002.

[48] H. Bruyninckx, "Open robot control software: The OROCOS project", en *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2001*, vol. 3, 2001, pp. 2523–2528, doi: 10.1109/ROBOT.2001.933002

[49] **"***Open Management Group. CORBA: Common Object Request Broker Architecture*". Accedido el 27-sep-2019. [En línea]. Disponible en: http://www.corba. org/

[50] D. C. Santini y W. F. Lages, "An architecture for robot control based on the OROCOS framework", en *Proceedings of the 4th Workshop on Applied Robotics and Automation*, Sociedade Brasileira de Automática, Bauru, SP, Brasil, 2010.

[51] T. Huntsberger *et al*., "CAMPOUT: A Control Architecture for Tightly Coupled Coordination of Multirobot Systems for Planetary Surface Exploration", en *IEEE Transactions os Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 33, no. 5, 2003, pp. 550–559, doi: 10.1109/TSMCA.2003.817398

[52] P. Pirjanian y M. J. Matarić, "Multi-robot target acquisition using multiple objective behavior coordination", en *Proc. IEEE Int. Conf. Robotics Automation*, San Francisco, CA, EE.UU., 2000, pp. 101–106.

[53] N. Muscettola, G. Dorais, C. Fry, R. Levinson y C. Plaunt, "Idea: Planning at the core of autonomous reactive agents", en *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

[54] F. Mondada *et al*., "Swarm-Bot: A New Distributed Robotic Concept", *Autonomous Robots*, vol. 17, pp. 193–221, sept. 2004.

[55] M. Dorigo, "SWARM-BOT: an experiment in swarm robotics", en *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005, SIS 2005*, Pasadena, CA, EE.UU., 2005, pp. 192–200, doi: 10.1109/SIS.2005.1501622

[56] R. O'Grady, R. Groß, F. Mondada, M. Bonani y M. Dorigo, "Self-assembly on Demand in a Group of Physical Autonomous Mobile Robots Navigating Rough Terrain", en *Advances in Artificial Life, ECAL 2005 Lecture Notes in Computer Science*, vol. 3630, Springer, Berlín, Heidelberg, 2005.

[57] G. Baldassarre, D. Parisi y S. Nolfi, "Distributed Coordination of Simulated Robots Based on Self-Organization", *Artificial Life*, vol. 12, no. 3, pp. 289–311, jul. 2006, doi: 10.1162/artl.2006.12.3.289

[58] E. Tuci, C. Ampatzis, Christos, F. Vicentini y M. Dorigo, "Operational Aspects of the Evolved Signalling Behaviour in a Group of Cooperating and Communicating Robots", en *Symbol Grounding and Beyond. EELC 2006. Lecture Notes in Computer Science*, Springer, Berlín, Heidelberg, vol. 4211, 2006, pp. 113–127, https://doi.org/10.1007/11880172_10

[59] A. L. Christensen, M. Dorigo, L. M. Rocha, L. S. Yaeger y M. A. Bedau, "Evolving an Integrated Phototaxis and Hole-avoidance Behavior for a Swarm-bot", en *Artificial Life X: Proceedings of ALIFE X*, MIT Press, Cambridge, 2006, pp. 248–254.

[60] T. H. Labella, M. Dorigo y J. L. Deneubourg, "Division of labor in a group of robots inspired by ants' foraging behavior", *ACM Trans. Auton. Adapt. Syst*., vol. 1, no. 1, pp. 4–25, sept. 2006.

[61] M. Mouad, L. Adouane, P. Schmitt, D. Khadraoui y P. Martinet, "Architecture Controlling Multi-Robot System using Multi-Agent based Coordination Approach", en *ICINCO'11, 8th International Conference on Informatics in Control, Automation and Robotics*, Noordwijkerhout, Holanda, 2011.

[62] P. Schmitt, C. Bonhomme y B. Gâteau, "Easy programming of agent based electronic institution with UTOPIA", en *10th international conference on New*

Technologies of Distributed Systems, NOTERE 2010, Tozeur, Túnez, 2010, pp. 211–217, doi: 10.1109/NOTERE.2010.5536694

[63] B. Gâteau, D. Khadraoui y E. Dubois, "MOISE_Inst: An Organizational Model for Specifying Rights and Duties of Autonomous Agents", en *1st International Workshop on Coordination and Organisation (CoOrg 2005) affiliated with the 7th International Conference on Coordination Models and Languages*, Namur, Bélgica, 2005, pp. 484–485.

