UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA EN COMPUTACIÓN

ANÁLISIS DE EXTENSIBILIDAD, REESTRUCTURACIÓN Y DESEMPEÑO DE SOFTWARE PARA ROBOTS MÓVILES

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:
JOSÉ MAURICIO MATAMOROS DE MARIA Y CAMPOS

TUTOR
JORGE LUIS ORTEGA ARJONA
INSTITUTO DE INVESTIGACIONES EN
MATEMÁTICAS APLICADAS Y SISTEMAS

MÉXICO, D.F. ENERO 2013

Agradecimientos:

A CONACyT por el apoyo recibido durante mis estudios de posgrado, tal como se acuerda en el contrato de beca.

PAPIIT IN117612, " Robot de Servicio para Asistencia a Adultos Mayores y en Sistemas Hospitalarios" por el apoyo recibido durante la investigación, tal como solicitaron.

Agradecimientos:

A mis padres y hermanos, que desde pequeño me formaron, ayudaron a crecer y siempre me han dado su cariño y apoyo incondicional.

Al Maestro Carlos de Elías Mondragón, por enseñarme que escribir es más que colocar letras juntas.

Al Dr. Jesús Savage, por apoyarme, darme su confianza y permitirme cumplir mi sueño de hacer robots.

Al Dr. Jorge Luis Ortega, por ser mi guía durante el posgrado y enseñarme cómo escribir bien una tesis.

A todos mis profesores y maestros, por todos sus conocimientos, enseñanzas, y por que en muchas ocasiones fueron como amigos.

A la Universidad, el alma mater, por darme formación, porque gracias a ella ahora soy profesionista, pude cumplir mis más grandes sueños y en ella encontré a mis mejores amigos.

A Isaac Asimov, por toneladas de inspiración.

Y en especial a Erick, Jaime y Juan, mis mejores amigos, por estar siempre allí cuando más los necesito.

RESUMEN

Los robots móviles de propósito general son mucho más que hardware; su funcionamiento se basa en un software tan complejo (o incluso más) que el hardware del que están compuestos. Dicho software suele requerir de computadoras rápidas y con una alta capacidad de procesamiento; especialmente durante la etapa de desarrollo, en la que se deben probar y afinar diversos métodos y algoritmos que acerquen a la solución de los problemas que el robot móvil debe resolver. Se han propuesto diversas soluciones a estos problemas, pero por su naturaleza práctica tienden a imponer restricciones tales como el lenguaje de programación o el sistema operativo. O bien dan por supuesto una arquitectura de hardware, sin ofrecer una guía real para la solución del problema. En el presente trabajo de tesis, se analiza el uso de una arquitectura *Blackboard* para el monitoreo y comunicación entre los diversos programas que operan a un robot móvil, como potencial solución al problema.

Para dicho análisis se compara el impacto que tiene el uso de una arquitectura distribuida (basada en *Peer-to-Peer*) contra la arquitectura centralizada propuesta (basada en *Blackboard*), tanto durante la etapa de desarrollo del sistema de software del robot como en su desempeño. El impacto del uso de cada arquitectura durante la etapa de desarrollo del sistema de software se analiza estimando el número de cambios o actualizaciones a realizar en los componentes sistema cada vez que se modifica un componente de éste, considerándose una mejor arquitectura aquella que reduzca el número de cambios o actualizaciones. Así mismo, se analiza el desempeño de ambas arquitecturas, tanto a nivel teórico, como con base en el tiempo que un robot tarda en ejecutar una tarea de navegación utilizando software desarrollado con base en cada una de las arquitecturas.

Los resultados de este análisis muestran que el número de cambios a realizar en el sistema de software que opera al robot cuando se tiene una arquitectura basada en *Blackboard* son menos que cuando se tiene una arquitectura basada en *Peer-to-Peer*, lo que se traduce en un menor tiempo de desarrollo y mantenimiento, especialmente cuando se considera que dicho software está en constante desarrollo. Aunado a esto, el análisis teórico del desempeño, así como su comprobación experimental, muestran que los desempeños de ambas arquitecturas son equivalentes cuando la granularidad del sistema es media o gruesa.

CONTENIDO

1. INTRODUCCIÓN

No tengo miedo a los ordenadores.
A lo que tengo miedo es a la falta de ellos.

Isaac Asimov

1.1. Contexto

Más que en el complejo hardware del que están compuestos, los robots móviles basan su funcionamiento en un software cuya complejidad es proporcional al nivel de interacción que el robot tiene con su entorno. Por lo que no es de extrañar que el software llegue a ser por mucho más complejo que el hardware, requiriendo por lo general de más de una computadora. Por esta razón, la programación de los robots móviles es un tema interesante que debe ser estudiado a fondo.

Especialmente durante la fase de desarrollo, el software que controla a un robot móvil suele estar dividido en diversos módulos independientes altamente especializados. Éstos se espera sean robustos, actualizables y fácilmente intercambiables, dada la necesidad en esta etapa de probar diversos algoritmos equivalentes.

En consecuencia, se han generado diversas propuestas de solución, principalmente plataformas (*Frameworks*), software intermediario (*middlewares*) y sistemas operativos especializados. Sin embargo, dichas propuestas suelen estar enfocadas a la implementación. Están influenciadas por arquitecturas existentes. Son propietarias, o están pensadas (o diseñadas) para una plataforma específica. Por lo que ninguna provee de una guía que permita modelar el software con independencia de la estructura y componentes del robot mismo.

1.2. Problema a resolver

La coordinación de los diversos módulos que operan un robot móvil es crucial para la correcta operación del mismo. Así mismo, los recursos (principalmente de hardware) que conforman al robot son limitados, y los diversos módulos se ven irremediablemente en la necesidad de competir por ellos cuando la tarea a realizar así lo requiera. En concreto, los módulos de reconocimiento de personas, marcas y objetos competirán por el uso de los dispositivos de adquisición de imágenes; mientras que los módulos de localización y navegación competirán por sensores como láseres, sonares, e incluso podrían llegar a requerir información de las imágenes procesadas. Es necesario que el acceso a estos recursos compartidos esté siempre disponible para todos los módulos por igual.

Durante la etapa de investigación y desarrollo del software es necesario probar con diversos algoritmos e implementaciones, lo que genera múltiples aplicaciones que realizan funciones similares (o incluso la misma función) y que pese a su similitud, su respuesta ante datos de entrada similares puede diferir mucho dado el dinamismo del entorno. Por ejemplo, para realizar la tarea de reconocer una persona utilizando imágenes, una implementación basada en segmentación por color podría ser muy rápida, pero también muy imprecisa, mientras que otra implementación que utilizara características invariantes podría funcionar bien para reconocer a una persona de frente o incluso en tres cuartos de perfil, pero ser lenta y muy costosa computacionalmente. No obstante, ambas serían ineficaces si la tarea consistiera en seguir a la persona (que seguramente daría la espalda a la cámara) donde podría funcionar un reconocedor basado en modelo oculto de Markov.

Durante la etapa de investigación es deseable que módulos equivalentes sean intercambiables entre sí, a fin de reducir los tiempos de desarrollo y prueba. Es decir, que el cambio de una aplicación o módulo por otro de función equivalente requiera de pocas o ninguna modificación en los demás componentes del sistema, aún cuando el nuevo módulo se ejecute en una plataforma diferente. Es necesario además, proveer medidas que hagan las comunicaciones constantes y fiables, que administren correctamente los recursos disponibles y que incluso permitan cambiar dinámicamente las aplicaciones durante la ejecución.

Existen diversas implementaciones como *Frameworks* o sistemas operativos, cuyo objetivo es ofrecer una solución a los problemas mencionados anteriormente. Sin embargo, al ser implementaciones, imponen las restricciones de su propio diseño y carecen del nivel de abstracción suficiente para ofrecer una solución más general. La falta de arquitecturas que establezcan los fundamentos y provean una guía para el diseño del software que opera a los robots móviles ha causado que el desarrollo de dicho software sea más un arte que una ciencia, prolongando los tiempos de desarrollo y frenando el avance de la investigación en este campo.

1.3. Hipótesis

El objetivo principal de la presente investigación es confirmar la hipótesis presentada a continuación:

> "El uso de una arquitectura basada en *Blackboard* ofrece mejor extensibilidad, reestructuración y desempeño similar comparado con una arquitectura basada en *Peer-to-Peer*, cuando la granularidad del sistema es media o gruesa."

Dentro del contexto de la hipótesis, es necesario acotar el significado de "extensibilidad", "reestructuración" y "desempeño", para las cuales se presentan las siguientes definiciones:

EXTENSIBILIDAD

Buschmann, F. et al. en [1] definen extensibilidad de la siguiente manera:

> *Extensibilidad. Se enfoca en la extensión de un sistema de software con nuevas características, así como el reemplazo de componentes con versiones mejoradas y la supresión de características y componentes innecesarios o no deseados…*[1]

REESTRUCTURACIÓN

Buschmann, F. et al. en [1] definen reestructuración de la siguiente manera:

> *Reestructuración. Se ocupa de la organización de los componentes de un sistema de software y de las relaciones entre ellos, por ejemplo cuando se cambia el lugar de un componente moviéndolo a un subsistema diferente…*[2]

DESEMPEÑO

Según el estándar ISO-9126-1 en [2] la eficiencia o desempeño de un producto de software está relacionado con su comportamiento en el tiempo, uso de recursos y apego a las regulaciones establecidas de eficiencia. Para el enfoque de esta tesis, es de importancia el comportamiento en el tiempo el cual en ISO-9126-1 (según [2]) se define como:

> *Comportamiento en el tiempo (desempeño). La capacidad del producto de software para proporcionar un tiempo de respuesta adecuado, tiempo de procesamiento y tasas de rendimiento cuando realiza su función bajo las condiciones establecidas.*

Con esto, y según ISO-9126-1 en [2], el comportamiento en el tiempo de una arquitectura puede medirse como la suma del comportamiento en el tiempo de cada componente, más la suma del comportamiento en el tiempo de cada conector. Así pues, a menor tiempo de comportamiento, mejor desempeño.

---

[1] *Extensibility*.
[2] *Restructuring*.

Contar con una arquitectura que ofrezca mejores características de extensibilidad y reestructuración durante la etapa de investigación y desarrollo del software de un robot móvil afecta benéficamente el desarrollo, reduciendo su tiempo y complejidad; por lo que se puede dedicar más tiempo a la investigación y solución de los problemas y tareas específicos que debe resolver el robot.

1.3.1. Alcance

La validez de la hipótesis se comprueba para los sistemas de software que operan robots móviles de gama media. Sin embargo, podría ser válida para otros tipos de robot. En adelante el término *robot* se utilizará para referirse a los robots móviles de gama media, por ser estos el objetivo del estudio de la presente tesis.

Los términos extensibilidad y reestructuración son analizados a nivel teórico, calculándose la complejidad de realizar adaptaciones al sistema de software en ambas arquitecturas, con base en el número de modificaciones necesarias para actualizarlo dado un cambio en alguno de sus componentes, tal como se describe en el Capítulo 4.

En lo referente al desempeño del sistema, se analiza a nivel teórico, comparando el tiempo de ejecución de las funciones realizadas por el sistema en ambas arquitecturas (Capítulo 5), lo cual es aterrizado mediante los experimentos presentados en el Capítulo 6.

1.4. Aproximación

El problema se aborda proponiendo una arquitectura basada en *Blackboard*, cuyas características de cambiabilidad y mantenibilidad igualen o superen a las de una arquitectura *Peer-to-Peer*, sin que se presente un cambio importante en el desempeño del sistema. De acuerdo a Buschmann, F. et al. en [1], un *Blackboard* es un patrón arquitectónico fiable que soporta cambiabilidad y mantenibilidad, y que centraliza el control del sistema al proveer un repositorio común de datos para todos los componentes que conforman el sistema. Sus principales carencias son: una baja eficiencia, dificultad para probar el sistema, no existe garantía de encontrar una solución satisfactoria, complejidad de desarrollo y no soporte para paralelismo.

En el desarrollo de aplicaciones para robots móviles la eficiencia es un factor clave, ya sea por las limitaciones del hardware disponible, o por la complejidad de los algoritmos que se requieren para realizar dichas tareas. Por esta razón la existencia de un elemento centralizado que pueda llegar a producir un cuello de botella en el sistema debe ser estudiada con cuidado.

Tomando en cuenta los puntos mencionados anteriormente, se evaluarán las siguientes características.

- ✓ Extensibilidad
- ✓ Reestructuración
- ✓ Desempeño

Para los primeros dos puntos (extensibilidad y reestructuración) se analizarán parámetros a partir de los cuales la arquitectura propuesta iguala o supera una arquitectura *Peer-to-Peer*. Para el análisis del desempeño se busca fijar parámetros en la arquitectura que, tomando en cuenta el tiempo de respuesta tanto de sensores como de actuadores (dispositivos que suelen denominarse "lentos") el número de módulos y la granularidad del sistema, tengan una respuesta aproximadamente igual a la respuesta de un sistema equivalente en arquitectura *Peer-to-Peer* y garanticen evitar un cuello de botella.

En cuanto a la dificultad para probar el sistema a la que Buschmann, F. et al. se refieren, es una característica ligada al no-determinismo y falta de reproducibilidad del estado del *Blackboard*. Para el caso de los robots móviles, no debe representar un mayor problema, ya que usualmente operan sobre entornos altamente dinámicos. Inclusive este factor aleatorio puede ser utilizado provechosamente por parte de la inteligencia artificial del robot, dando la apariencia de improvisación o iniciativa. La falta de garantía para encontrar una solución satisfactoria y la complejidad de desarrollo derivan de las limitaciones del componente de control central y de los programas especializados que trabajan conjuntamente, tal como lo describen Buschmann, F. et al. en [1]. En el caso de los robots móviles, esta tarea recae directamente en el motor de inteligencia artificial que es el encargado de tomar las decisiones pertinentes según el estado del robot y del entorno. Por lo que la solución de estos puntos no es responsabilidad de la arquitectura, sino de la implementación de la inteligencia del robot según su propósito.

Es importante mencionar que en el contexto de la presente tesis, se considera a cada uno de los módulos como una caja negra que realiza una o varias funciones bien definidas. No importa si éstos son programas atómicos que se ejecutan en el contexto de un solo procesador/memoria, o bien están conformados por varias instancias que operan de manera paralela o distribuida en uno o más procesadores con memoria compartida o distribuida. En el segundo caso sólo una de las instancias tendrá acceso al módulo *Blackboard*.

1.5. Motivación

1.6. Estructura del presente documento de Tesis

Capítulo 1: Introducción. En este capítulo se presenta una introducción general al trabajo de tesis, planteándose 1) el contexto, 2) el problema a resolver, 3) la hipótesis que plantea de manera formal la solución que se da al problema y 4) la metodología a aplicar para probar que dicha hipótesis se verifica.

Capítulo 2: Antecedentes. En este capítulo se presenta una introducción a la robótica, definiendo qué es un robot, estableciendo la clasificación de los mismos y dando un panorama general de las metodologías de desarrollo utilizadas en la programación de robots móviles. Además se describen arquitecturas de software que pueden ser utilizadas en la programación de robots móviles.

Capítulo 3: Trabajo Relacionado. En este capítulo se presentan los trabajos relacionados más relevantes que tratan el diseño de software para robots móviles, implementaciones de software para robots móviles y del uso de arquitecturas basadas en *Blackboard* en diversos sistemas que requieren del manejo de recursos compartidos.

Capítulo 4: Análisis de Extensibilidad y Reestructuración. En este capítulo se realiza el análisis de las características de extensibilidad y reestructuración de ambas arquitecturas *Peer-to-Peer* y *Blackboard*, así como un análisis comparativo de las mismas.

Capítulo 5: Análisis teórico del desempeño. En este capítulo se realiza el análisis teórico del desempeño de ambas arquitecturas *Peer-to-Peer* y *Blackboard,* considerando tanto granularidad media como granularidad gruesa, así como un análisis comparativo de las mismas.

Capítulo 6. Comparación experimental del desempeño. En este capítulo se describen los experimentos realizados sobre un robot móvil de gama media orientado a servicio doméstico y se comparan los datos obtenidos en cada experimento.

Capítulo 7. Conclusiones. En este capítulo se presentan las conclusiones obtenidas con base en los resultados tanto del análisis teórico como de los resultados obtenidos experimentalmente en los capítulos anteriores. Además en este capítulo se presenta el trabajo futuro.

2. ANTECEDENTES

Primera Ley: Un robot no puede dañar a un ser humano ni, por inacción,
permitir que un ser humano sufra daño.
Segunda Ley: Un robot debe obedecer las órdenes dadas por los seres humanos
excepto cuando tales órdenes entren en conflicto con la Primera Ley.
Tercera Ley: Un robot debe proteger su propia existencia
hasta donde esta protección no entre en conflicto con la Primera o Segunda Ley.

Las Tres Leyes de la Robótica, Isaac Asimov

En este capítulo se presentan los conceptos básicos y definiciones sobre las cuales está fundamentado el presente trabajo de tesis. El capítulo se encuentra dividido en cuatro partes; las dos primeras definen y dan un panorama general del área de la robótica, en especial de la robótica móvil; mientras que las dos últimas describen arquitecturas de software que pueden ser utilizadas en la programación de robots móviles.

La primera parte se presenta una breve introducción a la robótica, definiendo qué es un robot y estableciendo la clasificación de los mismos. En la segunda parte se describen a *grosso modo* las metodologías de desarrollo más utilizadas en la programación de los robots móviles. La tercera parte detalla el patrón arquitectónico *Blackboard* y en la cuarta y última parte se describe la arquitectura *Peer-to-Peer*.

2.1. Acerca de los robots

Cada día la ciencia y la técnica avanzan juntas en un campo que antes era considerado poco más que ciencia ficción: la robótica. Los robots, que comenzaron como herramientas de apoyo para el hombre en entornos de trabajo inaccesibles o riesgosos, hoy buscan salir de las plantas de producción y las grandes fábricas para ganarse un lugar al lado del ser humano, realizando las tareas que a éste le son desagradables o entreteniéndole, con la meta a futuro de poderle ofrecer compañía.

2.1.1. Algunas definiciones

Aún cuando no existe una definición de robot que satisfaga a todo el mundo, cualquier persona es capaz de reconocer a un robot cuando lo ve (y reaccionar en consecuencia). Y esto tiende a aplicarse a cualquier máquina cuya apariencia se asemeje a un ser humano o animal (o a una parte de este), siempre que no tenga partes biológicas. Sin embargo, existen máquinas denominadas robots cuya forma en nada asemeja a ser humano o animal o alguna parte de ellos. Y quizá en un futuro no muy lejano las máquinas que hoy consideramos robots cuenten con partes orgánicas y no por ello dejar de ser máquinas.

Incluso de manera más informal, el vocablo se utiliza también para definir agentes virtuales cuyo comportamiento puede ser descrito por las definiciones anteriores dentro de un entorno virtual, aunque para diferenciales se les suele llamar *bots*.

DEFINICIÓN DE ROBOT

Según la *International Standard Organization* en el estándar ISO 8373, se define robot como:

> *Un manipulador automáticamente controlado, reprogramable, multiuso, programable en tres o más ejes, que pueden estar fijos en un lugar o movilizarse para ser usado en aplicaciones de automatización industrial [3; 4].* [3]

Se espera que en un futuro la ISO termine modificaciones al estándar ISO 8373 para que este incluya a los robots de servicio doméstico [5]. Por otro lado la *Robot Institute of America* define un robot como:

> *Un manipulador multifuncional reprogramable diseñado para mover materiales, partes, herramientas o dispositivos especializados a través de diversos movimientos programados para la realización de diversas tareas [3].* [4]

Aunque no exista una definición clara de lo que es un robot capaz de satisfacer a todos [3], la palabra es derivada del vocablo checo "*Robota*" que significa: "sirviente" [6], etimología que se utiliza aquí para acotar el término robot, excluyendo de éste modo a cualquier implementación de software que simule inteligencia e interactúe de manera virtual con su entorno, sea éste virtual o real, y a cualquier implementación física con capacidades motrices tele-operadas .

Para los propósitos d esta tesis, se define Robot como:

> *Máquina artificial autónoma capaz de percibir e interactuar con su entorno; reaccionando a estímulos predeterminados.*

---

[3] *"An automatically controlled, reprogrammable, multipurpose, manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications"* ISO 8373.

[4] *"A reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks"*, RIA, 1979.

2.1.2. Acerca de la clasificación de robots

La *International Federation of Robotics* separa a los robots en dos grupos: robots industriales y robots de servicio. Adoptando para la primera categoría la definición establecida por la ISO, y propone para la segunda la siguiente definición preliminar: [3]

> *Un robot de servicio es un robot que opera parcial o completamente de manera autónoma para desempeñar servicios útiles para el bienestar de los seres humanos y equipos, excluyendo tareas de manufactura.*[5]

Para esta definición, la *International Federation of Robotics* menciona además (aunque no en la definición) que aunque los robots manipuladores (que la IFR denomina industriales) que no realicen tareas de manufactura, podrían considerarse robots de servicio. Los robots de servicio son, por lo general, robots móviles que en algunos casos cuentan con uno o más brazos manipuladores integrados [3].

CLASIFICACIÓN EN BASE A SU FUNCIÓN

Tomando como base las definiciones dadas anteriormente, los robots se pueden clasificar en dos grupos: robots manipuladores (también llamados robots industriales) y robots móviles [3; 5]. Gracias a su enorme contribución en la industria manufacturera, los robots manipuladores son a la fecha la clase de robots más investigados [3; 4; 5]. Estos robots se caracterizan por su capacidad de mover con gran velocidad y precisión objetos muy pesados o herramientas especializadas dentro de un volumen de trabajo bien delimitado [3; 4; 5]. Un ejemplo de este tipo de robots son los bien conocidos brazos manipuladores que forman, ensamblan y pintan automóviles en las ensambladoras automotrices [3; 4; 5].

El otro grupo es el de los robots móviles, que se caracterizan por poder moverse en su entorno dentro de las limitaciones de sus capacidades motrices. Por lo general este entorno es dinámico, es decir, la distribución de sus componentes varía con el paso del tiempo, y los robots móviles deben de conservar su capacidad de operar e interactuar con el entorno, independientemente de los cambios que éste sufra. Es bastante común que los robots móviles cuenten con un manipulador a pequeña escala de los mencionados en la otra categoría [7].

CLASIFICACIÓN POR GRADO DE INTERACCIÓN

Otra forma de clasificar a los robots es por su grado de interacción con el entorno, sin embargo, esta clasificación suele ser exclusiva de los robots móviles, debido a que los robots manipuladores tienen un entorno (llamado también volumen de trabajo) bien conocido y cuyos cambios están previstos [7].

No existe una escala formal para clasificar el grado de interacción de los robots móviles con su entorno. Sin embargo se puede hablar sin pérdida de generalidad de interacción baja, media y alta. Esta clasificación se adopta *de facto* por los robotcistas.

---

[5] *A service robot is a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations.* [3]

Una interacción baja es, por ejemplo, la de un robot-aspiradora, cuya interacción consiste básicamente en detectar los objetos y personas a nivel de piso para evitar chocar con ellos, y responder a estímulos simples como un par de órdenes verbales simples. Un nivel medio de interacción se observa en los robots humanoides jugadores de *soccer*; los cuales necesitan identificar la pelota, porterías, a sus compañeros y contrincantes, además de reaccionar a la posición de cada uno de los elementos en el campo de juego y de las órdenes que del árbitro con su silbato.

Un alto nivel de interacción es el que se espera de los robots móviles de servicio. Un ser humano espera que uno de estos robots sea capaz de responder a estímulos visuales (por ejemplo reconocer a una persona por su rostro), comprender y reaccionar a órdenes verbales complejas, gestos, señas. Que sea capaz de aprender, de resolver problemas simples y que, además, realice sus tareas con eficacia. Por ejemplo, se espera que el robot tome la decisión de abrir una puerta si la encuentra cerrada y ésta obstruye su paso. Que se retire si estorba el paso de un ser humano, o que busque a una persona para recordarle una cita o evento importante. Y todo esto por "iniciativa propia" [7].

CLASIFICACIÓN GENERALIZADA

Existe una clasificación empírica utilizada por quienes estudian a los robots, que toma en cuenta la relación entorno-robot, así como el hardware y software necesario para llevar a cabo dicha interacción, tomando como punto de comparación las capacidades humanas:

ROBOTS DE GAMA BAJA

Esta categoría comprende a los robots móviles con un número pequeño de sensores y actuadores, capaces de operar solamente en aquellos entornos para los cuales fueron diseñados, realizando tareas específicas, con un nivel bajo de interacción con el entorno. Por lo general son de tamaño "pequeño" (en comparación con los seres humanos) y todo el procesamiento de los datos se realiza dentro del robot. Ejemplos de este tipo de robots son los robots seguidores de línea, robots *micromouse*, robots aspiradora, etc. [7].

ROBOTS DE GAMA MEDIA

Esta categoría comprende robots móviles de propósito general capaces de navegar en cualquier espacio adecuado a sus capacidades motrices, operar en entornos altamente dinámicos (es decir, que cambian constantemente la configuración de sus componentes), y de interactuar plenamente con su medio. Por lo general un robot de gama media puede desplazarse en cualquier entorno en que un ser humano pueda hacerlo; su tamaño no excede el tamaño promedio de un ser humano y el procesamiento de los datos se realiza dentro del robot, aunque pueden auxiliarse de agentes externos [7].

ROBOTS DE GAMA ALTA

Esta categoría comprende a los robots que están dotados de sensores/actuadores altamente especializados. Son capaces de desplazarse por entornos altamente dinámicos e incluso imprevisibles, o que pudiesen resultar peligrosos a los seres humanos. Este tipo de robots no carecen de limitantes respecto a su tamaño y el lugar en el que se procesan los datos, aunque el robot debe ser capaz de seguir operando aunque la conexión con el centro externo de datos o procesamiento se vea interrumpida. Ejemplos de este tipo de robots son los robots mineros, robots de rescate y robots de exploración espacial

2.1.3. Características de los robots de gama media

En un robot de gama media, todo el procesamiento de datos se realiza dentro del espacio físico del robot mismo. Sin embargo, existen robots móviles de gama media diseñados para juegos de futbol soccer que requieren de un agente externo, el cual, con base en las posiciones de los robots y la pelota, elige la jugada más conveniente, además de auxiliar en las operaciones de arbitraje. No obstante los robots jugadores realizan el resto de las tareas de forma autónoma y pueda llevar a cabo un buen partido sin este árbitro.

Se espera que un robot de gama media de propósito general tenga las mismas capacidades que un ser humano, o bien, capacidades equivalentes que le permitan interactuar con la misma facilidad en su entorno como lo haría un ser humano. Esto aplica no solo para las capacidades motrices (navegación, manipulación, etc.), sino también para las capacidades de percepción y cognición del robot, tales como aprendizaje, razonamiento, inferencia, etc. Así un robot de gama media puede tomarse varios minutos para "pensar" en la mejor manera de llegar a su destino, a través de una vía densamente transitada, o para elegir la mejor jugada en un juego de ajedrez. Naturalmente, es lógico suponer que los robots superen por mucho a los seres humanos en algunas habilidades, como por ejemplo en el cómputo numérico, y tener una comprensión limitada en otras como la comprensión y percepción de emociones.

Por regla general, los robots móviles autónomos deben contar con su propio sistema de alimentación (usualmente un conjunto de baterías), el cual debe ser lo más compacto y liviano posible para alimentar sensores, actuadores y unidades de proceso. Dicha restricción de peso tiende a estar relacionada con la cantidad de energía disponible antes de que el robot requiera recargarse o reemplazar las baterías. Es común que las unidades de proceso tengan una alimentación independiente, y por lo general se utilizan una o más laptops/notebooks, cuyo número suele variar entre una y tres. Acorde a la experiencia en el trabajo con robots se ha observado que las laptops/notebooks utilizadas son de arquitectura Von Neumann, con procesadores x86/x64 o compatible de entre dos y ocho núcleos y entre 1GB y 4GB en RAM, utilizándose para las comunicaciones entre computadoras redes *Fast Ethernet* o *Gigabit Ethernet* bajo los estándares 100BASE-T y 1000BASE-T. Por otra parte, en la mayoría de los robots de gama media, el control de los sensores/actuadores se realiza mediante interfaces de hardware, en general microcontroladores, que se ocupan de las tareas de envío o adquisición de datos en tiempo real.

2.2. Programación de software para robots

La programación de un sistema de cómputo ha sido desde sus orígenes un tema oscuro, considerado con frecuencia como un paso trivial entre la creación del algoritmo y su "ejecución" sobre algún conjunto de datos. Tradicionalmente, se considera a la programación de sistemas de cómputo como un arte en el que el programador, con base en sus conocimientos de la arquitectura, el lenguaje y principalmente en su experiencia, implementa un algoritmo que opera sobre un conjunto de datos generalmente desconocidos de antemano; incluyendo todas las funciones auxiliares y heurísticas necesarias.

Esto ha llevado a la tradicional y bien conocida organización (o método de desarrollo) Programa-Principal/Subrutinas [8; 9]; donde una función o procedimiento principal describe la ejecución del algoritmo, auxiliada por una colección de funciones o procedimientos secundarios denominados subrutinas que realizan las tareas más específicas. Tales tareas secundarias, aunque pueden corresponder a otros algoritmos, no están especificadas en el algoritmo principal (se consideran triviales o dadas), por lo que es tarea del programador ingeniárselas y proponer una solución que no comprometa la ejecución del algoritmo principal que se está implementando.

Si bien en las últimas décadas ha habido un importante progreso en Ingeniería de Software, éste se ha focalizado principalmente en las aplicaciones de negocios o para usuario final, dejando un poco de lado las otras áreas quizá por ser menos redituables.

Es común encontrar software desarrollado en lenguajes funcionales u orientados a objetos (por mencionar algunos paradigmas) que aún están organizados como un programa principal y subrutinas. En la mayoría de los casos, no por desconocimiento del lenguaje por parte del programador, sino por una arraigada costumbre. Lo mismo se aplica para la programación de los robots. Por sencillo que sea, todo robot cuenta con sensores y actuadores que tienen tiempos de activación y respuesta. Y por lo general, se crean subrutinas que realizan la adquisición de datos de los sensores y la transmisión de datos a los actuadores, aún cuando dichas subrutinas se reduzcan solamente una instrucción.

Durante la ejecución de un programa que controle el funcionamiento de un robot mediante un determinado algoritmo, suele ser importante que los datos estén listos a tiempo. Sin embargo, carece de importancia para el algoritmo si los datos se solicitan al sensor en el instante preciso en que se ejecuta ese paso del algoritmo, o si éestos fueron solicitados de antemano por un agente externo y almacenados en un buffer para estar disponibles cuando el algoritmo lo requiera. Esto hace inevitable pensar en concurrencia, e incluso para robots de mayor tamaño que requieran más poder de procesamiento: cómputo paralelo y distribuido.

2.2.1. Concurrencia, paralelismo y cómputo distribuido en software para robots

A diferencia de otro tipo de sistemas, en un robot la mayoría de las tareas concurrentes (y en algunos casos, todas) trabajan de manera cooperativa para el correcto funcionamiento del sistema completo, aunque con determinada autonomía unas de otras. Por esta razón, la organización tradicional de programa principal con subrutinas mencionada anteriormente es inadecuada en la mayoría de los casos, aunque mucha de la programación para robots es realizada con esta organización.

En muchos robots, especialmente en los manipuladores, el sensado es tan importante que se incluye hardware especializado que realiza las mediciones de manera automática. Como menciona Veli-Pekka en [10], ésto puede realizarse integrando hardware especializado que escribe directamente en la memoria del microcontrolador. O bien, incorporando otro microprocesador completo que opere al sensor y se comunique con el procesador principal, ya sea por memoria compartida o por paso de mensajes. Cabe mencionar que lo mismo es válido en el caso de los actuadores.

Ante este escenario, es más natural pensar en un robot como un conjunto de subsistemas que trabajan cooperativamente, orquestadas por un coordinador central, que como una función principal que ejecuta llamadas a subrutinas. Con base en lo anterior se puede deducir que a mayor complejidad y autonomía de los subsistemas que componen a un robot, mayores inconvenientes presentará una programación procedural o estructurada para la operación del mismo. En el caso de los robots móviles con un alto grado de interacción con el entorno, el software de cada subsistema puede llegar a ser lo suficientemente complejo como para requerir un clúster completo. En estos casos es necesario recurrir a las arquitecturas, paradigmas y herramientas para el diseño e implementación software paralelo y distribuido.

2.2.2. Arquitectura Igual a Igual (*Peer-to-Peer*)

En esta sección se presenta una introducción, así como una breve descripción de la arquitectura *Peer-to-Peer* o igual-a-igual; que en parte de la literatura es referida como P2P.

El término *Peer-to-Peer* se refiere a una clase de sistemas y aplicaciones que utilizan recursos distribuidos para desempeñar una función de manera descentralizada [11]. En un sistema distribuido, los *peers* (o iguales) son procesos que se comunican todos entre sí, y a diferencia de las arquitecturas cliente-servidor, pueden actuar como cliente o como servidor a conveniencia [1]. Los sistemas *Peer-to-Peer* se han usado principalmente para sistemas personales, aunque su uso comienza a extenderse a aplicaciones de negocios. En estos sistemas descentralizados el cómputo puede realizarse en cualquier nodo en la red, sin que exista una clara distinción entre clientes y servidores [12].

DEFINICIONES

El cómputo entre iguales o *Peer-to-Peer* es un tema muy controversial. Muchos expertos creen que existe poca innovación en este campo. Además existe mucha confusión respecto a preguntas como: ¿Qué exactamente constituye *Peer-to-Peer*?, ¿es el cómputo distribuido realmente *Peer-to-Peer*? ó ¿cuáles son las limitantes de los sistemas *Peer-to-Peer*? [11].

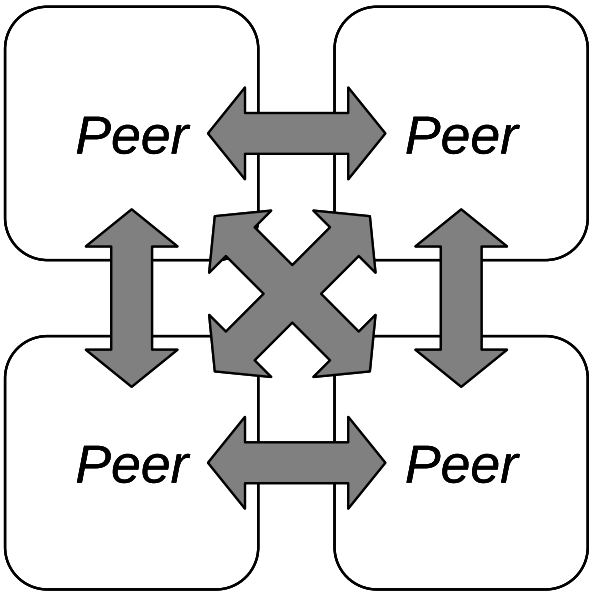
Sommerville define una arquitectura *Peer-to-Peer* en [12] como:

> *Arquitectura descentralizada donde no existe distinción entre clientes y servidores. El cómputo puede ser distribuido sobre muchos sistemas bajo diferentes organizaciones.*

Otras definiciones se presentan por Milojicic en [11], por ejemplo:

- *Colaborar con recursos de cómputo y servicios por intercambio directo entre sistemas.*[6] *(Grupo de trabajo P2P de Intel)*
- *El uso de dispositivos en la periferia de Internet con capacidades de no-cliente.*[7] *(Alex Weytsel)*
- *"*Peer-to-Peer *es una clase de aplicaciones que toman ventaja de los recursos (ciclos de procesador, almacenamiento, contenido, presencia humana) disponible en la red. Debido a que el acceso de estos recursos descentralizados conlleva operar en un entorno de conexiones inestables y direcciones IP impredecibles, los nodos deben operar fuera de un sistema DNS y tener una fuerte o total autonomía de servidores centrales".*[8] *(*Clay Shirky, O'Reilly*)*

ESTRUCTURA



[6] "*The sharing of computer resources and services by direct exchange between Systems*"
[7] *"The use of devices on the internet periphery in a non-client capacity"*
[8] "*P2P is a class of applications that takes advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers*"

Figura 2.1 Estructura de una arquitectura *Peer-to-Peer*.

Según Sommerville [12], las aplicaciones y sistemas *Peer-to-Peer* están diseñados para tomar ventaja del poder de cómputo y del almacenamiento disponible en una red de computadoras potencialmente grande, y todos los estándares y protocolos requeridos para la comunicación se integran en la aplicación, por lo cada nodo debe ejecutar una copia de la aplicación. Por otro lado Milojicic en [11] enfoca los sistemas *Peer-to-Peer* no en el poder de cómputo, sino en compartir recursos en comunidad de iguales. Estos recursos pueden ser poder de cómputo, información (almacenamiento y contenido), ancho de banda, computadoras, humanos, etc., por lo que la función crítica puede ser computo distribuido, contribuir con datos/contenido, o préstamo de servicios.

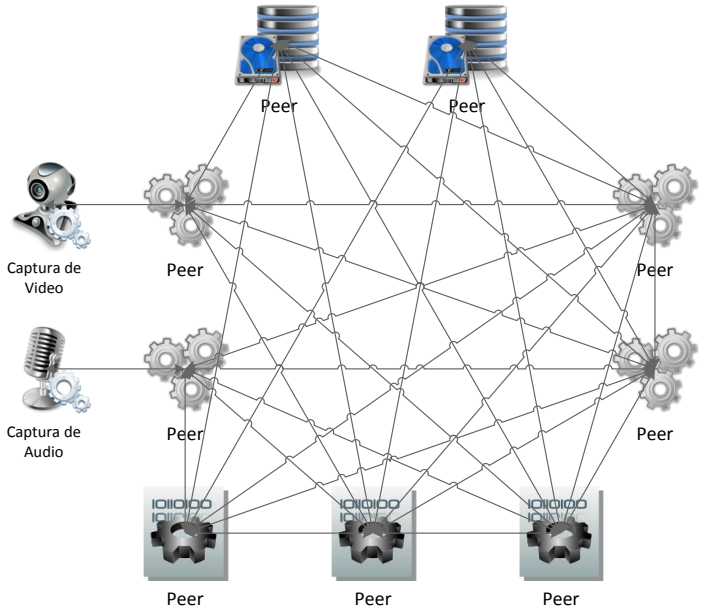


Figura 2.2 Arquitectura *Peer-to-Peer*.

En una arquitectura descentralizada, los nodos no sólo fungen como elementos funcionales en la red, sino que además toman el papel de enlaces que permiten enrutar paquetes entre un nodo y otro, lo que permite ofrecer varias rutas (alta redundancia) a los mensajes, permitiendo que éstos lleguen a destino aún cuando una de las rutas se interrumpa o sobresature, lo que genera una tolerancia a fallos respecto a la caída de nodos en la red [12]. Además, al no depender de elementos centralizados, se elimina la necesitad de infraestructura costosa, lo que aumenta la escalabilidad, pues la comunicación directa entre los *peers* facilita la incorporación de nuevos recursos [11].

La descentralización puede aplicar tanto a algoritmos, programas, datos, meta datos o a todos ellos; sin embargo, una arquitectura *Peer-to-Peer* no limita la inclusión de elementos que operen de manera centralizada [11].

ARQUITECTURA *PEER-TO-PEER* SEMI-CENTRALIZADA

Un problema de las arquitecturas *Peer-to-Peer* puras es que pueden existir sobrecargas debido a la cantidad de nodos que pueden procesar el mismo paquete, es decir, la replicación excesiva produce sobrecargas [12]. Como solución a este problema se establece una arquitectura semi-centralizada donde, dentro de la red, uno o más nodos actúan como servidores para facilitar la comunicación entre los nodos. En esta variante, el papel del servidor es ayudar a establecer contacto entre los *peers*, o coordinar los resultados de un cómputo [12].

En un sistema *Peer-to-Peer* donde una tarea computacionalmente pesada (uso intensivo del procesador) se distribuye en un número grande de nodos. Es normal que algunos nodos se distingan de otros y en vez de realizar trabajo de cómputo relativo a la tarea, se encarguen de distribuir el trabajo entre los otros nodos y de cotejar los resultados del cómputo [12]. A pesar de que la arquitectura *Peer-to-Peer* ofrece un cómputo más eficiente que las arquitecturas tradicionales orientadas a servicios, existen aún muchos problemas con las arquitecturas *Peer-to-Peer* como los problemas de seguridad y confianza, que siguen aún sin resolverse. Esto reduce a los sistemas *Peer-to-Peer* a un uso donde no exista proceso de información crítica [12].

COMPARACIONES

En la investigación sobre varios casos de estudio, se realiza un análisis cuyos resultados se presentan en la Tabla 2.1. En dicho análisis se comparan los requisitos del sistema o aplicación y características clave respecto a tres organizaciones: a) una organización completamente centralizada, donde el elemento central tiene control absoluto de lo que realizan los demás nodos (por ejemplo un dominio); b) una organización cliente/servidor, donde el servidor tiene control respecto a los clientes que acceden al servicio y c) una organización *Peer-to-Peer* [11]:

Requisitos del Sistema/Aplicación	**Tipo de Sistema**		
	Centralizado	**Cliente / Servidor**	*Peer-to-Peer*
Descentralización	Baja (ninguna)	Alta	Muy Alta
Conectividad *ad-hoc*	No	Media	Alta
Costo de propiedad	Muy Alta	Alta	Baja
Anonimato	Baja (ninguna)	Media	Muy Alta
Escalabilidad	baja	Media	Alta
Desempeño	Individual: Alta En conjunto: baja	Media	Individual: Baja En conjunto: Alta
Resistencia a fallas	Individual: Alta En conjunto: baja	Media	Individual: Baja En conjunto: Alta
Auto organización	Media	Media	Media
Transparencia	Baja	Media	Media
Seguridad	Muy Alta	Alta	Baja
Interoperabilidad	Estandarizada	Estandarizada	En progreso

Tabla 2.1. Resultados de la comparación de los requisitos del sistema/aplicación para tres tipos de organizaciones.

En la Tabla 2.2 se muestran los resultados de la comparación de características clave, tanto para el usuario final, el desarrollador como el IT Pro (Profesional en tecnologías de información)[9], para tres organizaciones: a) una organización completamente centralizada, b) una organización cliente, y c) una organización *Peer-to-Peer* según [11].

Objetivo	Criterio	Tipo de Sistema		
		Centralizado	**Cliente / Servidor**	*Peer-to-Peer*
Usuario	**Capacidad de Penetración (*Pervasiveness*)**	Baja	Media	Alta
	Estado del arte	Baja	Alta	Media
	Complejidad	Alta	Baja	Media
	Confiabilidad y reputación	Alta	Media	Baja
Desarrollador	**Complejidad**	Alta	Inmediata	Típico: No Atípico: Sí
	Sostenibilidad	Baja	Alta	Media
	Herramientas	Media (Propietarias)	Alta (Estandarizado)	Baja (Pocas Herramientas)
	Compatibilidad	Media	Alta	Baja
IT Pro	**Rendición de cuentas**	Alta	Media	Baja
	Bajo control	Alta (Total)	Media	Baja
	Capacidad de gestión	Media	Alta	Baja
	Estándares	Media (Propietarios)	Alta	Baja (Inexistente)

Tabla 2.2. Resultados de la comparación de características clave para tres tipos de objetivos con base en tres tipos de organizaciones.

2.2.3. Arquitectura *Blackboard*

En esta sección se presenta una breve descripción del patrón arquitectónico *Blackboard*, basada principalmente en la descripción realizada por Buschmann, F. et al. en [1].

La arquitectura *Blackboard* se diseña originalmente como un método para manejar problemas complejos y mal definidos. El primer ejemplo famoso es el sistema de reconocimiento de voz *Hearsay II*. Como ejemplo más reciente, el componente PLAN del sistema de control de misiones para RADSAT-1 es un sistema *Blackboard* [13].[10] Otros ejemplos de uso incluyen procesamiento de señales, reconocimiento de patrones y reconocimiento del habla [8].

[9] IT Pro o Profesional en Tecnologías de Información se refiere comúnmente a una persona encargada de la gestión y administración de la seguridad y los recursos en una red de computadoras entre otras actividades.
[10] RADARSAT-1 es un sofisticado satélite de observación de la Tierra desarrollado por Canadá para monitorear cambios en los recursos naturales del planeta [13].

A continuación se describe la arquitectura *Blackboard* con la forma de un patrón de software.

RESUMEN

Cuando no se conoce una solución o estrategia determinística para resolver un problema, el patrón arquitectónico *Blackboard* puede ser útil. En un *Blackboard,* diversos subsistemas especializados se conjuntan para crear una solución parcial o aproximada al problema planteado [1].

Una arquitectura *Blackboard* es una colección de programas independientes que trabajan cooperativamente sobre una estructura de datos común. El componente *Blackboard* es el almacén de datos central, del que los demás componentes leen y escriben coordinados por un componente de control (véase Figura 2.3) [1].

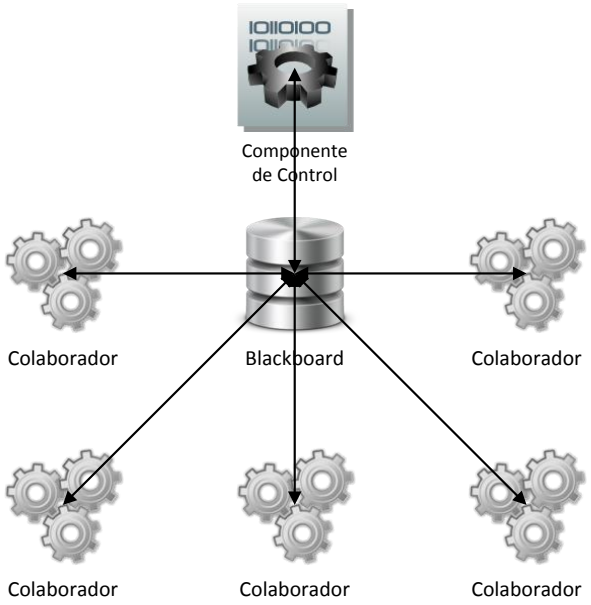


Figura 2.3 Arquitectura *Blackboard.*

Una forma más intuitiva de describir un sistema *Blackboard* es la siguiente: imagine un grupo de humanos especialistas expertos en un campo sentados alrededor de un pizarrón. Los expertos trabajan cooperando unos con otros con el fin de resolver un problema, utilizando el pizarrón como espacio común para intercambiar conceptos e ideas durante el desarrollo de la solución [13]. La solución del problema comienza con la presentación del problema y escribiendo los datos iniciales en el pizarrón; tarea que realiza el moderador. Los especialistas están pendientes de la información contenida en el pizarrón a la espera de una oportunidad de contribuir a la solución del problema. Si un especialista encuentra la oportunidad, registra su contribución escribiendo en el pizarrón, con la esperanza de que ésta sea útil a los otros especialistas. Esto se repite hasta que se encuentra una solución al problema [13].

CONTEXTO

Un dominio inmaduro para el que se desconoce un enfoque cercano a una solución, o bien dicho enfoque no es factible.[11]

PROBLEMA

El patrón *Blackboard* aborda problemas para los que no existe una determinística factible, tales como reconocimiento del habla, reconocimiento de imágenes, visión, etc. Estos se caracterizan por ser un problema que al descomponerse en sub-problemas, abarca diversos ámbitos de experiencia. Las soluciones a los problemas parciales requieren de diferentes representaciones y paradigmas, no existiendo por lo general una estrategia predeterminada para combinar las soluciones parciales [1].

En algunos de los problemas, es posible que se tenga que trabajar con conocimiento incierto o aproximado. Cada paso de transformación puede generar varias alternativas de solución, pero no se garantiza que se encuentre una solución para todos los casos. En consecuencia, las limitaciones de un *Blackboard* deben ser documentadas con cuidado; verificando los resultados cuando si existen decisiones importantes que los afecten [1].

