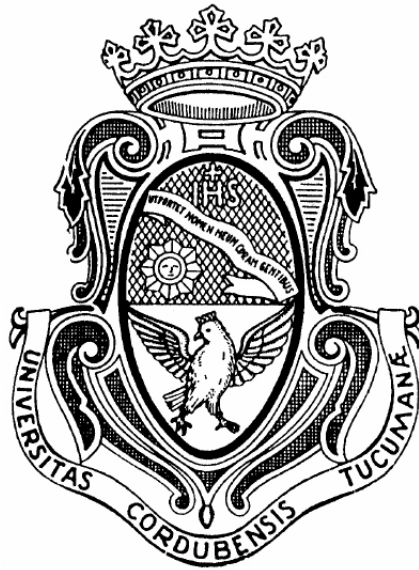


Proyecto Integrador

Interfaz JAUS para Herramientas de Desarrollo de Software de Robots



Facultad de Ciencias Exactas, Físicas y Naturales
UNC

Federico A. Bazán - Mauricio G. Jost

2009

Resumen

En la actualidad la robótica depende del software, y éste se ve afectado cada vez más por sus características de diseño. La modularización del software de un robot otorga importantes ventajas. Plantea a los equipos de desarrollo límites en cada módulo, facilitando así el trabajo en paralelo. Además, da lugar a la reutilización de un mismo módulo en otro sistema.

El Software Basado en Componentes gira en torno a esta idea. Este planteo requiere de ciertas definiciones, entre ellas, la de interfaces. El estándar JAUS define tanto un modelo de componentes con dichas interfaces como su arquitectura, en el contexto de la robótica. Las bondades del estándar dieron origen a un importante número de implementaciones en forma de SDK. Las mismas carecen de soporte multiplataforma, o las plataformas que soportan no siempre satisfacen las necesidades de los desarrolladores: desarrollo ágil de pruebas de concepto, recursos del lenguaje de programación (proyectos existentes, soporte de la comunidad, interfaz gráfica, etc).

Este trabajo extiende las posibilidades de implementación de un componente y permite que el mismo herede los beneficios de la plataforma seleccionada. Como resultado de su uso, se pueden integrar proyectos desarrollados en herramientas como MATLAB, Simulink, LabVIEW, las cuales son ampliamente utilizadas en robótica.

Agradecimientos

El presente trabajo no hubiera sido posible sin la buena predisposición y el asesoramiento de varias personas. Un sincero agradecimiento a todos aquellos que colaboraron con los autores en el desarrollo de I-JAUS.

En particular, se hace un reconocimiento a:

- Ing. Orlando Micolini, por su gran paciencia, guía y permanente apoyo.
- Integrantes del GRSI, por su disposición, aliento, y confianza en nosotros.
- Tom Galluzzo, por su soporte y amabilidad, y por poner a disposición del público el resultado de este trabajo.
- Don Riley, por su interés en nuestro desarrollo, y por permitirnos hacer uso de sus proyectos (*Robot Demo* y *3D Puma Robot Demo*).
- Dr. Ing. Bernd Plannerer, quien ha puesto a nuestra disposición el proyecto *Speech Recognition*, de manera desinteresada.
- Peter Aldrian, Uwe Meier, André Pura, por permitirnos hacer uso de su proyecto *Fast Eyetracking*.
- Ing. Pablo Passera por sus aportes prácticos sobre *Scrum*.

Por último y no menos importante, se agradece a nuestras familias por su apoyo y confianza en nuestra formación universitaria. En especial por su comprensión en aquellos momentos en que estuvimos ausentes abocados exclusivamente al estudio.

Índice general

1. Introducción	9
1.1. Motivación	9
1.2. Importancia	10
1.3. Alcance	10
1.4. Objetivos generales	10
1.5. Objetivos específicos	10
1.6. Metodología	11
 I Contexto de trabajo	 12
2. Obtención de Requerimientos	13
2.1. Situación del grupo <i>cliente</i>	14
2.1.1. Proyectos en curso	14
2.2. Requerimientos	16
2.3. Replanteo de requerimientos	17
2.4. Replanteo de objetivos	18
2.5. Análisis de Riesgos	18
2.5.1. Riesgos universales	19
2.5.2. Riesgos específicos	19

II	Marco teórico	24
3.	Desarrollo de Robots	25
3.1.	Robots en la actualidad	25
3.1.1.	Armamento y Seguridad	26
3.1.2.	Asistencia y Entretenimiento	26
3.2.	Software de robots	29
3.2.1.	Componentes en un robot	29
3.2.2.	Componentes de software	31
4.	JAUS	36
4.1.	Breve reseña histórica	36
4.2.	Documentación	37
4.2.1.	<i>Reference Architecture (RA)</i>	37
4.3.	Jerarquía de componentes	39
4.3.1.	Componentes JAUS	40
4.3.2.	Componentes de Comando y Control	40
4.3.3.	Componentes de Comunicaciones	40
4.3.4.	<i>Manipulator Components</i>	41
4.4.	JAUS y otros estándares	44
4.5.	Implementaciones y herramientas de JAUS	45
4.5.1.	Alternativas	45
4.5.2.	Comparación	46
4.6.	OpenJAUS	48
4.6.1.	Aspectos de implementación	49
4.6.2.	OpenJAUS como plataforma	54
4.6.3.	Manipulador con OpenJAUS	56
III	Desarrollo	57
5.	Interfaz JAUS para HDR	58
5.1.	Herramientas de desarrollo de robots	58

5.1.1. LabVIEW	58
5.1.2. MATLAB	59
5.1.3. Simulink	60
5.2. JAUS y Herramientas de Desarrollo en Robótica	60
6. Diseño y Uso de I-JAUS	63
6.1. Documentación	63
6.2. Estilo de codificación	63
6.2.1. Fuentes en C	63
6.2.2. Fuentes en MATLAB	64
6.3. Estructura de usuario	64
6.4. Arquitectura de Componentes	66
6.5. Integridad de Datos	67
6.5.1. OpenJAUS	67
6.5.2. OpenJAUS y HDR	68
6.6. Llamadas a la librería	70
6.6.1. Nombre de las llamadas	71
6.6.2. Puesta en funcionamiento de un componente	72
6.6.3. Envío de mensajes desde I-JAUS	74
6.6.4. Recepción de mensajes	74
6.6.5. Campos de tamaño variable	75
6.6.6. Solicitud de mensaje en espera	77
6.6.7. Acceso a mensajes	79
6.6.8. Otras llamadas	80
6.6.9. Finalización	81
6.7. Retorno de errores	81
6.8. Uso de I-JAUS en HDR	82
6.8.1. LabVIEW	82
6.8.2. MATLAB	84
6.8.3. Simulink	85
7. Validación y Verificación	86