[64] "*The CHARON toolkit Version 1.0*". Accedido el 16-ago-2020. [En línea]. Disponible en: https://rtg.cis.upenn.edu/mobies/charon/implementation.html

[65] "*Open Robot Control Software*". Accedido el 16-ago-2020. [En línea]. Disponible en: https://www.orocos.org/

[66] A. Clark y B. Claise, "RFC 6390: Guidelines for Considering New Performance Metric Development", BCP 170, RFC 6390, 2011

[67] T. Balch, "Hierarchic Social entropy: An information theoretic measure of robot team diversity", *Autonomous Robots*, vol. 8, no. 3, pp. 209–238, jun. 2000.

[68] K. Grimmelsen, F. N. Catbas, R. A. Barrish, M. Pervizpour, E. K. Kulcu y G. Uma, "Stigmergy – An Intelligence Metric for Emergent Distributed Behaviors", en *NIST Workshop on Performance Metrics for Intelligent Systems*, Gaithersburg, MD, EE.UU., 2000, pp. 186–192.

[69] P.P. Grassé, "La reconstruction du nid et les coordinations inter-individuelles chez Bellicositermes Natalensis et Cubitermes sp. La theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs", *Insect Sociology*, vol. 6, 1959, pp. 41–84.

[70] L. E. Parker, "On the Development of Metrics for Multi-Robot Teams within the ALLIANCE Architecture", en *NIST Workshop on Metrics for Intelligence*, Gaithersburg, MD, EE.UU., 2000, pp 205–210.

[71] L. E. Parker. "ALLIANCE: An architecture for fault tolerant multi-robot cooperation", en *IEEE Transactions on Robotics and Automation*, Vol. 14, no. 2, 1998, pp 220–240.

[72] L. E. Parker, "Lifelong adaptation in heterogeneous teams: Response to continual variation in individual robot performance", *Autonomous Robots*, vol. 8, no. 3, pp. 239–267, jul. 2000.

[73] L. E. Parker, "On the design of behavior-based multi-robot teams", *Journal of Advanced Robotics*, vol. 10, no. 6, pp. 547–578, 1996.

[74] L. E. Parker, "Designing control laws for cooperative agent teams", en *Proceedings of the IEEE Robotics and Automation Conference*, Atlanta, GA, EE.UU., 1993, vol. 3, pp. 582–587, doi: 10.1109/ROBOT.1993.291842

[75] L. E. Parker, "Distributed control of multi-robot teams: Cooperative baton-passing task", en *Proceedings of the 4th International Conference on Information Systems Analysis and Synthesis, ISAS '98*, Orlando, Florida, EE.UU., 1998, vol. 3, pp. 89–94.

[76] L. E. Parker, "Cooperative robotics for multi-target observation", *Intelligent Automation and Soft Computing*, special issue on Robotics Research at Oak Ridge National Laboratory, vol. 5, no. 1, pp.5–19, 1999.

[77] Z. Yan, L. Fabresse, J. Laval y N. Bouraqadi, "Metrics for Performance Benchmarking of Multi-robot Exploration", en *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, Hamburgo, Alemania, 2015, pp. 3407–3414, doi: 10.1109/IROS.2015.7353852

[78] K. Mohamed, A. Elshenawy, H. M. Harb, "An Evaluation of Multi-robot Systems Exploration Algorithms", *Journal of Al-Azhar University Engineering Sector*, vol. 14, no. 51, pp. 559–574 abr. 2019.

[79] D. Goldberg y M. J. Matarić, "Interference as a Tool for Designing and Evaluating Multi-Robot Controllers", en *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Conference on Innovative Applications of Artificial Intelligence*, 1997, pp. 637–642.

[80] L. E. Parker y B. Kannan, "Adaptive causal models for fault diagnosis and recovery in multi-robot teams", en *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems,* Beijing, 2006, pp. 2703–2710, doi: 10.1109/IROS.2006.281993

[81] B. Kannan y L. E. Parker, "Metrics for quantifying system performance in intelligent, fault-tolerant multi-robot teams", en *2007 IEEE/RSJ International*

Conference on Intelligent Robots and Systems, San Diego, CA, EE.UU., 2007, pp. 951–958, doi: 10.1109/IROS.2007.4399530

[82] B. Kannan y L. E. Parker, "Fault-Tolerance Based Metrics for Evaluating System Performance in Multi-Robot Teams", en *Performance Metrics for Intelligent Systems Workshop*, Gaithersburg, Maryland, EE.UU., 2006.

[83] G. S. Sukhatme y G. A. Bekey, "An evaluation methodology for autonomous mobile robots for planetary exploration", en *Proceedings of The First ECPD International Conference on Advanced Robotics and Intelligent Automation*, Atenas, Grecia, 1995, pp. 558–563.