FUERZAS

Un *Blackboard* es útil cuando se presentan los siguientes casos [1]:

- ✓ Cuando una búsqueda en el espacio de solución no es factible en un tiempo razonable.
- ✓ Cuando se necesita experimentar con diferentes algoritmos para la misma tarea. Por esta razón, los módulos individuales deberían ser fácilmente intercambiables.
- ✓ Existen diferentes algoritmos para resolver problemas parciales.
- ✓ La entrada, así como los resultados intermediarios y finales, tienen diferentes representaciones y los algoritmos son implementados de acuerdo a diferentes paradigmas.
- ✓ Un algoritmo usualmente trabaja sobre el resultado de otros algoritmos.
- ✓ Están involucradas soluciones aproximadas y datos imprecisos.
- ✓ El empleo de algoritmos disjuntos induce potencialmente al paralelismo. Cuando sea posible debe evitarse una solución estrictamente secuencial.

VENTAJAS

Una arquitectura basada en *Blackboard* tiene las siguientes ventajas [1]:

- ✓ Experimentación. En dominios en los cuales no existe un enfoque cerrado y una búsqueda completa del espacio de solución no es factible, el patrón *Blackboard* permite la experimentación con diferentes algoritmos y diferentes heurísticas de control.

[11] *An immature domain in which no closed approach to a solution is known or feasible.*

- ✓ Soporte para cambiabilidad y mantenibilidad. La arquitectura *Blackboard* soporta cambiabilidad y mantenibilidad porque las fuentes de conocimiento individuales, el algoritmo de control y el almacén de datos centralizados están estrictamente separados. Sin embargo, los módulos se pueden comunicar mediante el *Blackboard*.
- ✓ Fuentes de conocimiento reusables. Las fuentes de conocimiento son especialistas independientes para determinadas tareas. Una arquitectura *Blackboard* ayuda a hacerlas reusables, siempre que ambas partes manejen el mismo protocolo y estructura de datos.
- ✓ Soporte para tolerancia a fallas y robustez. En una arquitectura *Blackboard* todos los resultados son hipótesis, solo aquellas que son fuertemente soportadas por datos u otras hipótesis sobreviven. Esto provee tolerancia a datos con ruido y conclusiones inciertas

DESVENTAJAS

Una arquitectura basada en *Blackboard* tiene las siguientes desventajas [1]:

- ✓ Dificultad para realizar pruebas. Debido a que los cálculos realizados por un sistema basado en *Blackboard* no siguen un algoritmo determinístico, sus resultados no son reproducibles; además una hipótesis incorrecta es parte del proceso de solución.
- ✓ No se garantiza una buena solución. Usualmente los sistemas basados en *Blackboard* pueden resolver un cierto porcentaje de las tareas dadas correctamente.
- ✓ Dificultad para establecer una buena estrategia de control. La estrategia de control no puede ser diseñada de manera directa, sino que requiere una aproximación experimental.
- ✓ Baja eficiencia. Los sistemas basados en *Blackboard* presentan sobrecargas cuando se rechazan hipótesis erróneas. Sin embargo, si no existe un algoritmo determinista, un sistema de baja eficiencia es mejor que no tener sistema.
- ✓ Alta complejidad de desarrollo. La mayoría de los sistemas basados en *Blackboard* tardan años para evolucionar.
- ✓ Carencia de soporte para paralelismo. La arquitectura *Blackboard* no limita el uso de estrategias de control que exploten el potencial de fuentes de conocimiento paralelas; aunque tampoco las provee. Sin embargo el acceso al almacén de datos centralizado debe estar sincronizado

SOLUCIÓN

La idea detrás de una arquitectura *Blackboard* es una colección de programas independientes que trabajan cooperativamente en una estructura de datos común. Cada programa está especializado en resolver una parte particular de la tarea total, y todos los programas trabajan juntos en encontrar la solución, pero son independientes unos de otros; por lo que no se llaman ni existe una secuencia predeterminada de activación, sino que la dirección tomada por el sistema se determina según el estado actual del progreso, con la coordinación de un componente central de control [1].

ESTRUCTURA

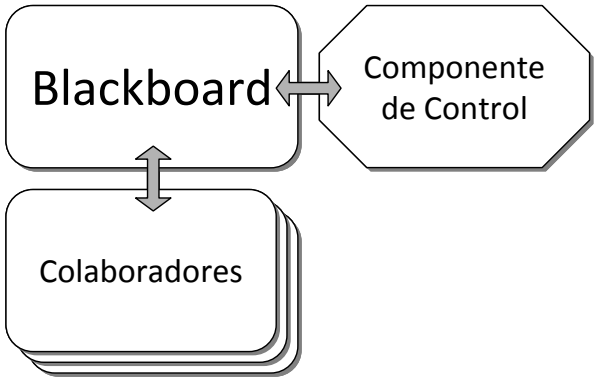


Figura 2.4 Estructura de una arquitectura *Blackboard*.

El sistema se divide en tres componentes: un componente central llamado *Blackboard*, una colección de programas especializados y un componente de control centralizado [1].

El componente *Blackboard* es el centro de almacenaje de datos. Los elementos del espacio de solución y datos de control son almacenados aquí. El *Blackboard* debe proveer una interfaz que permita a todos los demás programas leer y escribir en el [1].

El componente de control centralizado se encarga de orquestar el funcionamiento de los demás programas que trabajan cooperativamente en la búsqueda de la solución. Además suele atribuírsele la tarea de seleccionar la más apta de todas las soluciones posibles, si existe una [1]. Los programas especializados son eso, programas enfocados en resolver una sub-tarea muy particular con la información contenida en el componente *Blackboard* y entregar una o varias soluciones [1].

DINÁMICA

El siguiente escenario ilustra el comportamiento de la arquitectura *Blackboard* en el ejemplo de reconocimiento de habla presentado por Buschmann et al. en [1]:

1. Se inicia el ciclo principal del componente de control.
2. Control llama el procedimiento *nextSource()* para activar el siguiente origen de conocimiento.
3. Primero, *nextSource()* revisa el *Blackboard* y determina qué orígenes de conocimiento son contribuyentes potenciales. En este ejemplo, se asume que los candidatos son Segmentación, Creación de Sílabas y Creación de Palabras.
4. A continuación, *nextSource()* invoca la parte condicional de los orígenes de conocimiento candidatos. En el ejemplo, las partes condicionales de Segmentación, Creación de Sílabas y Creación de Palabras revisan el *Blackboard* y determinan si pueden contribuir al estado actual de la solución.
5. El componente de control elige un origen de conocimiento a invocar y una hipótesis del conjunto de hipótesis para trabajar. En el ejemplo la elección se realiza de acuerdo a los resultados de las partes condicionales; en este caso la parte activa de Creación de Sílabas, que examina el *Blackboard*, crea una nueva sílaba y actualiza el *Blackboard*.

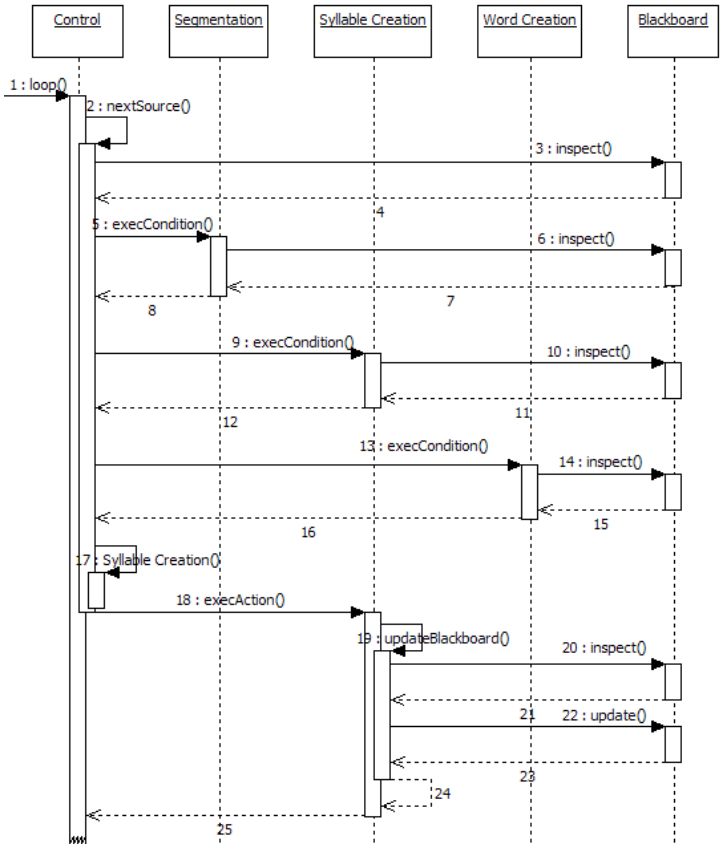


Figura 2.5 Dinámica de una arquitectura *Blackboard* [1].

2.3. Arquitectura básica del software de un robot móvil

Como se describe en la sección 1.3.1, los conceptos de cambiabilidad y robustez se descomponen en distintas propiedades que pueden ser evaluadas en tiempo de desarrollo o en tiempo de ejecución. Las características de tiempo de desarrollo se analizan en la sección 4.2, mientas que las características de tiempo de ejecución se evalúan en la sección 4.3.

Un robot está compuesto principalmente de sensores, actuadores y unidades de procesamiento. Los sensores proveen información del entorno y del robot mismo a los programas que se ejecutan en las unidades de procesamiento (a las que comúnmente se les denomina como unidad de control o controlador), que controlan a los actuadores con base en la información obtenida y al estado del sistema.

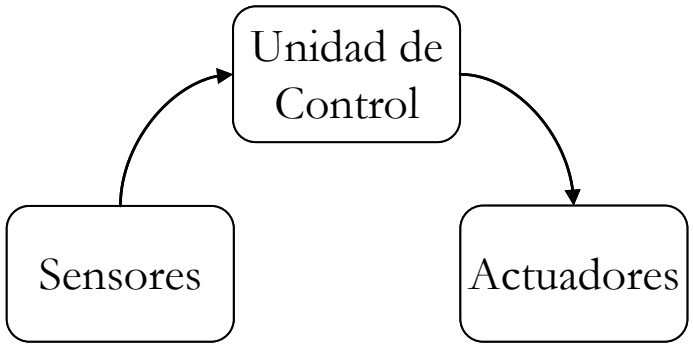


Figura 2.6. Estructura básica del software de un robot móvil (primera aproximación).

Como primera aproximación podemos considerar esta separación para el software que controla a los sensores, actuadores, y el que opera al robot mismo, es decir, la unidad de control. Bajo esta arquitectura la unidad de control debe hacerse cargo del funcionamiento completo del robot, lo que en un robot móvil de gama media incluirá el poder realizar la mayoría de las siguientes tareas:

- ✓ Localización
- ✓ Manipulación de objetos
- ✓ Navegación
- ✓ Reconocimiento de marcas / objetos / personas / voz
- ✓ Síntesis de voz
- ✓ Planeación de acciones o tareas

En la práctica, la ejecución de cada una de estas tareas es computacionalmente costosa, ya que requiere de software complejo. Es deseable que estos programas puedan ser desarrollados simultáneamente, y que el fallo en la ejecución de uno de estos programas no afecte la ejecución de los otros programas. Esta independencia sugiere modularidad, por lo que es posible considerar la siguiente estructura como una segunda aproximación:

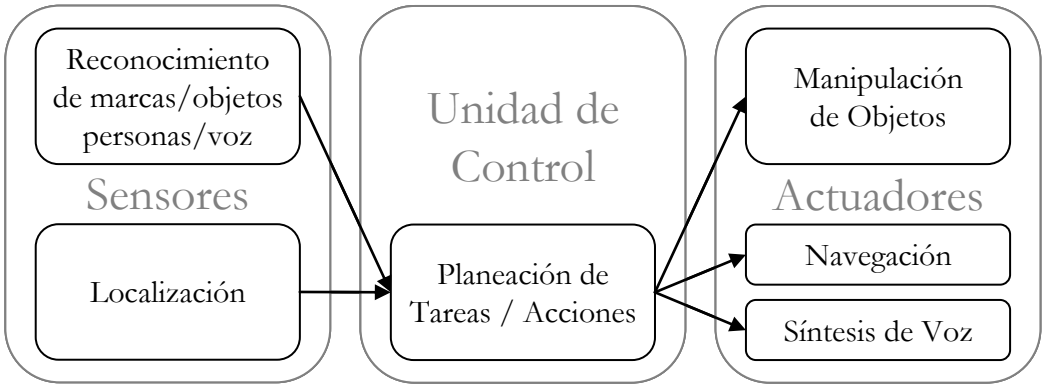


Figura 2.7. Estructura básica del software de un robot móvil (segunda aproximación).

En esta segunda aproximación se deja la tarea del reconocimiento de marcas / objetos / personas / voz, y la localización a los sensores; mientras que la responsabilidad de la manipulación de objetos, la navegación y la síntesis de voz se relega a los actuadores, dejando para la unidad de control la planeación de tareas / acciones.

Un análisis empírico revela que este esquema no sólo compromete al software en un nivel muy estrecho con el hardware, sino que además carece de realismo y practicidad. La localización, por ejemplo, es realizada por muchos seres vivos mediante la identificación de objetos, patrones y marcas usando visión. Del mismo modo, la manipulación de objetos es por lo general un proceso realimentado constantemente por los sentidos del tacto, la vista y el conocimiento empírico de la posición de la configuración física del manipulador.

Por otro lado, si se desease probar otro algoritmo de navegación, habría que reprogramar directamente al actuador encargado de dicha tarea; y la integración de un sensor telemétrico (como un láser) para evitar colisiones durante la navegación sería imposible sin modificar directamente el hardware. Esto sugiere que el software que realiza la tarea cuente con el mayor grado de independencia posible del hardware, con la excepción del software que controla al hardware mismo.

Con base en lo anterior, se propone una tercera aproximación mostrada en la Figura 2.8. En esta aproximación la parte del software relacionada con los sensores se compone sólo del software de adquisición de datos que sirve de interfaz con el hardware de sensado. La parte del software relacionada con los actuadores se compone sólo del software que permite la interfaz y el control del hardware de actuación. Por último la unidad de control se compone de todos los módulos de software que realizan procesamiento de datos y toma de decisiones.

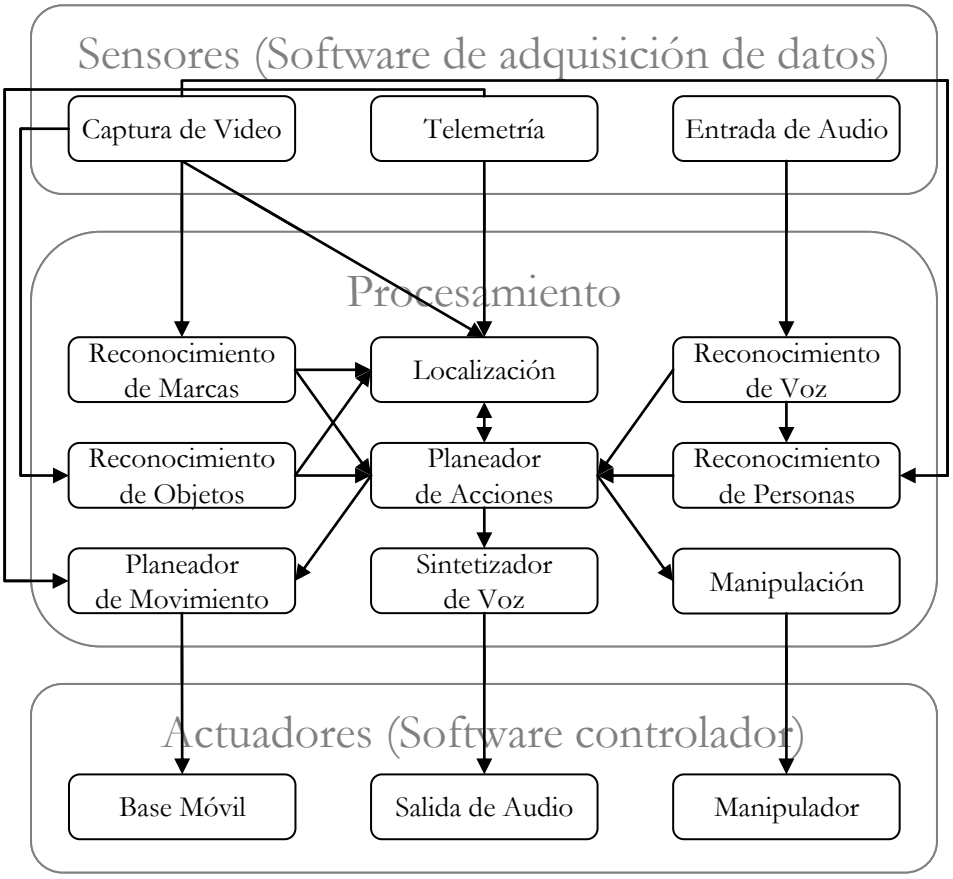


Figura 2.8. Estructura básica del software de un robot móvil (tercera aproximación).

Nótese que en esta tercera aproximación, el software que sirve de interfaz con el hardware ha sido separado y el resto del software que realiza las tareas se ha especializado según su función. Nótese también que las flechas del diagrama representan el flujo de datos mínimo necesario para la operación del robot, por lo que no se contemplan flujos de datos para el control y monitoreo de las aplicaciones.

Como sucede en el caso del sensor telemétrico, otros sensores y actuadores pueden ser agregados si es necesario, y acorde con ello, el software necesario para proveer a los módulos restantes la nueva información accesible (si la requiriesen) o acceso a los nuevos actuadores.

Así como una persona siente dolor, debilidad o algún malestar cuando algo no está bien en su sistema, es deseable que el planeador de acciones tenga información concerniente al estado de cada uno de los componentes del sistema para poder tomar las medidas pertinentes en caso de fallas. Para lograr esto, es necesario establecer una comunicación bilateral entre este componente y cada uno de los componentes del sistema.

2.3.1. Arquitectura *Peer-to-Peer*

Incluyendo esta red de comunicaciones para permitir al planeador de acciones monitorear cada una de las aplicaciones, hace al sistema más robusto, pero incrementa el número de enlaces de comunicación entre los módulos que conforman al sistema. En esta cuarta aproximación, la existencia de comunicaciones muchos-a-uno y uno-a-muchos sugiere la idea de una red de comunicaciones *Peer-to-Peer* como solución natural al problema (véase Figura 2.9):

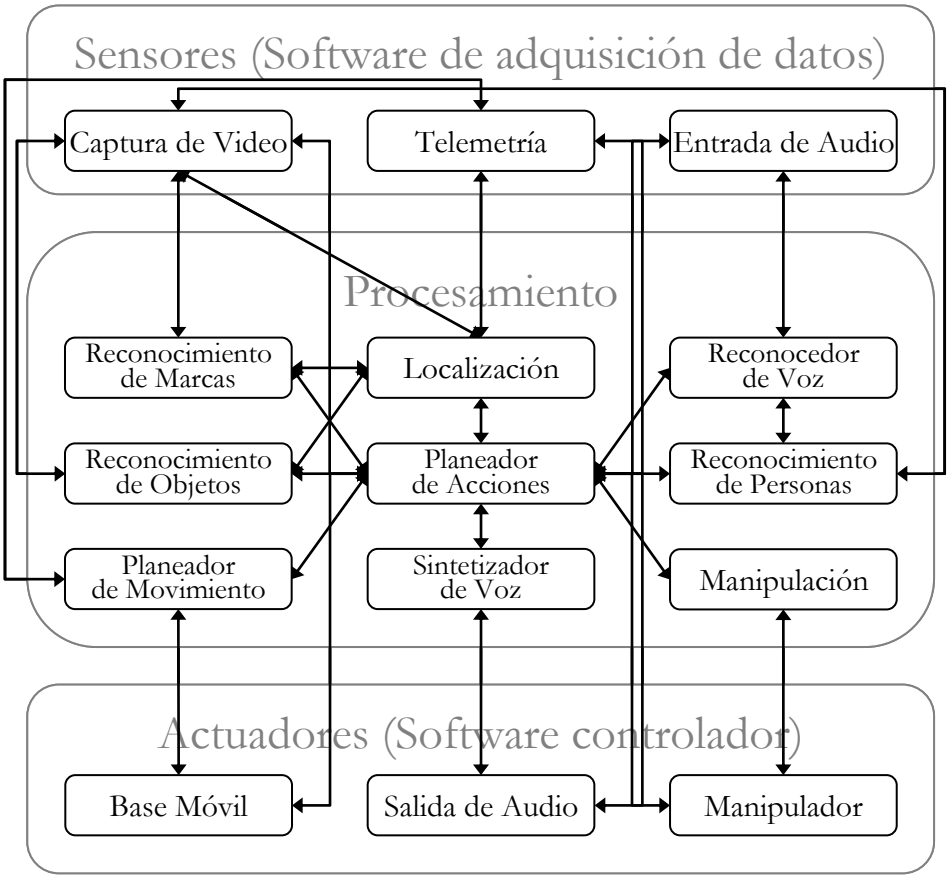


Figura 2.9. Estructura básica del software de un robot móvil basada en arquitectura *Peer-to-Peer*.

En esta organización, cada módulo necesita conocer a cada uno de los otros módulos con los que se conecta e implementar canales de comunicación y protocolos de comunicación para cada uno de ellos, además de proveer soporte para todos los errores que puedan presentarse en cada una de las conexiones.

Supóngase que se agrega un nuevo módulo para obtener datos de odometría. Bajo este escenario, dicho módulo tendrá que conectarse con los módulos de Localización, Planeación de Acciones y Planeación de Movimiento; lo cual requere que además se realicen modificaciones en estos componentes del sistema no sólo para procesar los datos, sino para permitir la comunicación con el nuevo módulo.

Hasta el momento se ha considerado para el análisis de la arquitectura un módulo central que se encarga de la planeación de acciones y de la coordinación del sistema completo. Sin embargo, la planeación de acciones (especialmente las acciones complejas) puede requerir de mucho tiempo de cómputo y en ocasiones es necesario que el robot responda de manera inmediata a un determinado estímulo.

Tomando como ejemplo a los seres humanos, éstos cuentan con varios niveles de reacción a diferentes estímulos, siendo los reflejos los de más bajo nivel (un nivel inconsciente e inmediato). Mientras que un razonamiento complejo o la solución de un problema puede tomar varios minutos e inclusive horas. Así como un ser humano retira de inmediato la mano al recibir una quemadura, un robot tendría que detener su movimiento para evitar una colisión, independientemente de qué tarea esté realizando.

En el caso de los seres humanos, gran parte de los actos reflejos se llevan a cabo en la médula espinal, mientras que las funciones vitales son coordinadas desde el bulbo raquídeo y las actividades cognitivas se realizan en la corteza cerebral; estando las partes relacionadas entre sí con una estructura jerárquica que permite estímulos inhibitorios. La misma idea podría aplicarse a los robots; por lo que el planeador de acciones podría descomponerse en varios módulos independientes e interconectados.

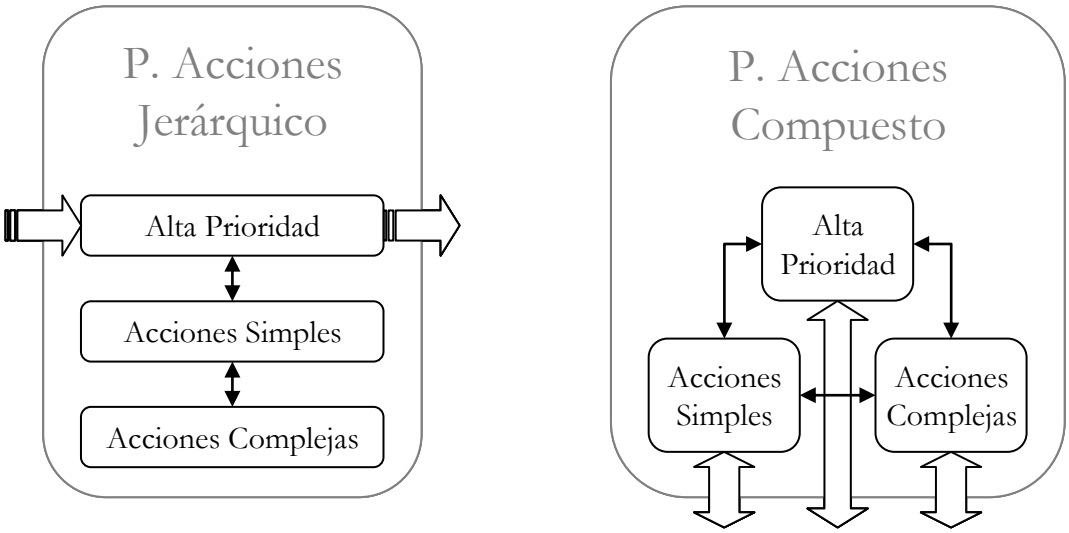


Figura 2.10. Especialización (descomposición) de un módulo del software de un robot móvil basada en arquitectura *Peer-to-Peer*.

La descomposición del planeador de acciones en dos o más módulos aumenta necesariamente el número de conexiones entre los módulos; lo cual puede realizarse con un número pequeño de conexiones conectando los módulos en serie o en una jerarquía de árbol (lo que podría introducir retrasos debidos a la propagación de los mensajes). O bien, un número grande, si cada una de las partes del planeador de acciones opera con absoluta independencia de las demás (planeador de acciones compuesto), en cuyo caso el número de conexiones nuevas es proporcional al número de módulos por el número de partes del planeador de acciones (preservando el desempeño pero aumentando la complejidad del sistema). Lo anterior es aplicable no sólo para el Planeador de Acciones, sino a cualquier otro módulo que requiera ser dividido.

Cuando un módulo es dividido en una estructura jerárquica, se considera al módulo de más alta prioridad como módulo raíz y cada especialización se comunica con el resto del sistema mediante el módulo raíz. En este caso los demás módulos no necesariamente resultan afectados.

Si por el contrario el módulo es descompuesto en varios módulos independientes, todos los módulos con los que el módulo original se comunica resultan afectados y deben adecuarse. Aunado a esto, si tras la descomposición del módulo dos o más partes comparten un recurso (ya sea el recurso un sensor, actuador, u otro módulo), el acceso a dicho recurso compartido debe realizarse con cuidado, a fin de evitar condiciones de competencia que perjudiquen el desempeño del sistema o el envío de órdenes contradictorias. Comúnmente, al realizar una descomposición de este tipo, suele agregarse un módulo adicional que controle arbitre el acceso al recurso compartido.

2.3.2. Arquitectura *Blackboard*

Blackboard representa otra forma de organizar el software que opera un robot. Para ello es necesario agregar dos componentes nuevos a la aproximación presentada anteriormente: un repositorio común de datos llamado *Blackboard* y un componente de control.

Bajo esta arquitectura, todos los componentes se conectan directamente al *Blackboard*, donde se almacenan las variables de estado que determinan el comportamiento del robot, comunicándose entre sí mediante el cambio del valor de dichas variables. El componente de control actúa como moderador, permitiendo que los diferentes módulos operen de manera cooperativa sin estorbarse unos a otros.

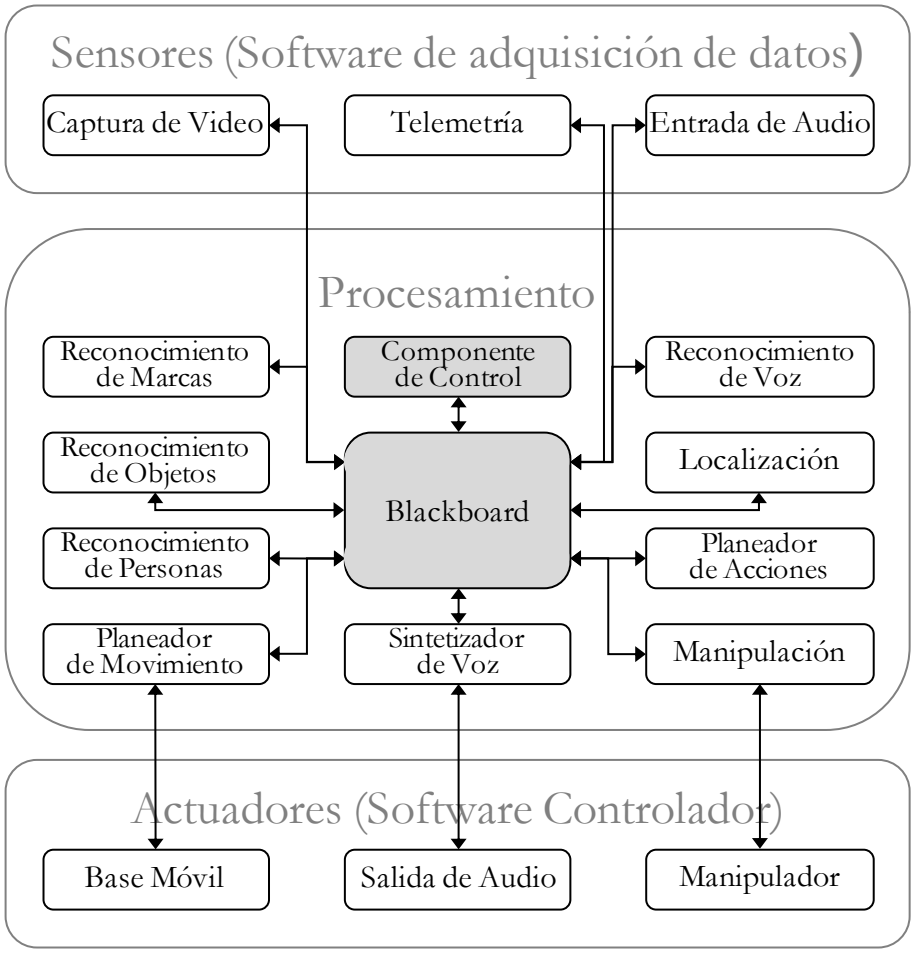


Figura 2.11. Estructura básica del software de un robot móvil basada en arquitectura *Blackboard*.

Esta arquitectura reduce notablemente el número de canales de comunicación necesarios para interconectar todos los módulos a una constante proporcional al número de módulos que conforman al sistema. Sin embargo, al no existir una comunicación directa entre los módulos, el tiempo invertido en las comunicaciones aumenta.

Nótese en la Figura 2.11 que mientras que los módulos que operan los sensores están conectados al *Blackboard*, los módulos que operan los actuadores no. Esto se debe a que el *Blackboard* es una estructura simple que funciona como repositorio de datos y, mientras que los módulos que operan los sensores pueden escribir los valores leídos en el *Blackboard*, los actuadores requieren de una relación muy estrecha con un único controlador que los opere. Lo mismo aplica para cualquier recurso que requiera de acceso exclusivo o para el cual el rendimiento sea un factor crítico.

Supóngase que se agrega un nuevo módulo para obtener datos de odometría. En este escenario con arquitectura *Blackboard*, dicho módulo sólo tiene que alimentar al *Blackboard* con la nueva información disponible. Cualquier módulo que requiera dicha información, como Localización, Planeación de Acciones y Planeación de Movimiento, solo tiene que realizar una operación de lectura de variable al *Blackboard* mediante su canal de comunicación. Ahora bien, si el módulo agregado requiere de interacción con otros módulos o compite con ellos para llevar a cabo una acción, debe también de modificarse el Módulo de Control, pues es éste el que decide que acción o hipótesis tomar de todas las propuestas.

Tal como se menciona en la sección 2.3.1, puede ser necesario descomponer el módulo de Planeación de Acciones (o cualquier otro módulo) en varios módulos independientes e interconectados. El hecho de dividir un módulo no es muy diferente de agregar uno nuevo ya que todos los módulos se comunican únicamente con el *Blackboard*, y por lo general, no necesitan saber de dónde proviene la información contenida en las variables almacenadas en el *Blackboard*, mientras tengan acceso a ellas. Al igual que cuando se agrega un módulo, si alguna de las partes del módulo descompuesto requiere de interacción con otros módulos o compite con ellos para llevar a cabo una acción, debe también de realizarse en el Módulo de Control los cambios pertinentes.

2.4. Arquitectura sistema ViRbot

El robot utilizado para las pruebas de desempeño del capítulo 6 basa su funcionamiento a nivel conceptual en la arquitectura propuesta por Savage et. al. en [14]: el sistema ViRbot. Dicha arquitectura se describe a continuación.

ViRbot es un sistema diseñado para operar robots móviles autónomos capaces de llevar a cabo tareas de servicio en casas, oficinas y fábricas. Este sistema divide la operación de un robot móvil en varios módulos con una función específica que contribuye a la operación final del robot [14].

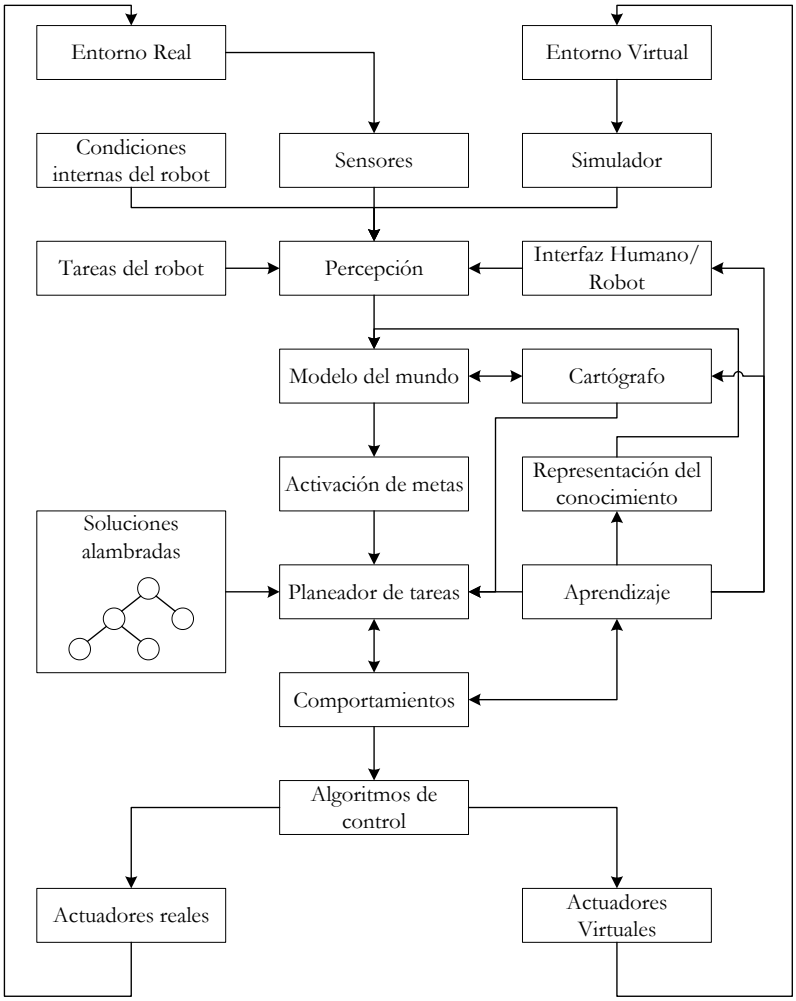


Figura 2.12. Módulo que integran la arquitectura ViRbot [14].

Tal como se presenta en la Figura 2.12, los módulos que componen al sistema ViRbot son los siguientes [14]:

- ✓ Sensores internos. Interfaz para los sensores del robot que proveen información sobre el estado del robot mismo, tales como sensor de baterías, odometría, etc.

- ✓ Sensores externos. Interfaz para los sensores del robot que proveen información sobre el entorno, tales como cámaras, micrófonos, sensores de contacto, sensores telemétricos, etc.
- ✓ Simulador. Provee valores simulados de los sensores internos y externos del robot con base en modelos matemáticos.
- ✓ Tareas del robot. Conjunto de tareas que el robot necesita realizar.
- ✓ Interfaz humano/robot. Permite la interacción humano robot mediante reconocimiento de lenguaje natural, síntesis de voz y expresiones faciales.
- ✓ Percepción. Recibe una representación simbólica de los datos que vienen de los sensores, y genera un conjunto de creencias sobre del estado del mundo.
- ✓ Cartógrafo. Almacena diferentes tipos de mapas para la representación del entorno.
- ✓ Representación del conocimiento. El conocimiento del robot es representado por reglas usando un sistema experto.
- ✓ Modelo del mundo y activación de metas. Las creencias generadas por el módulo de percepción se validan en los módulos de cartógrafo y representación del conocimiento, generando una situación de reconocimiento. Dada una situación de conocimiento, un conjunto te metas son activadas.
- ✓ Soluciones alambradas. Almacena un conjunto de procedimientos almacenados que permiten la solución parcial de problemas, como tomar un objeto o desplazarse.
- ✓ Planeador de tareas. Se divide en dos capas, la capa superior realiza la planeación de acciones mediante un sistema basado en reglas, mientras que la capa inferior planea los movimientos del robot basándose en redes topológicas.
- ✓ Aprendizaje. Permite que el robot aprenda.
- ✓ Comportamientos. Intenta alcanzar cada uno de los nodos provistos por el planeador de movimientos.
- ✓ Algoritmos de control. Encapsula todos los algoritmos de control necesarios para la operación de los actuadores, tanto reales como virtuales.

La arquitectura del sistema ViRbot muestra cómo debe organizarse el software de un robot móvil orientado a servicio e interacción con los humanos [14], que pueden englobarse de acuerdo a lo presentado en la sección 2.1.2 y 2.1.3 como robots de gama media. Sin embargo, si se considera que estos robots cuentan con un gran número de sensores y/o actuadores, es aconsejable que los módulos "sensores" y "actuadores" (tanto reales como simulados) se especialicen a fin de facilitar el mantenimiento del sistema de software.

Aunado a esto, los módulos "planeador de tareas" e "interfaz humano/robot" se dividen en varios sub-módulos que no aparecen en la Figura 2.12 [14], mientras que la mención del uso de sistemas expertos sugiere el uso de diversos lenguajes de programación para implementar dicha arquitectura.

Estas razones conllevan a cuestionar la factibilidad de implementar la arquitectura ViRbot tal y como se presenta en la Figura 2.12, por lo que ésta puede considerarse como una arquitectura a nivel conceptual de cómo organizar los módulos que operan al robot en un alto nivel de abstracción alejado de las fuerzas y limitaciones del hardware.

Por otro lado, considerando las líneas que unen a los módulos de la arquitectura como conexiones entre los módulos sugiere una arquitectura *Peer-to-Peer* en lo que a las comunicaciones se refiere debido a las conexiones uno-a-muchos y muchos-a-uno que aparecen en la Figura 2.12. No obstante, así como la arquitectura básica del software de un robot móvil propuesta en la sección 2.3 puede ser modelada como arquitectura *Peer-to-Peer* (sección 2.3.1) o como arquitectura Blackboard (sección 2.3.2), lo mismo es posible adaptar a la arquitectura de alto nivel de abstracción ViRbot para que opere sobre *Peer-to-Peer* o Blackboard.

2.5. Descripción formal de un sistema de software para robots

Para el análisis realizado en los capítulos siguientes se proponen las siguientes definiciones:

Un sistema S es de la forma:

$$S = (M, C)$$

donde:

- ✓ M es el conjunto de módulos que conforman al sistema S, tales que $M = \{m_1, m_2, \dots, m_n\} \neq \emptyset$
- ✓ C es el conjunto de conexiones entre los módulos tal que $C \subseteq \{(m_i, m_j) \in M \times M | m_i \neq m_j\}$

De esta forma, un sistema se asemeja a la descripción de grafo utilizada por Cormen et. al. en [15], correspondiendo los módulos a los vértices del grafo y las conexiones a las aristas. Es necesario mencionar que la dirección de las aristas del grafo establece la relación de dependencia entre los módulos con base en el flujo de información, no por la dirección del enlace; es decir, la flecha apunta al módulo que realiza la petición y recibe los datos o resultado de la operación (aún cuando esto se reduzca a informar si la ejecución tuvo éxito o no). Para este propósito los canales de comunicación se consideran bidireccionales.

De manera similar se propone que un módulo m del sistema S sea de la forma:

$$m = \{f_1^m, f_2^m, \dots, f_n^m\}$$

donde:

- ✓ $\bar{y}_i = f_i^m(\bar{x}_i)$ es la i-ésima función del conjunto de funciones que puede ejecutar el módulo m.
- ✓ $\bar{y}_i$ es el resultado de la ejecución de la función $f_i^m(\bar{x}_i)$.
- ✓ $\bar{x}_i$ es el argumento o parámetros que requiere f_i para ejecutarse.

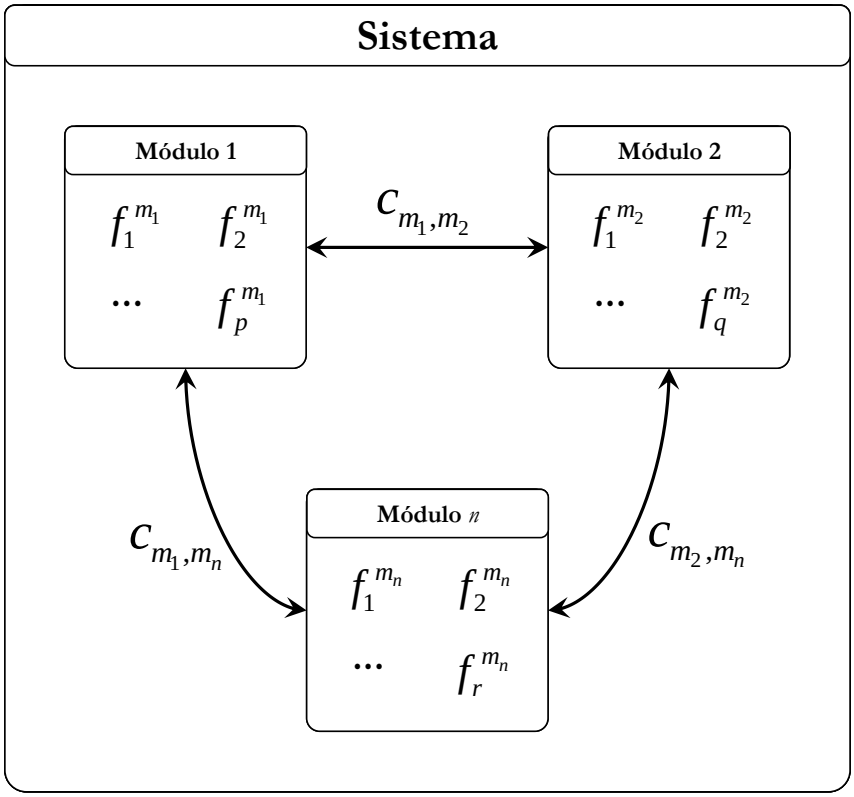


Figura 2.13. Ejemplo de un sistema con tres módulos. Los módulos m_1, m_2 y m_n tienen p, q y r funciones respectivamente

Se definen las siguientes funciones que brindan información sobre el sistema y sus módulos.

- ✓ $\{m_i\}$ Conjunto de módulos que conforman al sistema.
- ✓ $|\{m_i\}|$ Número de módulos que conforman al sistema.
- ✓ $D(m_i)$ La función de dependencia D obtiene el conjunto de módulos que dependen de alguna de las funciones del módulo m_i.
- ✓ $|D(m_i)|$ Número de módulos que dependen de alguna de las funciones del módulo m_i.
- ✓ $\left\{f_j^{m_i}\right\}$ Conjunto de funciones que conforman al módulo m_i.
- ✓ $|m_i| = \left|\left\{f_j^{m_i}\right\}\right|$ Número de funciones que conforman al módulo m_i.
- ✓ $D\left(f_j^{m_i}\right)$ La función D. Obtiene el conjunto de módulos que dependen de la función $f_j^{m_i}$.
- ✓ $\left|D\left(f_j^{m_i}\right)\right|$ Número de módulos que dependen de la función $f_j^{m_i}$.
- ✓ $m_k \xrightarrow{call} f_j^{m_i}$ Llamada a la función $f_j^{m_i}$ desde el módulo m_k.
- ✓ $\left|m_k \xrightarrow{call} f_j^{m_i}\right|$ Número de llamadas a la función $f_j^{m_i}$ desde el módulo m_k.

Además, se puede definir la función tiempo que mapea elementos del conjunto potencia de S al dominio del tiempo, brindando información sobre el tiempo de ejecución de los componentes ejecutables del sistema, denotada por $t: 2^S \rightarrow T$ donde:

- ✓ t es la función tiempo.
- ✓ S es el sistema (o para la función t, el conjunto de elementos que componen al sistema).
- ✓ $S^2 = \{(m_i, m_j) | m_i, m_j \in S\}$ el conjunto de todos los módulos que conforman a S tomados en pares.
- ✓ T representa al dominio del tiempo.

Con esto, pueden definirse los siguientes casos de uso de la función $t: S^2 \rightarrow T$:

- ✓ $t(f_i^m)$ Tiempo que tarda el módulo m en ejecutar la función f_i^m.
- ✓ $t\left(c_{m_i,m_j}\right)$ Tiempo de comunicación entre los módulos m_i y m_j. Se asume que $t(c_{m_i,m_i}) = 0$.
- ✓ $t(m_j, f_k^{m_i})$ Tiempo que tarda el módulo m_j en ejecutar la función f_i^m del módulo m_i.

Es importante hacer notar que la ejecución de una función f_i^m se considera una operación atómica, haciendo $t(f_i^m)$ el tiempo total requerido por f_i^m para su ejecución. Por esta razón, si la ejecución de la función f_i^m requiere de comunicaciones con otros módulos, $t(f_i^m)$ considera los tiempos de ejecución de todas las dependencias de f_i^m. Por otro lado, se considera que $t(f_i^m)$ está acotado, por lo que un módulo siempre recibe una respuesta para cada petición hecha; aún cuando dicha respuesta sea un error generado por la capa de comunicaciones del mismo módulo por un exceso de tiempo de espera.

Como las comunicaciones son gestionadas por una plataforma existente, no es necesario profundizar en su funcionamiento. Suponiendo que los módulos m_i y m_j están conectados directamente puede decirse que si $t(f_k^{m_j}) \gg t\left(c_{m_i,m_j}\right)$ entonces:

$$t\left(c_{m_i,m_j}\right) \approx t\left(c_{m_j,m_i}\right) \cong t_k$$

Donde t_k es una constante equivalente al tiempo que tarda un módulo en enviar un paquete de datos a otro. Considerando que toda comunicación entre módulos está formada por una petición o comando y una respuesta, es decir, es bidireccional, el tiempo invertido en la comunicación t_c entre dos módulos puede expresarse como:

$$t_c = 2t_k \cong t\left(c_{m_i,m_j}\right) + t\left(c_{m_j,m_i}\right)$$

Donde t_c es una constante equivalente al tiempo de comunicación promedio entre dos módulos cualesquiera. Sin pérdida de generalidad, en muchos casos, este tiempo de comunicación se considera como despreciable cuando el tiempo que tarda un módulo m en ejecutar la función f_i^m es suficientemente grande, es decir, $t(f_i^m) \gg t_c$.

3. TRABAJO RELACIONADO

"No basta con sentir respeto por todos los seres humanos,
uno debe respetar a todos los seres inteligentes."

Robots e Imperio. Isaac Asimov.

Este capítulo provee un panorama general sobre los desarrollos de software existentes para robots en general, y específicamente robots móviles. En la primera parte, se presentan dichos desarrollos al nivel de la arquitectura que presenta dicho software. En la segunda parte se revisa más a detalle dichas implementaciones, con un enfoque menos centrado en la arquitectura y más basado en la implementación del mismo (soporte del lenguaje, plataforma, etc.) y su grado de abstracción. Finalmente se analizan soluciones no relacionadas directamente con la robótica pero que pueden adaptarse o utilizarse para el desarrollo de software para robots móviles.

El trabajo relacionado comentado en este capítulo incluye no sólo sistemas para el control de un hardware, modelo o arquitectura en particular como el Open-R SDK para el Aibo de Sony, sino aquellos que han sido creados pensando en servir a casi cualquier robot como es el caso de ROS.

3.1. Arquitecturas de Software para Robots Móviles

En la presente sección se presenta los resultados del análisis de cuatro arquitecturas de software para robots móviles según [9]. Para dicho análisis, se considera que un robot móvil debe cumplir con las siguientes tareas: [9]

- ✓ Adquisición de datos de entrada de los sensores.
- ✓ Controlar el movimiento de sus ruedas y otras partes móviles.
- ✓ Planear la trayectoria a seguir.