7.1. Matriz de trazabilidad	100
8. Demostraciones	102
8.1. PICOLEBOT	102
8.1.1. Utilizando <i>Joystick (OpenJAUS - C)</i>	103
8.1.2. Utilizando <i>Mouse (MATLAB)</i>	103
8.1.3. Utilizando <i>voz (MATLAB)</i>	105
8.1.4. Utilizando <i>seguimiento ocular (MATLAB)</i>	106
8.2. Robot Laparoscópico	108
8.3. Modelo PUMA 762 3D	108
8.3.1. Utilizando <i>GUI (LabVIEW)</i>	109
9. Conclusiones	111
Glosario	114
Bibliografía	117
Apéndices	119
A. Manual de Referencia de I-JAUS	120
B. Ejemplos de I-JAUS	121

Índice de cuadros

2.1. Análisis de riesgos universales.	19
2.2. Análisis de Riesgos de Requerimientos.	21
2.3. Análisis de Riesgos de Tecnología.	22
2.4. Análisis de Riesgos de Herramientas.	23
2.5. Análisis de Riesgos de Estimación.	23
4.1. Comparación entre algunos estándares de interoperabilidad.	44
4.2. Comparación entre algunas implementaciones de JAUS.	47
6.1. Tipos de uso más frecuente en I-JAUS.	70
6.2. Formato de algunas llamadas I-JAUS.	71
6.3. Resumen de las llamadas para el envío de mensajes.	74
6.4. Llamadas particulares de I-JAUS.	81
7.1. Matriz de trazabilidad.	101

Índice de figuras

2.1. Brazo Laparoscópico del GRSI en su plataforma de pruebas.	15
2.2. Modelo del Brazo Exoesqueleto del GRSI.	15
3.1. Vehículo R-Gator.	26
3.2. Pequeño HOAP-3.	27
3.3. HRP-2m Chromet.	27
3.4. Robot <i>Deka's Luke Arm</i>	28
3.5. Exoesqueleto HAL-5.	28
3.6. Componentes habituales en la arquitectura del software de un robot.	31
4.1. Jerarquía de elementos de JAUS.	39
4.2. Componentes <i>Manipulator</i>	42
4.3. Componente JAUS según implementación OpenJAUS.	49
5.1. Componente JAUS implementado en HDR.	61
5.2. Ejemplo de evolución de la implementación de un componente.	61
6.1. Disposición de los datos en estructuras ante el servicio <i>JPS</i>	65
6.2. Llamadas primitivas de I-JAUS.	66
6.3. Arquitectura de I-JAUS.	67
6.4. Hilo de ejecución en un componente OpenJAUS.	68
6.5. Productor/Consumidor como consecuencia de un segundo hilo de ejecución.	69
6.6. Memoria reservada por I-JAUS. Forma insegura.	76

6.7. Memoria reservada por HDR. Forma segura.	76
6.8. Comportamiento de las diferentes colas de preservación de orden de mensajes recibidos.	78
6.9. Procesos en ejecución dentro de un nodo JAUS.	82
6.10. Configuración desde LabVIEW de una llamada a la librería de I-JAUS.	83
6.11. Componente JAUS EEPD corriendo en LabVIEW (al estilo panel de control). . . .	83
6.12. Componente Subsystem Commander corriendo en MATLAB (consola).	84
8.1. PICOLEBOT escribiendo.	102
8.2. Diagrama del subsistema: SSC y EEPD.	103
8.3. SSC en MATLAB donde la posición del puntero determina la del extremo efector. .	104
8.4. Diagrama del subsistema: SSC y EEPD/PM.	104
8.5. Resultados de la prueba.	105
8.6. Manipulación del extremo efector mediante la voz.	105
8.7. Diagrama del subsistema: SSC y EEPD/JPS.	106
8.8. Cuadrantes de reconocimiento de pupila.	106
8.9. Resultados del reconocimiento de ojos.	107
8.10. Comportamiento del PICOLEBOT ante un movimiento ocular.	107
8.11. Demostración haciendo uso del Robot Laparoscópico.	108
8.12. Modelo 3D del robot industrial PUMA 762.	109
8.13. Componentes de LabVIEW (panel SSC) y MATLAB (PUMA3D en EEPD) interactuando.	109

1

Introducción

1.1. Motivación

El equipo GRSI¹ dirige anualmente una elevada cantidad de proyectos del campo de la robótica, llevados adelante por los estudiantes de la Facultad. Cada uno de estos desarrollos representa una posibilidad importante de progreso. Los proyectos son interdisciplinarios, y adicionalmente involucran muchos integrantes. Cada uno de ellos aporta su bloque al sistema.

Los trabajos se producen en un contexto de tiempos reducidos, y ello generalmente impide a los alumnos converger en el uso de interfaces reconocidas. Cada uno crea su *protocolo* específico, entonces lograr su reutilización obliga al equipo a un proceso de interpretación y adaptación a una interfaz conveniente. Esto implica tiempo y esfuerzo. Muchas veces requiere también de fortuna, puesto que la documentación creada no siempre es útil, y los alumnos quedan fuera del alcance del equipo en unos pocos años. Como consecuencia la reusabilidad es parcial, y muchas veces son desperdiciados trabajos de gran calidad. Es necesaria la normalización de las interfaces para lograr una correcta integración de los bloques, incluso aquellos provenientes de otros grupos.

Por otro lado, la robótica se muestra un tema de vanguardia, que está y estará presente en la vida del hombre en todos sus aspectos. La medicina no es la excepción. De hecho, uno de los proyectos del grupo *cliente* está destinado a la ayuda de un hospital de la región. Contribuir es un orgullo para los autores.

¹Grupo de Robótica y Sistemas Integrados (GRSI) de la Facultad de Ciencias Exactas, Físicas y Naturales de la UNC. A lo largo del texto también se lo denominará grupo *cliente* puesto que establece los requerimientos del proyecto.

1.2. Importancia

Tanto el GRSI como los demás grupos involucrados en el desarrollo de software para robots se han topado alguna vez con este problema: reusabilidad. El *Departamento de Defensa de los EEUU* ha hecho su aporte a este tema, dando origen al estándar JAUS (ver Capítulo 4 JAUS).

JAUS posee numerosas implementaciones. Sin embargo, ninguna de ellas permite tomar provecho de herramientas de uso común en el campo de la robótica, como son MATLAB (ver Sección 5.1.2 MATLAB), Simulink (ver Sección 5.1.3 Simulink) y LabVIEW (ver Sección 5.1.1 LabVIEW). Como consecuencia, el grupo debería primero reescribir el código de un proyecto, para luego hacer uso de JAUS. Recién allí estaría en condiciones de lograr cierta reutilización. La importancia de este trabajo radica fundamentalmente en la solución de esta problemática.

1.3. Alcance

El dominio de la robótica y de JAUS es extenso. Sin embargo, las necesidades del GRSI son puntuales. Este proyecto hace foco en el software de la robótica de manipuladores, que es el área donde actualmente el equipo hace hincapié.

No se busca brindar los componentes JAUS en sí, sino las herramientas que permitan la creación de los mismos. Esto es importante: deja en claro que se brindan elementos que contribuyen a solucionar no sólo los problemas de interfaz de proyectos actuales, sino también los del futuro.