[84] R. B. McGhee y A. A. Frank, "On the stability properties of quadruped creeping gaits", *Mathematical Biosciences*, vol. 3, pp. 331–351, ag. 1968.

[85] D. Messuri y C. Klein, "Automatic body regulation for maintaining stability of a legged vehicle during rough-terrain locomotion", en *IEEE Journal on Robotics and Automation*, 1985, vol. 1, no. 3, pp. 132–141.

[86] B. Wilcox, "Mobility characteristic curve", *Jet Propulsion Labs*, IOM 3472-91-019, 1991.

[87] G. S. Sukhatme, M. A. Lewis y G. A. Bekey, "Mission reachability for extraterrestrial rovers," en *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japón, 1995, pp. 1964–1969.

[88] J. Craighead, R. Murphy, J. Burke y B. F. Goldiez, "A Survey of Commercial & Open Source Unmanned Vehicle Simulators", en *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Roma, 2007, pp. 852–857, doi: 10.1109/ROBOT.2007.363092

[89] "*USARSim v2.0.2 - a game based simulation of the nist reference arenas*," 15-jul-2015. Accedido el 16-ago-2020. [En línea]. Disponible en: http://sourceforge.net/projects/USARSim

[90] S. Carpin, M. Lewis, J. Wang, S. Balakirsky y C. Scrapper, "USARSim: a robot simulator for research and education", en *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Roma, 2007, pp. 1400–1405, doi: 10.1109/ROBOT.2007.363180

[91] M. Zaratti, M. Fratarcangeli y L. Iocchi, "A 3D Simulator of Multiple Legged Robots Based on USARSim", en *RoboCup 2006: Robot Soccer World Cup X,*

RoboCup 2006, Lecture Notes in Computer Science, Springer, Berlín, Heidelberg, 2007, vol. 4434, pp. 13–24.

[92] "*Gamebots,*" Dec. 19, 2002. . Accedido el 16-ago-2020. [En línea]. Disponible en: http://gamebots.sourceforge.net/

[93] "*The Mobility Open Architecture Simulation and Tools (MOAST),"* 21-jun-2019. Accedido el 16-ago-2020. [En línea]. Disponible en: https://sourceforge.net/projects/moast/

[94] "*The Stage Simulator*," 9-feb-2019. . Accedido el 16-ago-2020. [En línea]. Disponible en: https://github.com/rtv/Stage

[95] "How to Use Player/Stage," 15-sep-2015. Accedido el 16-ago-2020. [En línea]. Disponible en: https://player-stage-manual.readthedocs.io/

[96] B. Gerkey, R. T. Vaughan y Andrew Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems", en *Proceedings of the 11th International Conference on Advanced Robotics, ICAR'03*, Coímbra, Portugal, 2003, pp. 317–323.

[97] R. Vaughan, "Massively Multiple Robot Simulations in Stage", *Swarm Intelligence*, Springer, vol. 2, no. 2–4, pp. 189–208, dic. 2008.

[98] "*Player - one hell of a robot server*," 9-dic-2018. Accedido el 16-ago-2020. [En línea]. Disponible en: https://github.com/playerproject/player

[99] "*Gazebo*". Accedido el 16-ago-2020. [En línea]. Disponible en: http://gazebosim.org/

[100] "*Webots: robot simulator*". Accedido el 24-ago-2020. [En línea]. Disponible en: https://www.cyberbotics.com/

[101] "robotbenchmark". Accedido el 24-ago-2020. [En línea]. Disponible en: https://robotbenchmark.net/

[102] "*Simbad 3d Robot Simulator*," 1-may-2011. Accedido el 24-ago-2020. [En línea]. Disponible en: http://simbad.sourceforge.net/

[103] "*Eyewire Studio***".** Accedido el 11-dic-2007. [En línea]. Disponible en: http://eyewire.com/studio/

[104] "Microsoft Robotics Developer Studio 4". Accedido el 24-ago-2020. [En línea]. Disponible en: http://www.microsoft.com/robotics

[105] C. Otiniano, P. Geelen, *Microsoft Robotics Developer Studio - Tutorial en español***,** 16-jul-2015. Accedido el 24-ago-2020. [En línea]. Disponible en:

https://social.technet.microsoft.com/wiki/contents/articles/12955.microsoft-robotics-developer-studio-tutorial-en-espanol.aspx

[106] "*MissionLab v7.0,"* 7-dic-2006. Accedido el 24-ago-2020. [En línea]. Disponible en: https://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/