De acuerdo con lo anterior, se establece que las arquitecturas candidatas deben cumplir con los siguientes requisitos [9]:

- ✓ La arquitectura debe tener en cuenta comportamientos deliberativos y reactivos.
- ✓ La arquitectura debe admitir incertidumbre debido a que las circunstancias bajo las que opera el robot nunca son completamente predecibles.
- ✓ La arquitectura debe contemplar los peligros inherentes a la operación del robot en el entorno. Esto incorporando los atributos de tolerancia a fallas, seguridad y desempeño, además de ayudar en la conservación de la integridad del robot, los operadores y el entorno.

- ✓ La arquitectura debe dar flexibilidad al diseñador. El desarrollo de aplicaciones para robots móviles requiere experimentación y reconfiguración constante. Además, cambios en las tareas pueden llegar a requerir modificaciones regulares.

3.1.1. Ciclo de Control (*Control Loop*)

La mayoría de los robots industriales ofrecen un mínimo soporte para el manejo de eventos impredecibles (todas sus tareas suelen estar completamente predefinidas) y el robot carece de responsabilidad respecto a los que pasa en su entorno. En este caso se utiliza el paradigma de lazo abierto (*open loop*). Sin embargo, los robots móviles requieren realimentación del entorno, por lo que llevar éste paradigma a la robótica móvil requiere cerrar el ciclo. Entonces, el controlador realizará una acción, espera retroalimentación del mundo externo mediante los sensores y planea futuras acciones con base en esta información. [9]

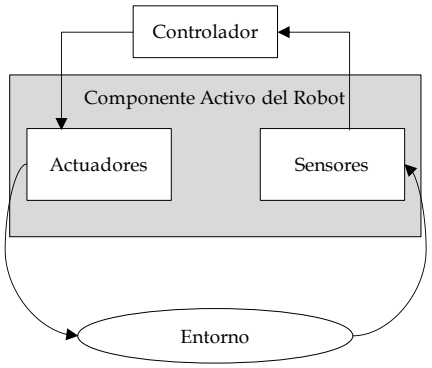


Figura 3.1 Ciclo de control [9].

VERIFICACIÓN DE REQUISITO 1: COMPORTAMIENTOS

La principal ventaja de este paradigma es su simplicidad. Sin embargo, en los entornos más impredecibles esta simplicidad es también un fuerte inconveniente. Una consecuencia del ciclo es que asume que los cambios en el entorno son lineales y requieren reacciones lineales. Además, el modelo no provee de guía alguna respecto a cómo manejar los eventos [9].

Para tareas complejas, el ciclo de control no da la oportunidad de descomponer el software en componentes cooperativos. Si los pasos de sensado, planeación y actuación tienen que ser redefinidos, es necesario recurrir a otros paradigmas [9].

VERIFICACIÓN DE REQUISITO 2: INCERTIDUMBRE

Para resolver la incertidumbre, un ciclo de control intenta solventar la falta de información mediante iteraciones. Un proceso de ensayo-error que elimina posibilidades en cada ciclo. Si se requiere de un proceso más delicado, la arquitectura no permite integrar componentes separados para delegarles la solución del problema [9].

VERIFICACIÓN DE REQUISITO 3: TOLERANCIA A FALLAS Y SEGURIDAD

Este paradigma soporta tolerancia a fallas y seguridad por el simple hecho de que, dada su simplicidad, hace fácil la duplicación y reduce la probabilidad de que se presenten errores en el sistema [9].

VERIFICACIÓN DE REQUISITO 3: FLEXIBILIDAD

La mayor parte de los componentes del robot están separados y pueden ser reemplazados de manera independiente. Un ajuste más fino tendrá lugar dentro de los módulos, a un nivel de detalle que no concierne a la arquitectura [9].

3.1.2. Arquitectura de Capas (*Layered Architecture*)

La Figura 3.2 muestra una versión idealizada de la arquitectura de capas que influencia el diseño del sistema de navegación y sonar *Dolphin* implementado en los robots *Terregator* y *Neptune*. [9]

- ✓ En el primer nivel (el más bajo) residen las rutinas de control.
- ✓ Los niveles 2 y 3 se encargan de manejar las entradas de los sensores, es decir perciben el mundo real.
- ✓ El nivel 4 genera y mantiene el "modelo del mundo", una conceptualización del mundo tal como la entiende el robot.
- ✓ El nivel 5 se encarga de la navegación.
- ✓ Los niveles 6 y 7 programan y planean las acciones del robot. El manejo de los problemas se realiza en el nivel 7.
- ✓ Las capas superiores se encargan de supervisar y proveer la interfaz de usuario.

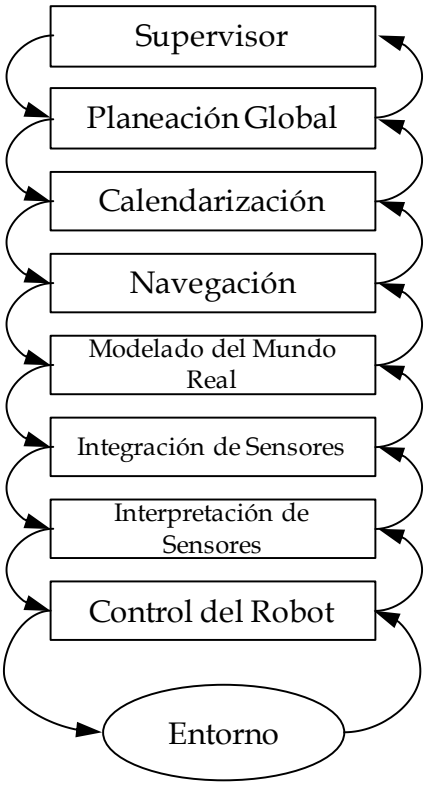


Figura 3.2 Arquitectura en capas [9].

VERIFICACIÓN DE REQUISITO 1: COMPORTAMIENTOS

Esta arquitectura deja de lado algunos de los problemas encontrados en la arquitectura Ciclo de Control (*Control Loop*), definiendo más componentes a los cuales las tareas requeridas pueden ser delegadas. Además se definen niveles de abstracción como guía en el diseño [9].

La arquitectura sugiere que los datos sean pasados entre componentes adyacentes, lo que puede impactar negativamente cuando se requiere una respuesta rápida. Otra imprecisión del modelo es que no separa las jerarquías de manera congruente por nivel de abstracción. Nótese que los módulos encargados de los sensores y el modelado del mundo están juntos (capas 2, 3 y 4); mientras que del control al control del robot y navegación (capas 1, 5 y 6) están separadas [9].

VERIFICACIÓN DE REQUISITO 2: INCERTIDUMBRE

La existencia de capas de abstracción infiere la necesidad de manejo de incertidumbre. Lo que es incierto en los niveles más bajos, puede volverse claro en las capas superiores. Por ejemplo, el contexto implícito en la representación del mundo, puede dar pistas para eliminar ambigüedad en los datos de los sensores cuando estos están en conflicto [9].

VERIFICACIÓN DE REQUISITO 3: TOLERANCIA A FALLAS Y SEGURIDAD

La tolerancia a fallas seguridad pasiva también son manejadas por el mecanismo de abstracción. Los datos y los comandos son analizados desde diferentes perspectivas, por lo que es posible incorporar muchas verificaciones y ajustes en el sistema. En cuanto a la seguridad activa y al desempeño puede requerir que los caminos de datos y comandos entre las capas se acorten agregando atajos [9].

VERIFICACIÓN DE REQUISITO 4: FLEXIBILIDAD

Las dependencias entre los módulos son un obstáculo para el fácil reemplazo de componentes, así como para agregar componentes nuevos. Las frágiles relaciones entre las capas se pueden volver más difíciles de descifrar con cada cambio [9].

3.1.3. Arquitectura de Control de Tareas e Invocación implícita (*Implicit Invocation*)

La arquitectura de control de tareas o TCA (por sus siglas en inglés *Task Control Architecture*) se basa en jerarquías de tareas, es decir, un árbol de tareas. Las tareas padre inician a las tareas hijo. El diseñador del software puede definir dependencias temporales entre pares de tareas [9].

En una arquitectura de control de tareas, las tareas se comunican enviando mensajes a un servidor central que redirecciona el mensaje a las tareas que pueden manejarla. Bajo este esquema, el remitente no necesita conocer al destinatario. Para ello, entre las características de TCA se incluyen tres mecanismos de invocación implícita: excepciones, escucha (*wiretapping*), y monitores [9].

Además, a diferencia de las arquitecturas por Capas y Ciclo de Control, TCA permite la ejecución concurrente de los agentes; pues muchas acciones pueden ejecutarse simultáneamente con mayor o menor independencia. Dicha concurrencia está limitada por las capacidades del servidor central [9].

VERIFICACIÓN DE REQUISITO 1: COMPORTAMIENTOS

El uso de árboles de tareas por una parte y excepciones, escucha y monitores por la otra, permite una separación clara entre las acciones (el comportamiento contenido en el árbol de tareas) y las reacciones (el comportamiento dictado por acciones y circunstancias externas) [9].

VERIFICACIÓN DE REQUISITO 2: INCERTIDUMBRE

Cuando se presenta incertidumbre puede generarse un árbol tentativo que pueda ser adaptado por los manejadores de excepciones. Sin embargo este manejo es poco claro [9].

VERIFICACIÓN DE REQUISITO 3: TOLERANCIA A FALLAS Y SEGURIDAD

La tolerancia a fallas se obtiene cuando muchos manejadores se suscriben para una misma señal, pues si uno de ellos deja de estar disponible, otro puede procesar las peticiones. Múltiples suscriptores también benefician el desempeño [9].

La Seguridad es manejada por los monitores y los escuchas [9].

VERIFICACIÓN DE REQUISITO 4: FLEXIBILIDAD

El uso de la invocación implícita hace del desarrollo incremental y del reemplazo de componentes una tarea inmediata: es suficiente con registrar nuevos manejadores, excepciones, escuchas o monitores en el servidor central y ningún otro componente nota el cambio [9].

3.1.4. Arquitectura *Blackboard* (*Blackboard Architecture*)

En esta arquitectura, todos los componentes se comunican mediante el componente característico de un sistema *Blackboard*: una base de datos central. Cuando un módulo requiere determinada información, la lee directamente de la base de datos si está disponible, o se bloquea en espera de que algún otro módulo la proporcione escribiéndola en la base de datos. La arquitectura *Blackboard* analizada corresponde al componente NAVLAB del sistema CODGER [9].

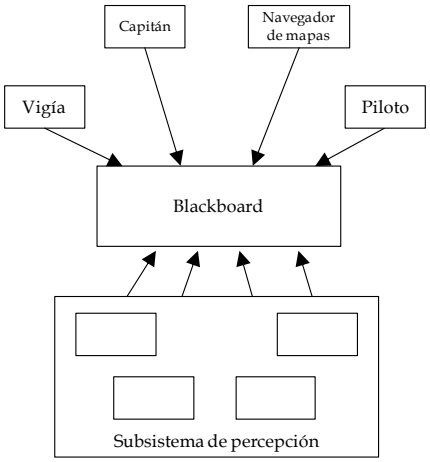


Figura 3.3. Una arquitectura *Blackboard* [9].

Los módulos del sistema CODGER son los siguientes [9]:

- ✓ Capitán. Supervisor general.
- ✓ Navegador de mapas. Planeador de trayectorias de alto nivel.
- ✓ Vigía (*lookout*). Módulo que monitorea el entorno buscando puntos de referencia.
- ✓ Piloto. Planeador de trayectorias de bajo nivel y controlador de motores.
- ✓ Subsistema de Percepción. Conjunto de módulos que leen información RAW de los sensores y les dan una interpretación coherente.

VERIFICACIÓN DE REQUISITO 1: COMPORTAMIENTOS

La comunicación se realiza mediante la base de datos compartida. Un problema de esta arquitectura es que todo el flujo de control debe encajar en el mecanismo de la base de datos, aún cuando una interacción directa entre los componentes podrías ser más natural [9].

VERIFICACIÓN DE REQUISITO 2: INCERTIDUMBRE

El *Blackboard* es también el medio para resolver conflictos e incertidumbres. El mejor ejemplo de esta actividad es en la integración de datos de los sensores, realizada por el subsistema de percepción para conciliar las entradas de los diversos sensores [9].

VERIFICACIÓN DE REQUISITO 3: TOLERANCIA A FALLAS Y SEGURIDAD

La comunicación mediante la base de datos es similar a la comunicación mediante el servidor central de mensajes del TCA. Mecanismos de manejo de excepciones, escucha y monitoreo pueden implementarse en CODGER, definiendo módulos que supervisen la base de datos en busca de situaciones problemáticas [9].

VERIFICACIÓN DE REQUISITO 4: FLEXIBILIDAD

La arquitectura *Blackboard* ofrece soporte para concurrencia y desacopla a los remitentes de los receptores de los mensajes; lo que se refleja en flexibilidad y mantenibilidad [9].

3.1.5. Análisis comparativo.

De las cuatro arquitecturas analizadas, dos (capas y *blackboard*) son muy específicas y dan información precisa de los componentes que se esperan en un robot. Las otras dos (ciclo de control y control de tareas e invocación implícita) no definen componentes funcionales y se centran en los mecanismos. Una arquitectura más específica ayuda a tener una mejor comprensión de las abstracciones y tareas involucradas en un robot autónomo [9].

Finalmente se presenta la Tabla 3.1, que resume las características de las arquitecturas presentadas [9].

	Ciclo de control	Capas	Control de tareas e invocación implícita	*Blackboard*
Coordinación de Tareas	Regular	Pobre	Muy buena	Buena
Manejo de Incertidumbre	Pobre	Regular	Regular	Buena
Tolerancia a Fallos	Regular	Regular	Muy buena	Buena
Seguridad	Regular	Regular	Muy buena	Buena
Desempeño	Regular	Regular	Muy buena	Buena
Flexibilidad	Regular	Pobre	Buena	Buena

Tabla 3.1 Comparación de Fuerzas [9].

3.2. Tecnologías de software para robots móviles

El campo de la robótica es demasiado amplio para que exista una solución o plataforma que sirva de base para todos los robots [16]. Sin embargo, existe mucho trabajo realizado en el campo de la programación de robots, enfocado principalmente al desarrollo de plataformas o *Frameworks* [17]. Algunas de estas plataformas fueron diseñadas para satisfacer necesidades particulares, mientras que otras están diseñadas para un campo más general.

La mayoría de estas plataformas, como ROS o Player/Stage, operan sobre una arquitectura cliente/servidor o *Peer-to-Peer*, buscando evitar el problema de cuellos de botella [16; 17], para no depender de un componente centralizado que pudiese fallar y detener el funcionamiento de todo el sistema. Mientras que otras plataformas como OpenRDK utilizan un *Blackboard* para la comunicación de sus módulos.

En la presente sección se describen brevemente algunas de las diferentes implementaciones de plataformas (*Frameworks* y *middlewares*) existentes para robots. Tecnologías como OROCOS y MOAST no se incluyen, pues aunque pueden adaptarse para robots móviles, están diseñadas para realizar control, y su enfoque es más aplicado a robots manipuladores.

3.2.1. CARMEN

Carnegie Mellon Robot Navigation Toolkit es un conjunto de aplicaciones de software para el control de robots móviles. Está diseñado de manera modular, e incluye aplicaciones para navegación y localización. Más que proveer una plataforma para el desarrollo de robótica móvil, está orientado a funciones de mapeo y navegación [18].

En CARMEN existe un programa central (IPC) que permite la comunicación entre los otros programas. Los programas publican la información generada, la cual es entregada a sus programas suscriptores mediante el programa central que se encarga del reparto de información [18].

CARMEN cuenta con las siguientes características [18]:

- ✓ Arquitectura: Centralizada. Cliente/Servidor con repositorio común de variables.
- ✓ Plataformas Soportadas: Linux.
- ✓ Lenguajes: C, Java.
- ✓ Intercambio de información: Paso de Mensajes y variables compartidas
- ✓ Protocolo de Comunicación: IPC.

3.2.2. Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio es un conjunto de aplicaciones que incluye una plataforma de desarrollo de aplicaciones para robots, entorno de desarrollo, herramientas de análisis y entorno de simulación. El *framework* de la aplicación se encarga del manejo de la concurrencia y la coordinación, mientras que el modelo orientado a servicios permite que los módulos de alto nivel operen de manera independiente de los módulos de bajo nivel [19].

En *MS-RDS* las aplicaciones se ejecutan como servicios (orientados a estados). La comunicación entre programas se realiza como llamadas a servicios utilizando http y un protocolo basado en *SOAP* [19].

Microsoft Robotics Developer Studio cuenta con las siguientes características [19]:

- ✓ Arquitectura: Distribuida. Basada en servicios (cliente/servidor)
- ✓ Plataformas Soportadas: Windows.
- ✓ Lenguajes: Lenguaje visual, C#.
- ✓ Intercambio de información: Paso de Mensajes.
- ✓ Protocolo de Comunicación: WCF (protocolo basado en SOAP).

3.2.3. *MIRO*

Middleware for Autonomous Mobile Robots es una plataforma o *middleware* basado en CORBA y orientado a objetos que permite manejar los sensores y actuadores de un robot móvil como objetos en un entorno distribuido. Incluye servicios que realizan funciones de sensado y monitoreo [20; 21].

MIRO cuenta con las siguientes características [21]:

- ✓ Arquitectura: Distribuida. *Peer-to-Peer.*
- ✓ Plataformas Soportadas: Linux/Unix, OS X, Windows.
- ✓ Lenguajes: C/C++, Java, Python, etc. (Cualquiera soportado por CORBA).
- ✓ Intercambio de información: Objetos remotos.
- ✓ Protocolo de Comunicación: TAO CORBA.

3.2.4. MOOS

Mission Oriented Operating Suite es un *middleware* multiplataforma escrito en C++ para investigación en robótica. Está estructurado en dos capas principales: el núcleo (*Core MOOS*) o capa de comunicaciones que provee la plataforma para la comunicación entre las aplicaciones y la capa de aplicación (*Essential MOOS*) que se ejecuta sobre la capa de comunicaciones y se encarga del manejo y control de ejecución de las aplicaciones [22].

La comunicación entre los programas en *MOOS* se realiza mediante la lectura y escritura de las variables almacenadas en la base de datos centralizada *MOOSDB,* que es el centro de *CoreMOOS*. Las comunicaciones entre el cliente y el servidor son siempre iniciadas por el cliente, y las aplicaciones cliente no necesitan saber que otras aplicaciones cliente existen [22].

MOOS cuenta con las siguientes características [22]:

- ✓ Arquitectura: *Blackboard.*
- ✓ Plataformas Soportadas: Linux, OS X.
- ✓ Lenguajes: C++.
- ✓ Intercambio de información: Variables compartidas.
- ✓ Protocolo de Comunicación: Paquetes TCP. El acceso a las variables es gestionado por el *middleware.*

3.2.5. OpenRDK

OpenRDK es un *framework* modular diseñado para facilitar el desarrollo de sistemas robóticos distribuidos, en base a las necesidades de los usuarios. Existe un modulo principal llamado *agente* que se encarga del lanzamiento de los demás módulos, que se ven como hilos de un solo proceso dentro del *framework* [23].

Los módulos se comunican entre sí utilizando un objeto tipo *Blackboard* llamado repositorio, en el cual publican algunas de sus variables internas denominadas propiedades. Las propiedades se definen al arranque y posteriormente pueden ser accesadas para lectura o escritura por cualquiera de los otros módulos. Las propiedades se accesan mediante direcciones tipo URL [23].

OpenRDK cuenta con las siguientes características [23]:

- ✓ Arquitectura: *Blackboard*.
- ✓ Plataformas Soportadas: Linux/Unix, OS X, Windows (emulación).
- ✓ Lenguajes: C/C++.
- ✓ Intercambio de información: variables compartidas.
- ✓ Protocolo de Comunicación: Paquetes TCP. El acceso a las variables es gestionado por el *Framework*.

3.2.6. Orca

Orca es un *framework* para el desarrollo de sistemas robóticos basados en componentes. Esta desarrollado sobre ICE (siglas en inglés de *Internet Communications Engine*, un *middleware* orientado a objetos para comunicación entre procesos [24]), lo que le permite trabajar de manera distribuida sobre múltiples plataformas con independencia de lenguaje [25].

La comunicación entre las aplicaciones que constituyen un software se realiza mediante invocaciones a objetos remotos; por lo que una aplicación funcionará siempre que el objeto este definido e instanciado en la aplicación remota que lo opera [25].

Orca cuenta con las siguientes características [25]:

- ✓ Arquitectura: *Peer-to-Peer*.
- ✓ Plataformas Soportadas: Linux/Unix, OS X , Windows.
- ✓ Lenguajes: C++, C#, Java, PHP, Python, Ruby (Cualquiera soportado por ICE)
- ✓ Intercambio de información: Objetos remotos.
- ✓ Protocolo de Comunicación: Slice (protocolo de comunicaciones de ICE: *Internet Communications Engine*).

3.2.7. Player/Stage

Player/Stage son dos aplicaciones separadas para la ejecución de aplicaciones para robot tanto en entornos reales como simulados. *Player* es un servidor de dispositivos que provee una interfaz flexible para una amplia gama de sensores y actuadores. *Stage* es un simulador de múltiples robots móviles en un entorno bidimensional. Posteriormente se incluyó *Gazebo* que ofrece las mismas características que *Stage* pero la simulación es realizada en tres dimensiones [26].

En *Player/Stage* los dispositivos son servicios accesibles mediante un proxy, en el cual se leen y escriben los valores para los sensores y actuadores. La comunicación se realiza punto a punto mediante paquetes TCP por lo que cada cliente debe conocer los servidores a los que se conecta. Sin embargo existen librerías de cliente que manejan las comunicaciones para proveer una interfaz más clara [26].

Player/Stage cuenta con las siguientes características [26]:

- ✓ Arquitectura: *Peer-to-Peer*.
- ✓ Plataformas Soportadas: Linux/Unix, OS X, Windows.
- ✓ Lenguajes: C/C++, Java, Python (Servidor).
- ✓ Intercambio de información: Paso de mensajes. Acceso a dispositivos mediante interfaces "*Proxy Class*".
- ✓ Protocolo de Comunicación: Mensaje Player (TCP).

3.2.8. SPQR-RDK

Software Per Qualunque Robot – Robot Development Kit se desarrolla para robots móviles (principalmente para robots jugadores de soccer) enfocándose principalmente en los aspectos de eficiencia, modularidad y reusabilidad; permitiendo inspección remota, interfaces comunes de hardware, intercambio de información [27].

En *SQPR-RDK*, el paso de información entre los módulos se realiza mediante variables compartidas. Las variables están asociadas a un solo módulo el cual sólo tiene permisos de escritura sobre la variable, mientras que los demás módulos tienen permiso de lectura sobre la misma [27].

SPQR-RDK cuenta con las siguientes características [27]:

- ✓ Arquitectura: *Peer-to-Peer*.
- ✓ Plataformas Soportadas: Linux, OS X.
- ✓ Lenguajes: C/C++.
- ✓ Intercambio de información: Variables compartidas.
- ✓ Protocolo de Comunicación: Mensaje *SPQR-RDK* (TCP).

3.2.9. ROS

Robot Operating System a pesar de su nombre, no es un sistema operativo, sino una plataforma distribuida diseñada para resolver un conjunto específico de problemas que se presentan durante el desarrollo de robots de servicio de gran escala [16].

En ROS, los módulos se comunican entre sí de manera directa a través de objetos remotos que se definen en sencillos *IDL's*, llevándose el flujo de información en canales [16].

ROS cuenta con las siguientes características [16]:

- ✓ Arquitectura: *Peer-to-Peer*.
- ✓ Plataformas Soportadas: Linux/Unix, OS X, Windows.
- ✓ Lenguajes: C/C++, LISP, Octave, Python.
- ✓ Intercambio de información: Objetos remotos.
- ✓ Protocolo de Comunicación: XML/RPC utilizando *IDL's*.

3.2.10. Otras Tecnologías

Existen otras tecnologías diseñadas para facilitar la comunicación entre aplicaciones, como CORBA, Java-RMI y SOAP. Si bien dichas tecnologías no están diseñadas para su uso en robots, pueden ser utilizadas o adaptadas para dar soporte a la implementación de alguna arquitectura [17], como se observa en el caso de MIRO (CORBA) o *Microsoft Robotics Developer Studio* (WCF/SOAP).

Java-RMI es poco utilizada, debido a que no permite de manera directa interoperabilidad con otros lenguajes. CORBA permite interoperabilidad, pero el manejo de los ORBs puede ser complejo, y muchas de sus funciones orientadas a seguridad suelen ser innecesarias en el contexto de la robótica. En el caso de SOAP, es un protocolo mucho más extendido y por ende, ofrece mayores posibilidades. Sin embargo, su desempeño es cuestionable, debido a que maneja XML como medio para transferir la información, y el tamaño de las cadenas de texto es varios órdenes de magnitud mayor que transferencia de datos binarios [17].

3.2.11. Resumen

La siguiente Tabla 3.2 presenta de manera resumida las características de las tecnologías existentes mencionadas anteriormente:

	Arquitectura	Plataforma	Lenguajes	Intercambio de información	Protocolo de Comunicación
CARMEN	Centralizada Cliente/Servidor	Linux	C/C++, C#, Java, PHP, Python, Ruby	Paso de mensajes Variable Compartida	IPC
MS RDS	Distribuida Cliente/Servidor	Windows	C#, Lenguaje Visual	Paso de Mensajes	WCF
MIRO	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C/C++, Java, Python (Soporte de CORBA)	Objetos remotos	TAO CORBA
MOOS	Centralizada *Blackboard*	Linux, OS X	C++	Variable compartida	Mensaje MOOS
OpenRDK	Centralizada *Blackboard*	Linux, OS X	C/C++	Variable compartida	Mensaje OpenRDK
Orca	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C++, C#, Java, PHP, Python, Ruby (Soporte de ICE)	Objetos remotos	ICE
Player / Stage	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C/C++, C#, Java, PHP, Python, Ruby	Paso de mensajes	Mensaje Player
SPQR-RDK	Distribuida Cliente/Servidor	Linux, OS X	C/C++	Variable compartida	Mensaje SPQR-RDK
ROS	Distribuida *Peer-to-Peer*	Linux, OS X, Windows	C/C++, C#, Java, PHP, Python, Ruby	Objetos remotos	XML/RPC

Tabla 3.2 Resumen de Tecnologías.

4. ANÁLISIS DE EXTENSIBILIDAD Y REESTRUCTURACIÓN

"Vale la pena ser obvio, especialmente si eres famoso por tu sutileza."

Salvor Hardín.
Fundación. Isaac Asimov.

En este capítulo se realiza el análisis de las características de extensibilidad y reestructuración aplicable al tiempo de desarrollo de la arquitectura del software de un robot móvil, acorde a las definiciones presentadas en la sección 1.3. El análisis de cada uno de los aspectos se realiza mediante una aproximación de complejidad algorítmica de las actualizaciones mínimas necesarias al sistema de software, dado el cambio de un componente, es decir, los aspectos se detalla como una secuencia de pasos finita y ordenada, analizándose posteriormente su complejidad. Este análisis se realiza por separado tanto para una arquitectura *Peer-to-Peer* como para la arquitectura basada en *Blackboard*, descrita en la sección 2.3.2.

Para este propósito, se considera que cada arquitectura está compuesta por módulos, cada uno de los cuales se subdivide en al menos dos o tres capas, dependiendo de si el módulo interactúa con hardware o no, tal como se presenta en la Figura 4.1.

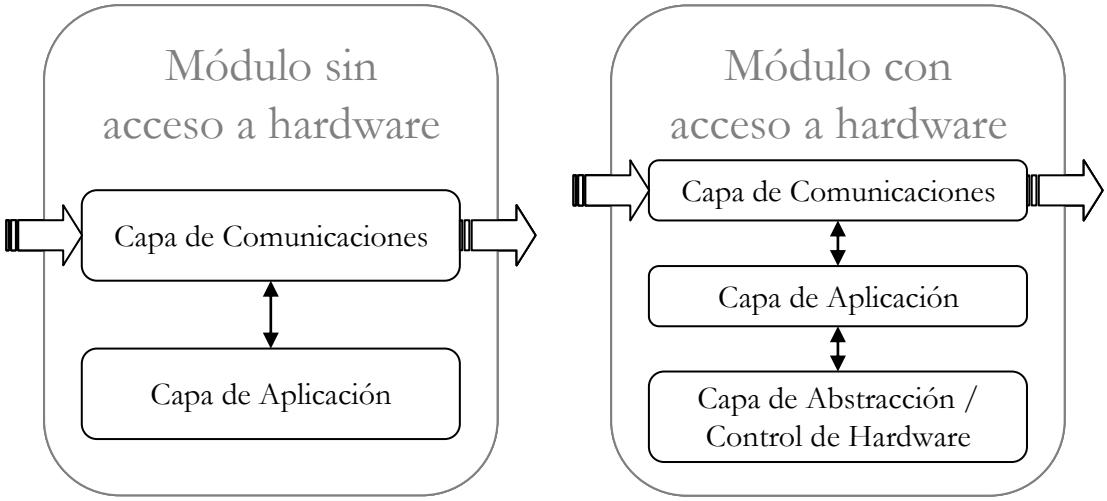


Figura 4.1. Estructura básica de un módulo.

Es común que los módulos cuenten con interfaz gráfica, tengan que realizar múltiples tareas de manera concurrente (por ejemplo procesar los datos, desplegar información en la interfaz gráfica y escuchar el canal de comunicación). Dichos elementos pueden agregarse como parte de un módulo sin pérdida de generalidad si el módulo así lo requiriese, por lo que puede describirse la estructura de cada módulo tal como se ilustra en la Figura 4.2.

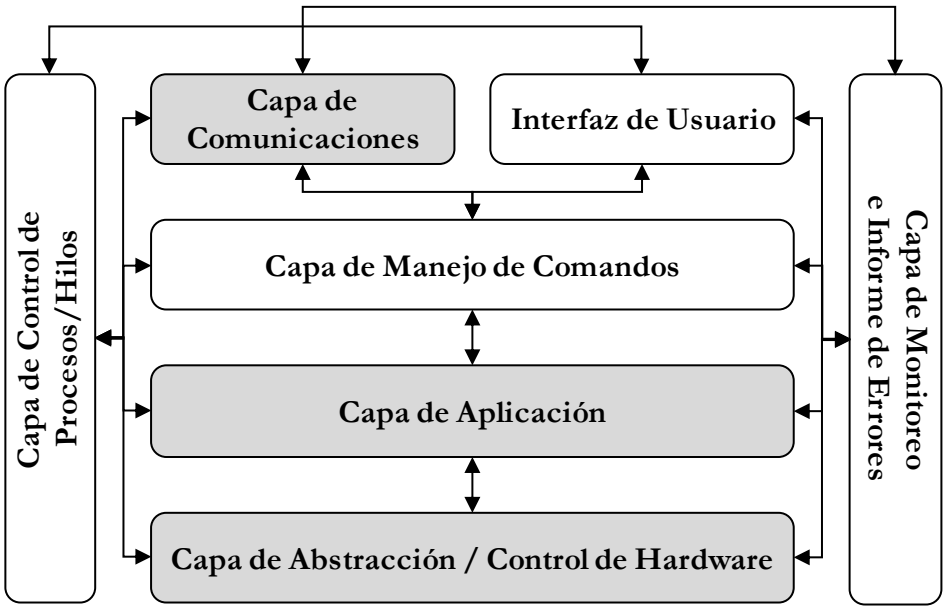


Figura 4.2. Propuesta de la estructura general de un módulo.

- ✓ Capa de aplicación: En esta capa se realizan las funciones propias del módulo.
- ✓ Capa de comunicaciones: Esta capa gestiona las comunicaciones con otros módulos en el sistema.
- ✓ Capa de manejo de comandos: Esta capa funge como intérprete entre la capa de comunicaciones e interfaz de usuario (si la hay) y la capa de aplicación, además de controlar el acceso a las funciones que realiza la capa de aplicación.
- ✓ Capa de Abstracción / Control de Hardware (capa opcional): Esta capa gestiona las comunicaciones con los dispositivos de hardware asociados al módulo, tanto para adquisición de datos como para control del mismo.
- ✓ Capa de Control de Procesos / Hilos (capa opcional): Esta capa se encarga de la gestión de los hilos o procesos que permiten la ejecución concurrente o en paralelo de las múltiples tareas que realiza el módulo, los elementos de sincronización y el control de acceso a los recursos compartidos dentro del mismo módulo.
- ✓ Capa de Monitoreo e Informe de Errores (capa opcional): Esta capa monitorea la ejecución de las otras capas, llevando a cabo acciones preventivas o correctivas cuando se presentan errores.
- ✓ Interfaz de Usuario (capa opcional): Esta capa se encarga de gestionar la GUI (Interfaz Gráfica de Usuario).

Si bien la Capa de Manejo de Comandos que aparece en la Figura 4.2 no está marcada como opcional, aunque no aparece en la Figura 4.1, se considera a esta capa un elemento fundamental. Aún cuando sus funciones estén integradas en la Capa de Comunicaciones o en la Capa de Aplicación su responsabilidad es adecuar los datos recibidos (por ejemplo des-serializando un objeto remoto o analizando el contenido de un mensaje) para su uso en la aplicación.

Es necesario recalcar que la estructura de módulo representada por la Figura 4.2 es una propuesta diseñada para facilitar el análisis del impacto de la arquitectura de un sistema de software en las características de cada uno de sus componentes, y que supone el diseño de software planeado para ser cambiable y fiable.

Bajo esta estructura, se observa que la influencia de la arquitectura utilizada para comunicar los módulos permea hasta la capa de manejo de comandos, pues la capa de aplicación solicita los datos (o ejecuciones si es el caso) a los demás módulos mediante estas capas, no importando el método que se utilice para adquirir dicha información.

Por otro lado, se supone que el sistema cuenta con una plataforma (middleware o *Framework*) que gestiona todas las comunicaciones y delega el manejo de los errores o excepciones a la capa inmediata superior. Dicha suposición es válida si se toma en cuenta la complejidad de desarrollo de las aplicaciones que operan a un robot móvil de gama media. Es natural pensar que se dedique el tiempo a los algoritmos de adquisición de datos, procesamiento de información, toma de decisiones y control de actuadores; y no a las comunicaciones.

4.1. Descripción del análisis

A lo largo del presente capítulo, el análisis de las características de extensibilidad y reestructuración de cada arquitectura de software para un robot móvil, se basa en estimar el costo de actualizar el sistema para que éste permanezca operacional, dado un cambio en el mismo. Esto es, en un sistema que está formado por múltiples elementos interdependientes, cuando uno de los elementos del sistema es modificado, los elementos que dependen del elemento modificado deberán actualizarse para que el sistema siga operando correctamente. Realizar esta actualización en las dependencias del elemento modificado tiene un costo proporcional al número de elementos que deben ser actualizados.

Este costo de actualización del sistema dado un cambio es un valor subjetivo asociable al tiempo y esfuerzo que un grupo de programadores o expertos deben dedicar a la tarea de actualización del sistema, lo cual depende en gran medida de la experiencia y habilidad del grupo, así como de su número de integrantes; lo cual es un factor no cuantizable.

No obstante, cuanto menor sea el número de cambios a realizar, menos tiempo y esfuerzo deberá dedicar el grupo de programadores o expertos en actualizar el sistema. Dado que la cantidad de actualizaciones a realizar es un valor medible, se tomará esto como métrica para estimar características de extensibilidad y reestructuración de cada arquitectura.

Es necesario aclarar que se asume que si un componente es modificado, esta modificación s realiza por ser necesaria e independientemente de la arquitectura elegida, por lo que no se contabiliza. Sin embargo, el número de actualizaciones a realizar sobre las dependencias de componente modificado en el resto del sistema sí dependen de la arquitectura, por lo que son éstas las únicas que se toman en cuenta.

4.1.1. Extensibilidad

> *"La extensibilidad se enfoca en la extensión de un sistema de software con nuevas características, así como el reemplazo de componentes con versiones mejoradas y la supresión de características y componentes innecesarios o no deseados…" [1]*

Para analizar la extensibilidad se plantean los siguientes escenarios de prueba:

- ✓ Cambio de un módulo por otro que realiza funciones equivalentes. El nuevo módulo se ejecuta en el mismo contexto que el módulo al que reemplaza. En caso contrario, se supone que ya se ha llevado a cabo un procedimiento de reestructuración.
- ✓ Cambio de la definición de las funciones de un módulo.
- ✓ Agregar funciones nuevas a un módulo existente.
- ✓ Eliminar funciones de un módulo existente.

INTERCAMBIO DE MÓDULOS EQUIVALENTES

En el caso de un cambio de módulo por otro de funciones equivalentes, se considera trivial cuando las funciones de ambos módulos tienen las mismas firmas (reciben exactamente los mismos argumentos y devuelven exactamente el mismo tipo de datos) y tienen tiempos de respuesta semejantes, pues los demás módulos no notarán la diferencia. Si por el contrario, hubiera variaciones importantes en los tiempos de respuesta o las firmas de algunas funciones cambiaran, hay que hacer modificaciones en los módulos que tengan una relación de dependencia con el módulo reemplazado. Este segundo caso es el que se analiza en las secciones posteriores.

CAMBIO DE LA DEFINICIÓN DE LAS FUNCIONES DE UN MÓDULO

Cuando se cambia la definición de las funciones de un módulo, se considera trivial cuando la nueva función tiene la misma firma (recibe exactamente los mismos argumentos y devuelve exactamente el mismo tipo de datos) y tiene un tiempo de respuesta semejante, ya que los demás módulos no notarán la diferencia. Si por el contrario, hubiera variaciones importantes en el tiempo de respuesta o la firma de la función sufriera cambios, hay que hacer modificaciones en los módulos que utilicen dicha función. Este caso es el que se analiza en las secciones posteriores.

AGREGACIÓN DE FUNCIONES

La agregación de funciones presenta dos escenarios: Agregación de funciones no utilizadas y agregación de funciones que ofrecen nuevas características al sistema. Estos escenarios para agregación de funciones no son analizados, ya que agregar funciones no utilizadas carece de trascendencia al no tener efecto en los otros módulos; y agregar funciones que provean nuevas características por lo general conlleva una fase de codiseño que queda fuera del marco de estudio.

ELIMINACIÓN DE FUNCIONES

El proceso de eliminar funciones presenta dos escenarios: eliminar funciones no utilizadas y eliminar funciones para eliminar características no deseadas. Estos escenarios para la eliminación de funciones no son analizados, ya que eliminar funciones no utilizadas carece de trascendencia al no tener efecto en los otros módulos; y eliminar funciones para suprimir características no deseadas por lo general conlleva una fase de codiseño que queda fuera del marco de estudio.

4.1.2. Reestructuración

> *La reestructuración se ocupa de la organización de los componentes de un sistema de software y de las relaciones entre ellos, por ejemplo cuando se cambia el lugar de un componente moviéndolo a un subsistema diferente… [1]*

Para probar la reestructuración se plantean los siguientes escenarios, los cuales contemplan tanto la relocalización de funciones individuales como de módulos completos:

- ✓ Cambio de ubicación de una función (relocalización de funciones).
- ✓ Separar una función en dos o más funciones que realicen, en suma, las mismas funciones que la función original (especialización de funciones).
- ✓ Unir dos o más funciones en una sola función que realice las mismas funciones que realizaban en conjunto las funciones originales (generalización de funciones).
- ✓ Cambio de ubicación o contexto de un módulo (relocalización de módulos).
- ✓ Separar un módulo en dos o más sub-módulos que realicen, en suma, las mismas funciones que el módulo original. Los sub-módulos pueden ejecutarse o no en el mismo contexto que el módulo al que reemplazan.
- ✓ Unir dos o más módulos en un nuevo súper-módulo que realice las mismas funciones que realizaban en conjunto los módulos originales. El súper-módulo puede ejecutarse o no en el mismo contexto que los módulos que reemplaza.

REESTRUCTURACIÓN FUNCIONAL

En ocasiones es necesario relocalizar una función, por ejemplo cuando se especializa un módulo o cuando la función está relacionada con un recurso que cambia de lugar. En estos casos se analiza el impacto que genera dicha relocalización funcional en los otros componentes del sistema (módulos), puesto que el tiempo invertido en la reprogramación (de ser necesaria) del cuerpo de la función por un cambio de lenguaje, contexto, etc., es independiente de la arquitectura empleada para el sistema. Así mismo se supone que ni la firma de la función ni su tiempo de ejecución sufren cambios a causa de la relocalización, puesto que recaen en un problema de extensibilidad (cambio de la definición de una función).

La misma suposición aplica para la separación (especialización, véase Figura 4.3) de una función en varias funciones especializadas y la unión (generalización, véase Figura 4.4) de varias funciones en una función generalizada: la firma de las funciones no sufre alteraciones y los tiempos de ejecución son similares. En caso contrario, al igual que con una relocalización, se grava el costo de una extensión (cambio de la definición de funciones existentes).

En el caso de la especialización de una función se contempla sólo el escenario en el cual las nuevas funciones especializadas que reemplazan a la función original se encuentran en el mismo módulo (véase Figura 4.3). En el caso de que algunas funcionalidades requirieran ejecutarse en otros módulos, se puede proceder con una relocalización de las funciones asociadas.

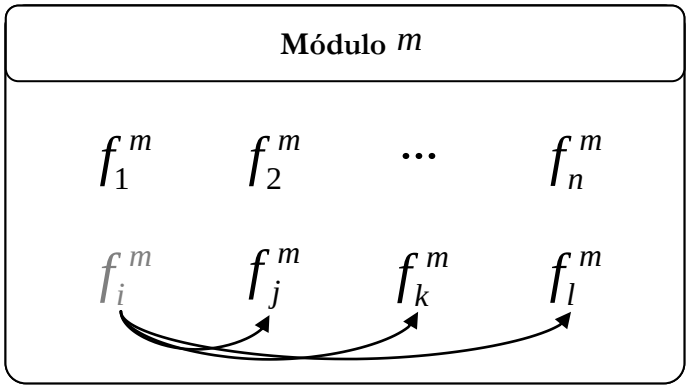


Figura 4.3. Especialización de funciones.
La función f_i^m se especializa y es reemplazada por las funciones f_j^m, f_k^m y f_l^m dentro del módulo m.

De manera similar, en el caso de la generalización de funciones se contempla sólo el escenario en el cual las funciones a unir se encuentran todas en el mismo módulo, y que la nueva función generalizada que las reemplaza se encuentra también en ese módulo (véase Figura 4.4). En el caso de que algunas de las funciones que se quieren integrar se encuentren en diferentes módulos, tiene que procederse previamente con una relocalización de dichas funciones al módulo en el cual se establecerá la nueva función generalizada.

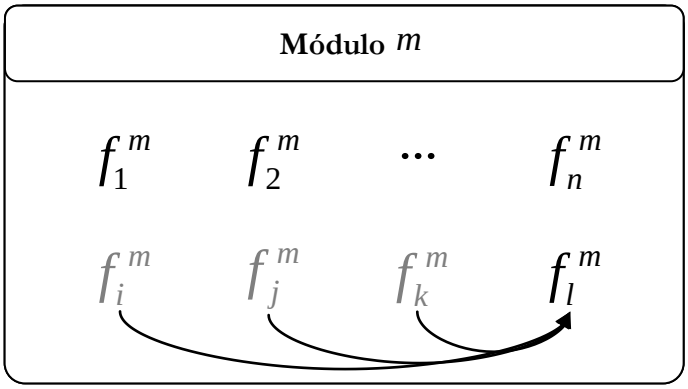


Figura 4.4. Generalización de funciones.
Las funciones f_i^m, f_j^m y f_k^m de módulo m se generalizan y son reemplazadas por las función f_l^m.

En ambos casos (especialización y generalización) se considera que si las funciones involucradas operan sobre un recurso en común, el lenguaje en el que están implementadas provee los mecanismos necesarios para gestionar el acceso a éste recurso, quedando fuera del marco de estudio los cambios que deban de realizarse en el cuerpo de dichas funciones para garantizar un uso adecuado del recurso compartido.

REESTRUCTURACIÓN MODULAR

En ocasiones es necesario relocalizar un módulo, como cuando se necesita realizar un balance de carga, incompatibilidad del hardware operado, recursos insuficientes, reemplazo de equipo, etc. Si dicha relocalización conlleva un cambio brusco del contexto, como por ejemplo un sistema operativo, plataforma o lenguaje diferente, en ocasiones es aconsejable reprogramar el módulo en lugar de adaptarlo.

El análisis realizado para la relocalización de un módulo supone que los contextos son similares y que el módulo simple y sencillamente es "movido" a otra ubicación igualmente accesible que la anterior y con aproximadamente las mismas características, analizándose solamente el impacto que tiene la relocalización en los módulos con los que exista una dependencia directa.

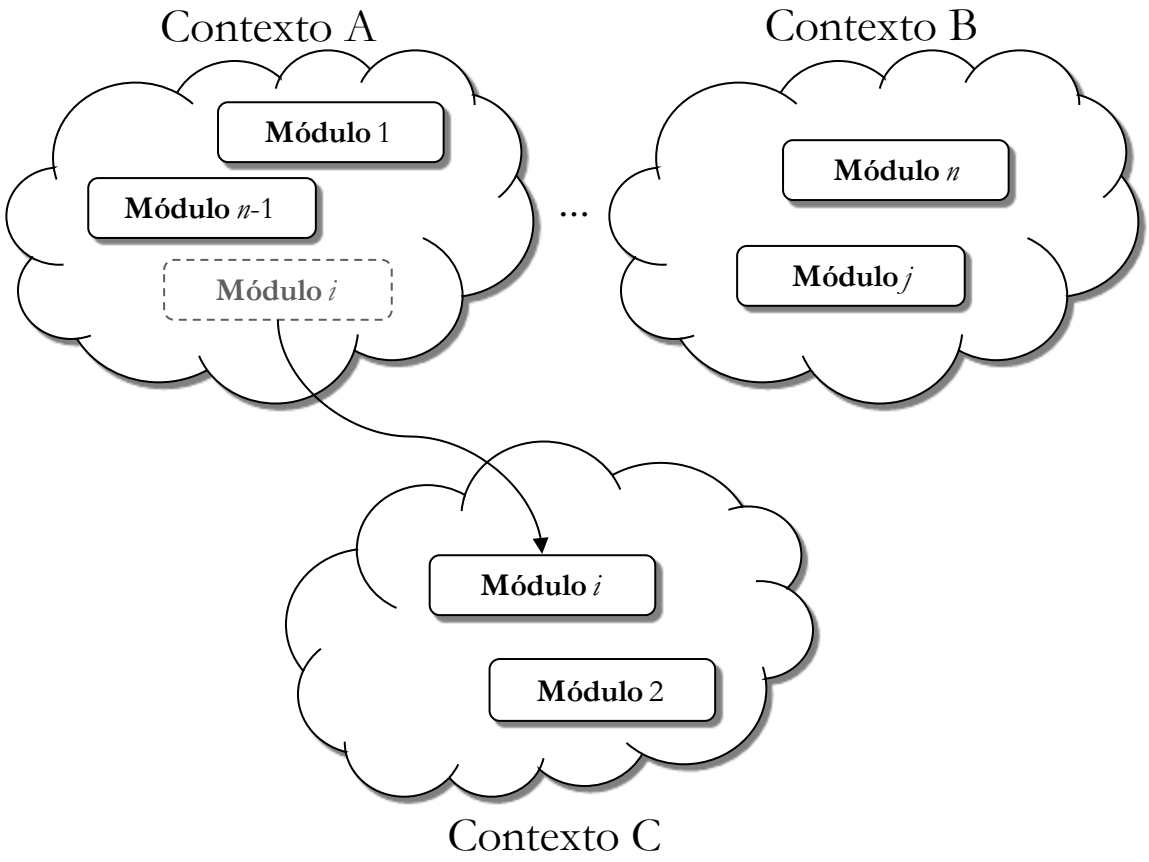


Figura 4.5. Relocalización de un módulo. El módulo *i* se relocaliza moviéndose del Contexto A al Contexto C.