1.4. Objetivos generales

El objetivo principal que persigue este trabajo es:

- Proveer una herramienta de trabajo basada en un estándar, para que el equipo *cliente* pueda explotar los beneficios del software basado en componentes dentro de la robótica de los manipuladores.

1.5. Objetivos específicos

Los objetivos específicos del proyecto son:

- Evaluar el estándar JAUS como alternativa.
- Determinar bajo qué entornos de software es conveniente que la herramienta funcione.

- Realizar una prueba de concepto con la herramienta.
- Estimar el impacto del producto final.

En la Sección [2.4 Replanteo de objetivos](#) se presenta un replanteo de objetivos, que da origen al siguiente objetivo adicional.

- Crear una herramienta de desarrollo para MATLAB, Simulink y LabVIEW que permita adaptar proyectos existentes en el GRSI al estándar JAUS.

1.6. Metodología

Se obtuvo información sobre el contexto y las necesidades del equipo *cliente* a través de entrevistas.

La metodología de desarrollo adoptada para el producto de este trabajo es [Scrum](#) [[Palacio, 2006](#)]. La razón de la elección fue la elevada probabilidad de cambio de requerimientos (ver Sección [2.5.2 Riesgos de Requerimientos](#)), dado lo innovador del tema. Los modelos que presume esta metodología son *evolutivo* e *incremental*. La duración de los ciclos o [Sprints](#) adoptada fue de dos a tres semanas, según se consideraba conveniente acorde a los objetivos de cada uno.

Fueron validados los requerimientos mediante pruebas experimentales y prototipos, presentadas a lo largo de diversas reuniones.



Contexto de trabajo

2

Obtención de Requerimientos

En este proyecto el cambio de rumbo fue una constante durante las primeras fases. Esto se debió a muchas razones, entre ellas la falta de experiencia relacionada a la robótica y a los estándares. También se debió en gran medida a la falta de claridad de requerimientos funcionales: se debe entender que existió un tiempo de ajuste en el cual los dos grupos (desarrolladores y *clientes*) delimitaron sus responsabilidades, y aprendieron qué es conveniente realizar en cada uno.

El grupo *cliente* se trata de un grupo de investigación, y sus proyectos muchas veces surgen gracias a la libertad con la cual se desenvuelven. Realizan pruebas exhaustivas de tecnologías y metodologías, y adoptan proyectos de los más diversos campos (como Ing. en Rehabilitación, Procesos Industriales, etc.).

Para este caso, se considera que la mencionada libertad permitió un mejor enfoque en los problemas del grupo, y entender cuál era la mejor manera de contribuir. Bajo la situación de cambio de requerimientos con frecuencia, el proceso de desarrollo adoptado fue *Scrum*[Kniberg, 2007]. Consecuentemente, las técnicas de obtención de requerimientos han sido principalmente las descriptas a continuación.

- **Entrevistas.** Han sido de enorme importancia durante todo el desarrollo del proyecto. Se han utilizado tanto *entrevistas abiertas* como *cerradas* (o *estructuradas*). Las primeras estuvieron destinadas a dar la posibilidad al grupo *cliente* de expplayarse en los puntos que consideraban importantes. Las segundas, permitieron indagar sobre cuestiones de interés para los desarrolladores. Los temas tratados estuvieron principalmente orientados hacia las necesidades del grupo, pero en reiteradas ocasiones fue necesario investigar la forma de trabajo, los hábitos, entre

otras cosas. Esto último permitió enfocar el proyecto hacia ciertas soluciones. Por ejemplo, las demostraciones fueron hechas mediante implementaciones que se esperaba el usuario realice. Si esas demostraciones se crean siguiendo los lineamientos del grupo *cliente* se contribuye sin introducir cambios innecesarios en su forma de trabajo, y se prueba que el producto es realmente útil.

- **Demostraciones.** Consistieron en pequeños trozos de proyecto susceptibles de ser entregados, prototipos. Esta forma de presentación está muy relacionada al ciclo de desarrollo escogido, y permite tener en todo momento una realimentación del cliente.

Sin embargo, las demostraciones consumen tiempo, principalmente cuando su frecuencia es elevada (1 demostración cada 2 semanas). Aún con esta desventaja, es necesario destacar que una demostración dispara una serie de ideas y correcciones que el *cliente* considera oportunas, y que validan las suposiciones del desarrollador.

2.1. Situación del grupo *cliente*

Durante las primeras reuniones el grupo *cliente* fue indagado sobre algunos ítems específicos. Estos permitieron una primera aproximación al problema, y serán descriptos a continuación.

2.1.1. Proyectos en curso

En esta sección serán detallados aquellos proyectos que se encuentran en desarrollo dentro del grupo *cliente*. Si bien existen otros, se citarán únicamente los que estén asociados a este trabajo.

Robot Laparoscópico

Uno de los emprendimientos del grupo consiste en un Brazo Laparoscópico, un robot especialmente diseñado para dar soporte al laparoscopia durante una operación. El laparoscopia es una cámara utilizada en ciertas intervenciones quirúrgicas.



Figura 2.1: Brazo Laparoscópico del GRSI en su plataforma de pruebas.

Aquí, una de las motivaciones más importantes es el costo de los brazos laparoscópicos comerciales. El grupo pretende construir un brazo de características similares a aquellos, pero con costos muy inferiores. Existe una segunda motivación: un brazo robótico es una herramienta potencialmente útil en los más diversos ámbitos. Intervenciones quirúrgicas, procesos industriales, rehabilitación, entretenimiento y accionar militar son algunas de las áreas donde tiene mayor aceptación y demanda esta clase de robots. Su reutilización (en varios de estos contextos) depende del grado de flexibilidad del diseño, y es un punto a desarrollar. Realizar ajustes en el mismo campo de trabajo para amoldarse a las necesidades puntuales no sólo de una situación, sino incluso de cierto profesional al mando, es más que valioso para cualquier usuario.

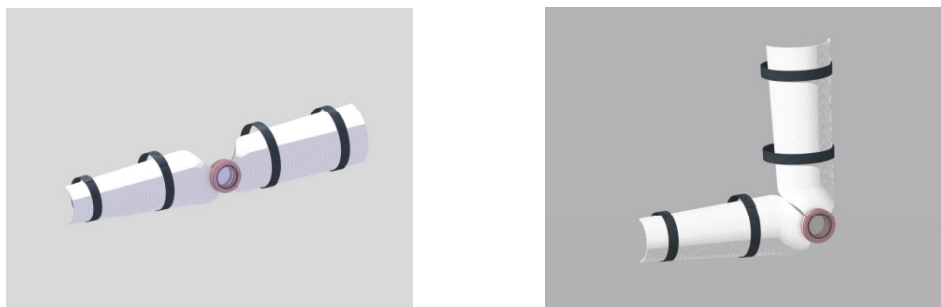


Figura 2.2: Modelo del Brazo Exoesqueleto del GRSI.