[107] "*SimRobot - Robotics Simulator*". Accedido el 24-ago-2020. [En línea]. Disponible en: http://www.informatik.uni-bremen.de/simrobot/index_e.htm

[108] "*Robocup Federation official website*". Accedido el 24-ago-2020. [En línea]. Disponible en: https://www.robocup.org/

[109] "*RoSiML Robot Simulator Markup Language*". Accedido el 28-ago-2020. [En línea]. Disponible en : http://www.informatik.uni-bremen.de/spprobocup/RoSiML.html

[110] T. Laue, K. Spiess, T. Röfer, "SimRobot – A General Physical Robot Simulator and Its Application in RoboCup", en *RoboCup 2005: Robot Soccer World Cup IX. RoboCup 2005. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2005, vol. 4020, pp. 173–183.

[111] "*SubSim - An Autonomous Submarine Simulation System*", 2006. Accedido el 28-ago-2020. [En línea]. Disponible en: https://robotics.ee.uwa.edu.au/auv/subsim.html

[112] "*RoBIOS Mobile Robot Library*", Ene-2020**.** Accedido el 28-ago-2020. [En línea]. Disponible en: https://robotics.ee.uwa.edu.au/eyebot/robios.html

[113] "*TeamBots 2.0*", 14-abr-2000. Accedido el 28-ago-2020. [En línea]. Disponible en: https://www.cs.cmu.edu/~trb/TeamBots/

[114] "*The Rossum Project*". Accedido el 28-ago-2020. [En línea]. Disponible en: http://rossum.sourceforge.net/

[115] "*Modular OpenRobots Simulation Engine*". Accedido el 31-ago-2020. [En línea]. Disponible en: https://morse-simulator.github.io/

[116] "*MORSE: the Modular Open Robots Simulator Engine*". Accedido el 31-ago-2020. [En línea]. Disponible en: http://github.com/morse-simulator/morse

[117] "*Robot Operating System (ROS)*". Accedido el 31-ago-2020. [En línea]. Disponible en: https://www.ros.org/

[118] "*YARP Yet Another Robot Platform*". Accedido el 31-ago-2020. [En línea]. Disponible en: https://www.yarp.it/

[119] "*pocolibs - openrobots*". Accedido el 31-ago-2020. [En línea]. Disponible en: https://www.openrobots.org/wiki/pocolibs/

[120] "*The MOOS Cross Platform Software for Robotics Research*". Accedido el 31-ago-2020. [En línea]. Disponible en: http://www.robots.ox.ac.uk/~pnewman/TheMOOS

[121] "*MAVLINK Micro Air Vehicle Communications Protocol*". Accedido el 31-ago-2020. [En línea]. Disponible en: https://mavlink.io/

[122] "*PPRZLINK*". Accedido el 31-ago-2020. [En línea]. Disponible en: https://github.com/paparazzi/pprzlink

[123] G. Echeverria *et al*., **"**Simulating Complex Robotic Scenarios with MORSE**",** **e**n *Simulation, Modeling, and Programming for Autonomous Robots. SIMPAR 2012, Lecture Notes in Computer Science*, Springer, Berlín, Heidelberg, 2012, vol. 7628, pp. 197–208.

[124] M.C. Carro, **"**REPRESENTACIÓN EN 3D DE OBJETOS CAPTADOS POR UN ROBOT" Proyecto de Fin de Carrera, Departamento de Electrónica y Sistemas, UDC, A Coruña, 2013.

[125] C. Fernandez-Lozano, M. Gestal, C. R. Munteanu, J. Dorado, A. Pazos, "A methodology for the design of experiments in computational intelligence with multiple regression models", *PeerJ*, 4:e2721, dic. 2016, https://doi.org/10.7717/peerj.2721

[126] S. Shapiro, M. B. Wilk, "An analysis of variance test for normality(complete samples)", *Biometrika*, vol. 52, no. 3-4, pp. 591–611, dic. 1965, https://doi.org/10.1093/biomet/52.3-4.591.48M

[127] M. S. Bartlett, "Properties of sufficiency and statistical tests", en *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 1937, vol. 160, no. 901, pp. 268–282, doi: 10.1098/rspa.1937.0109

[128] J. Derrac, S. García, D. Molina, F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms", *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, mar. 2011, https://doi.org/10.1016/j.swevo.2011.02.002

[129] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings", *Ann. Math. Statist.*, vol. 11, no. 1, pp. 86–92, mar. 1940, doi:10.1214/aoms/1177731944

[130] V. Bhapkar, "Practical nonparametric statistics", *Technometrics*, vol. 14, no. 4, pp. 977–979, abr. 2012.

Índice de figuras

Índice de tablas