Si se diera el caso de que la relocalización lleva al módulo a un contexto diferente, ocasionando cambios en el tipo de funciones que realiza o tiempos de respuesta diferentes al módulo original, al costo de la relocalización debe, además, gravársele el costo de una extensión (reemplazo de un módulo por otro equivalente).

La misma suposición aplica para la separación (especialización) de un módulo en varios sub-módulos y la unión (generalización) de varios módulos en un súper-módulo: la firma de las funciones no sufre alteraciones y los tiempos de ejecución son similares. En caso contrario, al igual que con una relocalización, se grava el costo de una extensión (cambio de la definición de funciones existentes).

En el caso de una especialización de módulos, se analizan dos escenarios:

- ✓ El módulo se divide en un conjunto de sub-módulos con una organización jerárquica.
- ✓ El módulo se divide en múltiples sub-módulos independientes.

De manera similar cuando se requiere una generalización de módulos existen dos escenarios equivalentes:

- ✓ Un conjunto de sub-módulos con una organización jerárquica se integran para formar un nuevo supermódulo.
- ✓ Varios módulos independientes que se integran para formar un nuevo supermódulo.

La integración de varios módulos independientes que no comparten recursos debe ser analizada cuidadosamente; pues reduce la modularidad generando módulos complejos y difíciles de mantener. Esta acción sólo debe llevarse a cabo cuando los módulos involucrados tienen una relación muy estrecha (granularidad fina) y su alto índice de comunicación afecta el desempeño del sistema.

4.2. Análisis en arquitectura Peer-to-Peer

En una arquitectura *Peer-to-Peer*, se considera que todos los módulos entre los que exista una relación de dependencia (m_i depende de m_j o m_j depende de m_i o ambos) están conectados mediante un enlace directo bidireccional. Así, si un módulo requiere una función provista por otro módulo, existe una conexión directa entre ambos módulos que les permite intercambiar datos. Expresado en otros términos, si existe $f_k^{m_i}(\bar{x})$ tal que $f_k^{m_i}(\bar{x}) = g(\overline{x_1}) \circ f_r^{m_j}(\overline{x_2}) \Longrightarrow \exists c_{m_i,m_j}$ donde $\overline{x_1} \cup \overline{x_2} \subseteq \bar{x}$.

Tal como se menciona en la sección 2.5 y para facilitar el análisis se considera que toda comunicación entre módulos está formada por una petición y una respuesta, por lo que si el módulo m_j requiere la ejecución de la función $f_k^{m_i}$ del módulo m_i, m_j envía primero una solicitud de ejecución de $f_k^{m_i}$ al módulo m_i junto con todos los datos necesarios para la ejecución de $f_k^{m_i}$, tras lo cual queda en espera de que el módulo m_i le envíe una respuesta con los resultados de la ejecución de $f_k^{m_i}$.

Se asume que un módulo m_i envía siempre una respuesta para todas las peticiones de ejecución recibidas (aún cuando esta respuesta indique que la ejecución no pudo llevarse a cabo). Se asume también que un módulo m_j espera un tiempo finito por una respuesta por parte de algún módulo m_i al que ha solicitado la ejecución de la función $f_k^{m_i}$ (cada función puede tener un tiempo máximo de respuesta diferente); y que de no llegar dicha respuesta dentro del tiempo máximo de espera, reacciona como si hubiera recibido una respuesta de fallo de ejecución.

4.2.1. Extensibilidad

En este apartado se analiza el costo de realizar los cambios descritos en la sección 4.1 al software de un robot en arquitectura *Peer-to-Peer*, en lo concerniente a extensibilidad. Es decir, se analiza el impacto a nivel sistema del intercambio de módulos equivalentes, y del cambio de la definición de las funciones de un módulo. El análisis mencionado refleja el costo o complejidad que tiene la actualización de los demás componentes del sistema cuando se presentan los cambios descritos.

INTERCAMBIO DE MÓDULOS EQUIVALENTES

Este escenario contempla el cambio de un módulo m_1 por otro módulo m_2 que realiza las mismas funciones y que se ejecuta en el mismo contexto del módulo original m_1. Formalmente:

Sean m_1, m_2 dos módulos de un sistema. m_1 y m_2 son intercambiables sí y sólo si:

$$\forall f_i^{m_1}(\bar{x}), \exists\, f_j^{m_2}(\bar{x}) \mid f_i^{m_1}(\bar{x}) = f_j^{m_2}(\bar{x})$$

Aún cuando los módulos son intercambiables quedan dos aspectos a considerar:

1. Los nombres de las funciones, y su firma (tipo de dato devuelto, y el tipo y orden de los parámetros que recibe).
2. Tiempo de ejecución de la función.

Cuando para todas las funciones, el nombre de la función y su firma coinciden, y sus tiempos de ejecución cumplen $t(f_j^{m_2}) \lesssim t(f_i^{m_1})$, el caso es trivial, pues los módulos son completamente compatibles y no hay que hacer ningún cambio en el sistema.

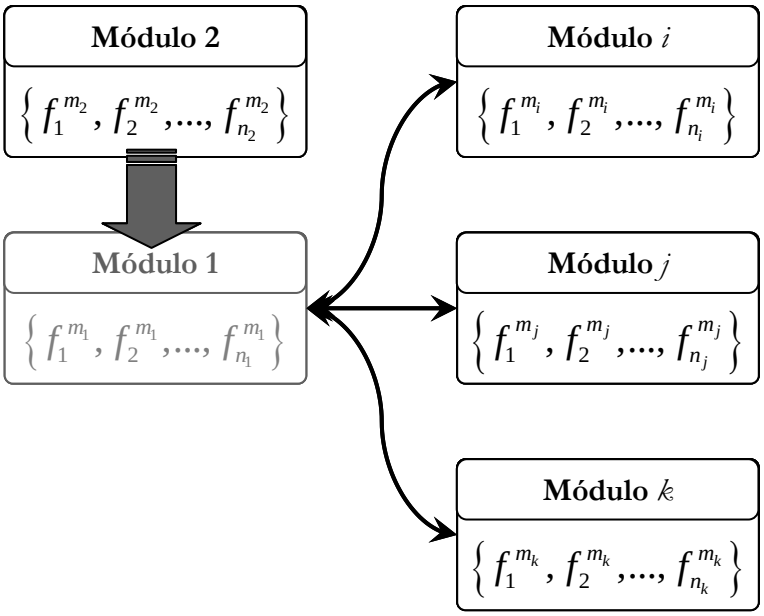


Figura 4.6. Intercambio de módulos equivalentes en *Peer-to-Peer*: Caso trivial.
Luego de reemplazar el módulo 1 por el módulo 2, ninguna actualización en el sistema es requerida.

Sin embargo, cuando existe discrepancia en los nombres de las funciones o en su firma, es necesario realizar ajustes. Se propone el algoritmo siguiente:

```
Para cada función f_i^{m_1} ∈ m_1
     Para cada módulo m_j ∈ D(f_i^{m_1})
          Para cada llamada a f_i^{m_1} en m_j
               Cambiar nombre y parámetros en la llamada a
               f_i^{m_1} por su equivalente para llamar a f_k^{m_2}
          Fin para cada
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|\{f_j^{m_1}\}\right|\right)\left(\left|D\left(f_i^{m_1}\right)\right|\right)\left(\left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$ lo cual da la idea de una complejidad de orden cúbico; para calcular la complejidad se toma:

$$n = min\left(\left|\{f_j^{m_1}\}\right|, \left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

$$N = max\left(\left|\{f_j^{m_1}\}\right|, \left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

Finalmente, para el intercambio de módulos equivalentes cuando existe discrepancia en los nombres de las funciones o en su firma se tienen las cotas $\Omega(n^3)$ y $O(N^3)$.

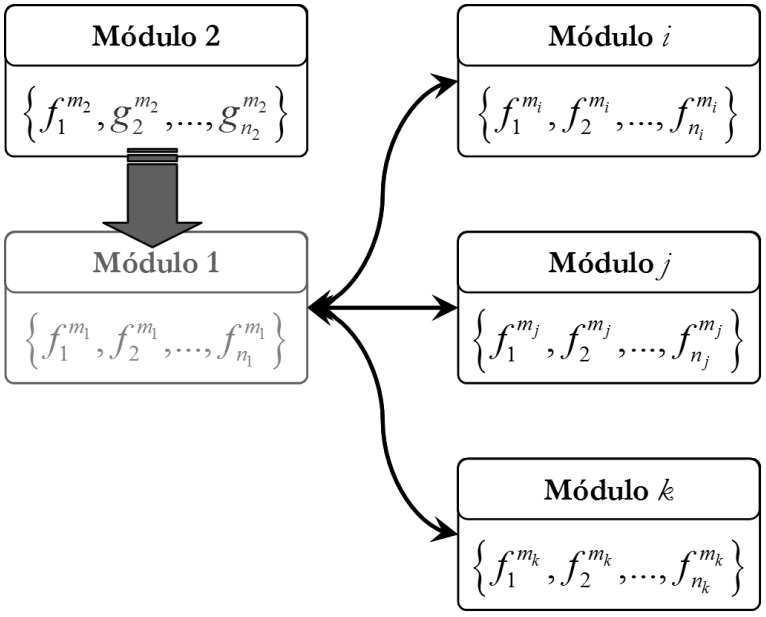


Figura 4.7. Intercambio de módulos equivalentes en *Peer-to-Peer*: Actualización requerida. Luego de reemplazar el módulo 1 por el módulo 2, se requiere actualizar los módulos *i*, *j*, y *k*.

Si el cambio se diera en el tiempo de ejecución de alguna de las funciones, es necesario actualizar los valores solamente si el nuevo tiempo de ejecución es mayor o si es mucho menor y se desea aumentar el rendimiento. Para este aspecto, se considera que cada módulo dependiente de una función realiza la verificación una sola vez, independientemente de cuantas veces se llame a la función remota. Por lo que sólo un cambio por módulo por función es necesario. Se propone el siguiente algoritmo:

```
Para cada función f_i^{m_1} ∈ m_1
     Para cada módulo m_j ∈ D(f_i^{m_1})
          Actualizar tiempo límite de ejecución para
          la llamada a f_i^{m_1} por el límite necesario para f_k^{m_2}
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|\{f_j^{m_1}\}\right|\right)\left(\left|D\left(f_i^{m_1}\right)\right|\right)$ lo cual da la idea de una complejidad de orden cuadrático; para calcular la complejidad se toma:

$$n = min\left(\left|\{f_j^{m_1}\}\right|, \left|D\left(f_i^{m_1}\right)\right|\right)$$

$$N = max\left(\left|\{f_j^{m_1}\}\right|, \left|D\left(f_i^{m_1}\right)\right|\right)$$

Finalmente, para el intercambio de módulos equivalentes cuando ha cambiado el tiempo de ejecución de las funciones se tienen las cotas $\Omega(n^2)$ y $O(N^2)$. Debido a que la complejidad de este cambio es menor que la complejidad de un cambio en el nombre o en la firma de una función, las cotas de orden cúbico $\Omega(n^3)$ y $O(N^3)$ se conservan.

CAMBIO DE LA DEFINICIÓN DE LAS FUNCIONES DE UN MÓDULO

Este escenario contempla el cambio de una función $f_i^{m_1}$ por otra función equivalente $g_i^{m_1}$ que realiza funciones semejantes y que se ejecuta en el mismo módulo m. Formalmente:

Sean f_i^m, g_i^m dos funciones de un módulo m. La función g_i^m puede reemplazar a f_i^m sí y sólo si:

$$f_i^m(\bar{x}) = g_i^m\ (\bar{y})$$

Hay dos aspectos a considerar:

1. La firma de las funciones (tipo de dato devuelto, y el tipo y orden de los parámetros que recibe).
2. Tiempo de ejecución de la función.

Cuando la firma de la función nueva coincide con la firma anterior, y sus tiempos de ejecución cumplen $t(g_i^m) \lesssim t(f_i^m)$, el caso es trivial, pues las funciones son completamente compatibles y no hay que hacer ningún cambio en el sistema.

Cuando existe discrepancia en la firma de las funciones, es necesario realizar ajustes. Se propone el algoritmo siguiente:

```
Para cada módulo m_j ∈ D(f_i^{m_1})
     Para cada llamada a f_i^{m_1} en m_j
          Cambiar nombre y parámetros en la llamada a
          f_i^{m_1} por su equivalente para llamar a g_i^{m_1}
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|D\left(f_i^{m_1}\right)\right|\right)\left(\left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$, lo cual da la idea de una complejidad de orden cuadrático. Para calcular la complejidad se toma:

$$n = min\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

$$N = max\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

Finalmente, para la actualización del sistema dado una discrepancia en la firma de una función se tienen las cotas $\Omega(n^2)$ y $O(N^2)$

Si el cambio se diera en el tiempo de ejecución de la función, es necesario actualizar los valores solamente si el nuevo tiempo de ejecución es mayor, o si es mucho menor y se desea aumentar el rendimiento. Para este aspecto se considera que cada módulo dependiente de una función realiza la verificación una sola vez, independientemente de cuantas veces se llame a la función remota, por lo que sólo un cambio por módulo por función es necesario. Se propone el siguiente algoritmo:

```
Para cada módulo m_j ∈ D(f_i^{m_1})
     Actualizar tiempo límite de ejecución para
     la llamada a f_i^{m_1} por el límite necesario para g_i^{m_1}
Fin para cada
```

La complejidad de este algoritmo es proporcional al número de dependencias de la función, es decir $O\left(\left|D\left(f_i^{m_1}\right)\right|\right)$, un tiempo de actualización lineal. Debido a que la complejidad de este cambio es menor que la complejidad de un cambio en la firma de una función, las cotas de orden cuadrático se conservan.

4.2.2. Reestructuración

En este apartado se analiza el costo de realizar los cambios descritos en la sección 4.1 al software de un robot en arquitectura *Peer-to-Peer*, en lo concerniente a reestructuración, es decir, se analiza el impacto a nivel sistema de realizar alguno de los siguientes cambios:

- ✓ Relocalización de una función
- ✓ Especialización de una función
- ✓ Generalización de una función

- ✓ Relocalización de un módulo
- ✓ Especialización de un módulo
- ✓ Generalización de un módulo

Tanto para la especialización como para la generalización modular, se realiza el análisis tanto para estructuras jerárquicas como para módulos independientes con y sin recurso compartido. El análisis mencionado refleja el costo o complejidad que tendría actualizar los demás componentes del sistema cuando se presentan los cambios descritos.

RELOCALIZACIÓN DE UNA FUNCIÓN

Este cambio contempla la relocalización de una función $f_i^{m_1}$ del módulo m_1 al módulo m_2, analizándose el impacto en el sistema de mover $f_i^{m_1}$ de un módulo a otro.

Formalmente:

$$f_i^{m_1} \xrightarrow{\text{move}} f_j^{m_2}$$

donde:

$$f_i^{m_1}(\bar{x}) = f_j^{m_2}(\bar{x})$$

Ejemplificando en la Figura 4.8:

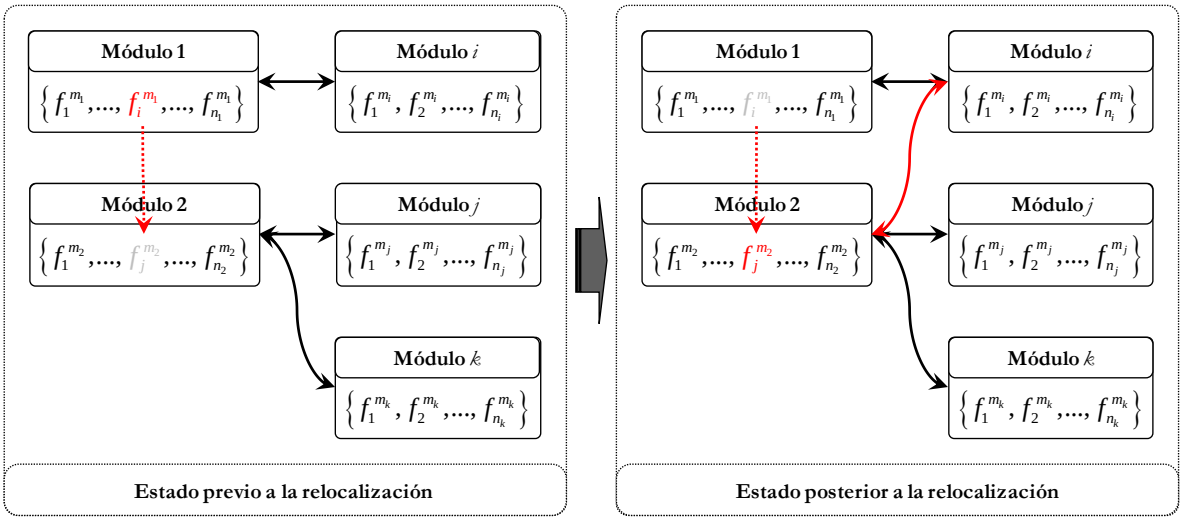


Figura 4.8. Relocalización de una función en arquitectura *Peer-to-Peer*.
La función $f_i^{m_1}$ se del módulo m_1 se mueve al módulo m_2 como $f_j^{m_2}$.

Es necesario mencionar que el análisis contempla exclusivamente el impacto que la relocalización de una función tiene sobre los demás componentes del sistema, suponiendo que la función "se mueve tal y como está", sin sufrir modificaciones en su nombre, firma, tiempo de ejecución o en el cuerpo mismo de la función. Por esto, si como parte de la relocalización la función requiriera ser adaptada o reprogramada (por ejemplo por un cambio del lenguaje en el que están desarrollados los módulos), aunque esta adaptación o reprogramación consume tiempo de desarrollo, éste es un cambio que debe realizarse independientemente de la arquitectura utilizada, por lo que dicho tiempo no es tomado en cuenta.

Si como consecuencia de la relocalización, la firma o tiempo de ejecución de la función resultan afectados, luego de la relocalización funcional debe llevarse a cabo un cambio de la definición de la función, caso que se estudia anteriormente en la sección anterior.

Cuando se realiza la relocalización de una función, es necesario actualizar todas las llamadas a la función en los módulos que hacen uso de esta función. Se propone el algoritmo siguiente:

```
Para cada módulo m_j ∈ D(f_i^{m_1})
     Para cada llamada a f_i^{m_1} en m_j
          Cambiar la llamada a f_i^{m_1} por una llamada a f_j^{m_2}
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|D\left(f_i^{m_1}\right)\right|\right)\left(\left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$ lo cual da la idea de una complejidad de orden cuadrático; para calcular la complejidad se toma:

$$n = min\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

$$N = max\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

Finalmente para la relocalización de una función en arquitectura *Peer-to-Peer* se tienen las cotas $\Omega(n^2)$ y $O(N^2)$.

ESPECIALIZACIÓN DE UNA FUNCIÓN

Este escenario contempla la división de una función $f_i^{m_1}$ del módulo m_1 en múltiples funciones dentro del mismo módulo m_1, analizándose el impacto en el sistema de descomponer $f_i^{m_1}$ en varias funciones especializadas.

Formalmente:

$$f_i^{m_1}(\bar{x}) = g_1^{m_1}(\bar{x}_1) \circ g_2^{m_1}(\bar{x}_2) \circ \ldots \circ g_n^{m_1}(\bar{x}_n)$$

donde:

- ✓ $f_i^{m_1}$ es la función a descomponer.

- ✓ $\bar{x}$ es el argumento de la función a descomponer.
- ✓ $g_1^{m_1}, \ldots, g_n^{m_1}$ son funciones especializadas que en conjunto y llamadas en orden realizan la misma función que $f_i^{m_1}$.
- ✓ $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ son los argumentos de las funciones especializadas.

Nótese que existe un caso en el cual $f_i^{m_1}$ es descompuesta en $g_1^{m_1}, \ldots, g_n^{m_1}$ pero se conserva $f_i^{m_1}$ como parte del módulo (aún cuando internamente llame a $g_1^{m_1}, \ldots, g_n^{m_1}$, es decir, como un alias de $g_1^{m_1}, \ldots, g_n^{m_1}$). En este caso no existe impacto alguno en el sistema, ya que los demás módulos seguirán haciendo referencia a $f_i^{m_1}$ sin importar cómo realice ésta sus procesos.

El otro caso ocurre cuando $f_i^{m_1}$ es descompuesta en $g_1^{m_1}, \ldots, g_n^{m_1}$ pero $f_i^{m_1}$ deja de estar disponible. Es en este caso cuando se vuelve necesario actualizar a los demás componentes del sistema y el cual es analizado a continuación. Esto se ejemplifica en la Figura 4.9.

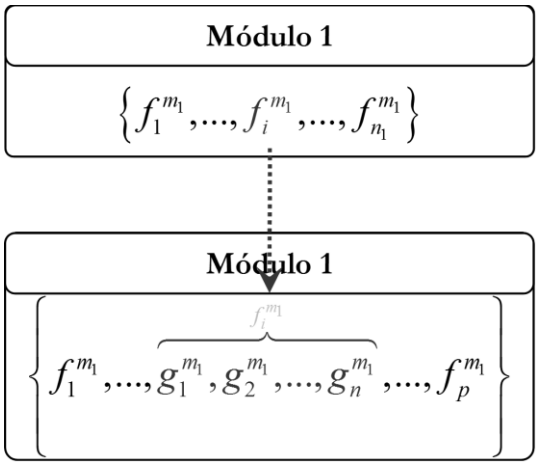


Figura 4.9. Especialización de una función. La función $f_i^{m_1}$ es reemplazada por las funciones especializadas $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$.

Es decir, la función $f_i^{m_1}$ es reemplazada por las funciones especializadas $g_1^{m_1}, \ldots, g_n^{m_1}$. Es de suponerse que en el sistema no existen llamadas a ninguna de las funciones especializadas $g_1^{m_1}, \ldots, g_n^{m_1}$ y sí existen llamadas a $f_i^{m_1}$, mismas que podrán ser reemplazadas por una llamada en orden de todas y cada una de las funciones especializadas $g_1^{m_1}, \ldots, g_n^{m_1}$ que sustituyen a $f_i^{m_1}$, tal como se muestra en la Figura 4.10:

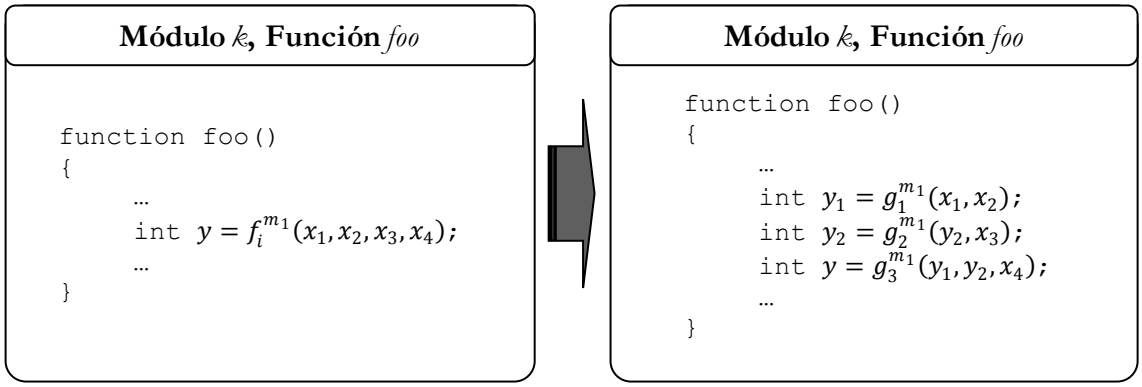


Figura 4.10. Un modo de llamar a la función original a partir
de las funciones especializadas en arquitectura *Peer-to-Peer*.
A la izquierda la función foo llama a la función original $f_i^{m_1}$.
A la derecha la función foo llama a las nuevas funciones especializadas g.

Es necesario mencionar que el análisis contempla exclusivamente el impacto que la especialización de una función tiene sobre los demás componentes del sistema, por lo que el tiempo requerido para descomponer la función no se toma en cuenta.

Con lo anterior, se propone el siguiente algoritmo para actualizar todas las llamadas a la función que ha sido especializada:

```
Para cada módulo m_j ∈ D(f_i^{m_1})
     Para cada llamada a f_i^{m_1} en m_j
          Cambiar la llamada a f_i^{m_1} por llamadas a g_1^{m_1}, …, g_n^{m_1}
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|D\left(f_i^{m_1}\right)\right|\right)\left(\left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$ lo cual da la idea de una complejidad de orden cuadrático. Para calcular la complejidad se toma:

$$n = min\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

$$N = max\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

Finalmente, para la especialización de una función en arquitectura *Peer-to-Peer* se tienen las cotas $\Omega(n^2)$ y $O(N^2)$

GENERALIZACIÓN DE FUNCIONES

Este escenario contempla la unión de varias funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ del módulo m_1 en una sola función $f_i^{m_1}$ del mismo módulo m_1, analizándose el impacto en el sistema de unir las funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ en una función generalizada.

Formalmente:

$$g_1^{m_1}(\bar{x}_1) \circ g_2^{m_1}(\bar{x}_2) \circ \ldots \circ g_n^{m_1}(\bar{x}_n) = f_i^{m_1}(\bar{x})$$

donde:

- ✓ $g_1^{m_1}, \ldots, g_n^{m_1}$ son funciones que se desean unir para formar a $f_i^{m_1}$.
- ✓ $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ son los argumentos de las funciones a generalizar.
- ✓ $f_i^{m_1}$ es la función generalizada.
- ✓ $\bar{x}$ es el argumento de la función generalizada.

Nótese que existe un caso en el cual las funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ se unen en $f_i^{m_1}$ pero se conservan $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ como parte del módulo (es decir, $f_i^{m_1}$ internamente sólo llama a las funciones $g_1^{m_1}, \ldots, g_n^{m_1}$). En este caso no existe impacto alguno en el sistema, ya que los demás módulos seguirán haciendo referencia a la correspondiente función $g_1^{m_1}, \ldots, g_n^{m_1}$.

El otro caso ocurre cuando $g_1^{m_1}, \ldots, g_n^{m_1}$ siempre son llamadas en conjunto (de otro modo no tendría sentido una generalización, pues $g_1^{m_1}, \ldots, g_n^{m_1}$ siguen siendo requeridas de manera independiente) y estas múltiples llamadas serán reemplazadas por una sola llamada a $f_i^{m_1}$, por lo que $g_1^{m_1}, \ldots, g_n^{m_1}$ dejarán de estar disponibles. Es en este caso cuando se vuelve necesario actualizar a los demás componentes del sistema y el cual es analizado a continuación. Esto se ejemplifica en la Figura 4.11

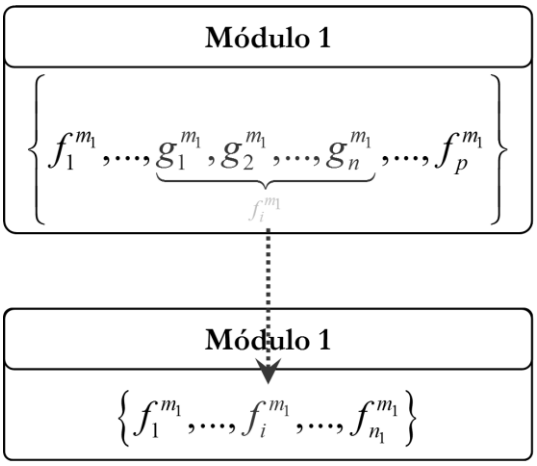


Figura 4.11. Generalización a una función. Las funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ son reemplazadas por la función generalizada $f_i^{m_1}$.

Es decir, las funciones $g_1^{m_1}, \dots, g_n^{m_1}$ son reemplazadas por la función generalizada $f_i^{m_1}$. Así pues, toda llamada al conjunto de funciones $g_1^{m_1}, \dots, g_n^{m_1}$ puede ser sustituido por una sola llamada a $f_i^{m_1}$ tal como se muestra en la Figura 4.12:

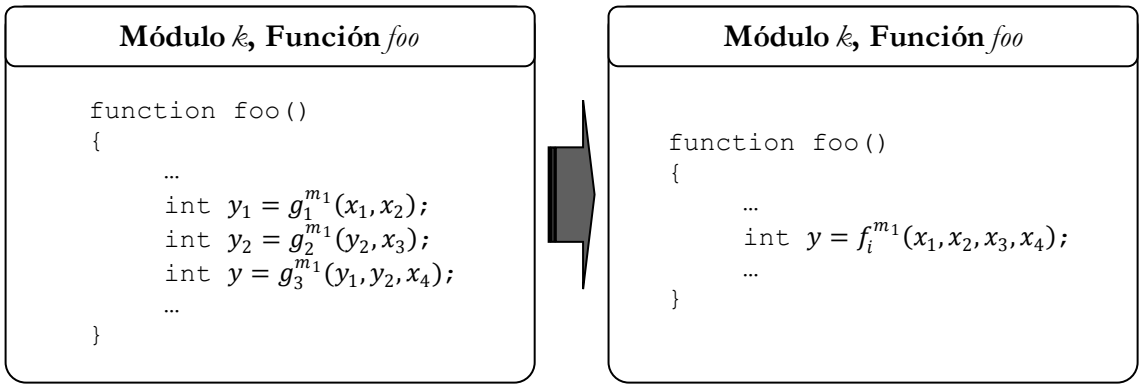


Figura 4.12. Un modo de llamar a las funciones originales a partir de la función generalizada en arquitectura *Peer-to-Peer*.
A la izquierda la función foo llama a las funciones originales g.
A la derecha la función foo llama a la nueva función generalizada $f_i^{m_1}$.

Es necesario mencionar que el análisis contempla exclusivamente el impacto que la generalización de una función tiene sobre los demás componentes del sistema, por lo que el tiempo requerido para unir las funciones no se toma en cuenta.

Con lo anterior, se propone el siguiente algoritmo para actualizar todas las llamadas al conjunto de funciones que han sido generalizadas:

```
Para cada módulo m_j ∈ D(g_1^{m_1}, …, g_n^{m_1})
    Para cada llamada a g_1^{m_1}, …, g_n^{m_1} en m_j
        Cambiar las llamadas a g_1^{m_1}, …, g_n^{m_1}
        por una llamada a f_i^{m_1}
    Fin para cada
Fin para cada
```

Se tiene que $\left(\left|D\left(g_1^{m_1}, \dots, g_n^{m_1}\right)\right|\right)\left(\left|m_j \xrightarrow{call} \left(g_1^{m_1}, \dots, g_n^{m_1}\right)\right|\right)$ lo cual da la idea de una complejidad de orden cuadrático; para calcular la complejidad se toma:

$$n = min\left(\left|D\left(g_1^{m_1}, \dots, g_n^{m_1}\right)\right|, \left|m_j \xrightarrow{call} g_1^{m_1}, \dots, g_n^{m_1}\right| \right)$$

$$N = max\left(\left|D\left(g_1^{m_1}, \dots, g_n^{m_1}\right)\right|, \left|m_j \xrightarrow{call} g_1^{m_1}, \dots, g_n^{m_1}\right| \right)$$

Finalmente, para la generalización de funciones en arquitectura *Peer-to-Peer* se tienen las cotas $\Omega(n^2)$ y $O(N^2)$

RELOCALIZACIÓN DE UN MÓDULO

En este apartado se contempla la relocalización de un módulo, es decir mover el módulo de su contexto actual de ejecución a un nuevo contexto (lo que comúnmente se traduce a un cambio de máquina). Dependiendo de la tecnología utilizada, la relocalización de un módulo puede ser inmediata o puede requerir cambios que van desde ligeros ajustes hasta una reprogramación completa del módulo involucrado. En esta sección se analiza únicamente el impacto que tiene en el sistema la relocalización de un módulo y el costo de la actualización del sistema luego de dicha relocalización, por lo que el costo de adaptar el módulo a migrar al nuevo contexto no se toma en cuenta.

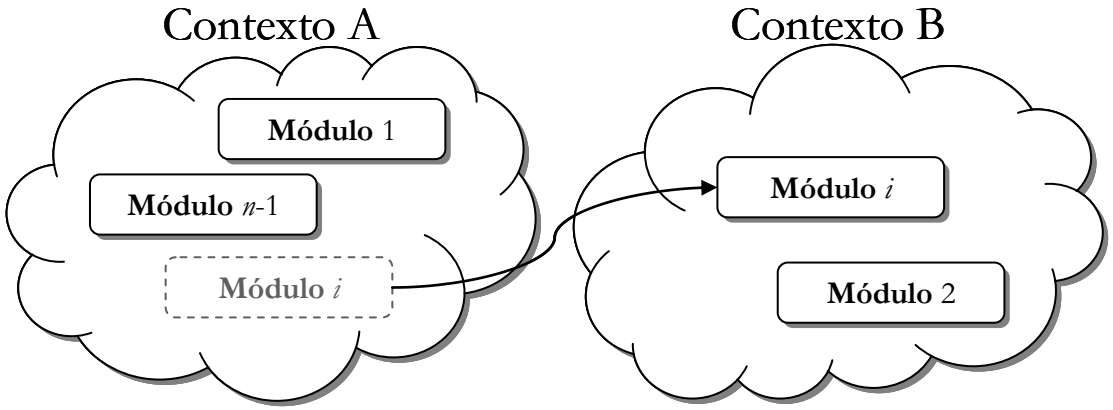


Figura 4.13. Cambio de contexto de un módulo.
El módulo *i* se mueve del contexto A al contexto B.

Para la relocalización de un módulo existen dos escenarios a contemplar: cuando se utiliza un servicio de resolución de nombres (provisto por la tecnología utilizada, *Framework* o *middleware*) encargado de gestionar el acceso a los componentes con independencia de su ubicación, y cuando dicho servicio no existe y cada componente debe conocer la ubicación de los demás componentes del sistema con los que interacciona.

A fin de simplificar el estudio del impacto de una relocalización, se considera que cada módulo en el sistema cuenta con un nombre y un punto de acceso (que puede verse reflejado como una *url*, un *pipe*, dirección IP y puerto, etc.) mediante el cual son accesibles todas las funciones que provee el módulo. Así, en el primer escenario los módulos no requerirán conocer de antemano dicho puntos de acceso de los módulos con los que interaccionan, pues éstos serán gestionados por el servicio de resolución de nombres; mientras que en el segundo escenario sí será necesario éste conocimiento.

Cuando se relocaliza un módulo bajo el primer escenario, basta con actualizar el punto de acceso a éste en el servicio de resolución de nombres, por lo que en este escenario la relocalización de un módulo no afecta al sistema más allá de este cambio, quedando la complejidad del cambio en $O(1)$.

Cuando se relocaliza un módulo bajo el segundo escenario, es necesario actualizar en cada módulo con el que existe una relación de dependencia el punto de acceso al módulo que ha sido relocalizado. Se toma la suposición de que los puntos acceso a los módulos con los que existe dependencia son configurados una sola vez en cada módulo (aunque pueden ser referenciados múltiples veces en base a esta única configuración).

Así pues, el algoritmo propuesto para una relocalización modular es el siguiente:

```
Para cada módulo m_j ∈ D(m_1)
     Actualizar punto de acceso a m_1
Fin para cada
```

Se tiene $|D(m_1)|$, lo cual da la idea de una complejidad de orden lineal. Para calcular la complejidad se toma $n = |D(m_1)|$ por lo que, para la relocalización de un módulo en arquitectura *Peer-to-Peer* cuando no se cuenta con un servicio de resolución de nombres se tiene la cota $O(n)$.

ESPECIALIZACIÓN DE UN MÓDULO USANDO ESTRUCTURA JERÁRQUICA

Este apartado contempla la especialización de un módulo, descomponiéndolo en varios sub-módulos interrelacionados mediante una estructura jerárquica. Generalmente esto es representable por un árbol, tal como se ilustra en la Figura 4.14.

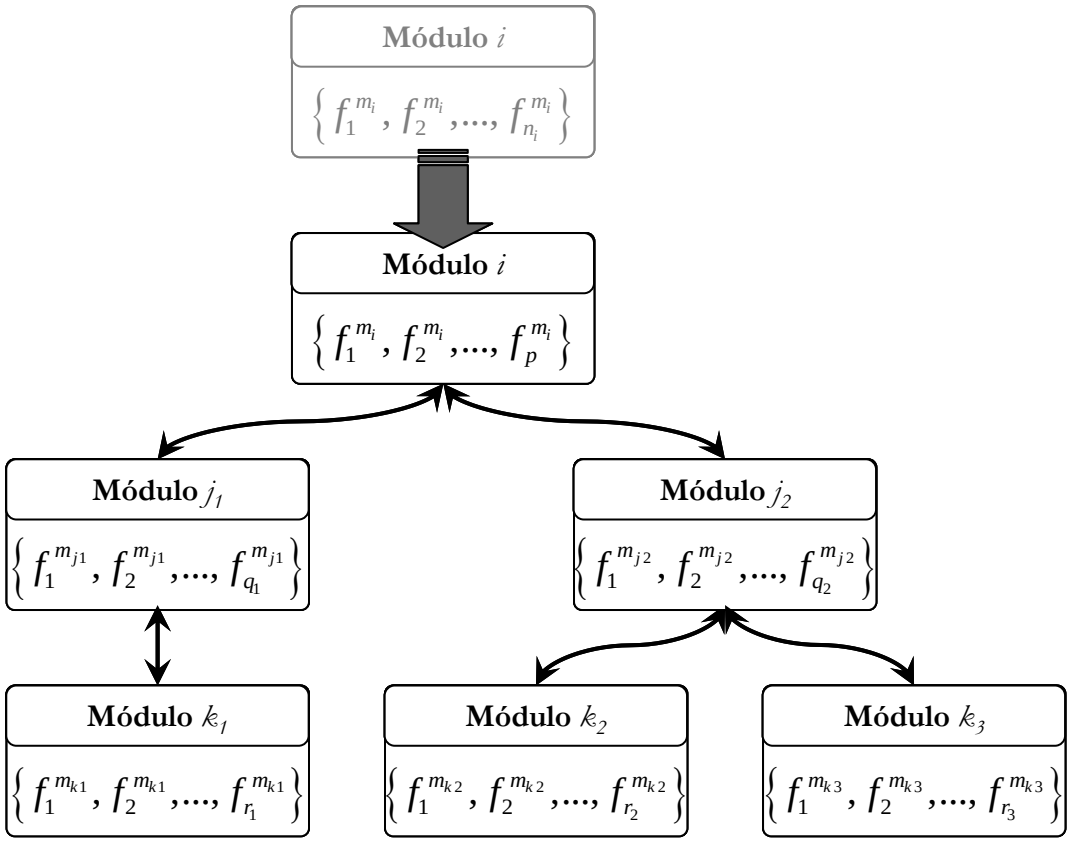


Figura 4.14. Especialización jerárquica de un módulo en arquitectura *Peer-to-Peer*.
El módulo i se descompone en los sub-módulos especializados j_1, j_2, k_1, k_2 y k_3. El módulo i se conserva.

Esta especialización se realiza bajo el supuesto de que al menos el primer módulo de la jerarquía se ejecuta en el mismo contexto que el módulo original y que los tiempos de ejecución y las firmas de las funciones que provee o a las que da acceso no sufren cambios. En caso contrario debe llevarse a cabo una relocalización de un módulo o un cambio de la definición de las funciones afectadas, según corresponda.

En este tipo de especialización, el primer módulo en la jerarquía (módulo m_i) conserva todas las conexiones con los módulos con los que estaba conectado el módulo original, además de las mismas funciones en lo que a nombre y firma se refiere. No necesariamente conserva el cuerpo de las mismas, el cual puede ser movido íntegramente a otro módulo de menor jerarquía, o ser descompuesto en varias funciones más especializadas distribuidas en los diferentes sub-módulos, a los cuales la función en el módulo original tiene que hacer referencia.

Por ejemplo, supóngase un escenario en el que el módulo m_i es especializado bajo una estructura jerárquica y la ejecución de la función $f_2^{m_i}$ módulo m_i se delega al nuevo sub-módulo de tercer nivel m_{k_2} (el cual está dedicado completamente a la ejecución de esa función). Bajo este escenario, una vez realizada la especialización, m_i conserva una función $f_2^{m_i}$ la cual hace referencia a la función $f_3^{m_{j_1}}$ del sub-módulo de segundo nivel m_{j_1}, y que a su vez hace referencia a la función $f_1^{m_k}$ del sub-módulo de tercer nivel m_k, la cual se encarga de realizar el procesamiento. Por lo que es responsabilidad tanto de $f_2^{m_i}$ como de $f_3^{m_{j_1}}$ el gestionar el acceso a $f_1^{m_k}$ (Véase Figura 4.15).

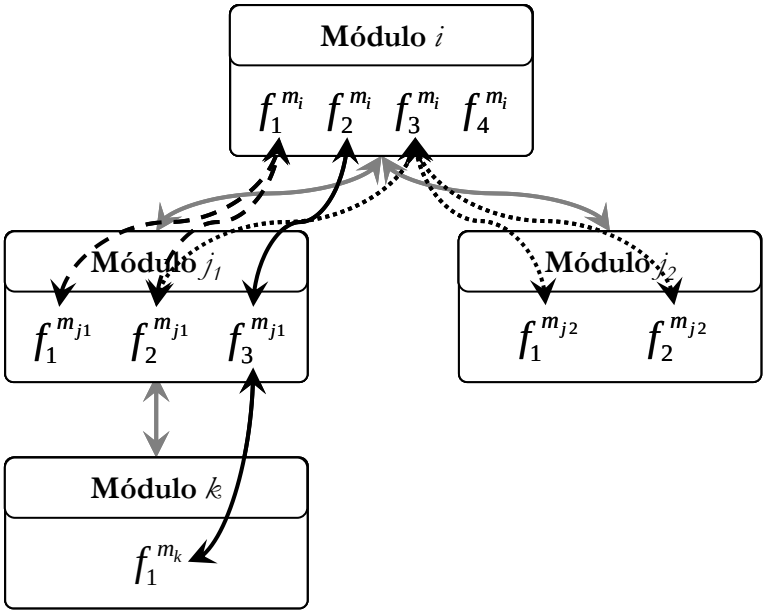


Figura 4.15. Un ejemplo de especialización jerárquica de un módulo en arquitectura *Peer-to-Peer*.

En el ejemplo de la Figura 4.15 se muestra no solo que la función $f_2^{m_i}$ del módulo m_i es un alias que permite el acceso a la función $f_1^{m_k}$, sino que la función $f_1^{m_i}$ del módulo m_i llama a las funciones $f_1^{m_{j_1}}$ y $f_2^{m_{j_1}}$ del módulo m_{j_1}, así como la función $f_3^{m_i}$ llama a las funciones $f_2^{m_{j_1}}$, $f_1^{m_{j_2}}$ y $f_2^{m_{j_2}}$ de los módulos m_{j_1} y m_{j_2}. Mientras que la función $f_4^{m_i}$ sí es ejecutada en el módulo m_i.

Al preservarse en el primer módulo de la jerarquía todas las funciones disponibles en el módulo original, los otros módulos del sistema no se ven afectados. Por lo tanto, es válido decir que la especialización de un módulo utilizando una estructura jerárquica no impacta a los demás módulos del sistema.

ESPECIALIZACIÓN DE UN MÓDULO A MÓDULOS INDEPENDIENTES

Este apartado contempla la especialización de un módulo, descomponiéndolo en varios sub-módulos independientes más especializados, tal como se ilustra en la Figura 4.16.

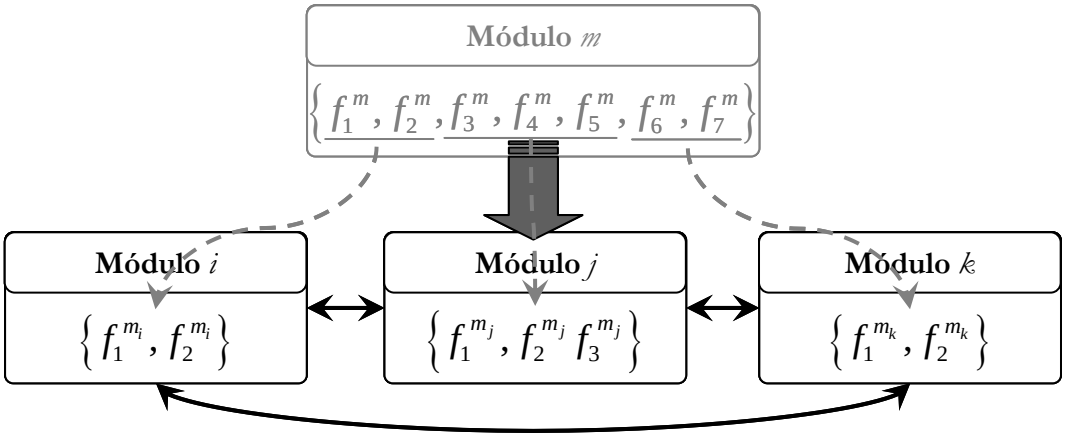


Figura 4.16. Especialización de un módulo a módulos independientes sin recurso compartido. El módulo m, que se descompone en los módulos especializados m_i, m_j y m_k.

Supóngase que existe una relación de dependencia entre los módulos m_1, m_2, m_3 y el módulo m, que se descompone en los módulos especializados m_i, m_j y m_k, tal como se ilustra en la Figura 4.17:

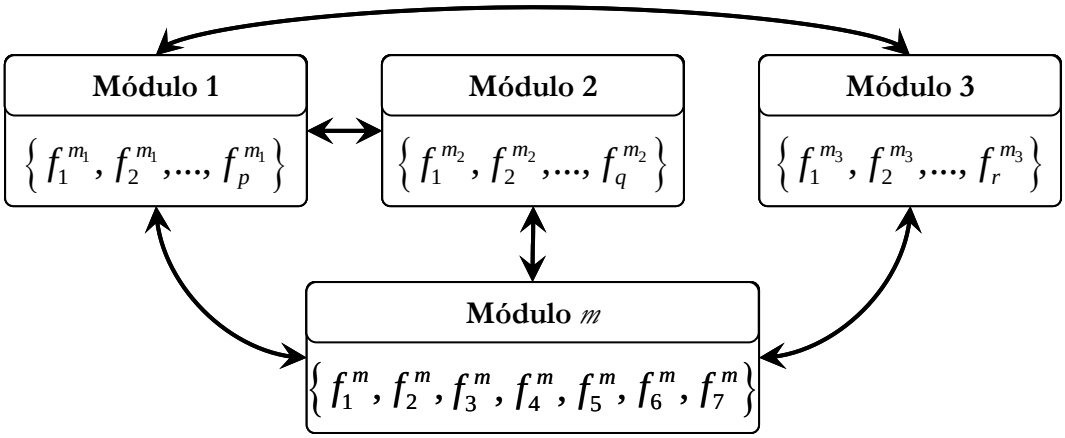


Figura 4.17. Estado del subsistema formado por el módulo m y sus interdependencias previo a la especialización del módulo m. Las flechas representan canales de comunicación.

En este supuesto, tras la especialización del módulo m, se tiene que actualizar las referencias a las funciones utilizadas del módulo original m por los módulos m_1, m_2 y m_3 de manera similar a como sucedería si estas funciones se hubieran "movido" a otro módulo (relocalización de funciones), tal como se ejemplifica en la Figura 4.18:

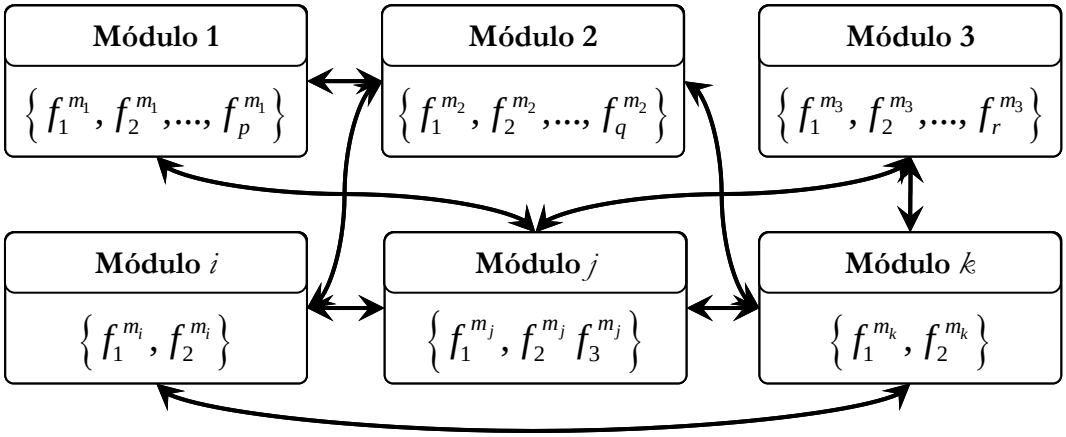


Figura 4.18. Estado del subsistema formado por los módulos m_i, m_j, m_k y sus interdependencias, posterior a la especialización del módulo m.

Para simplificar el análisis formal del impacto de esta operación de especialización, se puede suponer que se crean tantos nuevos módulos vacíos (sin funciones), como el número de módulos a especializar menos uno; después se "mueven" las funciones del módulo original a los nuevos módulos creados (los que inicialmente estaban vacíos) hasta que se tiene la especialización deseada.

Con esto se presentan las siguientes definiciones:

- ✓ $F_R \subset m$ es el conjunto de funciones a relocalizar.
- ✓ M_E es el conjunto de módulos en los que se divide el módulo m tras su especialización.

Luego entonces, formalmente:

$$\forall f_j^m \in F_R, \qquad f_j^m \xrightarrow{move} f_k^{m_i}$$

donde:

$$f_j^m(\bar{x}) = f_k^{m_i}(\bar{x})$$

y

$$m_i \in M_E$$

Es decir, se tienen $|F_R|$ funciones a relocalizar, lo cual como se estudia en la sección Relocalización de una función, tiene las cotas $\Omega(n^2)$ y $O(N^2)$.

Además, es de suponerse que durante una especialización se mueven el menor número posible de funciones, por lo que si un módulo con diez funciones es descompuesto en dos módulos especializados con dos y ocho funciones respectivamente, es natural pensar que solamente 2 funciones se mueven. Así, en el peor escenario donde un módulo m se descompondrá en k sub-módulos cada uno con el mismo número de funciones, se mueven un máximo de $\frac{|m|}{k-1}$ funciones, por lo que $|F_R| \leq \frac{|m|}{k-1}$.

Tomando las cotas mencionadas anteriormente para la relocalización de funciones y el valor $|F_R|$, se tienen finalmente cotas $\Omega(|F_R| \cdot n^2)$ y $O(|F_R| \cdot N^2)$ para la especialización de un módulo en varios módulos independientes bajo una arquitectura basada en *Peer-to-Peer*.

GENERALIZACIÓN DE MÓDULOS CON ESTRUCTURA JERÁRQUICA

Este apartado contempla la unión de varios módulos con una organización jerárquica (generalmente representable por un árbol) en un solo súper-módulo, tal como se ilustra en la Figura 4.19.

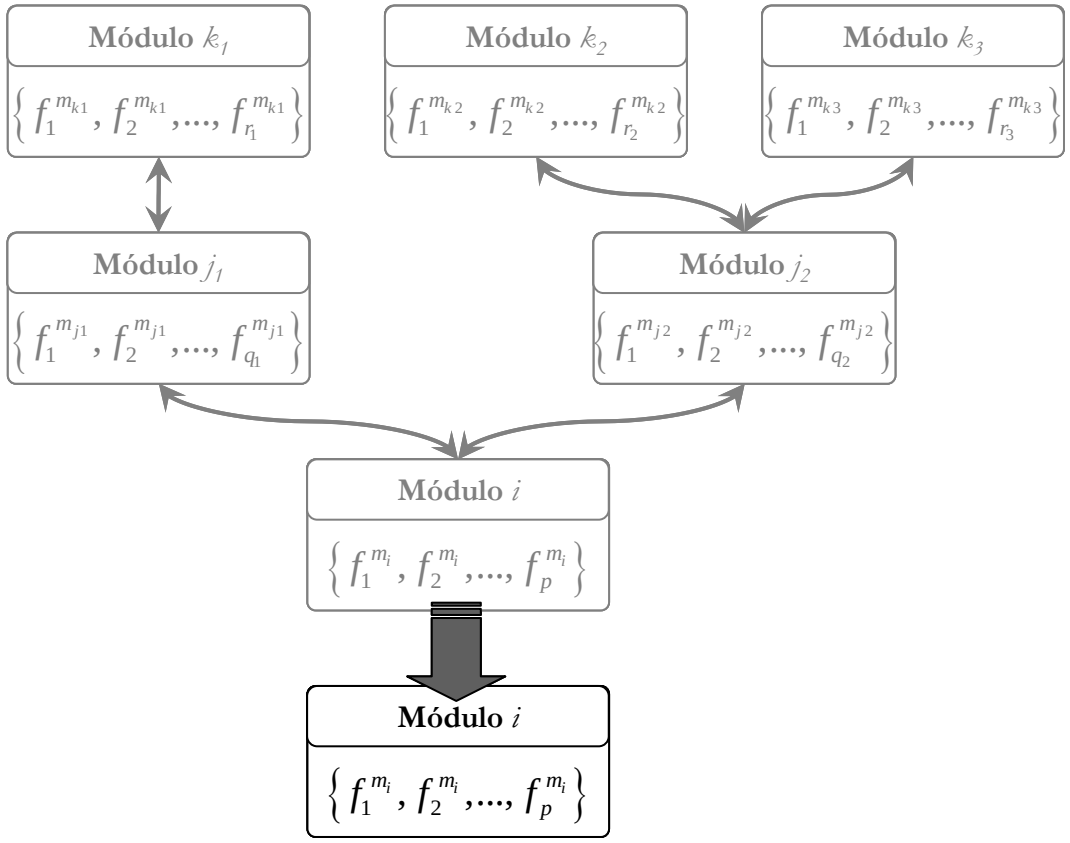


Figura 4.19. Generalización jerárquica de un módulo en arquitectura *Peer-to-Peer*.
Los módulos i, j_1, j_2, k_1, k_2 y k_3 se unen en un solo módulo i.

Esta generalización se realiza bajo el supuesto de que todos los módulos a unir se ejecutan en el mismo contexto, y que los tiempos de ejecución y las firmas de las funciones que provee o a las que da acceso no sufren cambios tras la generalización. En caso contrario debe llevarse a cabo primero una Relocalización de un módulo o un cambio de la definición de las funciones afectadas, según corresponda para los módulos afectados.

Además, la existencia de una estructura jerárquica obliga a que las funciones contenidas en módulos de menor jerarquía sean accesibles solamente mediante llamadas realizadas por el módulo inmediato superior en la jerarquía. Es decir, que la única manera de acceder a las funciones provistas por el conjunto de módulos a unir es mediante el módulo de más alta jerarquía.

Por ejemplo, supóngase un escenario en el que los módulos m_i, m_{j_1}, m_{j_2} y $, m_k$, conectados tal como se muestra en la Figura 4.20 se unen, generalizándose en un súper-módulo m. En este escenario, la función $f_1^{m_k}$ es ejecutada por el módulo m_k, pero accesible al sistema mediante la función $f_2^{m_i}$ módulo m_i con $f_3^{m_{j_1}}$ del sub-módulo de segundo nivel m_{j_1} como intermediario. Tras la generalización, la definición o cuerpo de la función $f_1^{m_k}$ se "mueve" a la función $f_2^{m_i}$ que deja de ser un mecanismo de control y redirección para ejecutar la función propiamente dicha.

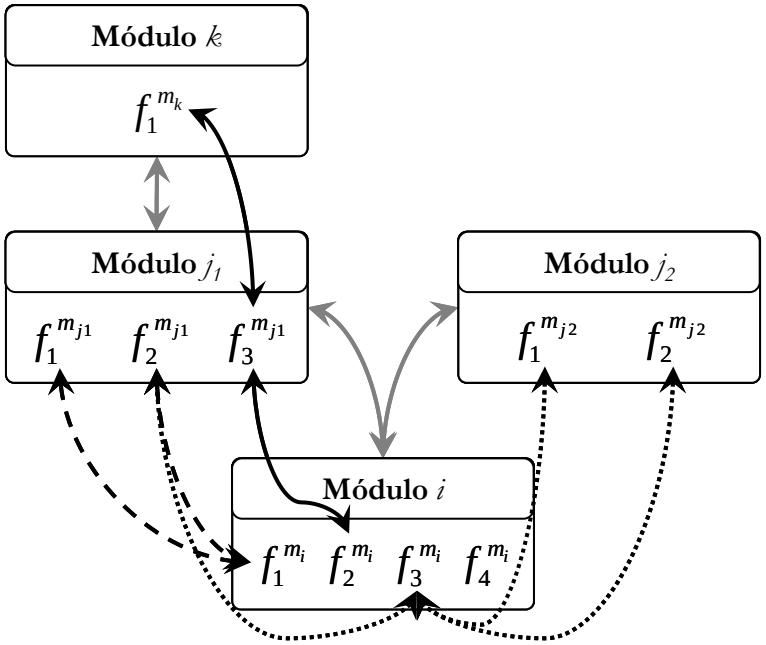


Figura 4.20. Un ejemplo de generalización a partir de una organización jerárquica de módulos en arquitectura *Peer-to-Peer*.

Así mismo, las definiciones de $f_1^{m_{j_1}}$ y $f_2^{m_{j_1}}$ se unen para formar parte de $f_1^{m_i}$, al igual que las definiciones de $f_2^{m_{j_1}}, f_1^{m_{j_2}}$ y $f_2^{m_{j_2}}$ se unen para formar parte de $f_3^{m_i}$. Finalmente los módulos m_{j_1}, m_{j_2} y $, m_k$ desaparecen, quedando solo m_i que integra todas las funciones de los módulos originales.

Al preservarse en el primer módulo de la jerarquía todas las funciones disponibles en el módulo original, los otros módulos del sistema no se ven afectados. Por lo tanto, es válido decir que la generalización de un módulo utilizando una estructura jerárquica no impacta a los demás módulos del sistema.

GENERALIZACIÓN DE MÓDULOS INDEPENDIENTES

Este apartado contempla la unión de varios módulos independientes para conformar un nuevo súper-módulo generalizado, tal como se ilustra en la Figura 4.21.

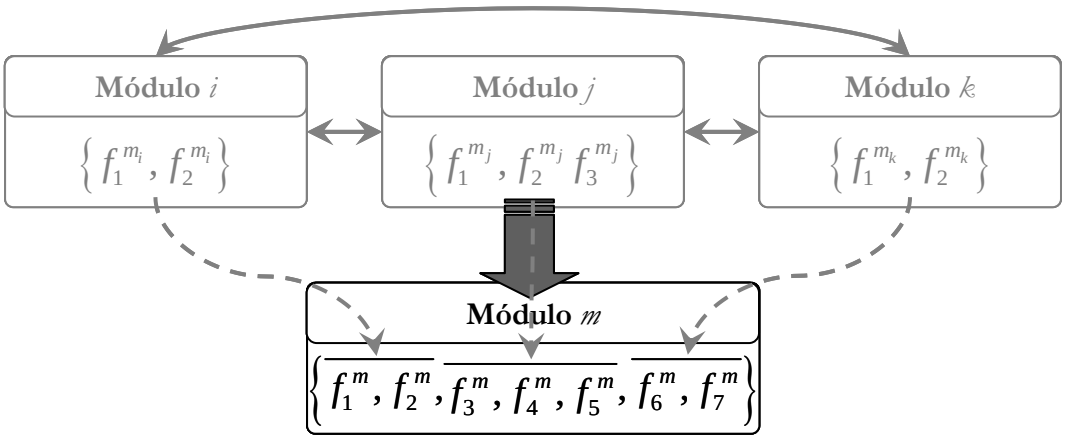


Figura 4.21. Generalización de módulos independientes.
Los módulos m_i, m_j y m_k se unen para conformar un nuevo módulo m.

Supóngase que existe una relación de dependencia entre los módulos m_1, m_2, m_3 y los módulos m_i, m_j y m_k que se unen para conformar el nuevo módulo m, tal como se ilustra en la Figura 4.22.

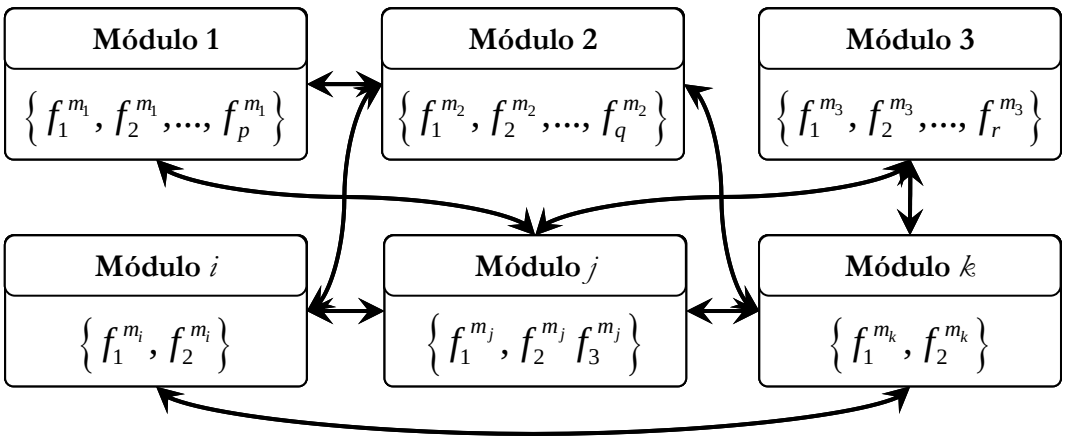


Figura 4.22. Estado del subsistema formado por los módulos m_i, m_j, m_k y sus interdependencias previo a la generalización.

En este supuesto, tras la generalización que da como resultado el módulo m, se tienen que actualizar las referencias a las funciones utilizadas de los módulos originales m_i, m_j y m_k por los módulos m_1, m_2 y m_3, de manera similar a como sucedería si estas funciones se hubieran "movido" a otro módulo (relocalización de funciones), tal como se ejemplifica en la Figura 4.23.

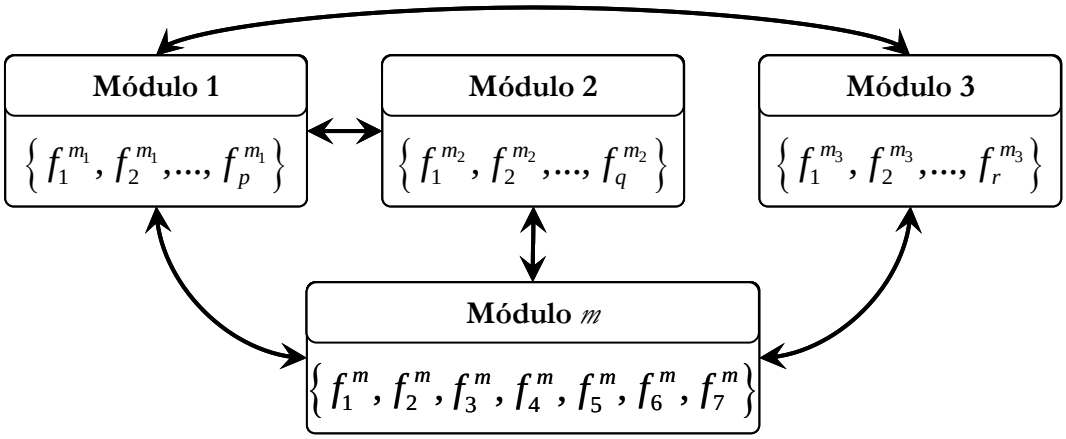


Figura 4.23. Estado del subsistema formado por el módulo m y sus interdependencias, posterior a la generalización de los módulos m_i, m_j, m_k.

Para simplificar el análisis formal del impacto de esta operación de generalización, se puede suponer que se escoge al módulo con mayor número de funciones del conjunto de módulos que se desea unir como base; y después se "mueven" las funciones de los otros módulos al módulo base hasta que se tiene la generalización deseada.

Con esto se presentan las siguientes definiciones:

- ✓ M_E es el conjunto de módulos especializados que se unen para conformar al nuevo módulo generalizado.
- ✓ m_0 es el módulo escogido como módulo base, es decir, el que posteriormente es el módulo generalizado.
- ✓ $F_R \subset M_E - \{m_0\}$ es el conjunto de funciones a relocalizar.

Luego entonces, formalmente:

$$\forall f_j \in F_R, \qquad f_j^{m_i} \xrightarrow{move} f_k^{m_0}$$

donde:

$$f_j^{m_i}(\bar{x}) = f_k^{m_0}(\bar{x})$$

y

$$m_i \in M_E - \{m_0\}$$

Es decir, se tienen $|F_R|$ funciones a relocalizar, lo cual como se estudia en la sección Relocalización de una función, tiene las cotas $\Omega(n^2)$ y $O(N^2)$.

Además, es de suponerse que durante una generalización se mueven el menor número posible de funciones, por lo que si se desean unir dos módulos con dos y ocho funciones respectivamente, es natural pensar que solamente 2 funciones se mueven. Así, en el peor escenario, donde k módulos cada uno con el mismo número de funciones se unen en un nuevo súper-módulo, se mueven un máximo de $\frac{|m|}{k-1}$ funciones, por lo que $|F_R| \leq \frac{|m|}{k-1}$.

Tomando las cotas mencionadas anteriormente para la relocalización de funciones y el valor $|F_R|$, se tienen finalmente cotas $\Omega(|F_R| \cdot n^2)$ y $O(|F_R| \cdot N^2)$ para la generalización de varios módulos independientes en un solo módulo bajo una arquitectura basada en *Peer-to-Peer*.

4.3. Análisis en arquitectura basada en Blackboard

En una arquitectura *Blackboard*, se considera que todos los módulos están conectados al *Blackboard,* y se comunican mediante la lectura/escritura de las variables almacenadas en el propio *Blackboard*.

Como se expresa en la sección 2.3.2, en una arquitectura basada en *Blackboard* hay tres tipos de módulos: el módulo que funge como *Blackboard*, los módulos que interactúan directamente con el hardware actuador y que se conectan al *Blackboard* mediante uno y sólo un intermediario, y los módulos colaboradores (incluyendo al componente de control, si éste no está integrado en el módulo *Blackboard*).

Con esto, en una arquitectura basada en *Blackboard,*

$$\forall m_i \in M_b \; \exists c_{m_i,m_b}$$

y

$$\forall m_j \in M_a \subset M \; \exists c_{m_j,m_i}$$

Por lo tanto, el número de conexiones $|C| = |M| - 1$.

donde:

- ✓ M es el conjunto de todos los módulos del sistema.
- ✓ M_a es el conjunto de módulos que interactúan directamente con el hardware.
- ✓ M_b es el conjunto de módulos que se comunican directamente con el *Blackboard*, es decir, $M_b = M - M_a - \{m_b\}$.
- ✓ m_b es el módulo que funciona como *Blackboard*.
- ✓ m_i es cualquier otro módulo del sistema tal que $m_i \in M_b$.

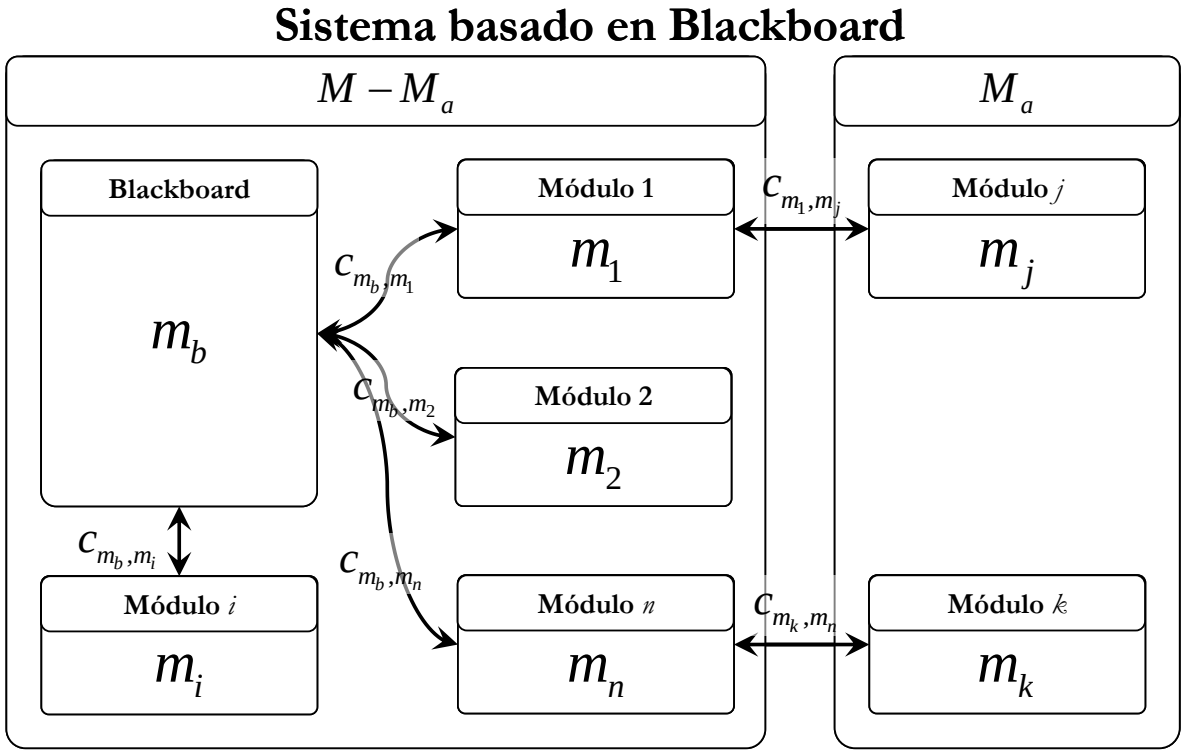


Figura 4.24. Sistema basado en *Blackboard*: elementos y comunicaciones.

Además se definen:

$$V = \{x_1, x_2, \dots, x_n\} \cup \{y_1, y_2, \dots, y_m\} \cup \left\{f_1^{m_1}, \dots, f_p^{m_1}, f_1^{m_2}, \dots, f_q^{m_2}, \dots, f_1^{m_n}, \dots, f_r^{m_n}\right\}$$

Como el conjunto de variables del sistema, $v \in V$ es una variable del sistema, y $read^{m_i}(v_j), write^{m_i}(v_j)$ las funciones de lectura y escritura en el *Blackboard* de la variable v_j realizada por el módulo m_i, donde:

- ✓ V es el conjunto de todas las variables del sistema, formado por las funciones que pueden ejecutar los módulos del sistema, los parámetros que reciben y sus resultados.
- ✓ $\{x_1, x_2, \dots, x_n\}$ es el conjunto de variables de entrada para las funciones del sistema.
- ✓ $\{y_1, y_2, \dots, y_m\}$ es el conjunto de variables resultado de la ejecución de funciones del sistema.
- ✓ $\left\{f_1^{m_1}, \dots, f_p^{m_1}, f_1^{m_2}, \dots, f_q^{m_2}, \dots, f_1^{m_n}, \dots, f_r^{m_n}\right\}$ es el conjunto de variables que indican a los módulos del sistema qué funciones deben ejecutar, así como el estado de las mismas.
- ✓ $v \in V$ es una de las variables del sistema.
- ✓ $read^{m_i}(v_j)$ es una operación de lectura sobre la variable v_j realizada por el módulo m_i.

- ✓ $write^{m_i}(v_j)$ es una operación de escritura sobre la variable v_j realizada por el módulo m_i.

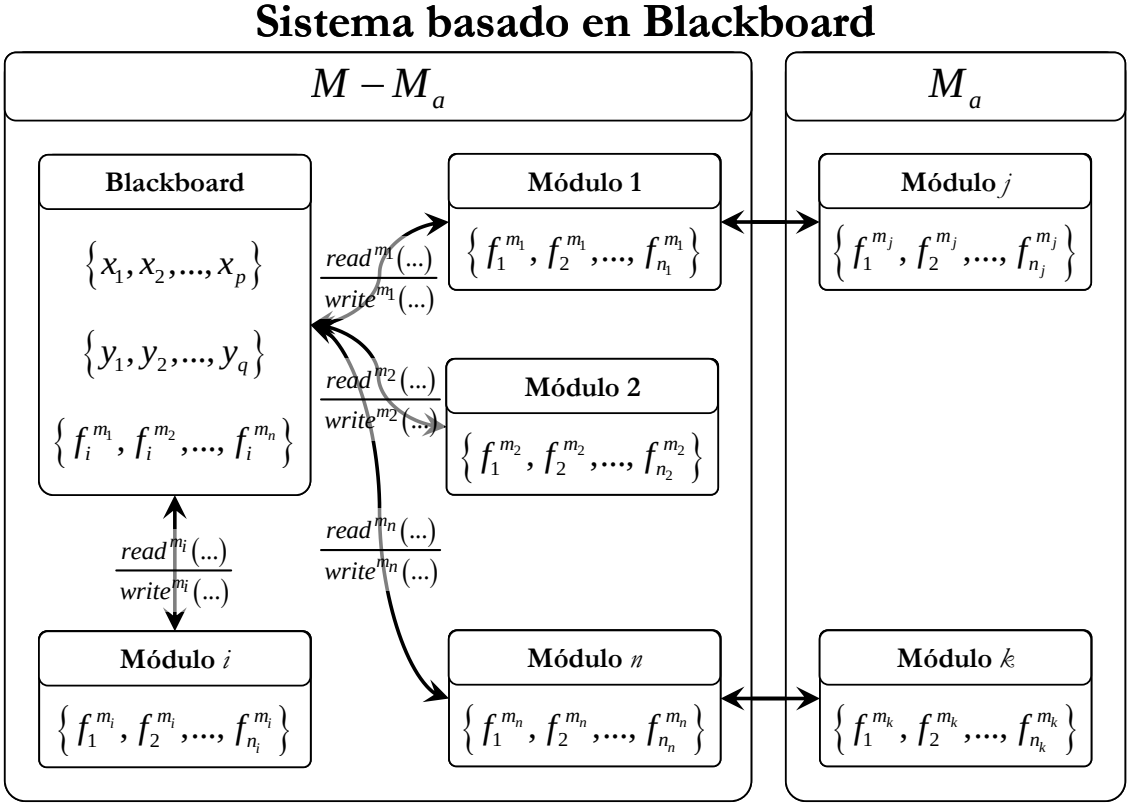


Figura 4.25. Sistema basado en *Blackboard*: funciones, variables y operaciones.

Tal como se menciona en la sección 4.2, se considera que toda comunicación entre módulos está formada por una petición o comando y una respuesta, las cuales en este caso se reducen únicamente a las peticiones de lectura/escritura y sus respuestas. Debido a esta limitación, la ejecución de una función debe ser desglosada en operaciones de lectura y escritura. Así, la ejecución de una función remota se desglosa como:

- ✓ $write^{m_i}\left(f_j^{m_k}\right)$. El módulo m_i escribe en el *Blackboard* una petición de ejecución de la función $f_j^{m_k}$ del módulo m_k.
- ✓ $read^{m_k}(\bar{x})$. El módulo m_k realiza una lectura de las variables de entrada $\bar{x} = \{x_1, x_2, \dots, x_n\}$ en el *Blackboard*.
- ✓ Se ejecuta en el módulo m_k la función $f_j^{m_k}$ con los argumentos de $\bar{x}$ generando el vector de resultados $\bar{y} = \{y_1, y_2, \dots, y_m\}$.
- ✓ $write^{m_k}(\bar{y})$. El módulo m_k realiza una escritura en el *Blackboard* del vector $\bar{y}$ resultado de la función $f_j^{m_k}$ del módulo m_k.
- ✓ $read^{m_i}(\bar{y})$. El módulo m_i realiza una lectura del vector resultado $\bar{y}$.

donde:

- ✓ $|\bar{x}|, |\bar{y}|$ son los tamaños (número de componentes) de los vectores de entrada y salida de la función $f_j^{m_k}$, respectivamente.

De manera similar a como se establece en la sección 4.2, se asume que los módulos que realizan una petición de ejecución de una función esperan un tiempo finito por la ejecución de la función solicitada. El control de los tiempos límite de ejecución de las funciones es llevado por el componente de control. Esto es debido a que el componente de control es quien gestiona el acceso a las variables almacenadas en el *Blackboard* y, en consecuencia, qué funciones se ejecutan y su estado de ejecución. Puede controlar también (aunque de manera indirecta) los tiempos límite de ejecución de las funciones cambiando las variables que informan el estado de las funciones, y permitiendo o denegando el acceso a los recursos del *Blackboard* por parte de los módulos que contienen dichas funciones. No obstante, nada limita a que las funciones implementen sus propias rutinas de verificación, control de acceso a los recursos, manejo de tiempos de espera, etc.

4.3.1. Extensibilidad

En este apartado se analiza el costo de realizar los cambios descritos en la sección 4.1 al software de un robot en arquitectura *Blackboard*, en lo concerniente a extensibilidad. Es decir, se analiza el impacto a nivel sistema del intercambio de módulos equivalentes, y del cambio de la definición de las funciones de un módulo.

El análisis mencionado refleja el costo o complejidad que tendría actualizar los demás componentes del sistema cuando se presentan los cambios descritos.

INTERCAMBIO DE MÓDULOS EQUIVALENTES

Este escenario contempla el cambio de un módulo m_1 por otro módulo m_2 que realiza las mismas funciones y que se ejecuta en el mismo contexto del módulo original m_1. Formalmente:

Sean m_1, m_2 dos módulos de un sistema. m_1 y m_2 son intercambiables sí y sólo si:

$$\forall f_i^{m_1}(\bar{x})\ \exists\ f_j^{m_2}(\bar{x}) \mid f_i^{m_1}(\bar{x}) = f_j^{m_2}\ (\bar{x})$$

Aún cuando los módulos son intercambiables quedan dos aspectos a considerar:

1. Los nombres de las funciones, y su firma (tipo de dato devuelto, y el tipo y orden de los parámetros que recibe).
2. Tiempo de ejecución de la función.

Cuando para todas las funciones, el nombre de la función y su firma coinciden, y sus tiempos de ejecución cumplen $t\left(f_j^{m_2}\right) \lesssim t\left(f_i^{m_1}\right)$, el caso es trivial, pues los módulos son completamente compatibles y no hay que hacer ningún cambio en el sistema.

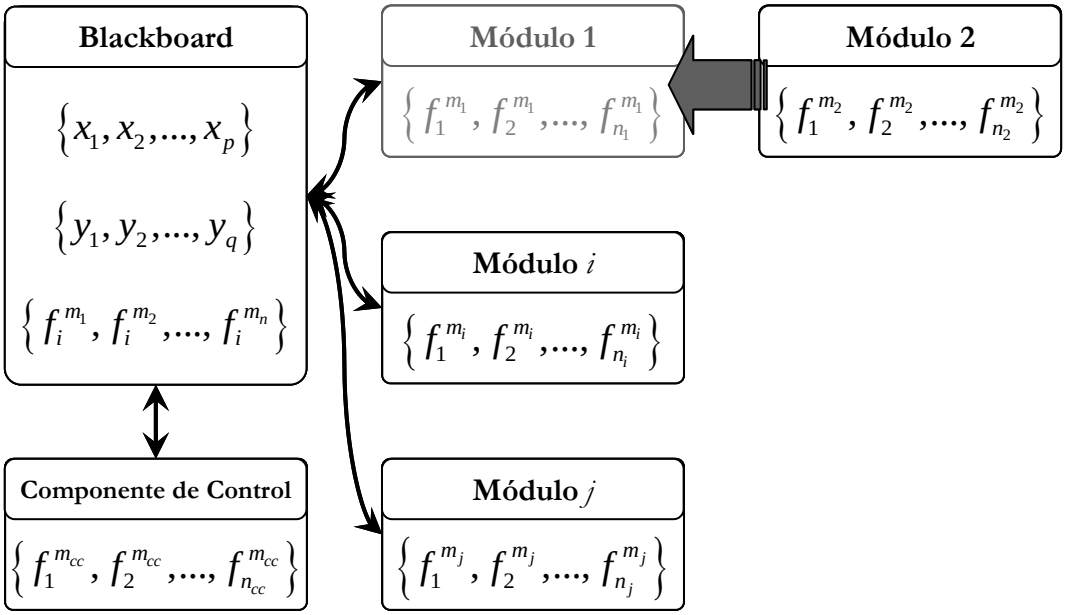


Figura 4.26. Intercambio de módulos equivalentes en *Blackboard*: caso trivial.
Luego de reemplazar el módulo 1 por el módulo 2, ninguna actualización en el sistema es requerida.

Sin embargo, cuando existe discrepancia en los nombres de las funciones o en su firma, puede ser necesario realizar ajustes. Como se describe en la sección 4.3, en arquitectura basada en *Blackboard* hay dos subconjuntos de módulos de principal interés: M_b y M_a, es decir, aquéllos que se comunican con el *Blackboard* y aquéllos que, por estar vinculados a un dispositivo de hardware, no se comunican directamente con el *Blackboard*, sino mediante uno y solamente uno de los otros módulos que sí se conectan con el *Blackboard*.

Para el primer caso y tal como se especifica en la sección 4.3, la ejecución de una función para los módulos que se comunican con el *Blackboard* comienza con una petición de ejecución de dicha función, la cual posteriormente se encargará tanto de leer los datos que requiere del *Blackboard*, como de escribir sus resultados en el *Blackboard*. Por lo anterior, un cambio en la firma de una función no afecta a los demás componentes (ya que esta gestiona por sí misma su acceso a la información y puede trabajar sólo con los datos existentes en el *Blackboard*). Aunque un cambio en el nombre de la función o en su tiempo de ejecución sí.

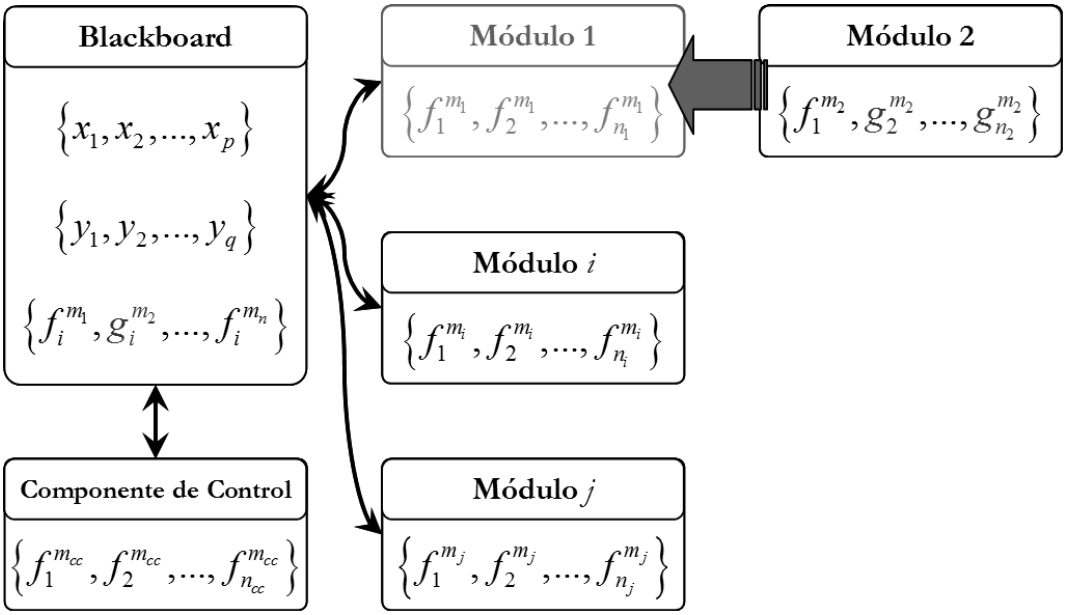


Figura 4.27. Intercambio de módulos equivalentes en *Blackboard*: actualización requerida.
Luego de reemplazar el módulo 1 por el módulo 2, se requiere actualizar el blackboard y los módulos *i*, *j*, y *k*.

Debido a que el tiempo límite de ejecución de una función es gestionado por el componente de control, se requiere una actualización en éste por cada función que cambie su tiempo de ejecución, por lo que en el peor caso se tiene $O(|\{f_j^{m_1}\}|) = O(N)$, donde $N = |\{f_j^{m_1}\}|$.

Para resolver el problema de cambio de nombre se propone el algoritmo siguiente:

```
Para cada función f_i^{m_1} ∈ m_1
     Para cada módulo m_j ∈ D(f_i^{m_1})
          Para cada petición de ejecución de f_i^{m_1} en m_j
               Cambiar nombre de f_i^{m_1} por el nombre de f_k^{m_2}
          Fin para cada
     Fin para cada
Fin para cada
```

Se tiene que $(|\{f_j^{m_1}\}|)(|D(f_i^{m_1})|)\left(\left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$, lo cual da la idea de una complejidad de orden cúbico. Para calcular la complejidad se toma:

$$n = min\left(|\{f_j^{m_1}\}|, |D(f_i^{m_1})|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

$$N = max\left(|\{f_j^{m_1}\}|, |D(f_i^{m_1})|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

Finalmente, para la actualización del sistema dado una discrepancia en la firma de una función se tienen las cotas $\Omega(n^3)$ y $O(N^3)$

Para el segundo caso, correspondiente al subconjunto M_a, debido a que estos módulos se conectan con uno y sólo uno de los módulos de $m_k \in M_b$, de manera directa, hay que actualizar cada llamada en m_k a $f_i^{\mathrm{m}_1}$ por su equivalente a $f_j^{m_2}$.

Se propone el algoritmo siguiente:

```
Para cada función f_i^{m_1} ∈ m_1
     Para cada petición de ejecución de f_i^{m_1} en m_k
          Actualizar petición de ejecución de f_i^{m_1}
          por su equivalente f_j^{m_2}
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|\{f_j^{m_1}\}\right|\right)\left(\left|m_k \xrightarrow{call} f_i^{m_1}\right|\right)$ lo cual da la idea de una complejidad de orden cuadrático. Para calcular la complejidad se toma:

$$n = min\left(\left|\{f_j^{m_1}\}\right|, \left|m_k \xrightarrow{call} f_i^{m_1}\right|\right)$$

$$N = max\left(\left|\{f_j^{m_1}\}\right|, \left|m_k \xrightarrow{call} f_i^{m_1}\right|\right)$$

Por lo que, para la actualización del sistema dado un cambio en el tiempo de ejecución de una función se obtienen las cotas $\Omega(n^2)$ y $O(N^2)$.

Sin embargo hasta el momento no se ha tomado provecho del *Blackboard* como componente centralizado. Como se describe en la sección 4.3 bajo la arquitectura propuesta, la ejecución de las funciones depende del *Blackboard*, por lo que si éste permite la existencia de seudónimos o alias para las variables, los módulos pueden referirse a la misma función por diferentes nombres. Incluso es posible que el *Blackboard* controle los tiempos de ejecución de las funciones que se ejecuten en el sistema, almacenando como parte de sus variables los tiempos mínimos y máximos de ejecución de éstas.

Bajo este esquema, las actualizaciones mencionadas anteriormente al resto del sistema no serían necesarias. Bastaría con realizar las modificaciones pertinentes en el *Blackboard* para que el sistema se adapte al nuevo módulo.

Sea un módulo m_2 que reemplaza al módulo m_1. Se actualiza o agrega un alias en el *Blackboard* para cada $f_i^{\mathrm{m}_1}$ por su equivalente a $f_j^{m_2}$. Se propone el algoritmo siguiente:

```
Para cada función f_i^{m_1} ∈ m_1
     Actualizar/agregar f_j^{m_2} como alias de f_i^{m_1} en Blackboard
     Actualizar tiempos de ejecución de f_i^{m_1} en Blackboard
Fin para cada
```

Esta estrategia reduce considerablemente las actualizaciones al sistema a orden lineal, y su validez se extiende para los módulos que interactúan directamente con el hardware. Finalmente si se define $n = N = \left|\{f_j^{m_1}\}\right|$, entonces para el intercambio de módulos equivalentes se tienen las cotas $\Omega(n)$ y $O(N)$.

CAMBIO DE LA DEFINICIÓN DE LAS FUNCIONES DE UN MÓDULO

Este escenario contempla el cambio de una función $f_i^{m_1}$ por otra función equivalente $g_i^{m_1}$ que realiza las mismas funciones y que se ejecuta en el mismo módulo m. Formalmente:

Sean f_i^m, g_i^m dos funciones de un módulo m. La función g_i^m puede reemplazar a f_i^m sí y sólo si:

$$f_i^m(\bar{x}) = g_i^m(\bar{y})$$

Hay dos aspectos a considerar:

1. La firma de las funciones (tipo de dato devuelto, y el tipo y orden de los parámetros que recibe).
2. Tiempo de ejecución de la función.

Cuando la firma de la función nueva coincide con la firma anterior, y sus tiempos de ejecución cumplen $t(g_i^m) \lesssim t(f_i^m)$, el caso es trivial, pues las funciones son completamente compatibles y no hay que hacer ningún cambio en el sistema.

Nuevamente, tanto la adquisición de datos como la escritura de las variables de resultado son operaciones ejecutadas por la función misma, por lo que un cambio en la firma de la misma no afecta a los otros componentes del sistema, salvo quizá al componente de control para adaptar los permisos de lectura/escritura de dicha función (si así estuviese implementado). Por lo que la complejidad sería $O(1)$ para un cambio en la definición de una función de un módulo.

Si el cambio se diera en el tiempo de ejecución de la función, es necesario actualizar el tiempo máximo de ejecución de la función en el componente de control, lo cual también tiene una complejidad de $O(1)$.

4.3.2. Reestructuración

En este apartado se analiza el costo de realizar los cambios descritos en la sección 4.1 al software de un robot en arquitectura *Blackboard*, en lo concerniente a reestructuración. Es decir, se analiza el impacto a nivel sistema de realizar alguno de los siguientes cambios:

- ✓ Relocalización de una función.
- ✓ Especialización de una función.
- ✓ Generalización de una función.
- ✓ Relocalización de un módulo.
- ✓ Especialización de un módulo.
- ✓ Generalización de un módulo.

A diferencia de una arquitectura *Peer-to-Peer*, en una arquitectura basada *Blackboard* tal como se describe en la sección 2.3.2, toda descomposición involucra al menos un recurso compartido: el *Blackboard*. Además es jerárquica, siendo el *Blackboard* el primer nivel de la jerarquía y el módulo de control de actuador (si lo hubiere) el último elemento en la jerarquía. Por esta razón sólo se analiza un caso para la generalización y especialización. El análisis mencionado refleja el costo o complejidad que tendría actualizar los demás componentes del sistema cuando se presentan los cambios descritos.

RELOCALIZACIÓN DE UNA FUNCIÓN

Este escenario contempla la relocalización de una función $f_i^{m_1}$ del módulo m_1 al módulo m_2, analizándose el impacto en el sistema de mover $f_i^{m_1}$ de un módulo a otro.

Formalmente:

$$f_i^{m_1} \xrightarrow{\text{move}} f_j^{m_2}$$

donde:

$$f_i^{m_1}(\bar{x}) = f_j^{m_2}(\bar{x})$$

Ejemplificando en la Figura 4.28.

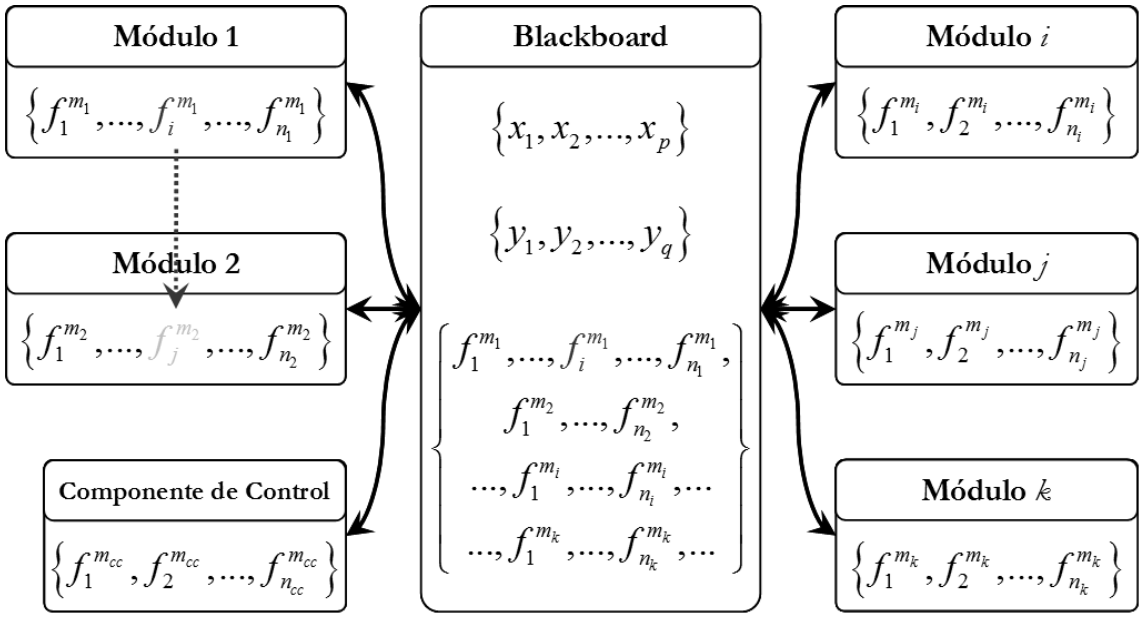


Figura 4.28. Relocalización de una función en arquitectura *Blackboard*: estado previo a la relocalización. Se desea mover la función $f_i^{m_1}$ del módulo m_1 al módulo m_2 como $f_j^{m_2}$.

Es necesario mencionar que el análisis contempla exclusivamente el impacto que la relocalización de una función tiene sobre los demás componentes del sistema, suponiendo que la función "se mueve tal y como está", sin sufrir modificaciones en su nombre, firma, tiempo de ejecución o en el cuerpo mismo de la función. Por esto, si como parte de la relocalización la función requiriera ser adaptada o reprogramada (por ejemplo por un cambio del lenguaje en el que están desarrollados los módulos), aunque esta adaptación o reprogramación consume tiempo de desarrollo, éste es un cambio que debe realizarse independientemente de la arquitectura utilizada, por lo que dicho tiempo no es tomado en cuenta.

Si como consecuencia de la relocalización la firma o tiempo de ejecución de la función resultan afectados, luego de la relocalización funcional debe llevarse a cabo un cambio de la definición de la función, caso estudiado en la sección 4.3.1.

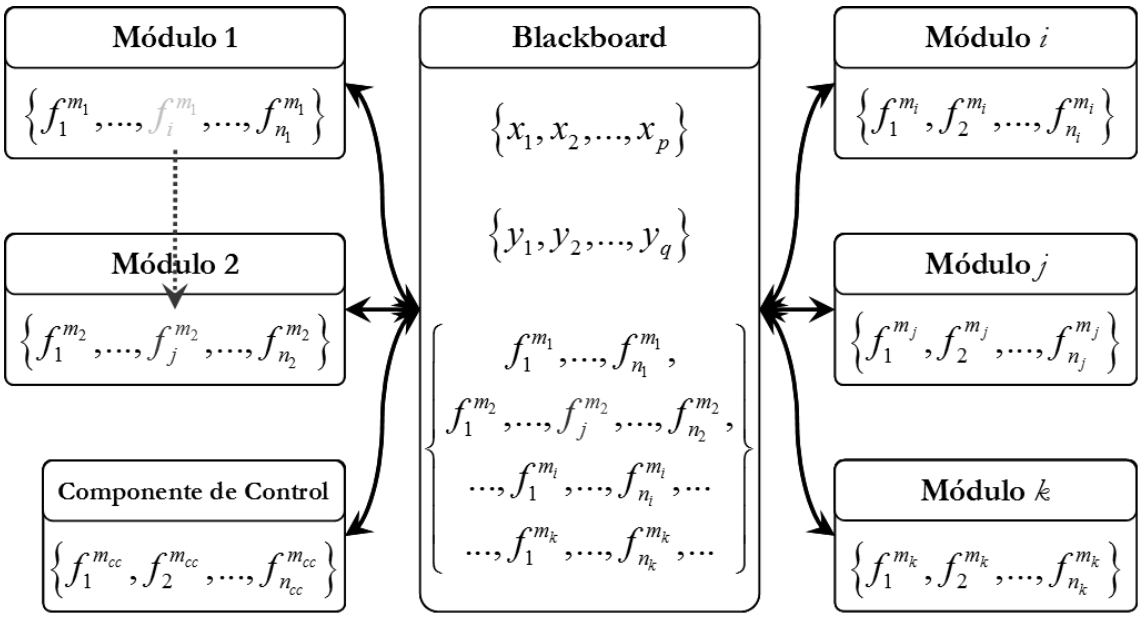


Figura 4.29. Relocalización de una función en arquitectura *Blackboard*: estado posterior a la relocalización. Se ha movido a la función $f_i^{m_1}$ del módulo m_1 al módulo m_2 como $f_j^{m_2}$.