"...Con el desarrollo propuesto, el robot será de arquitectura flexible capaz de adaptarse fácilmente a las necesidades del cirujano como a él mejor convenga, con una precisión similar a los importados y mejorable con el tiempo. Además, la ventaja que representa este sistema es la de ofrecer al cirujano laparoscópico, la posibilidad de elegir el sistema de navegación que le facilite operar. Es decir, adoptar al igual que procedimientos quirúrgicos, el sistema que le ofrezca mayor comodidad para obtener confort, ergonomía, rapidez y precisión para posicionar la endocámara. Se considera que la aplicación de la optoelectrónica combinada con algún programa de procesamiento de imágenes permitirá en un futuro cercano una navegación del laparoscopia en

la cavidad abdominal de manera automática de acuerdo a la posición de los instrumentos...”¹

Como es citado, es necesario que el brazo disponga de **varios puntos de entrada**. Esto quiere decir que, según el especialista considere conveniente, utilizará mando por voz, mando por joystick, control automático, etc. El desafío involucra también la posibilidad de soportar periféricos innovadores como *Eye Tracking*² y *OCZ*³, e incluso aquellos periféricos del futuro. Esta capacidad de interconexión entre diferentes bloques de un sistema requiere de interfaces bien definidas (sin ambigüedades), reconocidas (para disminuir tiempos de aprendizaje), maduras y fundamentalmente aceptadas. El estándar JAUS es entonces considerado como principal alternativa por ellos mismos, tal como puede verse más adelante.

JAUS

Dado que el grupo está inserto en un ambiente académico, dirige muchos trabajos finales de alumnos estudiantes de ingeniería: sistemas de control robóticos, de reconocimiento de voz, procesamiento de señales mioeléctricas, etc. Sin embargo, el uso de los mismos está generalmente limitado por su capacidad de interconexión.

El planteo del estándar JAUS como medio de acceso común a cada implementación busca solucionar este dilema. En el futuro, cada proyecto debería presentarse con una interfaz estandarizada, así cualquier otro desarrollo que cumpla el mismo estándar sería fácilmente conectado con el primero para constituir un sistema más complejo.

Otros proyectos

Otros proyectos en vista de desarrollo son *Sistema de Seguimiento por Visión y Teleoperación Industrial basada en Web*. Es importante notar que ambos proyectos pueden ser pensados como bloques: uno de entrada (o de manejo/administración) y un bloque de salida (o de actuación), en su forma más primitiva. Aún con esta sencilla configuración, la reusabilidad es posible.

2.2. Requerimientos

En un primer encuentro se pusieron de manifiesto los principales problemas presentes en el grupo *cliente*: interoperabilidad y reusabilidad. Resultan temas centrales a resolver puesto que el grupo lleva adelante proyectos de similares temáticas, que conforman bloques, cuya interconexión

¹Extraído de <http://www.efn.uncor.edu/investigacion/robotica/investigacion/robotlaparoscopico.htm>, octubre de 2009.

²Extraído de <http://www.nosolousabilidad.com/articulos/eye-tracking.htm>, octubre de 2009.

³Extraído de <http://www.dailytech.com/article.aspx?newsid=10379>, octubre de 2009.

es sinónimo de progreso. Además, por tratarse de un grupo de investigación, es una certeza que en un futuro cercano continúen trabajando en el mismo campo.

El grupo propuso el estándar JAUS como posible solución. El mismo está principalmente destinado a vehículos no tripulados, sin embargo cuenta con una sección dedicada a manipuladores.

También fueron puestas de manifiesto algunas de las ideas que se tienen a futuro, principalmente como parte de la evolución del brazo laparoscópico. Las mismas pretenden lograr un software eficiente, utilizar un sistema operativo *Real Time* y un embebido que reduzca el tamaño del hardware involucrado. Esto último es muy importante para facilitar el traslado y posicionamiento del producto en una sala de intervenciones quirúrgicas, o en el sitio que corresponda.

A continuación se enumeran los requerimientos resultantes de las reuniones:

- Elementos que permitan la reutilización de bloques existentes.
- Mecanismo estandarizado de interconexión de módulos existentes y nuevos.

2.3. Replanteo de requerimientos

Luego del estudio de los estándares que se ajustan a los requerimientos iniciales, se hizo un análisis comparativo. La elección fue JAUS. Su justificación puede verse en la Sección [4.4 JAUS y otros estándares](#).

Una vez definidos los requerimientos iniciales, se realizaron nuevas entrevistas para refinarlos. También se optó por un primer conjunto de demostraciones que fueron de gran utilidad.

En dichas demostraciones se presentaron pruebas de concepto desarrolladas en OpenJAUS, una de varias implementaciones del estándar JAUS. La justificación de esta elección puede encontrarse en la Sección [4.5 Implementaciones y herramientas de JAUS](#). Cada prueba fue desarrollada en lenguaje C y ejecutada en un sistema operativo Linux.

La primera importante realimentación fue dada a luz: tanto el sistema operativo Linux, como el lenguaje C eran ajenos a los hábitos de trabajo del GRSI. Si bien su aprendizaje es posible, esto no representaba una solución en el corto plazo. Desarrollar una herramienta para el lenguaje C constituía más un inconveniente que una solución. Esto generaría:

- Un impedimento para **reutilizar** el software desarrollado por el GRSI (bajo MATLAB, Simulink y LabVIEW).
- Un incremento en la complejidad para aprender el estándar seleccionado.

En consecuencia, se reconocieron aquellas herramientas comúnmente utilizadas en cualquiera de sus proyectos. Como se ha mencionado, eran MATLAB, Simulink y LabVIEW. Se trata de

herramientas conocidas por todos los integrantes del grupo, y frecuentemente utilizadas en el campo de la robótica.

Como resultado, se reformularon los **requerimientos**.

- 1. Herramienta sencilla utilizable desde MATLAB, Simulink y LabVIEW enfocada en manipuladores, que permita:
 - 1.1. Iniciar un componente JAUS que se integre a un sistema del estándar, y finalizarlo.
 - 1.2. Tener control sobre el estado de dicho componente.
 - 1.3. Enviar y recibir mensajes del estándar a través del mismo.
 - 1.4. Disponer de los mecanismos indispensables de JAUS.
- 2. Documentación asociada a dicha herramienta.
- 3. Ejemplos de uso.

2.4. Replanteo de objetivos

En consideración de los nuevos requerimientos, ha surgido un nuevo objetivo específico, que se añade a la lista inicial (Sección [1.5 Objetivos específicos](#)):

- Crear una herramienta de desarrollo para MATLAB, Simulink y LabVIEW que permita adaptar proyectos existentes en el GRSI al estándar JAUS.

Todos los objetivos del proyecto son mencionados en el Capítulo [1 Introducción](#).

2.5. Análisis de Riesgos

El riesgo puede ser definido como *una probabilidad de que una circunstancia adversa ocurra*[\[Sommerville, 2005\]](#). Los riesgos constituyen una amenaza tanto para el proyecto en sí, como para la organización que lo lleva a cabo. Su identificación y valoración son elementos importantes a la hora de realizar acciones que los eviten, en caso de ser esto posible. Por otro lado, en caso de ser inevitable su desencadenamiento, es indispensable identificarlos a tiempo y tomar medidas correctas para enfrentar sus consecuencias.

En primer lugar es necesario identificar las circunstancias no deseadas. Luego se debe asignar a cada una cierta probabilidad de ocurrencia e impacto sobre el proyecto (en caso de ocurrir).

A continuación se presentará el conjunto de riesgos significativos. En primer lugar se hará mención de aquellos que son comunes a la mayoría de los desarrollos (denominados *riesgos universales*), y luego se presentarán aquellos más específicos de este caso particular.

2.5.1. Riesgos universales

Definidos en [Sommerville, 2005] estos riesgos son recurrentes para la mayoría de los proyectos. Este caso no ha sido la excepción.

Riesgo	Probabilidad	Efecto
No disponibilidad de recursos. Existen algunos recursos que son necesarios para llevar adelante un proyecto. Muchas veces estos no están disponibles, o lo están pero no en el momento indicado. Para el caso se debe pensar principalmente en algún prototipo de brazo robótico que permita realizar verificaciones con un grado más de aproximación al campo. No disponer de esto limita al proyecto, y oculta a los desarrolladores maneras de facilitar el uso, que quizá serían viables con muy poco esfuerzo extra.	Moderada	Serio
Cambio de requerimientos. Desarrollado en la Sección 2.5.2.	–	–
Subestimación del tamaño. JAUS es el estándar de interoperabilidad a analizar. No disponer de personas idóneas en el tema, o alguna experiencia previa en su uso, es una indiscutible razón para dudar de las estimaciones de tiempo.	Alta	Serio
Cambio de tecnología. Ocurre cuando por alguna razón la tecnología (en el sentido amplio de la palabra) utilizada queda obsoleta. Se puede pensar tanto en <i>hardware</i> (cierto embebido, etc.), como en <i>software</i> (herramientas de desarrollo, lenguajes de programación, etc.). Es posible incluir en este punto la tecnología asociada al estándar: un estándar posee versiones, y a medida que evoluciona descarta elementos que se consideran innecesarios o inapropiados, e incluye mejoras. Hacer uso de partes del estándar sujetas a cambios es siempre un riesgo. Que el cambio ocurra durante la misma implementación es un efecto verdaderamente no deseado. Actualmente JAUS (el estándar en la mira) se encuentra en la versión 3.3.	Baja	Tolerable

Tabla 2.1: Análisis de riesgos universales.

2.5.2. Riesgos específicos

En la referencia bibliográfica se ha hecho una distinción entre los diferentes tipos de riesgos (no excluyentes entre sí):

- **Riesgos del Proyecto.** Afectan el calendario o los recursos del proyecto.
- **Riesgos del Producto.** Afectan la calidad y/o rendimiento del producto.
- **Riesgos del Negocio.** Afectan a la organización que desarrolla el producto.

Los *Riesgos del Negocio* existen, pero no son importantes para el caso, fundamentalmente por tratarse de un proyecto con fines académicos.

El conjunto de riesgos involucrados en este trabajo está dado principalmente por los *Riesgos del Proyecto*. Esto es así porque existe una gran cantidad de factores que introducen incertidumbre en el proceso: robótica, estándares de interoperabilidad, y JAUS, son algunos de los conceptos con los que es indispensable familiarizarse antes de realizar afirmaciones suficientemente fundadas. Adicionalmente, y basado en el hecho de que se busca realizar un aporte a un grupo de desarrollo existente, **deben** mostrarse avances útiles etapa tras etapa. Caso contrario la esencia del trabajo se estaría perdiendo.

Los *Riesgos del Producto* existen, y más que nada están asociados a su calidad: documentación, fiabilidad, extensibilidad, etc. Para el caso, se puede pensar sencillamente en ellos como los riesgos que están ubicados luego de la etapa de definición del **qué**. Una vez consolidada la idea que se persigue, es necesario un diseño robusto, y una implementación de fácil uso, con posibilidades de extensión, con gran documentación.

Existe otra clasificación para los riesgos:

- **Riesgos de Personal.** Asociado a las personas del equipo de desarrollo.
- **Riesgos de Requerimientos.** Cambios de requerimientos y medidas para reajustar el rumbo.
- **Riesgos de Tecnología.** Relacionado a *hardware* o *software* de terceros que se utiliza en el producto.
- **Riesgos Organizacionales.** Derivan del entorno donde el proyecto está siendo desarrollado.
- **Riesgos de Herramientas.** Asociado a las herramientas de apoyo utilizadas para el desarrollo.
- **Riesgos de Estimación.** Relacionado a las presunciones realizadas durante la planificación del proyecto.

A continuación se desarrollarán aquellos tipos de riesgos que se consideran significativos para este proyecto, de acuerdo a esta segunda clasificación.

Riesgos de Requerimientos

Estos riesgos están asociados al grupo *cliente*, y a factores influyentes en la/s etapa/s de obtención de requerimientos.

Riesgo	Probabilidad	Efecto
Cambio de requerimientos. Un grupo de investigación en muchos casos dispone de flexibilidad en su rumbo de desarrollo. La naturaleza de su trabajo se basa muchas veces en la posibilidad de ver en simples ideas una gran oportunidad. No obstante, en la posición de clientes, el grupo se ha mostrado abierto, y más que <i>requerimientos</i> ha detallado escenarios. Estos han sido estudiadas, y transformados en requerimientos de manera casi continua, con su validación posterior. Aunque el permanente cambio de requerimientos es esperado, los retrasos consecuentes son inevitables.	Muy alta	Tolerable
Disponibilidad del grupo cliente. El tiempo del que dispone el grupo <i>cliente</i> para el proyecto común puede no ser el esperado. La ausencia de un sólo miembro significa que durante las reuniones habrá más información que pasa desapercibida (en uno y otro sentido), y más medidas incorrectas y tomadas que son pasadas por alto. Principalmente en las primeras etapas, asociadas a la obtención de requerimientos, esto puede tener una repercusión considerable en la planificación.	Moderada	Tolerable
Falta de consenso en el cliente. No siempre los grupos saben resolver disputas internas. No poder encontrar una solución que satisfaga las necesidades de varios integrantes en desacuerdo constituye un riesgo. Aun cuando este problema no es de los desarrolladores, repercute en los tiempos de desarrollo.	Baja	Tolerable

Tabla 2.2: Análisis de Riesgos de Requerimientos.

Riesgos de Tecnología

Estos riesgos guardan relación con aquellos elementos (sean *hardware* o *software*) de terceros que son utilizados por el producto. Un claro ejemplo de esto son los [SDK's](#), *kit's* de desarrollo que poseen implementaciones parciales de un tema específico, y que tienden a reducir los tiempos de diseño e implementación de un producto.