Cuando se realiza la relocalización de una función, es necesario actualizar tanto en el *Blackboard* como en el componente de control los datos concernientes a la nueva ubicación de la función. Se propone el algoritmo siguiente:

```
Actualizar en Blackboard información de f_i^m1 cambiando a f_j^m2
Actualizar en componente de control información de f_i^m1
cambiando a f_j^m2
```

Con sólo dos operaciones requeridas para este cambio, la complejidad de una relocalización funcional es $O(1)$.

ESPECIALIZACIÓN DE UNA FUNCIÓN

Este escenario contempla la división de una función $f_i^{m_1}$ del módulo m_1 en múltiples funciones dentro del mismo módulo m_1, analizándose el impacto en el sistema de descomponer $f_i^{m_1}$ en varias funciones especializadas.

Formalmente:

$$f_i^{m_1}(\bar{x}) = g_1^{m_1}(\bar{x}_1) \circ g_2^{m_1}(\bar{x}_2) \circ \ldots \circ g_n^{m_1}(\bar{x}_n)$$

donde:

- ✓ $f_i^{m_1}$ es la función a descomponer.
- ✓ $\bar{x}$ son los argumentos de la función a descomponer.
- ✓ $g_1^{m_1}, \ldots, g_n^{m_1}$ son funciones especializadas que en conjunto y llamadas en orden realizan la misma función que $f_i^{m_1}$.
- ✓ $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ son los argumentos de las funciones especializadas.

Nótese que existe un caso en el cual $f_i^{m_1}$ es descompuesta en $g_1^{m_1}, \ldots, g_n^{m_1}$, pero se conserva $f_i^{m_1}$ como parte del módulo (aún cuando internamente solicite la ejecución de $g_1^{m_1}, \ldots, g_n^{m_1}$, es decir, sea un alias de $g_1^{m_1}, \ldots, g_n^{m_1}$). En este caso no existe impacto alguno en el sistema, ya que los demás módulos siguen solicitando la ejecución de $f_i^{m_1}$, sin importar cómo realice ésta sus procesos, aunque podría ser necesario verificar los tiempos de ejecución.

El otro caso ocurre cuando $f_i^{m_1}$ es descompuesta en $g_1^{m_1}, \ldots, g_n^{m_1}$ pero $f_i^{m_1}$ deja de estar disponible. Es en este caso cuando se vuelve necesario actualizar a los demás componentes del sistema y el cual es analizado a continuación. Esto se ejemplifica en la Figura 4.30:

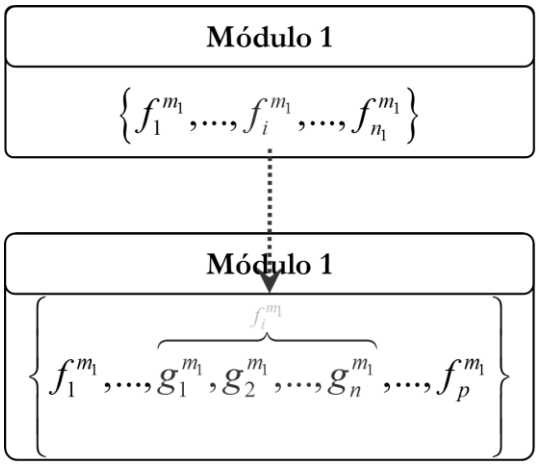


Figura 4.30. Especialización de una función. La función $f_i^{m_1}$ es reemplazada por las funciones especializadas $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$.

Es decir, la función $f_i^{m_1}$ es reemplazada por las funciones especializadas $g_1^{m_1}, \ldots, g_n^{m_1}$. Es de suponerse que en el sistema no existe ninguna referencia a las funciones especializadas $g_1^{m_1}, \ldots, g_n^{m_1}$ y sí existen referencias a $f_i^{m_1}$, mismas que pueden ser reemplazadas por una solicitud de ejecución en orden de todas y cada una de las funciones especializadas $g_1^{m_1}, \ldots, g_n^{m_1}$ que sustituyen a $f_i^{m_1}$, tal como se muestra en la Figura 4.31.

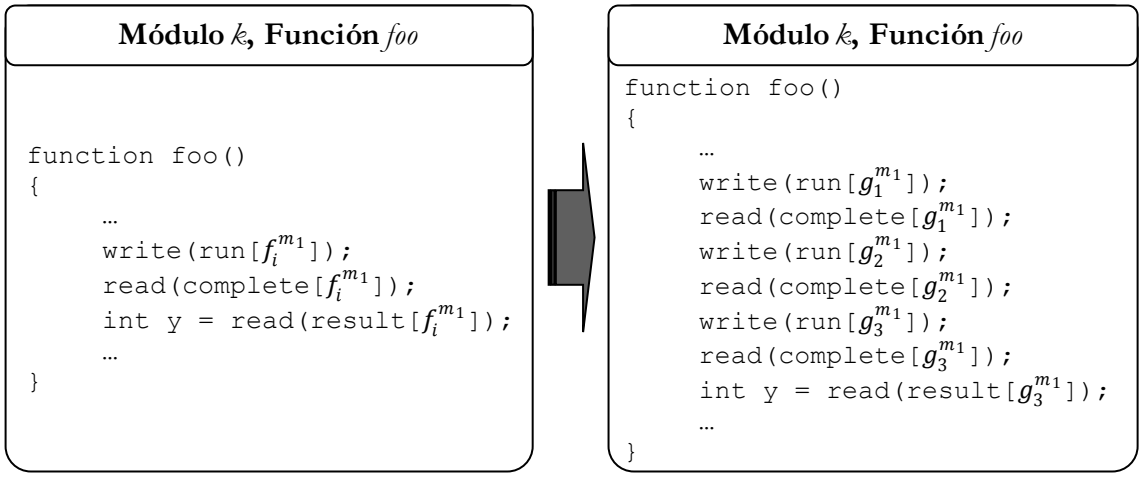


Figura 4.31. Un modo de llamar a la función original a partir
de las funciones especializadas en arquitectura *Blackboard*.
A la izquierda la función foo llama a la función original $f_i^{m_1}$.
A la derecha la función foo llama a las nuevas funciones especializadas g.

Es necesario mencionar que el análisis contempla exclusivamente el impacto que la especialización de una función tiene sobre los demás componentes del sistema, por lo que el tiempo requerido para descomponer la función no se toma en cuenta. Con lo anterior, se propone el siguiente algoritmo para actualizar todas las llamadas a la función que ha sido especializada:

```
Para cada módulo m_j ∈ D(f_i^{m_1})
     Para cada petición de ejecución de f_i^{m_1} en m_j
          Cambiar la petición de ejecución de
          la función f_i^{m_1} por peticiones de
          ejecución de las funciones g_1^{m_1}, …, g_n^{m_1}
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|D\left(f_i^{m_1}\right)\right|\right)\left(\left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$, lo cual da la idea de una complejidad de orden cuadrático. Para calcular la complejidad se toma:

$$n = min\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

$$N = max\left(\left|D\left(f_i^{m_1}\right)\right|, \left|m_j \xrightarrow{call} f_i^{m_1}\right|\right)$$

Finalmente, para la especialización de una función en arquitectura *Blackboard* se tienen las cotas $\Omega(n^2)$ y $O(N^2)$.

GENERALIZACIÓN DE FUNCIONES

Este escenario contempla la unión de varias funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ del módulo m_1 en una sola función $f_i^{m_1}$ del mismo módulo m_1, analizándose el impacto en el sistema de unir las funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ en una función generalizada.

Formalmente:

$$g_1^{m_1}(\bar{x}_1) \circ g_2^{m_1}(\bar{x}_2) \circ \ldots \circ g_n^{m_1}(\bar{x}_n) = f_i^{m_1}(\bar{x})$$

donde:

- ✓ $g_1^{m_1}, \ldots, g_n^{m_1}$ son funciones que se desean unir para formar a $f_i^{m_1}$.
- ✓ $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ son los argumentos de las funciones a generalizar.
- ✓ $f_i^{m_1}$ es la función generalizada.
- ✓ $\bar{x}$ son los argumentos de la función generalizada.

Nótese que existe un caso en el cual las funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ se unen en $f_i^{m_1}$ pero se conservan $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ como parte del módulo (es decir, $f_i^{m_1}$ internamente sólo solicita la ejecución de funciones $g_1^{m_1}, \ldots, g_n^{m_1}$). En este caso no existe impacto alguno en el sistema, ya que los demás módulos siguen haciendo referencia a la correspondiente función $g_1^{m_1}, \ldots, g_n^{m_1}$.

El otro caso ocurre cuando $g_1^{m_1}, \ldots, g_n^{m_1}$ siempre son llamadas en conjunto (de otro modo no tendría sentido una generalización, pues $g_1^{m_1}, \ldots, g_n^{m_1}$ siguen siendo requeridas de manera independiente) y estas múltiples llamadas son reemplazadas por una sola petición de ejecución de $f_i^{m_1}$; por lo que $g_1^{m_1}, \ldots, g_n^{m_1}$ dejan de estar disponibles. Es en este caso cuando se vuelve necesario actualizar a los demás componentes del sistema y el cual será analizado a continuación.

Ejemplificando en la Figura 4.32.

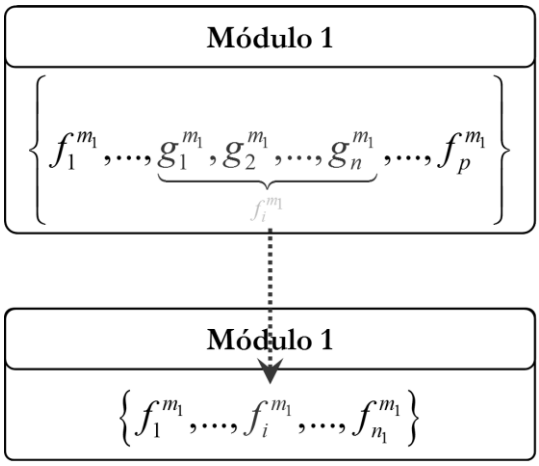


Figura 4.32. Generalización a una función. Las funciones $g_1^{m_1}, g_2^{m_1}, \ldots, g_n^{m_1}$ son reemplazadas por la función generalizada $f_i^{m_1}$.

Es decir, las funciones $g_1^{m_1}, \ldots, g_n^{m_1}$ son reemplazadas por la función generalizada $f_i^{m_1}$. Así pues, toda llamada al conjunto de funciones $g_1^{m_1}, \ldots, g_n^{m_1}$ puede ser sustituido por una sola llamada a $f_i^{m_1}$ tal como se muestra en la Figura 4.33.

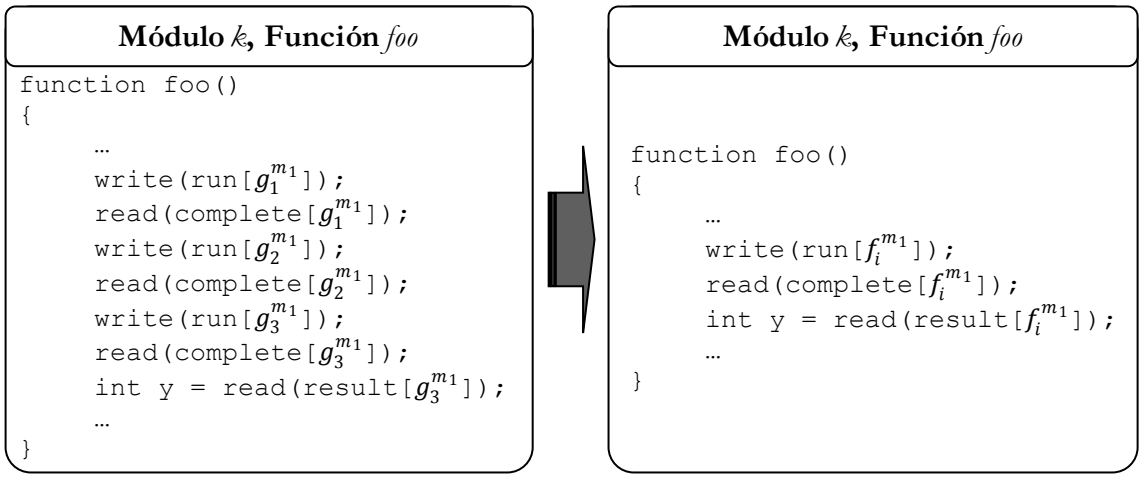


Figura 4.33. Un modo de llamar a las funciones originales a partir de la función generalizada en arquitectura *Blackboard*.
A la izquierda la función foo llama a las funciones originales g.
A la derecha la función foo llama a la nueva función generalizada $f_i^{m_1}$.

Es necesario mencionar que el análisis contempla exclusivamente el impacto que la generalización de una función tiene sobre los demás componentes del sistema, por lo que el tiempo requerido para unir las funciones no se toma en cuenta.

Con lo anterior, se propone el siguiente algoritmo para actualizar todas las llamadas al conjunto de funciones que han sido generalizadas:

```
Para cada módulo m_j ∈ D(g_1^{m_1}, ..., g_n^{m_1})
     Para cada llamada a g_1^{m_1}, ..., g_n^{m_1} en m_j
         Cambiar las llamadas a g_1^{m_1}, ..., g_n^{m_1}
         por una llamada a f_i^{m_1}
     Fin para cada
Fin para cada
```

Se tiene que $\left(\left|D\left(g_1^{m_1},\dots,g_n^{m_1}\right)\right|\right)\left(\left|m_j \xrightarrow{call} \left(g_1^{m_1},\dots,g_n^{m_1}\right)\right|\right)$, lo cual da la idea de una complejidad de orden cuadrático. Para calcular la complejidad se toma:

$$n = min\left(\left|D\left(g_1^{m_1},\dots,g_n^{m_1}\right)\right|, \left|m_j \xrightarrow{call} g_1^{m_1},\dots,g_n^{m_1}\right|\right)$$

$$N = max\left(\left|D\left(g_1^{m_1},\dots,g_n^{m_1}\right)\right|, \left|m_j \xrightarrow{call} g_1^{m_1},\dots,g_n^{m_1}\right|\right)$$

Finalmente, para la especialización de una función en arquitectura *Blackboard* se tienen las cotas $\Omega(n^2)$ y $O(N^2)$.

RELOCALIZACIÓN DE UN MÓDULO

En este apartado se contempla la relocalización de un módulo, es decir, mover el módulo de su contexto actual de ejecución a un nuevo contexto (lo que comúnmente se traduce a un cambio de máquina).

Dependiendo de la tecnología utilizada, la relocalización de un módulo puede ser inmediata o puede requerir cambios que van desde ligeros ajustes hasta una reprogramación completa del módulo involucrado. En esta sección se analiza únicamente el impacto que tiene en el sistema la relocalización de un módulo y el costo de la actualización del sistema luego de dicha relocalización, por lo que el costo de adaptar el módulo a migrar al nuevo contexto no se toma en cuenta.

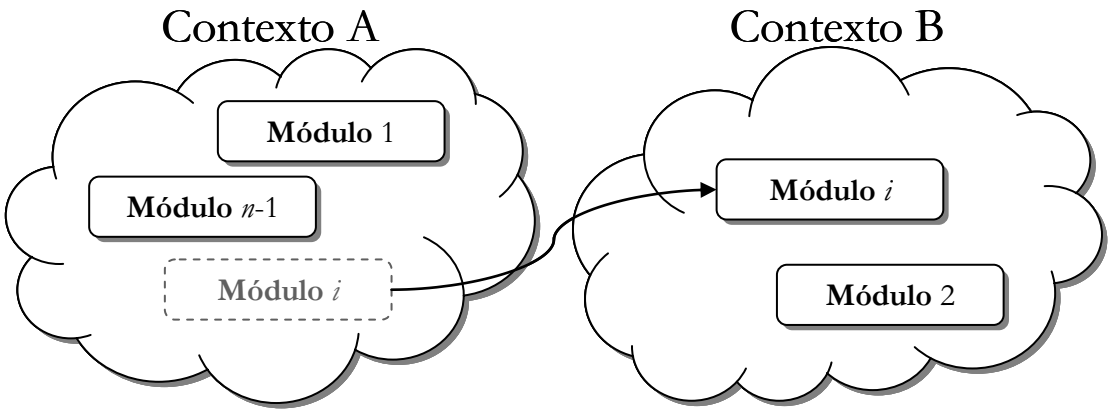


Figura 4.34. Cambio de contexto de un módulo.
El módulo *i* se mueve del contexto A al contexto B.

Para la relocalización de un módulo existen dos escenarios a contemplar: cuando se utiliza un servicio de resolución de nombres (provisto por la tecnología utilizada, *Framework* o *middleware*) encargado de gestionar el acceso a los componentes con independencia de su ubicación; y cuando dicho servicio no existe y cada componente debe conocer la ubicación de los demás componentes del sistema con los que interacciona.

A fin de simplificar el estudio del impacto de una relocalización, se considera que cada módulo en el sistema cuenta con un nombre y un punto de acceso (que puede verse reflejado como una *url*, un *pipe*, dirección IP y puerto, etc.) que permite al módulo comunicarse con el *Blackboard*.

Independientemente de si la tecnología utilizada provee un servicio de resolución de nombres o no, todos los módulos se conectan única y exclusivamente con el *Blackboard* (o con uno y sólo un módulo intermediario que los opera y que sí se comunica con el *Blackboard*). Por lo que bastará con actualizar el punto de acceso al módulo relocalizado en el servicio de resolución de nombres, el *Blackboard* o el módulo intermediario según sea el caso. Por lo anterior, la relocalización de un módulo en arquitectura *Blackboard* no afecta al sistema más allá de este cambio, quedando la complejidad en $O(1)$.

ESPECIALIZACIÓN DE UN MÓDULO

Este apartado contempla la especialización de un módulo, descomponiéndolo en varios sub-módulos tal como se ilustra en la Figura 4.35.

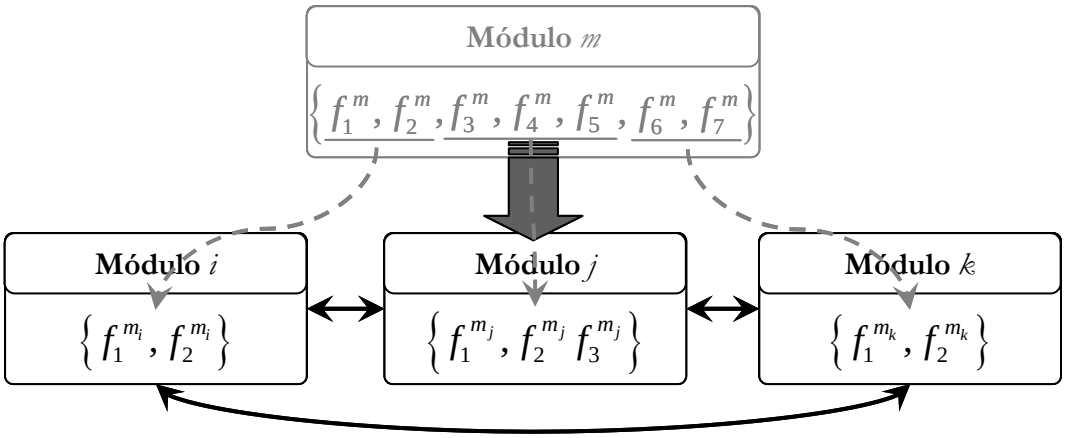


Figura 4.35. Especialización de un módulo.
El módulo m se descompone en los sub-módulos especializados i, j y k.

Supóngase que los módulos m_1, m_2, m_3 utilizan las funciones provistas por el módulo m que es descompuesto en los módulos especializados m_i, m_j y m_k, tal como se ilustra en la Figura 4.36.

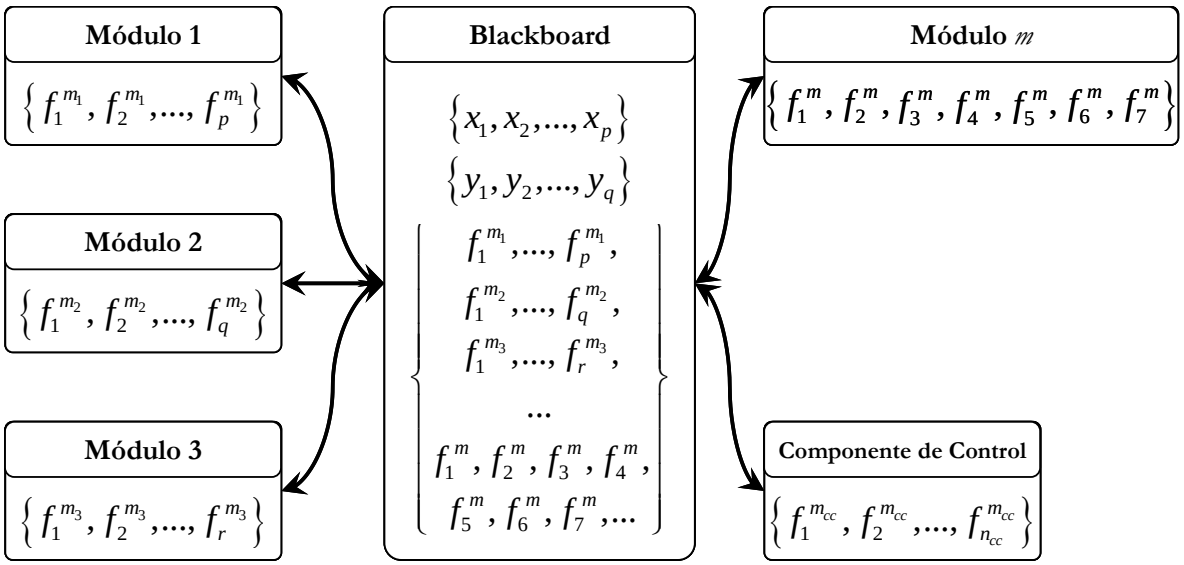


Figura 4.36. Estado del subsistema formado por el módulo m y sus interdependencias previo a la especialización del módulo m en los sub-módulos especializados m_i, m_j y m_k.

En este supuesto, tras la especialización del módulo m se tiene que actualizar el componente de control con la información de la nueva ubicación de las funciones que se han desplazado del módulo original m a los módulos m_1, m_2 y m_3, como si estas funciones se hubieran "movido" a otro módulo (relocalización de funciones), tal como se ejemplifica en la Figura 4.37.

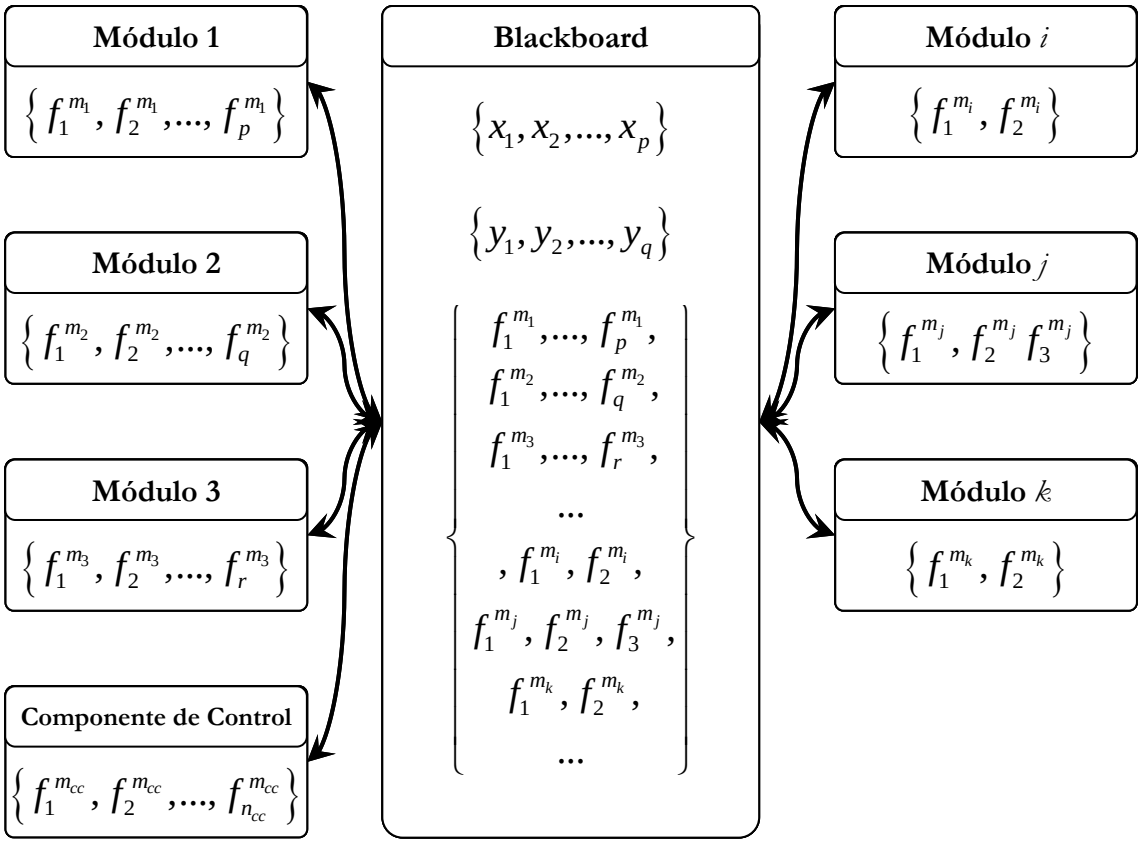


Figura 4.37. Estado del subsistema formado por los módulos m_i, m_j, m_k y sus interdependencias, posterior a la especialización del módulo m, el cual ha sido reemplazado por los sub-módulos especializados m_i, m_j y m_k .

Para simplificar el análisis formal del impacto de esta operación de especialización podemos suponer que se crean tantos nuevos módulos vacíos (sin funciones), como el número de módulos a especializar menos uno. Después se "mueven" las funciones del módulo original a los nuevos módulos creados (los que inicialmente estaban vacíos) hasta que se tiene la especialización deseada.

Con esto se presentan las siguientes definiciones:

- ✓ $F_R \subset m$ es el conjunto de funciones a relocalizar.
- ✓ M_E es el conjunto de módulos en los que se divide el módulo m tras su especialización.

Luego entonces, formalmente:

$$\forall f_i \in F_R, \qquad f_j^m \xrightarrow{move} f_k^{m_i}$$

donde:

$$f_j^m(\bar{x}) = f_k^{m_i}(\bar{x})$$

y

$$m_i \in M_E$$

Es decir, se tienen $|F_R|$ funciones a relocalizar, lo cual como se estudia en la sección Relocalización de una función, tiene una complejidad de $O(1)$.

Además, es de suponerse que durante una especialización se mueven el menor número posible de funciones, por lo que si un módulo con diez funciones es descompuesto en dos módulos especializados con dos y ocho funciones respectivamente, es natural pensar que solamente 2 funciones se mueven. Así, en el peor escenario donde un módulo m se descompondrá en k sub-módulos cada uno con el mismo número de funciones, se mueven un máximo de $\frac{|m|}{k-1}$ funciones, por lo que $|F_R| \leq \frac{|m|}{k-1}$.

Tomando las cotas mencionadas anteriormente para la relocalización de funciones y el valor $|F_R|$, se tienen finalmente que el costo por especializar un módulo en una arquitectura basada en *Blackboard* es de $O(|F_R|)$.

GENERALIZACIÓN DE UN MÓDULO

Este apartado contempla la unión de varios módulos en un nuevo súper-módulo generalizado tal como se ilustra en laFigura 4.38.

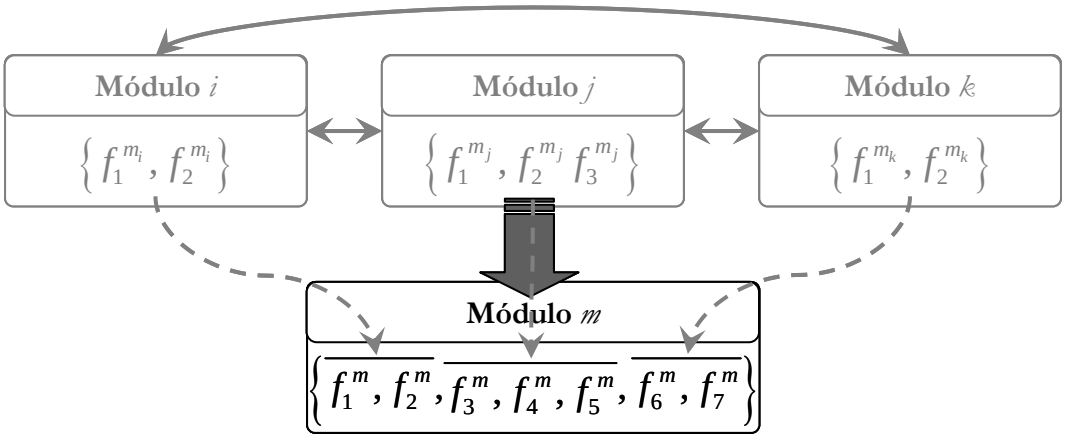


Figura 4.38. Generalización de módulos independientes. Los módulos m_i, m_j y m_k se unen en un solo módulo m.

Supóngase los módulos m_1, m_2, m_3 utilizan las funciones provistas por los módulos m_i, m_j y m_k que serán unidos para conformar el nuevo módulo m, tal como se ilustra en la Figura 4.39.

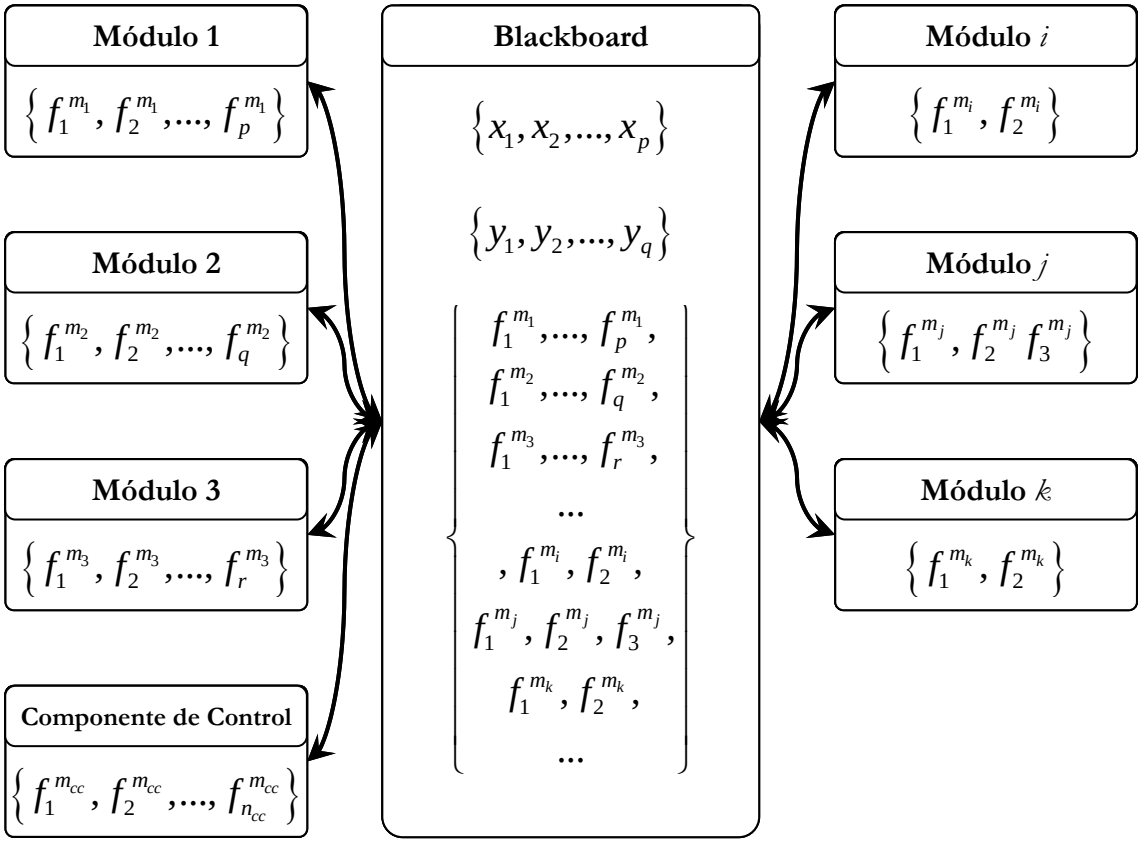


Figura 4.39. Estado del subsistema formado por los módulos m_i, m_j, m_k y sus interdependencias previo a la generalización. Se desea unir los módulos m_i, m_j y m_k en un solo módulo m.

En este supuesto, tras la generalización se tiene que actualizar el componente de control con la información de la nueva ubicación de las funciones que se han desplazado de los módulos m_i, m_j y m_k al nuevo módulo generalizado m, como si estas funciones se hubieran "movido" a otro módulo (relocalización de funciones), tal como se ejemplifica en la Figura 4.40.

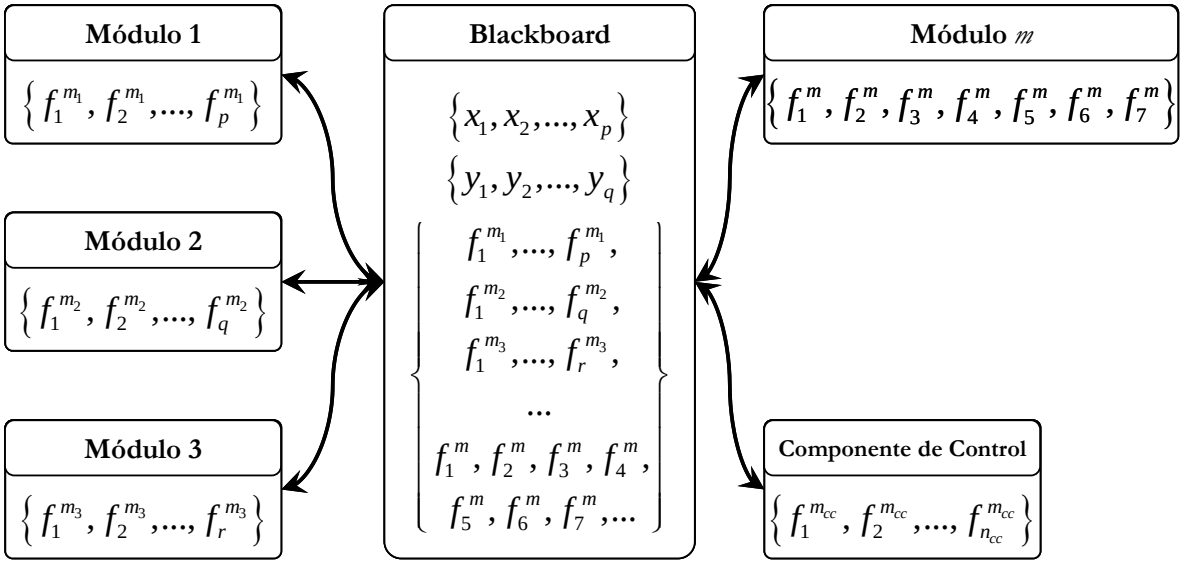


Figura 4.40. Estado del subsistema formado por el módulo m y sus interdependencias, posterior a la generalización de los módulos m_i, m_j, m_k.

Para simplificar el análisis formal del impacto de esta operación de generalización podemos suponer que se escoge al módulo con mayor número de funciones del conjunto de módulos que se desea unir como base; y después se "mueven" las funciones de los otros módulos al módulo base hasta que se tiene la generalización deseada.

Con esto se presentan las siguientes definiciones:

- ✓ M_E es el conjunto de módulos especializados que se unen para conformar al nuevo módulo generalizado.
- ✓ m_0 es el módulo escogido como módulo base; es decir, el que posteriormente es el módulo generalizado.
- ✓ $F_R \subset M_E - \{m_0\}$ es el conjunto de funciones a relocalizar.

Luego entonces, formalmente:

$$\forall f_j \in F_R, \qquad f_j^{m_i} \xrightarrow{move} f_k^{m_0}$$

donde:

$$f_j^{m_i}(\bar{x}) = f_k^{m_0}(\bar{x})$$

y

$$m_i \in M_E - \{m_0\}$$

Es decir, se tienen $|F_R|$ funciones a relocalizar, lo cual como se estudia en la sección Relocalización de una función, tiene una complejidad de $O(1)$.

Además, es de suponerse que durante una generalización se mueven el menor número posible de funciones, por lo que si se desean unir dos módulos con dos y ocho funciones respectivamente, es natural pensar que solamente 2 funciones se mueven. Así, en el peor escenario donde k módulos cada uno con el mismo número de funciones se unen en un nuevo súper-módulo, se mueven un máximo de $\frac{|m|}{k-1}$ funciones, por lo que $|F_R| \leq \frac{|m|}{k-1}$.

Tomando las cotas mencionadas anteriormente para la relocalización de funciones y el valor $|F_R|$, se tiene finalmente que el costo por generalizar varios módulos en una arquitectura basada en *Blackboard* es de $O(|F_R|)$.

4.4. Análisis comparativo

A continuación se presenta la Tabla 4.1, en la que se comparan las cotas inferior y superior del costo o dificultad de realizar actualizaciones o modificaciones al sistema de software del robot debido a cambios en alguno de sus componentes, según lo analizado en las secciones anteriores.

Característica		***Peer-to-Peer***		***Blackboard***	
		Cota inferior	**Cota superior**	**Cota inferior**	**Cota superior**
Extensibilidad	Intercambio de módulos equivalentes	$\Omega(n^3)$	$O(N^3)$	$\Omega(n)$	$O(N)$
	Cambio de la definición de una función	$\Omega(n^2)$	$O(N^2)$	$\Omega(1)$	$O(1)$
Reestructuración	Relocalización de una función	$\Omega(n^2)$	$O(N^2)$	$\Omega(1)$	$O(1)$
	Especialización de una función	$\Omega(n^2)$	$O(N^2)$	$\Omega(n^2)$	$O(N^2)$
	Generalización de funciones	$\Omega(n^2)$	$O(N^2)$	$\Omega(n^2)$	$O(N^2)$
	Relocalización de un módulo	$\Omega(n)$	$O(n)$	$\Omega(1)$	$O(1)$
	Especialización de un módulo (jerárquica)	0	0	$\Omega(\lvert F_R \rvert)$	$O(\lvert F_R \rvert)$
	Especialización de un módulo (independientes)	$\Omega(\lvert F_R \rvert \cdot n^2)$	$O(\lvert F_R \rvert \cdot N^2)$	-	-
	Generalización de varios módulos (jerárquica)	0	0	$\Omega(\lvert F_R \rvert)$	$O(\lvert F_R \rvert)$
	Generalización de varios módulos (independientes)	$\Omega(\lvert F_R \rvert \cdot n^2)$	$O(\lvert F_R \rvert \cdot N^2)$	-	-

Tabla 4.1. Análisis comparativo de extensibilidad y reestructuración entre arquitecturas *Peer-to-Peer* y basada en *Blackboard*.

Considerando

- ✓ n, N el menor y mayor número de cambios primitivos o atómicos necesarios para realizar una actualización, respectivamente.
- ✓ $|F_R|$ como el número mínimo necesario de funciones a relocalizar para realizar la actualización (si aplica).

Es importante hacer notar que las n y N mostradas en la Tabla 4.1 no necesariamente tienen el mismo valor para cada característica mostrada, tal como se muestra a continuación en la Tabla 4.2

Característica		***Peer-to-Peer***	***Blackboard***
Extensi-bilidad	Intercambio de módulos equivalentes	$n = min\left(\left\|\left\{f_j^{m_k}\right\}\right\|, \left\|D\left(f_i^{m_k}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_k}\right\|\right)$ $N = max\left(\left\|\left\{f_j^{m_k}\right\}\right\|, \left\|D\left(f_i^{m_k}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_k}\right\|\right)$	$n = N = \left\|\left\{f_j^{m_1}\right\}\right\|$
	Cambio de la definición de una función	$n = min\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$ $N = max\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$	N/A
Reestruc-turación	Relocalización de una función	$n = min\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$ $N = max\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$	N/A
	Especialización de una función	$n = min\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$ $N = max\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$	
	Generalización de funciones	$n = min\left(\left\|D\left(g_1^{m_1}, \dots, g_n^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} g_1^{m_1}, \dots, g_n^{m_1}\right\|\right)$ $N = max\left(\left\|D\left(g_1^{m_1}, \dots, g_n^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} g_1^{m_1}, \dots, g_n^{m_1}\right\|\right)$	
	Relocalización de un módulo	$n = N = \left\|D(m_1)\right\|$	N/A
	Especialización de un módulo (jerárquica)	N/A	
	Especialización de un módulo (independientes)	$n = min\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$ $N = max\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$	N/A
	Generalización de varios módulos (jerárquica)	N/A	
	Generalización de varios módulos (independientes)	$n = min\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$ $N = max\left(\left\|D\left(f_i^{m_1}\right)\right\|, \left\|m_j \xrightarrow{call} f_i^{m_1}\right\|\right)$	N/A

Tabla 4.2. Valores de n y N utilizadas para el análisis comparativo de extensibilidad y reestructuración entre arquitecturas *Peer-to-Peer* y basada en *Blackboard*.

En la Tabla 4.1 puede observarse que, en general, una arquitectura basada en *Blackboard* requiere de menos actualizaciones cada vez que uno de sus componentes cambia o, en el peor de los casos, tantas actualizaciones como una arquitectura basada en *Peer-to-Peer*.

Para los casos de especialización de un módulo con estructura jerárquica y generalización de módulos con estructura jerárquica, donde una arquitectura basada en *Blackboard* parece requerir de más cambios que arquitectura basada en *Peer-to-Peer*, es necesario contemplar que aunque la especialización/generalización de módulos con estructura jerárquica bajo *Peer-to-Peer* es inmediata. No es común para este tipo de arquitectura como lo es la especialización o generalización de módulos independientes (las cuales no existen en arquitecturas basadas en *Blackboard*). Si realizamos esta comparativa, una arquitectura basada en *Blackboard* ofrece menor cantidad de cambios al sistema.

Con base en lo anterior se presenta la Tabla 4.3, que resume la Tabla 4.1, presentando las cotas superiores para el mejor caso, caso promedio y peor caso de cada una de las características analizadas para extensibilidad y reestructuración en ambas arquitecturas.

Característica	***Peer-to-Peer***			***Blackboard***		
	Mejor caso	**Caso promedio**	**Peor caso**	**Mejor caso**	**Caso promedio**	**Peor caso**
Extensibilidad	$O(N^2)$	$O(N^3)$	$O(N^3)$	$O(1)$	$O(N)$	$O(N)$
Reestructuración	$O(N^2)$	$O(\|F_R\|N^2)$	$O(\|F_R\|N^2)$	$O(1)$	$O(\|F_R\|)$	$O(N^2)$

Tabla 4.3. Resumen de análisis comparativo de extensibilidad y reestructuración entre arquitecturas *Peer-to-Peer* y basada en *Blackboard*.

En la Tabla 4.3 se descarta la especialización/generalización modular jerárquica de la arquitectura *Peer-to-Peer*, por considerarse poco usual, y para los casos promedios se toma el costo de actualización de la operación más común: intercambio de módulos equivalentes para extensibilidad y especialización modular para reestructuración.

Finalmente, bajo los criterios explicados anteriormente, la Tabla 4.3 muestra mayor extensibilidad y reestructuración para una arquitectura basada en *Blackboard*.

4.5. Especialización del módulo Visión: un caso de ejemplo

En la presente sección se aterriza el análisis realizado en las secciones anteriores de este capítulo mediante un ejemplo de aplicación a un caso práctico. En este ejemplo se calcula el costo de actualización del sistema debido a la especialización de un módulo para ambas arquitecturas *Peer-to-Peer* y *Blackboard*.

El escenario inicial propuesto contempla la existencia de cuatro módulos en el sistema que se ejecuta en una misma computadora (sólo se muestran los módulos utilizados en el ejemplo independientemente de la arquitectura): un planeador de acciones (ACT-PLN), un planeador de tareas simples (ST-PLN), un módulo de control de cabeza (HEAD) y un módulo de visión (VISION). Cada uno de estos módulos ejecuta al menos una función, tal como se muestra en la Figura 4.41.

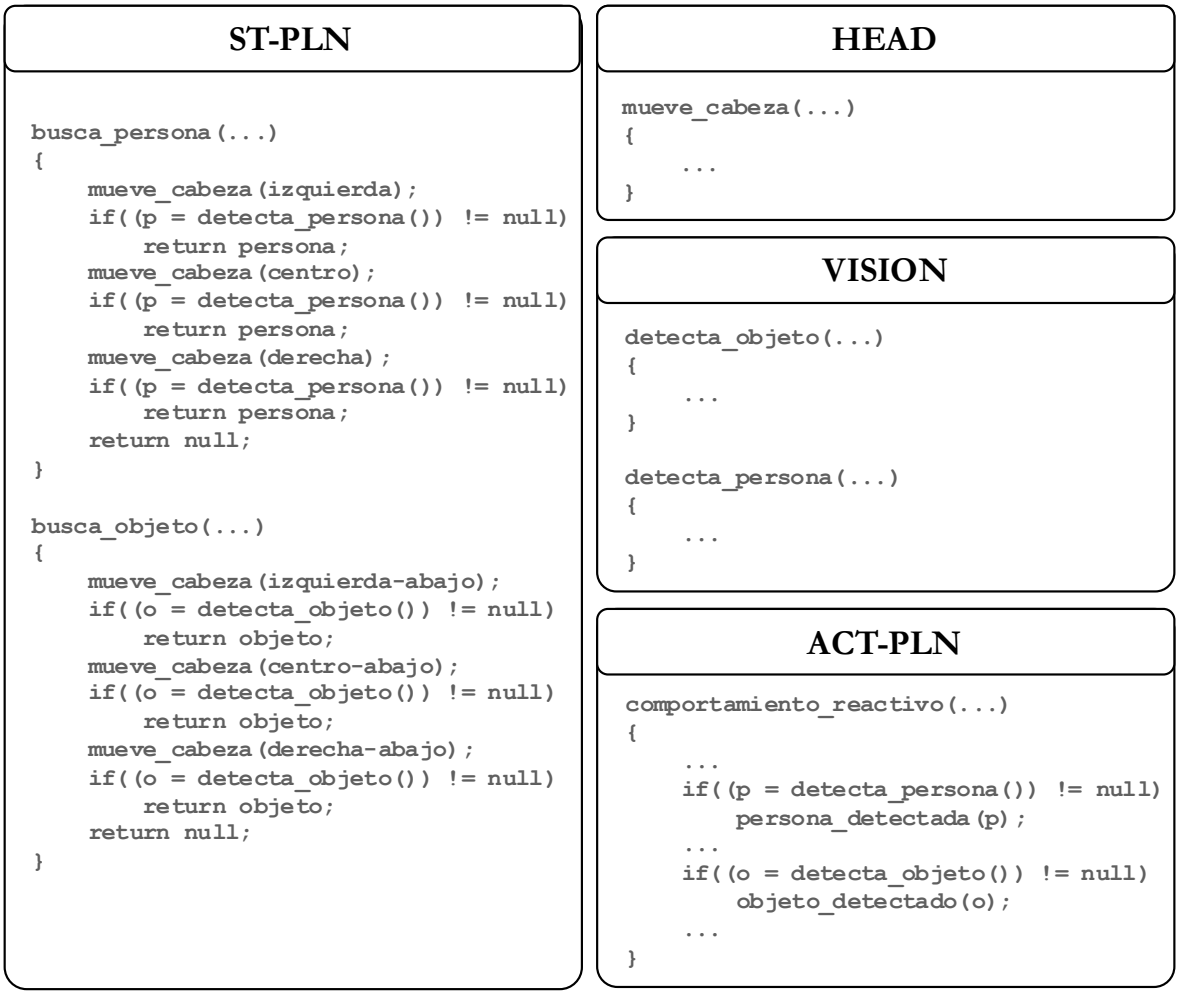


Figura 4.41. Escenario inicial propuesto (previo a la especialización de VISION).
Los módulos ACT-PLN y ST-PLN dependen de las funciones del módulo VISION.

En el escenario final propuesto, el modulo visión se ha descompuesto en un detector de personas que conserva el mismo nombre (VISION) y un detector de objetos (OBJ-FND) que ejecuta su función *detecta_objeto* más rápido. Además, el módulo detector de objetos se ejecuta en una segunda computadora por separado, tal como se muestra en la Figura 4.42.

Para lograr esto, el módulo VISION ha pasado por un proceso de especialización modular para dar lugar a un nuevo módulo llamado OBJ-FND que se ejecutará en otra computadora, es decir, es relocalizado (reestructuración: véanse secciones 4.2.2 y 4.3.2). Además, la función ha sufrido un cambio en su tiempo de ejecución, lo que implica un cambio en la definición de una función (extensibilidad) tal como se describe en las secciones 4.2.1 y 4.3.1. Ante estos cambios, el sistema necesita actualizarse a fin de permanecer operativo.

Nótese que el tiempo requerido para dividir el módulo inicial VISION en los módulos VISION y OBJ-FND no se toma en cuenta. Esto es debido a que el tiempo que toma especializar un módulo no depende de la arquitectura sino del módulo en sí.

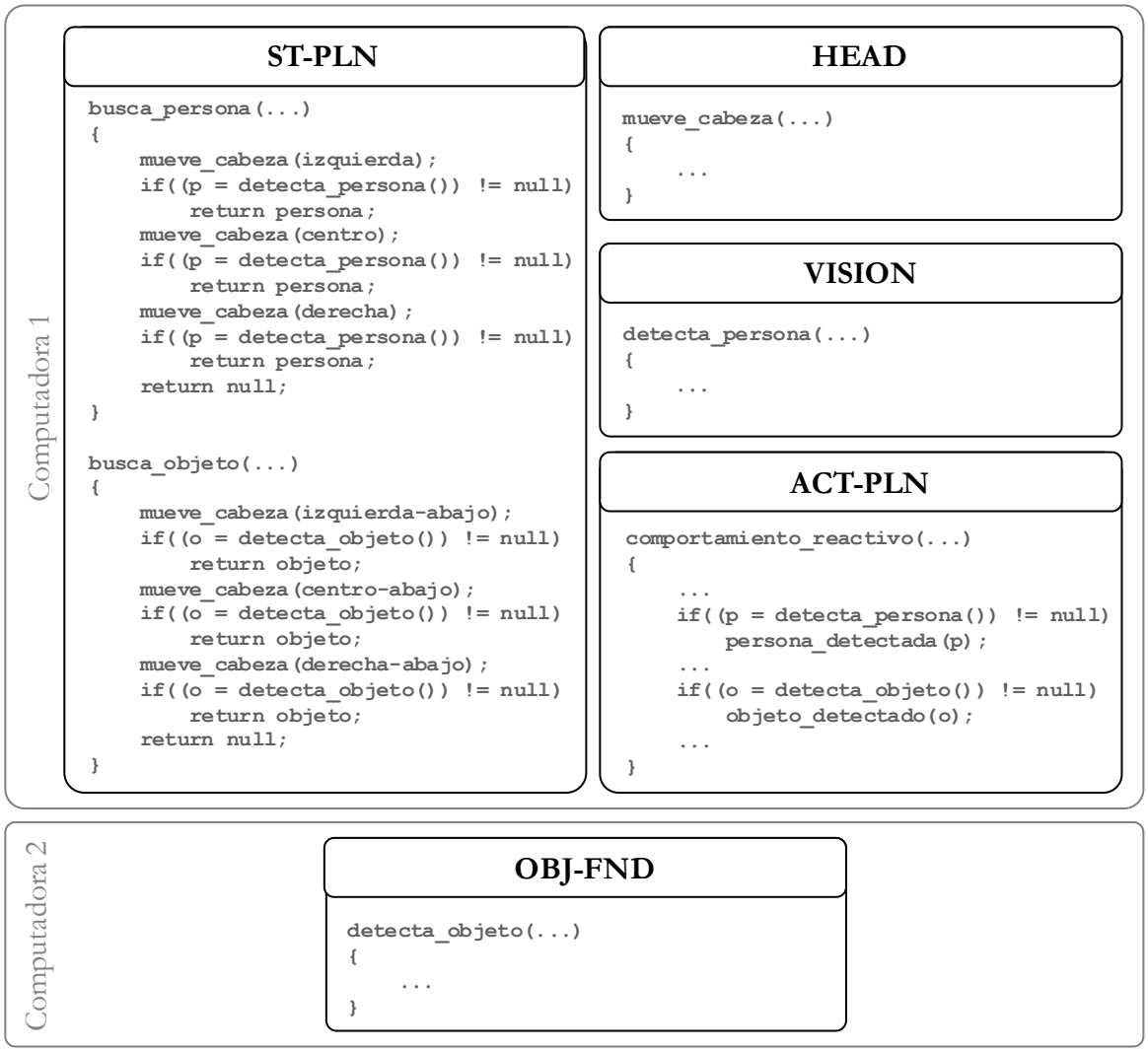


Figura 4.42. Escenario final propuesto (posterior a la especialización de VISION).
Ahora los módulos ACT-PLN y ST-PLN dependen de las funciones de los módulos VISION y OBJ-FND.

Como se menciona en las secciones 4.2.2 y 4.3.2, cuando se especializa un módulo, la operación puede verse como mover a un módulo vacío el menor subconjunto de funciones a relocalizar y luego (si aplica) renombrar el módulo original. Dado que en este ejemplo el módulo VISION sólo tiene 2 funciones, se considera que la función *detecta_objeto* se mueve al módulo originalmente vacío OBJ-FND, quedando en el módulo VISION únicamente la función *detecta_persona*. Posteriormente en cada módulo consumidor de la función *detecta_objeto* (los módulos ST-PLN y ACT-PLN) se debe actualizar el tiempo de ejecución de la misma. Esto es:

1. Actualizar en el sistema la pertenencia de *detecta_objeto* al módulo OBJ-FND.
2. Actualizar en el sistema la ubicación del módulo OBJ-FND.
3. Actualizar en el sistema el tiempo de ejecución de la función *detecta_objeto*.

4.5.1. Especialización del módulo Visión en arquitectura *Peer-to-Peer*

De acuerdo con el análisis realizado en "Especialización de un módulo a módulos independientes" de la sección 4.2.2, los cambios a realizar en el sistema tras descomponer el módulo inicial VISION en los módulos VISION y PRS-FND es equivalente a mover la función *detecta_objeto* a un nuevo módulo vacío.

De acuerdo al algoritmo propuesto en "Relocalización de una función" de la sección 4.2.2, el algoritmo para descomponer VISION es:

```
Para cada función a relocalizar de f_i^VISION
     Para cada módulo m_j ∈ D(f_i^VISION)
          Para cada llamada a f_i^VISION en m_j
               Cambiar la llamada a f_i^VISION
               por una llamada a f_j^OBJ−FND
          Fin para cada
     Fin para cada
Fin para cada
```

La función a relocalizar es una: *detecta_objeto*, por lo que el algoritmo siguiente se ejecuta sólo una vez:

```
Para cada módulo m_j ∈ D(detecta_objeto^VISION)
     Para cada llamada a detecta_objeto^VISION en m_j
          Cambiar la llamada a detecta_objeto^VISION
          por una llamada a detecta_objeto^OBJ−FND
     Fin para cada
Fin para cada
```

Las dependencias de la función $detecta\_objeto^{\mathrm{VISION}}$ representadas por la función $D(detecta\_objeto^{\mathrm{VISION}})$ corresponden únicamente a los módulos ACT-PLN y ST-PLN, es decir:

$$D(detecta\_objeto^{\mathrm{VISION}}) = \{\mathrm{ACT-PLN}, \mathrm{ST-PLN}\}$$

Ahora, analizando el pseudocódigo mostrado en la Figura 4.41, ACT-PLN realiza una llamada a *detecta_objeto*, mientras que ST-PLN realiza 3. Realizando la suma, actualizar el sistema conlleva hasta el momento un total de 4 cambios.

El siguiente paso consiste en actualizar en las dependencias de *detecta_objeto* (los módulos ACT-PLN y ST-PLN) la localización del módulo OBJ-FND que pasa de estar en la máquina 1 a la máquina 2, tal como se analiza en "Relocalización de un módulo" en la sección 4.2.2. Esta operación puede resolverse mediante un cambio de IP/Puerto, nombre de host, etc. Dado que son dos módulos los que se actualizan se suman 2 cambios para un total de 6 cambios al sistema.

Finalmente, de acuerdo con "Cambio de la definición de las funciones de un módulo" En la sección 4.2.1 se debe actualizar el tiempo de ejecución de la función *detecta_objeto* en cada módulo que la utiliza, lo que aumenta 2 al número de cambios a realizar en el sistema, por lo que el número total de cambios es de 8, y, suponiendo que cada cambio tomara 5 minutos, actualizar el sistema tomaría un total de 40 minutos.

4.5.2. Especialización del módulo Visión en arquitectura basada en *Blackboard*

De acuerdo con el análisis realizado en "Especialización de un módulo" de la sección 4.3.2, los cambios a realizar en el sistema tras descomponer el módulo inicial VISION en los módulos VISION y PRS-FND es equivalente a mover la función *detecta_objeto* a un nuevo módulo vacío.

De acuerdo al algoritmo propuesto en "Relocalización de una función" de la sección 4.3.2, el algoritmo para descomponer VISION es:

```
Para cada función a relocalizar f_i^VISION
     Actualizar en Blackboard información de f_i^VISION
     cambiando a f_j^OBJ−FND
     Actualizar en componente de control información
     de f_i^VISION cambiando a f_j^OBJ−FND
Fin para cada
```

La función a relocalizar es una: *detecta_objeto*, por lo que el algoritmo siguiente se ejecuta sólo una vez:

```
Actualizar en Blackboard información de detecta_objeto^VISION
cambiando a detecta_objeto^OBJ−FND
Actualizar en componente de control información
de detecta_objeto^VISION cambiando a f_j^OBJ−FND
```

Teniendo que actualizarse únicamente el componente de control y el *Blackboard*, lo que da un total de 2 cambios.

El siguiente paso consiste en actualizar el *Blackboard* localización del módulo OBJ-FND que pasa de estar en la máquina 1 a la máquina 2, tal como se analiza en "Relocalización de un módulo" en la sección 4.3.2. Esta operación puede resolverse mediante un cambio de IP/Puerto, nombre de host, etc. Dado que sólo un componente del sistema se actualiza, se suma uno al número de cambios a realizar al sistema para un total de 3.

Finalmente, de acuerdo con "Cambio de la definición de las funciones de un módulo" En la sección 4.3.14.2.1 se debe actualizar el tiempo de ejecución de la función *detecta_objeto* el *Blackboard*, lo que aumenta 1 al número de cambios a realizar en el sistema, por lo que el número total de cambios es de 4 cambios, y, suponiendo que cada cambio tomara 5 minutos, actualizar el sistema tomaría un total de 20 minutos, la mitad que en una 7arquitectura *Peer-to-Peer*.

4.6. Conclusiones preliminares sobre el análisis de extensibilidad y reestructuración en ambas arquitecturas

En cuanto al análisis de extensibilidad y reestructuración, y de acuerdo con la Tabla 4.3, se muestra en la Tabla 4.4 el costo de actualización promedio del sistema es para cada arquitectura.

	Peer-to-Peer	***Blackboard***
Extensibilidad	$O(N^3)$	$O(N)$
Reestructuración	$O(\|F_R\|N^2)$	$O(\|F_R\|)$

Tabla 4.4. Costo promedio de actualización del sistema por extensibilidad y reestructuración entre arquitecturas *Peer-to-Peer* y basada en *Blackboard*.

Según los resultados de los análisis mostrados en la Tabla 4.4, el número de modificaciones requeridas para actualizar el sistema dado una extensión o una reestructuración es menor en el caso de una arquitectura basada en *Blackboard* que en el caso de una arquitectura *Peer-to-Peer*. Inclusive una revisión más detallada de la Tabla 4.1 del presente capítulo (sección 4.4), revela que, en general, el número de modificaciones requeridas es menor para una arquitectura basada en *Blackboard* que para una arquitectura *Peer-to-Peer*.

Es importante recordar que se supone que la comunicación entre los módulos que conforman las arquitecturas se realiza mediante paso de mensajes y sin el uso de software intermediario que gestione las comunicaciones, lo que puede reducir el tiempo de desarrollo a costo de afectar el desempeño o hacer al sistema dependiente de la plataforma.

5. ANÁLISIS TEÓRICO DEL DESEMPEÑO

Vísteme despacio… que tengo prisa.

Napoleón Bonaparte

En este capítulo se realiza un análisis desde el punto de vista teórico del desempeño de ambas arquitecturas *Peer-to-Peer* y *Blackboard*, tomando en cuenta granularidades media y gruesa para sistemas en ambas arquitecturas.

La primera sección profundiza el análisis teórico del desempeño de arquitecturas basadas en *Peer-to-Peer*, mediante la estimación del tiempo de ejecución de una función del sistema con base en el tiempo de ejecución de la función misma (lo cual incluye el tiempo de ejecución de otras funciones de las cuales la función a ejecutar dependa) y el tiempo requerido para las comunicaciones. Dicho análisis se repite en la segunda sección para una arquitectura basada en *Blackboard* tomando las mismas consideraciones. Finalmente, en la tercera sección de este capítulo se comparan ambos desempeños.

En la sección 1.3 se define desempeño de acuerdo al estándar ISO-9126-1 como el comportamiento de un sistema de software en el tiempo de la siguiente manera:

> *Capacidad del producto de software para proporcionar un tiempo de respuesta adecuado, tiempo de procesamiento y tasas de rendimiento cuando realiza su función bajo las condiciones establecidas [2].*

Si bien el desempeño también está relacionado con el uso eficiente de los recursos, es necesario hacer notar que a diferencia de otros sistemas de software, el software que controla a un robot móvil de gama media no compite por el uso de los recursos con otros programas; ni tampoco es necesario que presente una interfaz que ofrezca una respuesta inmediata al usuario. Como se menciona en la sección 2.1.3, debido a que los robots de gama media son aquellos que más interactúan con los seres humanos, pretenden emular capacidades humanas. De esto, es lógico pensar que el tiempo de respuesta de un robot de gama media debe ser muy similar al tiempo de respuesta de un ser humano, a fin de hacer la interacción lo más natural posible.

Aunado a esto, si el sistema de software del robot es diseñado teniendo en cuenta una granularidad media o gruesa, la eficiencia en el uso de los recursos recae más en el nivel funcional de los módulos que componen al sistema de software y no en las comunicaciones que existen entre dichos módulos, por lo que el uso de una arquitectura u otra es de menor impacto.

Es por estas razones que la eficiencia en el uso de los recursos no es un factor crucial a tomar en cuenta como medida del desempeño, siempre que la respuesta del robot esté dentro de los límites que garanticen la satisfacción de los usuarios.

5.1. Análisis teórico del desempeño en arquitecturas basadas en Peer-to-Peer

Tal como se menciona en la sección 4.2, en una arquitectura *Peer-to-Peer* se considera que si un módulo requiere una función provista por otro módulo, existe una conexión directa entre ambos módulos. Expresado en otros términos, si existe $f_j^{m_k}(\bar{x})$ tal que $f_j^{m_k}(\bar{x}) = g(\overline{x_1}) \circ f_r^{m_n}(\overline{x_2}) \Longrightarrow \exists c_{m_k,m_n}$ donde $\overline{x_1} \cup \overline{x_2} \subseteq \bar{x}$

Tal como se menciona en la sección 2.5, y para facilitar el análisis, se considera que toda comunicación entre módulos está formada por una petición o comando y una respuesta, por lo que el tiempo de ejecución de una función $f_j^{m_k}$ del módulo m_k solicitada por el módulo m_i está dado por:

$$t\left(m_i, f_j^{m_k}\right) = t\left(f_j^{m_k}\right) + t_c$$

donde:

- ✓ $t\left(f_j^{m_k}\right)$ es el tiempo de ejecución de la función $f_j^{m_k}$ del módulo m_k.
- ✓ t_c es el tiempo total de las comunicaciones entre m_k y m_i.

Lo anterior para un sistema de granularidad media. En el caso de sistemas de granularidad gruesa (el tiempo de procesamiento es mucho mayor que el tiempo de comunicaciones) $t_c \ll t\left(f_j^{m_k}\right)$, por lo que es posible considerar a t_c como despreciable, reduciéndose el tiempo de ejecución de una función $f_j^{m_k}$ módulo m_k solicitada por el módulo m_i a:

$$t\left(m_i, f_j^{m_k}\right) \cong t\left(f_j^{m_k}\right)$$

donde:

$t\left(f_k^{m_i}\right)$ es el tiempo de ejecución de la función $f_k^{m_i}$ del módulo m_i.

Es importante mencionar que se considera a t_c, el tiempo de comunicaciones entre dos módulos, como un valor promedio constante e independiente del tamaño de los datos a transmitir. Así mismo se asume que no existen errores en las comunicaciones, tales como retrasos, mensajes erróneos, pérdida de mensajes, etcétera.

5.2. Análisis teórico del desempeño en arquitecturas basadas en Blackboard

Considerando el modelo de ejecución de una función mediante lectura y escritura de variables presentado en la sección 4.3, la ejecución de una función remota se desglosa a continuación tomando t_c como el tiempo de comunicación (comando-respuesta), t_r el tiempo de lectura de variable, t_w el tiempo de escritura de variable y $t\left(f_j^{m_k}\right)$ el tiempo de ejecución de la función $f_j^{m_k}$:

- ✓ $write^{m_i}\left(f_j^{m_k}\right)$. El módulo m_i escribe en el *Blackboard* una petición de ejecución de la función $f_j^{m_k}$ del módulo m_k. Esta operación tarda t_c+t_w.
- ✓ $read^{m_k}(\bar{x})$. El módulo m_k realiza una lectura de las variables de entrada $\bar{x} = \{x_1, x_2, \dots, x_n\}$ en el *Blackboard*. Esta operación tarda $|\bar{x}|(t_c+t_r)$.
- ✓ Se ejecuta en el módulo m_k la función $f_j^{m_k}$ con los argumentos de $\bar{x}$ generando el vector de resultados $\bar{y} = \{y_1, y_2, \dots, y_m\}$. Esta operación tarda $t\left(f_j^{m_k}\right)$.
- ✓ $write^{m_k}(\bar{y})$. El módulo m_k realiza una escritura en el *Blackboard* del vector $\bar{y}$ resultado de la función $f_j^{m_k}$ del módulo m_k. Esta operación tarda $|\bar{y}|(t_c+t_w)$.
- ✓ $read^{m_i}(\bar{y})$. El módulo m_i realiza una lectura del vector resultado $\bar{y}$. Esta operación tarda $|\bar{y}|(t_c+t_r)$.

Por lo que el tiempo de ejecución de una función $f_j^{m_k}$ del módulo m_k solicitada por el módulo m_i está dado por:

$$t\left(m_i, f_j^{m_k}\right) = t\left(f_j^{m_k}\right) + (|\bar{x}| + |\bar{y}|)(t_c+t_r) + (1 + |\bar{y}|)(t_c+t_w)$$

donde:

- ✓ $t\left(f_j^{m_k}\right)$ es el tiempo de ejecución de la función $f_j^{m_k}$ del módulo m_k.
- ✓ $\bar{x} = \{x_1, x_2 \dots x_n\}$ es el conjunto de variables de entrada para la función $f_j^{m_k}$.
- ✓ $\bar{y} = \{y_1, y_2 \dots y_m\}$ es el conjunto de variables producidas como resultado de la ejecución de la función $f_j^{m_k}$, es decir $\bar{y} = f_j^{m_k}(\bar{x})$.
- ✓ $|\bar{x}|, |\bar{y}|$ son los tamaños (número de componentes) de los vectores de entrada y salida de la función $f_j^{m_k}$, respectivamente.
- ✓ t_c es el tiempo total de las comunicaciones entre m_i o m_k y m_b, con m_b el *Blackboard*.
- ✓ t_r es el tiempo de lectura de los datos de una variable dentro del *Blackboard*. Cabe mencionar que varios módulos pueden leer de la misma variable al mismo tiempo si el diseño del *Blackboard* así lo permite; aunque el módulo no puede leer una variable en la que se esté realizando una operación de escritura.

- ✓ t_w es el tiempo de escritura de los datos de una variable dentro del *Blackboard*. Cabe mencionar que sólo un módulo puede acceder a la variable para realizar una operación de escritura a la vez; y que este tiempo contempla los tiempos de espera en cola para realizar la escritura. Lo anterior suponiendo que la lectura/escritura de cada componente de los vectores de entrada y salida $\bar{x}$, $\bar{y}$ de la función $f_j^{m_k}$ conllevan un acceso independiente al *Blackboard*, es decir, una comunicación.

Sin embargo, tomando en cuenta un entorno de memoria distribuida, es decir, con las aplicaciones ejecutándose en espacios de memoria diferentes y las comunicaciones realizadas a través de una red, donde el tiempo requerido en la lectura/escritura de variables en memoria (del orden de microsegundos o incluso menos) es mucho menor que el tiempo de comunicaciones (del orden de milisegundos), conviene realizar la lectura o escritura de todas las variables involucradas en una sola petición.

Con esto, se modifica al *Blackboard* para que las operaciones de lectura/escritura reciban no una, sino un conjunto de variables, redefiniéndose a $read^{m_i}(\overline{v_j})$ y $write^{m_i}(\overline{v_j})$ como las funciones de lectura y escritura en el *Blackboard* sobre el vector $\overline{v_j}$ realizada por el módulo m_i, donde:

- ✓ $\bar{v} = \{v_1, v_2, \dots, v_n\} \in V$ es un vector cuyas componentes son variables del sistema almacenadas en el *Blackboard*, las cuales son procesadas en una sola operación de lectura o escritura.
- ✓ $read^{m_i}(\overline{v_j})$ es una operación de lectura sobre el vector $\overline{v_j}$ realizada por el módulo m_i.
- ✓ $write^{m_i}(\overline{v_j})$ es una operación de escritura sobre el vector $\overline{v_j}$ realizada por el módulo m_i.

y

- ✓ $V = \{f_1^{m_1}, \dots, f_p^{m_1}, \dots, f_1^{m_n}, \dots, f_r^{m_n}\} \cup \{x_1, \dots, x_n\} \cup \{y_1, \dots, y_m\}$ es el conjunto de todas las variables del sistema, formado por las funciones que pueden ejecutar los módulos del sistema, los parámetros que reciben y sus resultados. En otras palabras V es el *Blackboard*.

Con lo anterior, el tiempo de ejecución de una función $f_j^{m_k}$ del módulo m_k solicitada por el módulo m_i se reduce a:

$$t\left(m_i, f_j^{m_k}\right) \cong t\left(f_j^{m_k}\right) + 2t_r + 2t_w + 4t_c$$

Como se parte de la suposición de un sistema de granularidad media a gruesa (el tiempo de procesamiento es cuando menos igual al tiempo de comunicaciones), entonces al ser $t_r, t_w \ll t_c$ es posible considerar a t_r y t_w como despreciables, reduciéndose el tiempo de ejecución de una función $f_j^{m_k}$ del módulo m_k solicitada por el módulo m_i a:

$$t\left(m_i, f_j^{m_k}\right) \cong t\left(f_j^{m_k}\right) + 4t_c$$

En el caso de sistemas de granularidad gruesa (el tiempo de procesamiento es mucho mayor que el tiempo de comunicaciones) $t_c \ll t\left(f_j^{m_k}\right)$, por lo que es posible considerar a t_c como despreciable, reduciéndose el tiempo de ejecución de una función $f_j^{m_k}$ módulo m_k solicitada por el módulo m_i a:

$$t\left(m_i, f_j^{m_k}\right) \cong t\left(f_j^{m_k}\right)$$

Para ejecutar una función mediante lectura y escritura de variables acorde a lo presentado en la sección 4.3, los módulos que ejecutan las funciones deben leer la variable que activa la ejecución de dicha función. La manera en que la lectura de la variable que dispara la ejecución de una función es realizada puede impactar en el desempeño. Este problema se trata a continuación.

5.2.1. Solución al problema desencadenamiento de funciones mediante lectura de variables compartidas.

La activación de la ejecución de una función mediante la escritura de una variable compartida en el *Blackboard* sugiere la lectura constante de dicha variable en un poleo. No obstante esta lectura constante conlleva a dos problemas: la saturación del canal de comunicación y un excesivo uso de procesador, lo que se traduce en un menor desempeño en las otras funciones ejecutadas concurrentemente.

Una solución a este problema consiste en realizar las lecturas a intervalos fijos con un periodo lo suficientemente grande como para que el uso del canal de comunicación y del procesador respecto a las otras tareas sea despreciable. Sin embargo, esta solución agrega un retraso al tiempo de ejecución de la función equivalente al tiempo transcurrido entre que el valor de la variable cambia y la lectura de la variable que desencadena la ejecución de la función.

Tal como se menciona en la sección 5.2, estas soluciones impactan negativamente en el desempeño del sistema, como posiblemente lo harían otras soluciones basadas en una petición de lectura por parte del módulo que ejecuta la función.

Para resolver este problema se propone un modelo basado en suscripciones similar al propuesto por Mary Shaw en [9] donde describe su Arquitectura de Control de Tareas e Invocación Implícita, y al utilizado en el *Framework* CARMEN descrito en [18], presentados en las secciones 3.1.3 y 3.2.1 respectivamente. La propuesta consiste en asignar al *Blackboard* la tarea de notificar el cambio del valor de una variable a los módulos del sistema que se encuentren suscritos a la misma.

De este modo, los módulos estarán suscritos a las variables utilizadas para solicitar la ejecución de cada una de sus funciones, y al cambiar el valor de estas, se les notifica y pueden proceder a la ejecución. Esta notificación puede realizarse simulando dentro del *Blackboard* una solicitud de lectura de variable por parte de cada uno de los módulos suscriptores, enviando a cada uno la respuesta.

Nótese que al ser una operación de escritura la que desencadena la notificación a los módulos suscriptores, no existe saturación del canal de comunicación ni tampoco se hace un uso excesivo de procesador al no ser una operación que opere dentro de un ciclo.

El modelo de suscripciones propuesto es extensible a cualquier variable almacenada en el *Blackboard*, lo que elimina necesidad de poleos para adquisición de datos de manera periódica por parte de los módulos que los requieren, además de eliminar las solicitudes de lectura para dicha operación. Dependiendo de la funcionalidad de los módulos, este modelo puede ayudar a reducir el uso del canal de comunicación y de recursos de procesador.

5.3. Comparación del desempeño teórico entre arquitecturas Peer-to-Peer *y* Blackboard

A continuación se presenta la Tabla 5.1 en la que se comparan los tiempos de ejecución de una función para granularidades media y gruesa en arquitectura basada en *Peer-to-Peer* y arquitectura basada en *Blackboard* según lo analizado en las secciones anteriores.

Granularidad	***Peer-to-Peer***	***Blackboard***
Media	$t\left(m_i, f_j^{m_k}\right) = t\left(f_j^{m_k}\right) + t_c$	$t\left(m_i, f_j^{m_k}\right) \cong t\left(f_j^{m_k}\right) + 4t_c$
Gruesa ($t_c \to 0$)	$t\left(m_i, f_j^{m_k}\right) \cong t\left(f_j^{m_k}\right)$	$t\left(m_i, f_j^{m_k}\right) \cong t\left(f_j^{m_k}\right)$

Tabla 5.1. Comparación del desempeño teórico entre arquitecturas *Peer-to-Peer* y *Blackboard*.

En la Tabla 5.1 es posible apreciar que no existe diferencia sobresaliente en el desempeño debido a la arquitectura cuando se tiene una granularidad gruesa. Además se observa que cuando la granularidad es media la diferencia entre ambas arquitecturas se hace menor (e incluso despreciable en la medida en que el tiempo de ejecución de una función sea mucho mayor que el tiempo de comunicaciones, es decir, $t\left(f_j^{m_k}\right) \gg t_c$.

6. COMPARACIÓN EXPERIMENTAL DEL DESEMPEÑO

Con su acción, reforzaron el complejo de Frankenstein de la humanidad; sus temores irracionales de que cualquier hombre artificial creado por ellos acabaría volviéndose contra su creador.

Keith Harriman.
Qué es el Hombre. Isaac Asimov.

En este capítulo se mide experimentalmente el desempeño de software que controla un robot móvil de gama media bajo ambas arquitecturas *Peer-to-Peer* y *Blackboard* mediante una prueba simple.

La primera sección establece la variable a medir, describe las características y capacidades del robot, y presenta los criterios de medición comparación y análisis utilizados durante la fase experimental. La segunda sección describe al software utilizado para la medición del desempeño de una arquitectura *Peer-to-Peer* y presenta los datos obtenidos en los experimentos. De forma análoga, la tercera sección describe al software utilizado para la medición del desempeño de una arquitectura *Blackboard* y presenta los datos obtenidos en los experimentos. Finalmente en la cuarta sección se comparan y analizan los datos experimentales obtenidos en las secciones anteriores.

6.1. Descripción del experimento

El desempeño del software que controla un robot móvil de gama media puede entenderse como la rapidez con la que el robot ejecuta una tarea. Es decir, cuanto menor sea el tiempo que tarde en ejecutar la tarea, mejor es el desempeño. Con esto, y acorde a la definición de la sección 1.3, la variable a medir es el tiempo.

Como se analiza en el capítulo 5, el cambio en el desempeño del software que controla un robot móvil de gama media debido a un cambio en su arquitectura se debe principalmente al cambio en los tiempos de comunicación entre los módulos que conforman al sistema. Sin embargo, el tiempo que tarda un robot en ejecutar una tarea depende también del tiempo de respuesta de sus sensores, actuadores y de las funciones que ejecuten los módulos. Razón por la cual la tarea escogida para los experimentos realizados involucra el uso de sensores, actuadores y módulos cuyo tiempo de respuesta puede ser considerado constante.

El robot utilizado para medir el desempeño de ambas arquitecturas es un robot móvil de gama media de propósito general orientado al servicio doméstico, el cual se muestra en la Figura 6.1. Entre las capacidades del robot se encuentran la gesticulación, manipulación de objetos livianos, navegación, reconocimiento del habla, objetos y rostros, síntesis de voz, entre otros.

Figura 6.1. Robot Justina.

Debido a que las tareas de reconocimiento y manipulación consumen tiempos muy variables, al verse alteradas por un gran número de variables no controlables (por ejemplo condiciones de iluminación), cualquier prueba que requiera de reconocimiento o manipulación queda descartada. Por otro lado, la navegación requiere una retroalimentación constante por parte de los sensores y los algoritmos se ejecutan en tiempos similares para trayectorias similares.

Tomando en cuenta lo anterior, la tarea que lleva a cabo el robot para la medición experimental del desempeño de ambas arquitecturas consiste en la navegación del robot desde un punto A hasta un punto B, a través de un entorno previamente conocido por el robot, cuyo mapa se muestra en la Figura 6.2, con los siguientes puntos de acuerdo con el diagrama de flujo de la Figura 6.3:

- ✓ El robot comienza en el punto A (pasillo) a la espera de que la puerta de la habitación a través de la cual se desplaza sea abierta.
- ✓ El robot anuncia que la puerta ha sido abierta cuando este evento ocurra. A partir de este momento comienza la toma de tiempo transcurrido.
- ✓ Una vez que el robot ha anunciado que se abre la puerta, navega de manera automática hasta el punto B.
- ✓ Una vez que el robot ha llegado al punto B, anuncia que ha llegado a destino. La toma de tiempo se detiene.

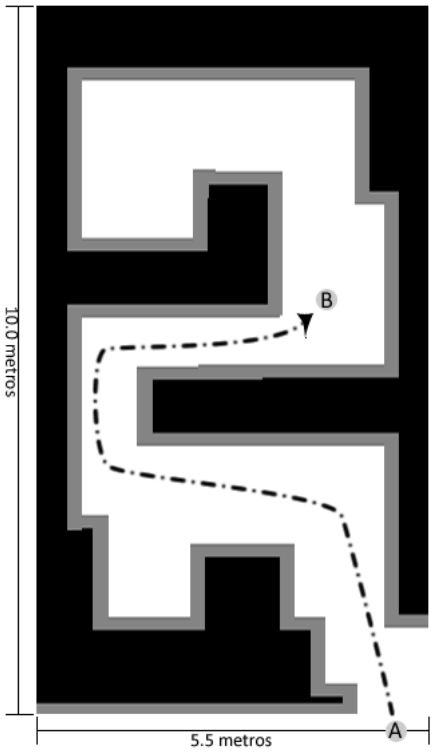


Figura 6.2. Trayectoria a seguir por el robot. La distancia media entre A y B es de 8.5m.

A continuación se presentan los dispositivos de hardware utilizados durante la tarea:

- ✓ Láser HOKUYO URG-04LX. Ofrece 10 lecturas por segundo.
- ✓ Base móvil de par diferencial. Velocidad linear máxima 0.3m/s.
- ✓ Bocinas conectadas a salida de audio estándar.

La base móvil de par diferencial recibe mediante puerto serie RS232 a 115kbps la velocidad de cada una de las ruedas de la computadora de control, respondiendo inmediatamente con el avance relativo de cada rueda desde la última instrucción de movimiento (aproximadamente 4000 pulsos por vuelta). Por motivos de seguridad, la base móvil del robot debe recibir una orden de movimiento cada 100 milisegundos o se detiene, lo cual, aparte de evitar accidentes garantiza un muestreo constante. También por seguridad la velocidad de la base móvil está limitada a un máximo de 0.3m/s.

Por otra parte, la síntesis de voz para anunciar que la puerta ha sido abierta y que el robot ha llegado a destino se ejecuta de manera asíncrona, es decir, una vez que llega la solicitud de ejecución con el texto a sintetizar, éste se analiza y encola, tras lo cual se envía un mensaje de respuesta. Esta operación suele tardar unos pocos milisegundos.

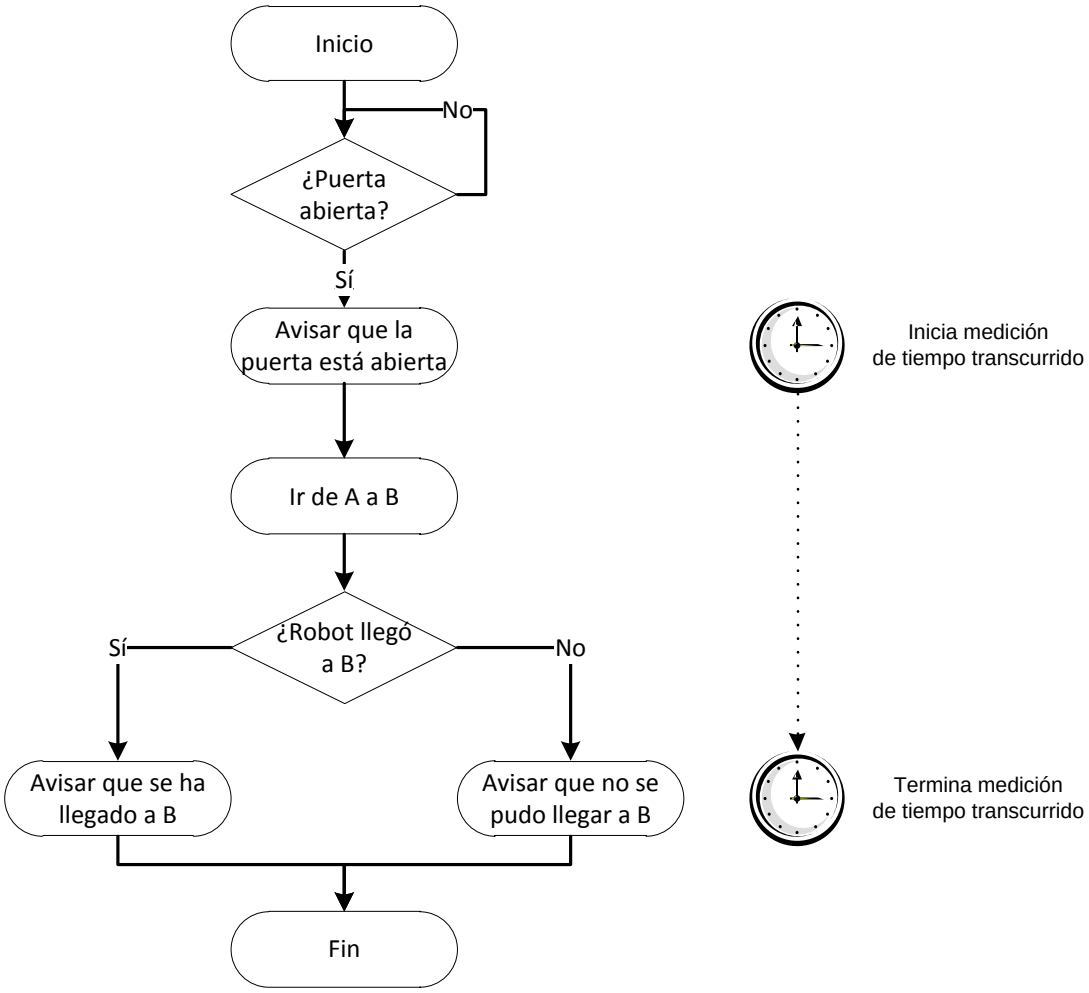


Figura 6.3. Diagrama de flujo de la tarea a ejecutar por el planeador de acciones.

La ejecución de la prueba requiere entonces de al menos los siguientes módulos, de acuerdo con las arquitecturas propuestas en la sección 2.3:

- ✓ Planeador de acciones.
- ✓ Planeador de movimientos.
- ✓ Control de base móvil.
- ✓ Módulo de sensado.
- ✓ Módulo de adquisición de datos del Láser.
- ✓ Sintetizador de voz.

Sin embargo, en el software actual del robot estos módulos se encuentran agrupados por razones de acceso al hardware como se muestra a continuación.

- ✓ MVN-PLN: Planeador de movimientos, control de base móvil, y módulo de sensado (Odometría).
- ✓ ACT-PLN: Planeador de acciones.

- ✓ SENSORS: Módulo de sensado y módulo de adquisición de datos del láser.
- ✓ SP-GEN: Sintetizador de voz.

Partiendo del supuesto de que tanto la adquisición de datos como el control de la base móvil se ejecutan a un intervalo constante de tiempo, puede decirse que estas operaciones requieren de tiempo constante. De manera similar la planeación de la misma trayectoria, la síntesis de voz para los mismos avisos y la ejecución de la misma rutina por parte del planeador de acciones son funciones que deben ejecutarse en tiempos similares que pueden considerarse constantes. Por lo tanto, de existir alguna variación significativa en el tiempo de ejecución de la tarea planteada, ésta se debe a los retardos inducidos por los tiempos de comunicación entre los módulos que conforman al sistema en cada una de las arquitecturas.

Actualmente, el software que controla al robot está compuesto por más de diez módulos, cada uno de los cuales está estructurado en capas entre las que destacan la capa funcional o de aplicación que realiza las funciones propias del módulo. Una capa encargada de las comunicaciones y una capa que provee acceso al hardware para los módulos que así lo requieren. Como parte de los criterios de selección de la tarea utilizada para medir el desempeño de ambas arquitecturas con los módulos existentes está el evitar la modificación de los componentes funcionales de los módulos a fin de reducir factores que produzcan incertidumbre en la prueba, por lo que solamente la capa de comunicaciones se modifica para adaptar los módulos a las distintas arquitecturas.

Aunque todos los módulos se ejecutan en entornos de un solo procesador de entre uno y cuatro núcleos y memoria compartida, el sistema de software completo requiere de más de una computadora para operar. Sólo se muestran los componentes necesarios para la ejecución de la tarea, indicándose en cada módulo el número de máquina en la que se ejecuta.

Los módulos utilizados se ejecutarán en dos computadoras conectadas entre sí por una red *Fast Ethernet* (100Mbps) y, debido a que las comunicaciones se realizan por paso de mensajes cuya longitud es menor a 8 kilobytes, puede asumirse un tiempo de comunicación promedio de 1 milisegundo. Las características de las computadoras M1 y M2 se especifican a continuación:

- ✓ M1: Intel Core 2 Duo 1.83GHz, 2GB RAM DDR2 667MHz
  Microsoft Windows XP SP3.
- ✓ M2: Intel Atom 1.6GHz, 1GB RAM DDR2 667MHz.
  Microsoft Windows XP SP3.

En resumen, la fase experimental consiste en la ejecución por parte del robot de trece veces la tarea descrita anteriormente para cada arquitectura, es decir, un total de 26 ejecuciones. Los tiempos medidos en cada experimento se reportan en la sección 6.4, así como el tiempo medio de ejecución bajo cada arquitectura y el análisis correspondiente de las distribuciones.

El análisis de los tiempos obtenidos realizado en la sección 6.4 parte del hecho de que un robot móvil de gama media cuenta con capacidades motrices similares a las de un ser humano dada su interacción con estos, tal como se describe en la sección 2.1.3. Por este motivo se consideran despreciables las variaciones en el tiempo de desplazamiento del robot dentro de un rango de ± 3 segundos, pues es un tiempo suficientemente pequeño como para pasar desapercibido por un ser humano.

6.2. Desempeño en arquitectura Peer-to-Peer

En esta sección se describe al software utilizado para la medición del desempeño de una arquitectura *Peer-to-Peer* tomando como parámetro base el tiempo que tarda el robot en realizar la tarea descrita en la sección 6.1, experimento que es repetido trece veces.

La arquitectura de software utilizada para la medición del desempeño considerando la distribución de los programas y las relaciones de dependencia funcional se muestra en la Figura 6.4.

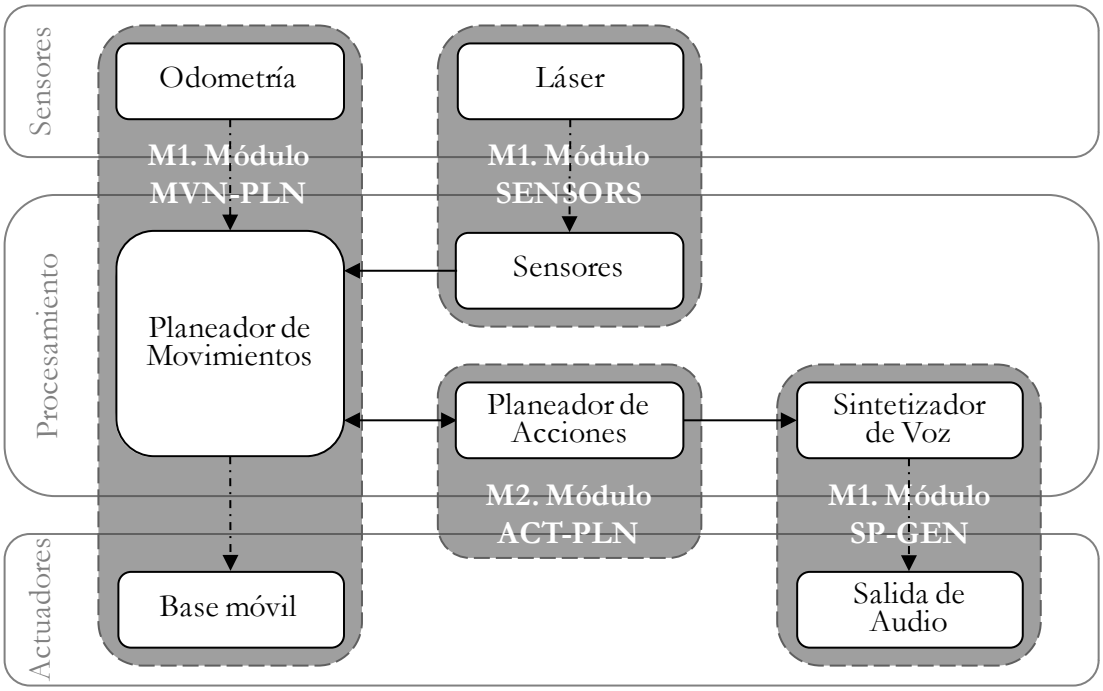


Figura 6.4. Distribución de los programas que conforman la tarea a ejecutar por el robot y su dependencia funcional en arquitectura *Peer-to-Peer*.

Acorde con la Figura 6.4 se utilizan únicamente cuatro módulos: ACT-PLN, MVN-PLN, SENSORS y SP-GEN, que se ejecutan en las dos computadoras descritas en la sección 6.1 conectadas entre sí por una red *Fast Ethernet*, con un tiempo de comunicaciones promedio de 1ms.

Siguiendo el esquema propuesto en el capítulo 2 y utilizado en los capítulos 4 y 5, debe existir una comunicación bidireccional en los módulos interrelacionados. Agregando las comunicaciones bidireccionales e integrando los módulos se obtiene la Figura 6.5 que describe la arquitectura utilizada en los experimentos.

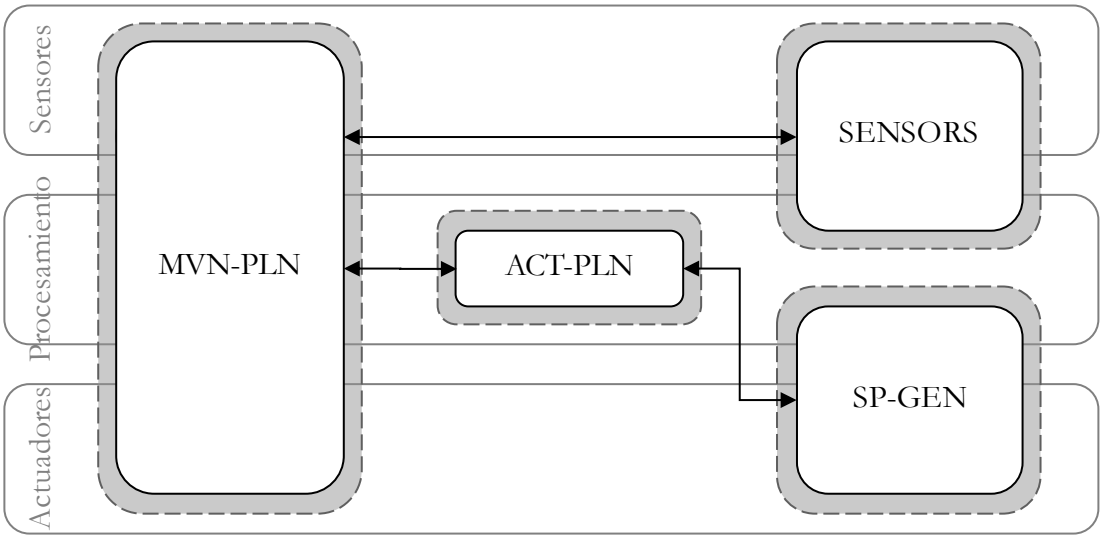


Figura 6.5. Arquitectura *Peer-to-Peer* utilizada en los experimentos.

La ejecución del algoritmo propuesto para los experimentos en la sección 6.1 utilizando la arquitectura mostrada en la Figura 6.5 se muestra a continuación mediante un diagrama de secuencia en la Figura 6.6.

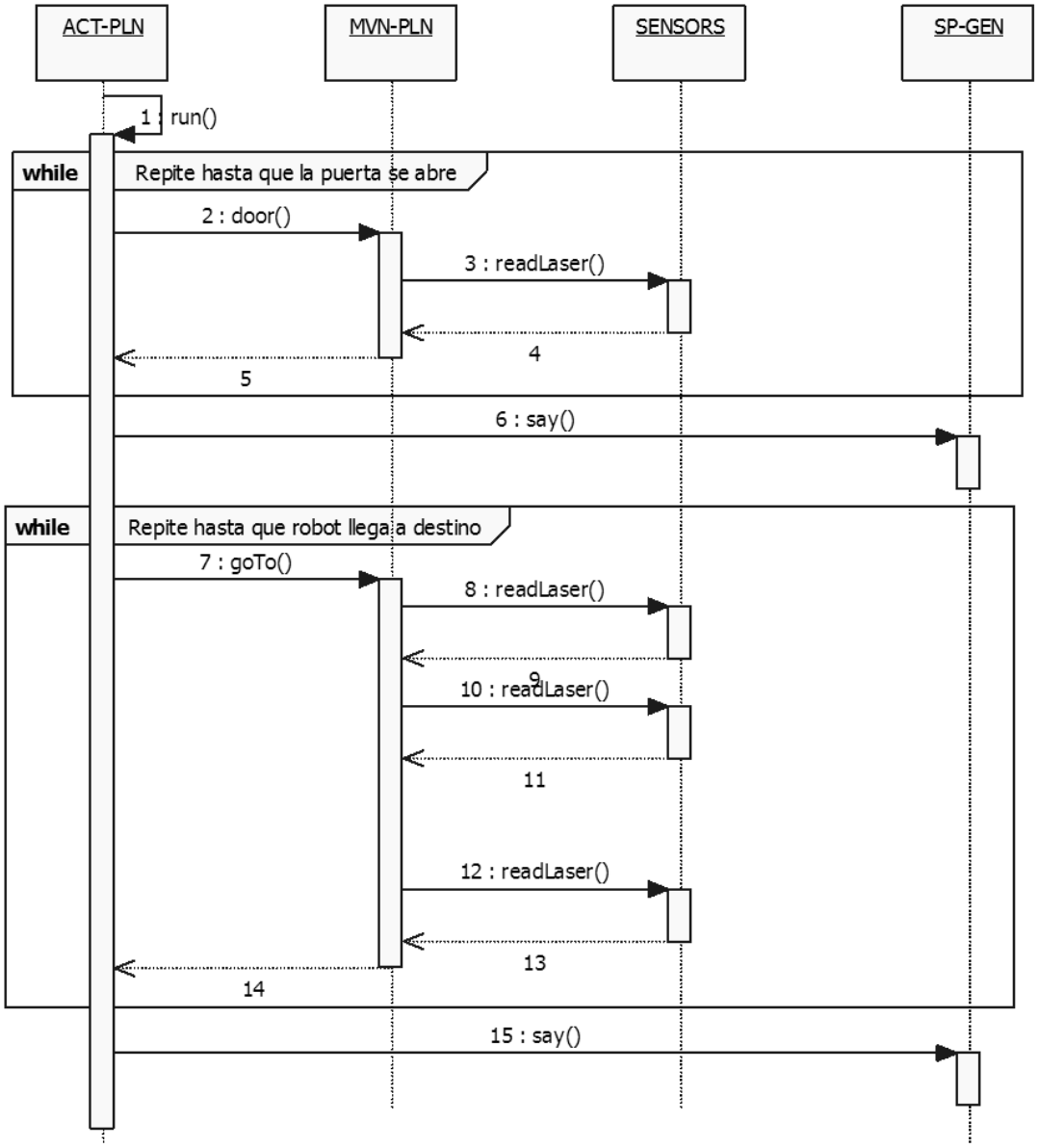


Figura 6.6. Diagrama de secuencia de la ejecución
del algoritmo propuesto en arquitectura *Peer-to-Peer*

6.3. Desempeño en arquitectura Blackboard

En esta sección se describe al software utilizado para la medición del desempeño de una arquitectura basada en *Blackboard*, tomando como parámetro base el tiempo que tarda el robot en realizar la tarea descrita en la sección 6.1, experimento que es repetido trece veces.

La arquitectura de software utilizada para la medición del desempeño considerando la distribución de los programas y las relaciones de dependencia funcional se muestra en la Figura 6.7.

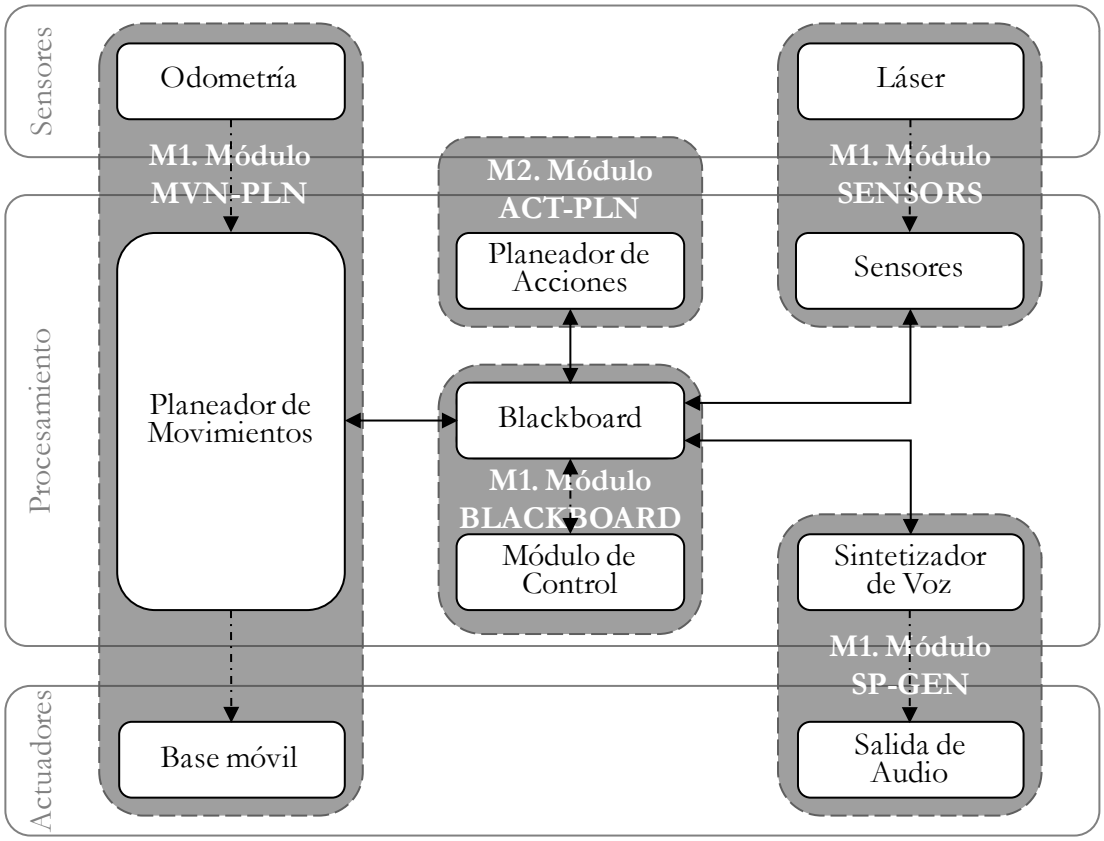


Figura 6.7. Distribución de los programas que conforman la tarea
a ejecutar por el robot y su dependencia funcional en arquitectura *Blackboard*.

Acorde con la Figura 6.7, se utilizan únicamente cuatro módulos: ACT-PLN, MVN-PLN, SENSORS y SP-GEN además del *BLACKBOARD*. Estos módulos se ejecutan en las dos computadoras descritas en la sección 6.1 conectadas entre sí por una red *Fast Ethernet*, con un tiempo de comunicaciones promedio de 1ms.

Siguiendo el esquema propuesto en el capítulo 2 y utilizado en los capítulos 4 y 5, debe existir una comunicación bidireccional en los módulos interrelacionados. Integrando los módulos se obtiene la Figura 6.8 que describe la arquitectura utilizada en los experimentos:

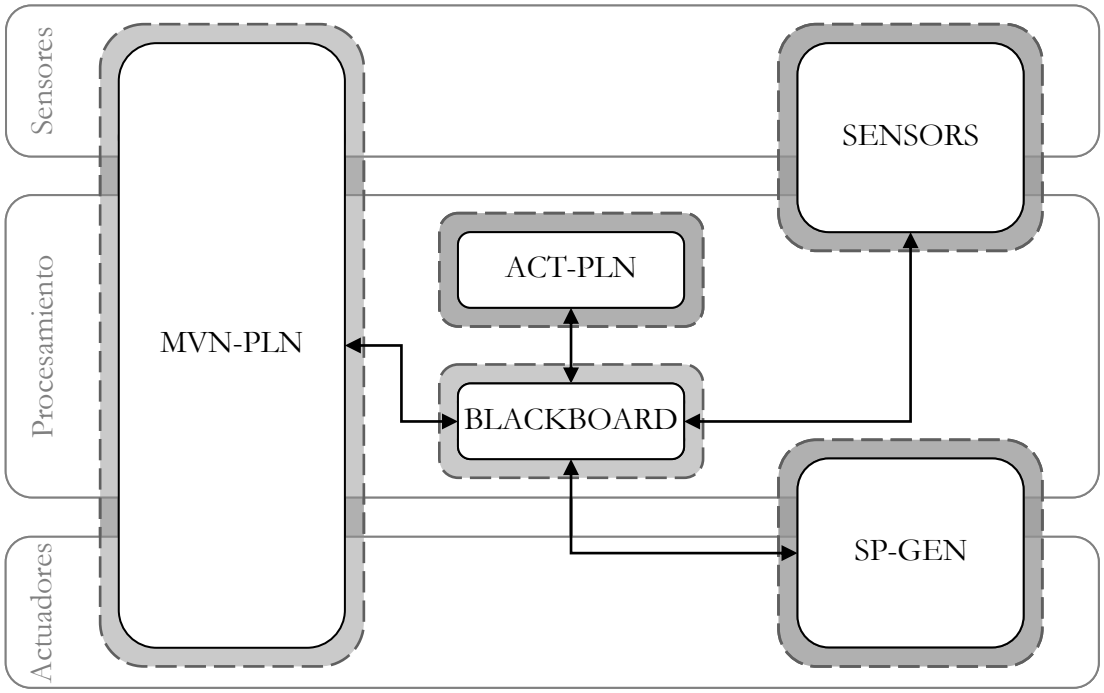


Figura 6.8. Arquitectura *Blackboard* utilizada en los experimentos.

La ejecución del algoritmo propuesto para los experimentos en la sección 6.1 utilizando la arquitectura mostrada en la se muestra a continuación mediante un diagrama de secuencia en la Figura 6.9.

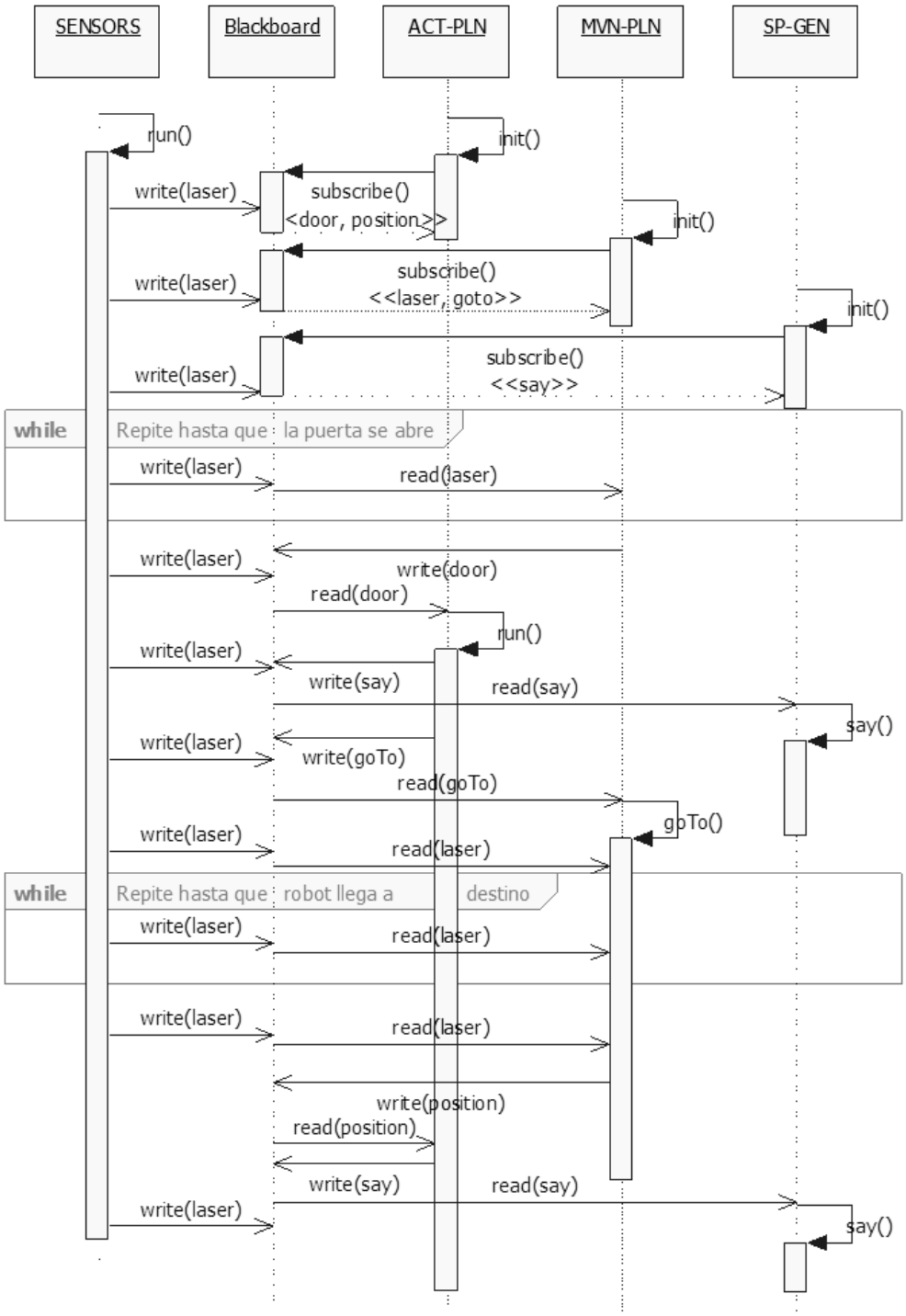


Figura 6.9. Diagrama de secuencia de la ejecución
del algoritmo propuesto en arquitectura *Peer-to-Peer*

6.4. Análisis comparativo de los datos experimentales

Aunque la velocidad máxima de la base móvil del robot es de 0.3m/s, esta velocidad sólo se alcanza cuando el robot se desplaza distancias grandes en línea recta y cuando no existen obstáculos de por medio, pues si durante el desplazamiento el robot debe realizar un giro mayor a 0.3 radianes (aproximadamente 17.2º) se detiene y gira sobre sí mismo, por lo que para determinadas trayectorias, la velocidad promedio se acerca al límite máximo.

A continuación, en la Tabla 6.1 se presentan los datos experimentales obtenidos para 13 ejecuciones del experimento planteado en la sección 6.1 bajo las arquitecturas *Peer-to-Peer* y *Blackboard* como se describe en las secciones 6.2 y 6.3. Los datos presentados corresponden al tiempo de desplazamiento del robot entre el punto A y el punto B, según la Figura 6.2.

Número de Prueba	**Arquitectura *Blackboard*** **Tiempo en segundos**	**Arquitectura *Peer-to-Peer*** **Tiempo en segundos**
1	72.70	73.09
2	73.86	74.02
3	72.28	72.06
4	72.58	73.11
5	72.45	72.73
6	73.98	73.64
7	73.94	72.03
8	72.69	72.69
9	73.22	73.88
10	73.28	73.96
11	73.66	73.27
12	73.94	73.63
13	72.70	72.19

Tabla 6.1. Resultados experimentales.
Tiempo de desplazamiento del robot bajo ambas arquitecturas.

Para analizar el conjunto de datos presentado en la Tabla 6.1 es necesario calcular la media, desviación estándar y error máximo correspondiente a cada arquitectura. El error máximo se calcula como la diferencia entre la media y el valor más alejado de la media con la fórmula siguiente:

$$Error\ máximo = \left(\frac{Max\left(Max(\bar{x}) - \mu, \mu - Min(\bar{x})\right)}{\mu}\right) \times 100\%$$

donde

- ✓ μ es la media o el promedio de los tiempos medidos en cada experimento.
- ✓ $\bar{x}$ es el conjunto de tiempos medidos en cada experimento.

finalmente

	Arquitectura *Blackboard*	**Arquitectura *Peer-to-Peer***
Media	73.17 segundos	73.10 segundos
Desviación estándar	0.6392 segundos	0.7162 segundos
Error máximo	±1.22%	±1.47%

Tabla 6.2. Datos estadísticos de los valores experimentales.

De los datos obtenidos se debe obtener el tiempo de ejecución promedio representativo bajo cada arquitectura. Debido a que se tiene una muestra pequeña de datos y que el error en las mediciones es menor al 1.5%, puede asumirse que el valor del promedio es tan próximo al real como si se hubiera obtenido estadísticamente con todas las mediciones reales. Así pues, es posible utilizar una distribución como la *t de Student* para calcular dicho tiempo de ejecución promedio con un nivel de confianza del 97.5% con la siguiente fórmula [28]:

$$Tiempo\ promedio\ de\ desplazamiento = \mu \pm (t_{0.975}) \frac{\sigma}{\sqrt{N-1}}$$

donde

- ✓ μ es la media o el promedio de los tiempos de ejecución para cada conjunto de experimentos
- ✓ $t_{0.975}$ es el coeficiente de confianza de 97.5%
- ✓ σ es la desviación estándar del conjunto de experimentos
- ✓ N es el número de experimentos.

(Nótese que se utiliza un coeficiente de confianza de 97.5% cuando el error máximo en las mediciones permitiría utilizar 98.5%. Sin embargo se utiliza una confianza de 97.5% por encontrarse en la literatura, específicamente en [28]).

Se presentan a continuación en la Tabla 6.3 los tiempos de desplazamiento promedio para las poblaciones en ambos experimentos.

	Tiempo promedio de desplazamiento
Arquitectura *Blackboard*	$\mathbf{73.17 \pm 2.35}$
Arquitectura *Peer-to-Peer*	73.10 ± 2.63

Tabla 6.3. Tiempo promedio de desplazamiento promedio del robot bajo ambas arquitecturas

En la Tabla 6.3, y acorde con el análisis realizado en la sección 5.3, se observa que el tiempo de desplazamiento promedio bajo una arquitectura *Peer-to-Peer* es menor que el tiempo de desplazamiento promedio bajo una arquitectura *Blackboard*. Sin embargo la incertidumbre bajo una arquitectura *Peer-to-Peer* es mayor que la incertidumbre bajo una arquitectura *Blackboard*. Esta diferencia de tiempos no es notoriamente diferente para el usuario.

Como se establece en la sección 5.1, para el caso de un robot móvil de gama media, dos ejecuciones se consideran iguales si sus tiempos de ejecución se encuentran a ± 3 segundos del tiempo de ejecución promedio, y se observa que el error obtenido se encuentra por debajo de esta cota. Es más, de todos los experimentos realizados aquel que se aleja más de la media es error más grande fue para el experimento #7 en arquitectura *Peer-to-Peer* con un valor de 72.03 segundos y alejado 1.07 segundos de la media (error: -1.47%), cantidad menor a los 3 segundos propuestos.

Finalmente puede decirse que acorde a los resultados de los experimentos propuestos en la sección 5.1, se comprueba que las arquitecturas *Peer-to-Peer* y *Blackboard* propuestas tienen desempeños similares.

6.5. Comprobación del análisis teórico del desempeño con los resultados experimentales obtenidos en ambas arquitecturas.

Según la Tabla 6.3 y considerando el hardware descrito en las secciones 6.2 y 6.3, se asume t_c como el tiempo de comunicaciones entre dos módulos del orden de 1 a 3 milisegundos. Se tomará el mayor valor $t_c = 3ms$.

Por otro lado, el experimento descrito en la sección 6.1 establece que el tiempo medido es el transcurrido entre que se abre la puerta y el robot llega a su destino. Así, el experimento puede dividirse en tres etapas:

1. El robot espera a que se abra la puerta.
2. Inicia la toma de tiempo. El robot anuncia que la puerta se ha abierto y comienza a desplazarse a su destino.
3. El robot anuncia que ha llegado a destino. Se detiene la toma de tiempo.

El punto 1 se descarta debido a que no contribuye en el tiempo medido. Los otros dos puntos pueden desglosarse como sigue, de acuerdo con lo especificado en las secciones 6.2 y 6.3, y a los diagramas de secuencia presentados en la Figura 6.6 y la Figura 6.9:

1. ACT-PLN inicia medición de tiempo.
2. ACT-PLN solicita a SP-GEN sintetizar el aviso de puerta abierta.
3. ACT-PLN solicita a MVN-PLN desplazarse a destino.
4. ACT-PLN queda a la espera de que MVN-PLN informe que el movimiento se ha completado. En arquitectura *Peer-to-Peer*, MVN-PLN solicita constantemente a SENSORS lecturas de láser. En arquitectura *Blackboard* MVN-PLN recibe constantemente lecturas del *Blackboard* escritas de manera automática por SENSORS.
5. MVN-PLN informa a ACT-PLN que ha llegado a destino.
6. ACT-PLN solicita a SP-GEN sintetizar el aviso de arribo a destino.

7. ACT-PLN inicia medición de tiempo.

Del listado anterior, los puntos 1 y 7 no contribuyen al tiempo medido; de los puntos 2 y 6 se deriva una función de síntesis de voz, y los puntos 3 y 5 operan como una función de desplazamiento.

De acuerdo con el capítulo 6, durante la ejecución del punto 4 (correspondiente al desplazamiento del robot) se realizan 10 lecturas por segundo por parte del láser, lo que implica un tiempo de ejecución de 100 milisegundos, pero al ser periódicas no se contabiliza, pues el tiempo de comunicaciones entre que el sensor entrega adquiere los datos y estos llegan al módulo destino agrega un retraso constante que puede verse como un offset en la frecuencia de adquisición.

No obstante, un retraso considerable en el arribo de dichas lecturas al MVN-PLN hará que el robot tenga un movimiento errático o que se detenga constantemente, lo que incrementaría el tiempo total de desplazamiento. Por otra parte, como se menciona en el capítulo 6, la síntesis de voz en el SP-REC se realiza de manera asíncrona, tardando la operación unos pocos milisegundos. En resumen, la operación de síntesis de voz es una operación de granularidad media y la navegación es una operación de granularidad gruesa que requiere de constantes operaciones de lectura de láser que pueden considerarse también como de granularidad gruesa.

En el capítulo 5 (y en especial en la sección 5.3) se afirma que el tiempo de comunicaciones no es significativo en la ejecución de funciones de granularidad gruesa y, teniendo el hardware un tiempo de respuesta al menos 30 veces superior al tiempo de comunicaciones, el sensado del láser y el movimiento con la base podrían considerarse operaciones de granularidad gruesa. No obstante, se considera el tiempo de comunicación como si fuesen de granularidad media a fin de observar su impacto en el desempeño del sistema y comprobar la afirmación teórica. Entonces el tiempo de ejecución teórico para el experimento es de la siguiente forma:

$$t = t\left(ACT - PLN, f_{say}^{SP-GEN}\right) + t\left(ACT - PLN, f_{goTo}^{MVN-PLN}\right) + t\left(ACT - PLN, f_{say}^{SP-GEN}\right)$$

En arquitectura *Peer-to-Peer* es equivalente a:

$$t_{Peer-to-Peer} = t\left(f_{say}^{SP-GEN}\right) + t_c + t\left(f_{goTo}^{MVN-PLN}\right) + t_c + t\left(f_{say}^{SP-GEN}\right) + t_c$$

$$t_{Peer-to-Peer} = t\left(f_{say}^{SP-GEN}\right) + t\left(f_{goTo}^{MVN-PLN}\right) + t\left(f_{say}^{SP-GEN}\right) + 3t_c$$

Y en arquitectura *Blackboard* es equivalente a:

$$t_{Blackboard} = t\left(f_{say}^{SP-GEN}\right) + 4t_c + t\left(f_{goTo}^{MVN-PLN}\right) + 4t_c + t\left(f_{say}^{SP-GEN}\right) + 4t_c$$

$$t_{Blackboard} = t\left(f_{say}^{SP-GEN}\right) + t\left(f_{goTo}^{MVN-PLN}\right) + t\left(f_{say}^{SP-GEN}\right) + 12t_c$$

Ahora se toma el mayor valor planteado para las comunicaciones $t_c = 3$ milisegundos y el menor valor para la función de síntesis de voz asíncrona $t(f_{say}^{SP-GEN}) = 1$ milisegundo, con el fin de realzar cualquier contribución negativa por parte de las comunicaciones entre módulos.

$$t_{Peer-to-Peer} = t(f_{goTo}^{MVN-PLN}) + 11ms$$

$$t_{Blackboard} = t(f_{goTo}^{MVN-PLN}) + 38ms$$

La tarea realizada consiste básicamente en que el robot se traslade de un lugar a otro aproximadamente 8.5 metros, y la hipótesis sostiene que no existe una diferencia apreciable en el desempeño por el uso de una arquitectura u otra. Dado que la tarea de trasladarse de un lugar a otro pocos metros suele ser del orden de segundos, una diferencia de 27 milisegundos no resulta significativa. Tal diferencia incluso no es perceptible por la mayoría de los seres humanos.

Aunado a esto, los resultados experimentales presentados en la sección 6.4 muestran que el tiempo de ejecución en una arquitectura *Blackboard* es de 73.17±2.35 segundos, y 73.10±2.63 segundos para una arquitectura *Peer-to-Peer*. Sustituyendo estos valores en las ecuaciones de tiempo anteriores se tiene:

$$t_{Peer-to-Peer} = t(f_{goTo}^{MVN-PLN}) + 0.011s = 73.10 \pm 2.63s$$

$$t_{Blackboard} = t(f_{goTo}^{MVN-PLN}) + 0.038s = 73.17 \pm 2.35s$$

Donde claramente los tiempos de comunicaciones son despreciables en comparación con el tiempo que tarda el desplazamiento del robot.

7. CONCLUSIONES

– Quiero dárselos, Señor.
– No los aceptaré, Andrew.
– A cambio de algo que usted puede darme, Señor.
– Ah, Qué es eso, Andrew?
– Mi libertad, Señor.
– Tu...
– Quiero comprar mi libertad, Señor.

Diálogo entre Andrew y Gerald Martin.
El Hombre Bicentenario. Isaac Asimov.

En este capítulo se presentan las conclusiones de la tesis obtenidas a partir del análisis realizado en los capítulos 4 y 5, así como de los resultados obtenidos en el capítulo 6 mediante un resumen de la investigación realizada, el replanteamiento de la hipótesis bajo los resultados obtenidos, las contribuciones y finalmente el trabajo futuro.

7.1. Sumario de la investigación (resumen)

La idea que inspira este trabajo de tesis es identificar la arquitectura de software para robots móviles de gama media más conveniente, mediante un análisis cuantitativo de las dos arquitecturas más utilizadas según la literatura existente.

La robótica móvil es un campo de investigación relativamente joven donde convergen muchas áreas de conocimiento, basándose no tanto en el desarrollo e investigación de nuevos algoritmos, como podrían ser de control o de inteligencia artificial, sino en la manera de integrarlos para que en conjunto permitan el funcionamiento de un robot, de manera autónoma, en un entorno dinámico.

En la investigación en robótica móvil es común que se prueben diferentes métodos de solución para un mismo problema. Por ejemplo el problema de la detección e identificación de objetos que se encuentran en situaciones no controladas de iluminación, posición, oclusión, e incluso con variantes de los objetos originales utilizados para desarrollar probar los algoritmos de detección y reconocimiento, requiere que varios de estos algoritmos sean probados e incluso adaptados, modificados o combinados a fin de obtener resultados satisfactorios.

Más aún, los procesos que ejecutan estos algoritmos no operan solos, compiten por recursos de cómputo que son utilizados por otros procesos con los que deben trabajar cooperativamente. En este escenario, el sistema pasa más tiempo en desarrollo que en ejecución, por lo que la elección de una arquitectura que reduzca los tiempos de desarrollo y aumente la cohesión del sistema se convierte en un factor muy importante.

Mucho de la investigación y desarrollo realizado en robótica móvil es realizado por estudiantes cuya permanencia en los proyectos es entre 2 y 5 años, tal como se menciona en [27; 29]. Un problema frecuente es que no se disponga del código fuente para adaptar un programa que funciona bien, o que los estudiantes no cuenten con los conocimientos para modificar los programas existentes. Por esta razón, una plataforma que permita la integración de componentes nuevos sin modificar los existentes, es una herramienta invaluable.

Para resolver estos problemas se han desarrollado diversas plataformas que facilitan la interacción de los programas que conforman el sistema que opera un robot móvil. Estas plataformas son desarrolladas basándose en la experiencia de otras áreas como la inteligencia artificial (*Blackboard*) o sistemas distribuidos (*Peer-to-Peer*) pero con el objetivo primario de resolver el problema de la comunicación y la coordinación de los componentes que conforman el sistema de software y sin detenerse a analizar el impacto que dicha arquitectura tiene en el sistema completo dentro de un área de investigación nueva y diferente.

7.2. Replanteamiento de la hipótesis

En la hipótesis (presentada en la sección 1.3) se plantea que:

> "El uso de una arquitectura basada en *Blackboard* ofrece mejor extensibilidad, reestructuración y similar desempeño comparado con una arquitectura basada en *Peer-to-Peer*, cuando la granularidad del sistema es media o gruesa."

Las características de extensibilidad y reestructuración señaladas en la hipótesis son aplicables al tiempo de desarrollo del sistema de software que opera un robot móvil de gama media, acorde al alcance definido en la sección 1.3.1, mientras que el desempeño del sistema sólo es válido en el contexto del tiempo de ejecución. Por esta razón, ambas partes se verifican por separado.

Para este fin, y en lo tocante al tiempo de desarrollo, se compara el costo de actualizar al sistema de software en cada una de las arquitecturas debido a cambios realizados en uno de sus componentes (extensibilidad) o en su organización (reestructuración), tomándose como medida de dicho costo el número de cambios a realizar de acuerdo con el análisis presentado en el capítulo 4.

Respecto al desempeño, el análisis teórico realizado en el capítulo 5 muestra que no debería existir una diferencia perceptible entre ambas arquitecturas. Este análisis debe ser corroborado con los datos experimentales obtenidos en el capítulo 6.

7.2.1. Discusión

Cuando se realiza investigación en robótica móvil, se pasa más tiempo desarrollando el sistema y adecuando los módulos a los cambios realizados en el sistema, que el tiempo que el sistema está en ejecución. Una arquitectura que permita reducir los tiempos de adaptación de los demás componentes del sistema cada vez que se realiza un cambio será muy valorada por los desarrolladores, que podrán dedicar su tiempo a adecuar el algoritmo y no a adecuar el sistema. No obstante, muchos desarrolladores dudarán el escoger una arquitectura que impacte negativamente en el desempeño del sistema, más aún cuando los recursos de cómputo son limitados como en el caso de los robots móviles, donde el robot debe llevar su propia fuente de alimentación y la mayor parte del espacio está reservada para sensores y actuadores.

Debido a que muchas de las métricas de software aplicables al tiempo de desarrollo del mismo son de índole cualitativo, no permiten establecer una comparación cuantitativa que pueda ser utilizada como factor ponderado o determinante en la elección de una arquitectura u otra. Por esta razón, se plantea una notación para describir un sistema de software y se propone un método que permita medir de forma cuantitativa características de tiempo de desarrollo del software, además de plantear un método para estimar el impacto en el desempeño del sistema el uso de una arquitectura, basándose en la comunicación entre sus componentes.

Se sabe que las arquitecturas centralizadas ofrecen un desempeño menor que las arquitecturas distribuidas por el cuello de botella que se genera en el elemento centralizado. Esto es cierto en sistemas donde componentes de software se comunican con otros componentes de software y los tiempos de respuesta son muy rápidos, del orden de microsegundos o menos. Sin embargo, los robots móviles dependen de sus sensores y sus actuadores, dispositivos físicos cuyo tiempo de respuesta es del orden de decenas de decenas o incluso centenares de milisegundo, es decir, dispositivos lentos.

Para un robot diseñado para interactuar con seres humanos que se mueve a una velocidad máxima de 0.5m/s y que percibe su entorno con un láser que ofrece 10 lecturas por segundo y una cámara de video que ofrece 30 cuadros por segundo, los procesos que realicen la adquisición de datos ofrecerán información nueva entre cada 30 y cada 100 milisegundos, cuando que los dispositivos de comunicación utilizados (comúnmente FAST Ethernet o Gigabit Ethernet) tienen una respuesta promedio menor a los 5 ms. Aún en el caso de que la carga en el procesador sea alta y esto ocasione que se pierdan una o dos lecturas de los sensores ¿qué significa una o dos lecturas en el entorno del robot? ¿Cuánto puede variar el entorno en medio segundo? ¿Es este medio segundo perceptible para los seres humanos que interactúan con el robot?

7.2.2. Interpretación y análisis de resultados

El análisis realizado a lo largo del Capítulo 4 plantea los escenarios más comunes de cambios que pueden presentarse durante el desarrollo de un sistema de software para robots móviles. Dicho análisis se realiza suponiendo que la comunicación entre los módulos se realiza directamente entre los módulos y mediante paso de mensajes.

De acuerdo a los resultados de los análisis de las características de tiempo de desarrollo: extensibilidad y reestructuración mostrados en la Tabla 4.1, el número de modificaciones requeridas para actualizar el sistema dado una extensión o una reestructuración es menor, en la mayoría de los casos, para una arquitectura *Blackboard* de los requeridos con una arquitectura *Peer-to-Peer*. Esta información se resume en Tabla 4.4, que presenta los casos promedio, eligiéndose la modificación más frecuente: cambio de definición de una función (su firma y tiempo de ejecución) para extensibilidad y la especialización de módulos en el caso de reestructuración.

Esta diferencia se muestra de manera más clara en el caso de ejemplo mostrado en la sección 4.5 donde, asumiendo un tiempo promedio de actualización de 5 minutos por operación (suponiendo apertura del código fuente, búsqueda de elementos a modificar, corrección y compilación), el tiempo total requerido para actualizar un sistema bajo arquitectura *Peer-to-Peer* de 40 minutos es el doble del tiempo requerido con un arquitectura *Blackboard* que toma solo 20 minutos. Si estas cantidades se extrapolan tomando como base las complejidades mostradas en la Tabla 4.1, es posible hacer estimaciones que no se basen sólo en la experiencia previa.

Es necesario aclarar que el análisis realizado en el Capítulo 4 supone que existe acceso al código fuente, es decir, que los módulos que componen al sistema pueden modificarse. En la práctica, esta suposición no siempre se cumple, pues el código fuente puede no estar disponible o su modificación no ser viable. Además, supone comunicación por paso de mensajes sin el uso de software intermediario que gestione las comunicaciones reduciendo el tiempo de desarrollo a costo de hacer al sistema dependiente de la plataforma.

Por ejemplo considérese el siguiente escenario: supóngase que un robot realiza detección de obstáculos usando un sensor láser y que el módulo del sensor envía los datos directamente al planeador de movimientos. Posteriormente desea añadirse la funcionalidad de localización usando el mismo dispositivo láser y no es posible modificar el módulo de láser. Bajo una arquitectura *Peer-to-Peer* hay serios problemas, pues es necesario reprogramar el módulo de láser para que envíe datos a dos programas; mientras que en arquitectura *Blackboard* los nuevos programas tendrán que leer los datos del laser del *Blackboard* por lo que no se requieren modificaciones. Por estas razones, cuantas menos modificaciones deban ser realizadas en el software que controla al robot, más fácilmente es desarrollar bajo esa arquitectura, ahorrando valioso tiempo que puede ser invertido en hallar soluciones y no en adaptar al sistema a los cambios.

Como se menciona a lo largo del Capítulo 3, existen tecnologías reducen significativamente los tiempos de actualización del sistema, como los servicios de resolución de nombres que eliminan la necesidad de una actualización dada la relocalización de un módulo, o plataformas como ROS, que enmascaran las comunicaciones en una arquitectura *Peer-to-Peer* facilitando el desarrollo a costa de restringir al uso de un sistema operativo y lenguaje de programación. El uso de estas tecnologías y plataformas es una opción viable para reducir los tiempos de desarrollo, sin embargo es necesario considerar y analizar las limitantes que imponen, así como uso de recursos que requieren para operar.

En lo referente al desempeño, los resultados experimentales comprueban el análisis teórico, pues el costo de las comunicaciones adicionales en una arquitectura basada en *Blackboard* no impacta de manera perceptible en el desempeño del robot, al punto de no hacer diferenciables las medidas entre el uso de una arquitectura u otra. Esto es válido sólo en el contexto de software para robots móviles de gama media que interactuarán con humanos.

Se hace hincapié en que el software que opera al robot está compuesto por funciones que pueden ser de granularidad media o gruesa, es decir, que el tiempo de ejecución de las funciones que componen un módulo puede ser tan pequeño como el tiempo de comunicación entre los módulos, o varios órdenes de magnitud mayor. No se considera granularidad fina (menor tiempo de ejecución que de comunicaciones). Si el tiempo de proceso es tan pequeño, conviene que la operación se realice dentro del mismo modulo, evitándose los tiempos de comunicación. Esto tiene sentido si se considera que los módulos agrupan funciones con un objetivo común y que los sensores y actuadores son lentos en comparación con la frecuencia de operación de un microprocesador.

7.3. Contribuciones

El objetivo del presente trabajo es analizar las fuerzas de las arquitecturas *Peer-to-Peer* y *Blackboard*, específicamente su extensibilidad, reestructuración y desempeño, lo que da lugar a las siguientes contribuciones:

1. Se plantea tanto una notación para describir un sistema de software (Sección 2.5) como un método para analizar de manera cuantitativa dos características de tiempo de desarrollo del software: la extensibilidad y la reestructuración, características que normalmente son de índole cualitativo (véase Capítulo 4). Aunque el análisis presentado se realiza dentro del contexto de la robótica móvil, las herramientas (notación y método) pueden ser utilizadas en otros contextos para el análisis de otros tipos de arquitectura.
2. Se presenta un método para analizar, desde el punto de vista teórico, el desempeño de dos arquitecturas de software equivalentes (*Peer-to-Peer* y *Blackboard*), con base en los tiempos de comunicación de sus elementos (Capítulo 5). Nuevamente, este método puede ser utilizado en otros contextos para el análisis de otras arquitecturas.
3. Se prueba que, cuando se trata de robots móviles de gama media (véase Capítulo 2), las arquitecturas *Peer-to-Peer* y *Blackboard* presentan un desempeño equivalente, ofreciendo la segunda mejores características de extensibilidad y reestructuración, tal como se muestra en los Capítulos 4, 5 y 6.

7.4. Trabajo Futuro

El presente trabajo plantea herramientas que permiten estimar el grado de extensibilidad y de reestructuración de un sistema de software con base en su arquitectura, así como para estimar el desempeño que tendrá el mismo en relación con la otra arquitectura. No obstante, existen otras características de tiempo de desarrollo y de ejecución que son importantes al diseñar un sistema de software, por ejemplo:

- ✓ Análisis de mantenibilidad: Es deseable que cuando se produzcan errores en un sistema de software estos puedan ser localizados y reparados de manera rápida, con un mínimo de cambios y efectos colaterales. Conviene realizar un análisis de mantenibilidad en ambas arquitecturas que permita conocer con qué arquitectura y bajo qué criterios es más fácil localizar y corregir los errores.
- ✓ Análisis de portabilidad: Cuando se diseña un sistema de software cuya ejecución puede ser distribuida o cuyo diseño puede tomar varios años, conviene elegir una arquitectura que ofrezca una mejor adaptación a cambios en la plataforma de hardware, sistemas operativos o incluso lenguajes de programación o compiladores.
- ✓ Tolerancia a fallas y robustez: Contar con elementos que permitan la selección de una arquitectura que facilite la creación de un software que se mantenga estable y operacional aún cuando se producen errores puede facilitar por mucho la tarea del desarrollador e impactar de manera positiva en el tiempo de desarrollo. Más aún en entornos donde el poder de cómputo es limitado y mecanismos como múltiple verificación y redundancia.

Además, esta tesis analiza solamente las arquitecturas *Peer-to-Peer* y *Blackboard* por ser las más utilizadas en la literatura de referencia. No obstante pueden existir otras, o incluso arquitecturas híbridas que se beneficien de las características de ambas.

8. ÍNDICE DE GRÁFICOS

9. ÍNDICE DE TABLAS

10. BIBLIOGRAFÍA

[1]. **BUSCHMANN, Frank, MEUNIER, Regine, ROHNERT, Hans, SOMMERLAND, Peter and STAL, Michael.** *Pattern-Oriented Software Architecture. A system of Patterns.* Chichester, United Kingdom. : John Wiley & Sons, Ltd., 1996.

[2]. *Towards a Standard EAI Quality Terminology.* **Losavio, Francisca, Ortega, Dinarle y Pérez, María.** Centro ISYS, Univ. Central de Venezuela, Venezuela : s.n., 2003. Chilean Computer Science Society, 2003. SCCC 2003. Proceedings. 23rd International Conference of the. págs. 119 - 129. 1522-4902.

[3]. **Robotics, International Federation of.** Industrial Robots. *International Federation of Robotics.* [En línea] [Citado el: 23 de Febrero de 2011.] http://www.ifr.org/.

[4]. **DOWLING, Kevin.** Robotics Frequently Asked Questions List. *Robot Institute of America.* [En línea] 1979. [Citado el: 23 de Febrero de 2011.] http://www.cs.cmu.edu/~chuck/robotpg/robofaq/.

[5]. **DEYLE, Travis.** Hizook { Robotics news for Academics & Professionals }. *An ISO Standard for Personal Care (Service) Robots.* [En línea] 18 de 07 de 2010. [Citado el: 25 de 03 de 2011.] http://www.hizook.com/blog/2010/07/18/iso-standard-personal-care-service-robots.

[6]. **HARPER, DOUGLAS.** Online Etimology Dictionary. [En línea] [Citado el: 26 de 02 de 2011.] http://www.etymonline.com/index.php?term=robot.

[7]. *Toward a framework for human–robot interaction.* **THRUN, Sebastian.** 2004, págs. 9-24.

[8]. **SHAW, Mary and GARLAN, David.** *An Introduction to Software Architecture.* Cernegie Mellon University : s.n., 1994.

[9]. **SHAW, Mary, GARLAN, David, ALLEN, Robert, KLEIN, Dan, OCKERBLOOM, John, SCOTT, Curtis and SCHUMACHER, Marco.** *Candidate Model Problems in Software Architecture.* Computer Science Department, Carnegie Mellon University : s.n., 1995. Version 1.3.

[10]. **VELI-PEKA, Eloranta, VESA-MATTI, Hartikainen, LEPPÄNEN, Marko, REIJONEN, Ville, HAIKALA, Ilkka, KOSKIMIES, Kai y MIKKONEN, Tommi.** *Patterns for Distributed Embedded Control System Software Architecture.* Tampere University of Technology : s.n., 2009.

[11]. **MILOJICIC, Dean, KALOGERAKI, Vana, LUKO, Rajan, NAGARAJA, Kiran, PRUYNE, Jim, RICHARD, Bruno, RILLINS, Sami and XU, Zhichen.** *Peer-to-Peer Computing.* HP Laboratories Palo Alto : s.n., 2003.

[12]. **SOMMERVILLE, Ian.** *Software Engineering.* 8th Edition. s.l. : China Machine Press, 2007.

[13]. **RUDENKO, D. and BORISOV, A.** *An overview of Blackboard architecture application for real tasks.* Institute of Information Technology, Riga Technical University : s.n., 2007.

[14]. *ViRbot: A system for the Operation of Mobile Robots.* **SAVAGE, Jesús, LLARENA, Adalberto, CARRERA, Gerardo, CUELLAR, Sergio y MINAMI, Yukihiro.** Mexico : s.n., 2008.

[15]. **CORMEN, Thomas H., LEISERSON, Charles E. y RIVEST, Ronald L.** Introduction to Algorithms. 2nd Edition. Cambridge, Massachusetts : The MIT Press, 2002.

[16]. **QUIGLEY, Morgan, GERKEY, Brian, CONLEY, Ken, FAUST, Josh, FOOTE, Tully, LEIBS, Jeremy, BERGER, Eric, WHEELER, Rob y NG, Andrew.** *ROS: An open-source Robot Operating System.* 2009.

[17]. **MOHAMMAD, Seyed.** *SRF: A Distributed robotic Framework.* 2010.

[18]. **MONTEMERLO, Michael, ROY, Nicholas, THRUN, Sebastian,HAEHNEL, Dirk, STACHNISS, Cyrill y GLOVER, Jared.** *CARMEN, Robot Navigation Toolkit.* [En línea] Carnegie Mellon, 1991. [Citado el: 26 de Junio de 2011.] http://carmen.sourceforge.net.

[19]. **Microsoft Corporation.** Microsoft Robotics Developer Studio. [En línea] 2006. [Citado el: 28 de Junio de 2011.] http://www.microsoft.com/robotics.

[20]. **ENDERLE, Stefan, UTZ, Hans, SABLATNÖG, Stefan, SIMON, Steffen, KRAETZSCHMAR, Gerhard y PALM, Günther.** *MIRO: Middleware for autonomous mobile robots.* . 2001.

[21]. **UTZ, Hans, SABLATNÖG, Stefan, ENDERLE, Stefan y KRAETZSCHMAR, Gerhard.** *Miro - Middleware for Mobile Robot Applications.* 2002.

[22]. **Oxford Mobile Robotics Group.** The MOOS homepage. [En línea] 2008. [Citado el: 25 de Julio de 2011.] http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php.

[23]. **CALISI, Daniele y CENSI, Andrea.** OpenRDK. [En línea] 2006. [Citado el: 27 de Junio de 2011.] http://openrdk.sourceforge.net.

[24]. **ZeroC.** ZeroC, the home of Ice. [En línea] ZeroC. [Citado el: 27 de 6 de 2011.] http://www.zeroc.com.

[25]. **MAKARENKO, Alexei, BROOKS, Alex y KAUPP, Tobias.** Orca: Components for robotics. [En línea] 2006. [Citado el: 25 de Julio de 2022.] http://orca-robotics.sourceforge.net.

[26]. **GERKEY, Brian, VAUGHAN, Richard y HOWARD, Andrew.** The Player Project. *Basic FAQ.* [En línea] 26 de Noviembre de 2010. [Citado el: 27 de Junio de 2011.] http://playerstage.sourceforge.net/wiki/Basic_FAQ.

[27]. **FARINELLI, Alessandro, GRISETTI, Giorgio y IOCCHI, Luca.** *SPQR-RDK a modular framework for programming mobile robots.* Università "La Sapienza" : s.n., 2004.

[28]. **SPIEGEL, Murray R.** *Probabilidad Y Estadistica (teoria y 760 problemas resueltos).* 2ª Edición. s.l. : McGraw Hill, 1979.

[29]. **CALISI, Daniele, CENSI, Andrea, IOCCHI, Luca Iocchi y NARDI, Daniele.** *OpenRDK: a modular framework for robotics software development.* 2008.

[30]. **MAKARENKO, Alexei, BROOKS, Alex y KAUPP, Tobias.** *Orca: Components for Robotics.* The University of Sydney, Australia : ARC Centre of Excellence in Autonomous Systems, 2006.

[31]. *An imitation of life.* **WALTER, W. Grey.** s.l. : Scientific American, 1950, págs. 42-45.

[32]. *Whose job is it anyway? A study of human-robot interaction in a collaborative task.* **HINDS, Pamela J., ROBERTS, Teresa L. y JONES, Hank.** 2004, Human-Computer Interaction, Vol. 19, págs. 151-181.