Riesgo	Probabilidad	Efecto
Imposibilidad de acceso a SDK's útiles. En la mayoría de los casos un proyecto surge a partir de otro. Al hablar de estándares de interconexión de bloques se podría esperar disponer de alguna implementación preexistente. Contar con un kit de desarrollo orientado a las necesidades del proyecto es de gran utilidad, principalmente a la hora de lograr demostraciones en etapas tempranas de desarrollo. Por los tiempos con los cuales cuenta el proyecto, el carácter de importancia de este punto aumenta considerablemente, sea tanto porque no existe tal herramienta, como porque su costo excede las posibilidades del proyecto.	Baja	Catastrófico
Bajo nivel de madurez de SDK's. Disponer únicamente de herramientas mal documentadas, incompletas (no contando con las partes esperadas o necesarias), difíciles de abordar, con elevada cantidad de errores, o con una comunidad de soporte de bajo nivel de actividad, constituyen riesgos de características similares al del punto anterior.	Baja	Catastrófico
Difícil aprendizaje de las SDK's. Por lo general, todo <i>kit</i> de desarrollo tiende a facilitar al usuario el abordaje de cierto tema. Sin embargo, existen casos donde se dan por supuesto conocimientos que son ajenos al común de los desarrolladores. No se trata de un riesgo importante, pero el estudio de conceptos <i>de base</i> debe ser considerado en la planificación.	Moderada	Tolerable
Imposibilidad de acceso a prototipos robóticos. Construir una herramienta y no ponerla en funcionamiento en el entorno para el cual fue diseñada, quita valor agregado al entregable. Tanto su validación como su verificación no serían hechas sino en un marco ficticio. La calidad de las conclusiones en base a esta realimentación crece a medida que los desarrolladores se involucran más en el campo al que está destinada la herramienta.	Baja	Catastrófico

Tabla 2.3: *Análisis de Riesgos de Tecnología.*

Riesgos de Herramientas

Las herramientas utilizadas para desarrollar (como *IDE's* de cierto lenguaje de programación) constituyen riesgos en sí mismas.

Riesgo	Probabilidad	Efecto
Imposibilidad de acceso a ciertas herramientas. Aunque siempre existen medios alternativos, las funcionalidades de cierta herramienta pueden ser indispensables por alguna razón. <i>Profiling</i> y depuración son funcionalidades no tan comunes (o poco maduras en ciertos casos) que pueden ser requeridas.	Moderada	Tolerable
Difícil aprendizaje de las herramientas. Al ingresar a un nuevo campo, es común enfrentarse con nuevos conceptos, y por ello con nuevas herramientas. No siempre estas se muestran amigables, y su aprendizaje puede llevar un tiempo importante.	Moderada	Tolerable

Tabla 2.4: *Análisis de Riesgos de Herramientas.*

Riesgos de Estimación

Riesgo	Probabilidad	Efecto
Falta de contactos idóneos. Al enfrentarse a un problema con elevada cuota de incertidumbre, es siempre una buena idea recurrir a ayuda. No disponer de un apoyo experimentado en el tema puede convertirse en un problema serio, principalmente si las dudas no se disipan con la maduración de los conocimientos y el tiempo. En el mejor de los casos se puede contar con una persona del entorno, aunque investigadores (incluso del exterior) están siempre dispuestos a ayudar.	Baja	Tolerable

Tabla 2.5: *Análisis de Riesgos de Estimación.*



Marco teórico

3

Desarrollo de Robots

En este capítulo se presentará la actualidad de la robótica, y se enunciarán consideraciones importantes asociadas al desarrollo de software en dicha área.

3.1. Robots en la actualidad

En el último siglo, los robots han tomado un papel trascendental en la industria. Han permitido realizar tareas de alto riesgo, en un tiempo menor, y suplantando a uno o más seres humanos. Esto ha provocado un crecimiento en la producción y la automatización de los procesos de fabricación, en cada una de sus etapas. Esto incluye desde la obtención de materia prima, hasta el ensamblaje del producto final. Las consecuencias se reflejan en una mayor eficiencia y reducción de costos.

En las últimas décadas, los robots han ganado mayor protagonismo en muchas otras áreas, como ser la ciencia y la tecnología, el entretenimiento y el armamentismo. Manipulan sustancias tóxicas, soportan condiciones extremas en tareas de exploración, manipulan explosivos, colaboran en cirugías, interactúan en forma lúdica con seres humanos, e incluso aprenden de su entorno.

Se presentan en distintas formas: aparatos rudimentarios, extremidades, humanoides, mascotas, insectos, vehículos, etc. Están capacitados para realizar distintos tipos de movimientos, según cuenten con ruedas, alas y/o articulaciones.

En distintos ámbitos, públicos y privados, como empresas, universidades y entidades nacionales, se llevan adelante proyectos de desarrollo, los cuales son una clara muestra de cómo se han insertado en la vida humana. Más aún, en el presente están en desarrollo prototipos que realizan diversas tareas cotidianas para el ser humano, como ser asesoramiento, recepción, vigilancia y educación.

Esta actualidad de la robótica pone en evidencia que el presente trabajo está en contacto con un área en expansión. Al igual que otros grupos de trabajo, es posible realizar un aporte que facilite el desarrollo de robots, tanto en el presente como en el futuro. A pesar de que este trabajo está centrado en la rama de los manipuladores, se considera importante mostrar algunos de los emprendimientos vanguardistas de la robótica general.

3.1.1. Armamento y Seguridad

En la actualidad las grandes potencias mundiales cuentan con vehículos no tripulados especialmente desarrollados para el combate. Se tiene conocimiento de autos, tanques, aviones, lanchas y submarinos comandados de forma remota desde una base. También han desarrollado robots dedicados a la seguridad, con capacidad para alertar sobre situaciones de riesgo y actuar ante ellas.

R-Gator

Se trata de un vehículo no tripulado utilizado en misiones militares (Figura 3.1). Es capaz de realizar tareas de exploración de forma comandada o autónoma. Cuenta con GPS y un sistema operativo Linux embebido, llamado *BlueCat*.



Figura 3.1: Vehículo R-Gator.

3.1.2. Asistencia y Entretenimiento

Por estos días es posible toparse con los más insólitos prototipos robóticos, que movilizan al amante de las novedades electrónicas y convocan multitudes turísticas generando importantes ganancias para sus propietarios.

A su vez, la interacción con las máquinas está en un profundo cambio. Gracias a la evolución de distintas disciplinas, como ser Inteligencia Artificial, Nanotecnología y Biotecnología, el ser

humano puede entablar relaciones con los robots, cada vez más parecidas a las que tiene con otros seres vivos. Es posible ver robots que expresan reacciones y sentimientos. Inclusive la apariencia de los mismos ha tomado formas cotidianas. En las grandes ciudades del mundo, se puede jugar, conversar y consultar a robots con rasgos humanos.

HOAP-3

Se trata de un robot capaz de aprender nuevos movimientos y utilizarlos en la situación que crea más conveniente (Figura 3.2). Interactúa con quien le enseña a través de un control remoto. Mientras aprende, habla para informar qué es lo que esta haciendo.



Figura 3.2: Pequeño HOAP-3.



Figura 3.3: HRP-2m Choromet.

HRP-2m Choromet

Este robot (Figura 3.3) realiza una gran cantidad de movimientos. Estos lo hacen apto para grupos de investigación académicos, que estudian sistemas de control aplicados a robots. Cuenta con un Sistema Operativo Linux.

Deka's Luke Arm

Se trata de un brazo robótico cuyo mentor es Dean Kamen, fundador de *DEKA Research and Development*. DEKA utiliza un sistema de control comandado desde el pie, con funcionamiento similar a un joystick: el usuario maniobra el brazo haciendo presión en diferentes partes de un conjunto de sensores ubicados en el pie.

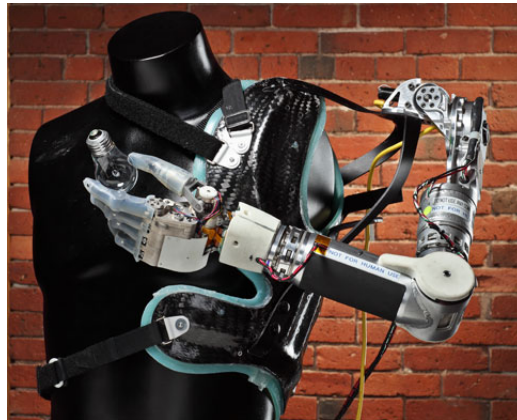


Figura 3.4: Robot Deka's Luke Arm.

HAL-5

Existen dispositivos robóticos que ayudan a personas en una etapa de rehabilitación. Este es el caso del HAL-5, presentado como un exoesqueleto que potencia las capacidades humanas, para mantenerse en pie y levantar cargas.



Figura 3.5: Exoesqueleto HAL-5.

3.2. Software de robots

En cualquier desarrollo de software es deseable obtener resultados conformados en bloques, módulos o componentes. La idea que gira en torno de este tipo de paradigmas es generar elementos abstractos, que se mapeen directamente con el modelo que pretenden representar. Así un sistema que simula al cuerpo humano podrá ser constituido por medio de objetos tales como un corazón, un hígado, dos de ojos, etc. Así también, un corazón podrá estar compuesto por objetos tales como válvulas, venas, arterias, etc. o bien tratarse de un objeto monolítico; basta con que el corazón funcione como tal ante el resto del sistema.

Cada bloque posee tanto elementos que determinan su estado, como elementos que determinan su comportamiento. En el mundo de la programación orientada a objetos se llamaría a estos elementos atributos y métodos, respectivamente. No es casual el gran crecimiento de estos modelos de programación en la actualidad.

Por otro lado, un bloque puede ser pensado como un componente. En tal caso contaría con un conjunto de funcionalidades, y con una interfaz definida, mediante la cual lograr la interconexión con el resto del sistema.

Los desarrollos actuales poseen requerimientos de tiempo menos holgados, los equipos son extensos, y los procesadores presentan mayores capacidades de procesamiento. Así un desarrollo abstracto no tiene un impacto negativo tan grande en el rendimiento final del sistema, y posibilita al equipo de desarrollo involucrado trabajar conjuntamente, y lograr resultados en tiempos más cortos. La modularidad es la base del trabajo grupal.

El software involucrado en robots no está exento de esta realidad.

3.2.1. Componentes en un robot

Herramientas de desarrollo de un componente

Los equipos de investigación de robótica utilizan una gran variedad de herramientas de desarrollo, lo que les permite generar las partes de un robot de manera más conveniente. Aplicaciones como MATLAB (*Matrix Laboratory*) y LabVIEW (*Laboratory Virtual Instrumentation Engineering Workbench*) son ejemplos de mucha aceptación en la comunidad. A lo largo del texto se hablará de estas [Herramientas de Desarrollo en Robótica \(HDR\)](#).

Las facilidades que brindan estas aplicaciones para desarrollar **prototipos** son excelentes. Permiten que un trabajo que sería engorroso en lenguajes de bajo nivel tenga un costo de apenas unas horas. Sin embargo, para etapas posteriores donde se busca eficiencia, la implementación en lenguajes como C es más conveniente. Ambas etapas son habituales en desarrollos de robótica.

No es sencillo sugerir un sólo entorno para todos los proyectos. Incluso, es difícil establecer un sólo entorno para un mismo proyecto. Imagínese un equipo destinado a implementar un robot de asistencia quirúrgica. Se tendrán objetivos y opiniones diversas, acerca de qué herramienta usar. El encargado del reconocimiento de imágenes se inclinará indudablemente por instrumentos que faciliten su labor, como LabVIEW y sus agregados especiales. Por otro lado, el conjunto de modelos matemáticos que estén involucrados será mucho más sencillo de integrar para entornos como MATLAB. Además se debe tener en cuenta que existen numerosas implementaciones libres que con poco esfuerzo pueden formar parte del sistema.

¿Cómo es posible generar un proyecto con tanta disparidad entre sus componentes?

Se pueden considerar dos alternativas. La primera consiste en promover a que todo el grupo trabaje bajo un mismo entorno. La segunda consiste en apoyar a los intereses de cada integrante, y lograr la interacción entre los componentes de diferentes entornos.

La naturaleza de este proyecto está enfocada en esta última idea: **lograr interoperabilidad entre componentes desarrollados bajo diferentes HDR, basándose para ello en un estándar ya existente.**

Interacción entre los componentes

Cada componente en un sistema cumple una funcionalidad. Es interesante notar que esta funcionalidad suele presentarse en diseños robóticos una y otra vez. Un ejemplo es el procesamiento de datos provenientes de sensores, que en la mayoría de los casos consiste en la conversión de un valor de tensión a la unidad física correspondiente. También se puede hablar de la generación de señales que controlan actuadores (como motores, luces, altavoces, etc.). Más aún, existen bloques de control, cuyo algoritmo determina la interacción entre los dos anteriormente mencionados. Esto significa que a partir de cierto dato en un sensor, se realiza cierta acción sobre un actuador.

Lograr que todos los elementos de un sistema se entiendan no es un trabajo menor. Partir de un proyecto robótico sencillo conduce muchas veces a desarrolladores novatos a crear un estándar de comunicación simplemente perfecto para el caso. Sin embargo, al dar más complejidad al sistema, el protocolo comienza a mostrarse falto de consistencia, haciendo que muchas veces sea necesario el replanteo del mismo, con todo el costo que esto conlleva.

Una aproximación más fructífera (aunque también más laboriosa) consiste en cumplir con un estándar ya existente de intercomunicación entre componentes. Así, se asegura la utilización de un idioma en común ampliamente aceptado y maduro, para la comunicación.

En la siguiente figura se muestra una posible configuración entre componentes y su interacción.

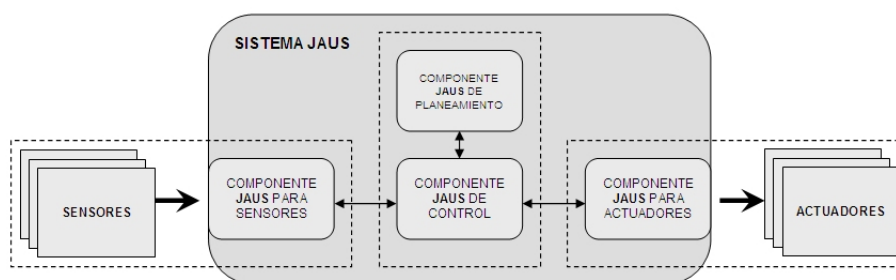


Figura 3.6: Componentes habituales en la arquitectura del software de un robot.

En este ejemplo, el componente para sensores se comporta como mediador. Desde el punto de vista de los sensores mismos, se muestra como un sistema **host** convencional, pues se comunica con él de la forma esperada (usando un *driver*, o un mensaje específico por cierta interfaz de la PC, etc.). Desde la perspectiva del resto de los componentes, se muestra como uno más, hablando el lenguaje común entre los mismos. Esto también ocurre con el componente para actuadores.

La Figura 3.6 muestra que los dos componentes periféricos ocultan al resto del sistema la tecnología de los sensores y los actuadores, respectivamente. Cambiar el componente intermedio, que lleva a cabo el control de los actuadores, no debería generar inconvenientes si se lo reemplaza por otro componente que hable el mismo idioma (respeta la misma interfaz). Asimismo, cambiar un sensor no debería ser problema si este posee su propio componente *envolvente*, que también habla el idioma de los demás.

¿Cuál es el idioma *en común*?

Debe ser un idioma apropiado, que contemple la modularidad. Debe también ser a medida de las necesidades de las intercomunicaciones en el campo de la robótica.

En el Capítulo 4 se presenta el idioma *en común*, denominado **JAUS**. Dado que es una arquitectura basada en componentes, se presenta a continuación una pequeña introducción teórica en busca de sus beneficios para el desarrollo de software.

3.2.2. Componentes de software

Antes que nada es necesario tener presente la definición del término:

"Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio." [Szyperski, 1998].

El componente constituye por naturaleza una unidad de composición de software. Su nacimiento se debió principalmente al crecimiento de los sistemas de software, a su incremental complejidad, a la mayor demanda de fiabilidad y cortos plazos.

El secreto del éxito de los componentes radica en la clara definición de sus interfaces, que son los medios mediante los cuales se accede a sus servicios. Estos deben ser tales que satisfagan los requisitos que cita la definición anterior.

Cuando en la definición se hace referencia a *"desarrollado [...] de forma independiente, en tiempo y espacio"* se está hablando de independencia total en términos de implementación. Dos componentes pueden ser desarrollados en ámbitos completamente diferentes, pero gracias a la claridad de la definición de sus interfaces cuentan con un mecanismo fijo de comunicación que les permite ser interconectados con éxito.

Durante la etapa de ejecución, los componentes cuentan también con cierta independencia espacial: estos no necesariamente tienen que ser ejecutados en el mismo sistema. Basta con que el mecanismo de comunicación entre ellos soporte (o sea soportado por) una red, para que se hable de componentes sobre sistemas distribuidos.

Características

Existe un conjunto de rasgos que son característicos de un componente [Sommerville, 2005]:

- **Estandarizado.** La estandarización de componentes significa que un componente usado en un proceso [CBSE](#) tiene que ajustarse a algún modelo estandarizado de componentes. Este modelo puede definir interfaces de componentes, metadatos de componentes, documentación, composición y despliegue.
- **Independiente.** Un componente debería ser independiente, debería ser posible componerlo y desplegarlo sin tener que utilizar otros componentes específicos. En las situaciones en las que el componente necesita servicios proporcionados externamente, éstos deberían hacerse explícitos en una especificación de interfaz del tipo «requiere».
- **Componible.** Para que un componente sea componible, todas las interacciones externas deben tener lugar a través de interfaces definidas públicamente. Además debe proporcionar acceso externo a la información sobre sí mismo, como por ejemplo a sus métodos y atributos.
- **Desplegable.** Para ser desplegable, un componente debe ser independiente y debe ser capaz de funcionar como una entidad autónoma o sobre una plataforma de componentes que implemente el modelo de componentes. Esto normalmente significa que el componente es binario y que no tiene que compilarse antes de ser desplegado.

- **Documentado.** *Los componentes tienen que estar completamente documentados para que los usuarios potenciales puedan decidir si los componentes satisfacen o no sus necesidades. La sintaxis e, idealmente, la semántica de todas las interfaces de componentes tienen que ser especificadas.*

Componentes y objetos

Probablemente el lector conozca el concepto de objeto, y se encuentre con dudas acerca de su relación con los componentes. En [Sommerville, 2005] se hace una interesante distinción entre ambas entidades:

- **Los componentes son entidades desplegables.** *Es decir, no son compilados en un programa de aplicación, sino que se instalan directamente sobre una plataforma de ejecución. Los métodos y atributos definidos en sus interfaces pueden ser accedidos por otros componentes.*
- **Los componentes no definen tipos.** *Una definición de clase define un tipo abstracto de datos y los objetos son instancias de ese tipo. Un componente es una instancia, no una plantilla que se utiliza para definir una instancia.*
- **Las implementaciones de los componentes son opacas.** *Los componentes están, al menos en principio, completamente definidos por la especificación de su interfaz. La implementación está oculta para los usuarios de los componentes. Los componentes a menudo se entregan como unidades binarias de forma que el comprador no tiene acceso a la implementación.*
- **Los componentes son independientes del lenguaje.** *Las clases de objetos tienen que seguir las reglas de un lenguaje de programación orientado a objetos particular, y generalmente, sólo pueden interoperar con otras clases escritas en dicho lenguaje. Si bien los componentes se implementan normalmente utilizando lenguajes orientados a objetos tales como Java, pueden implementarse en lenguajes de programación no orientados a objetos.*
- **Los componentes están estandarizados.** *A diferencia de clases de objetos que pueden implementarse de cualquier forma, los componentes deben ajustarse a algún modelo de componentes que restringe su implementación.*

Hasta este punto se ha nombrado reiteradas veces el concepto de *modelo de componentes*. Consiste en un conjunto de restricciones y reglas que todos los componentes de un mismo sistema deben respetar. JAUS es en sí un modelo de componentes, puesto que establece dichas reglas; pero adicionalmente JAUS brinda un conjunto de componentes de base que cumplen las mismas. Por esto JAUS es comúnmente citado como una *arquitectura de componentes*: involucra tanto un

modelo de componentes, como un conjunto de componentes definidos, así como la forma en que se interconectan para conformar un sistema.

Programación Orientada a Componentes

Los componentes representan beneficios al ajustarse a sistemas distribuidos, pero durante el desarrollo existen otras ventajas. Para conocerlas es necesario hablar de la Programación Orientada a Componentes (POC), mencionada en [Fuentes, n.d.] de la siguiente manera:

La POC nace con el objetivo de construir un mercado global de componentes software, cuyos usuarios son los propios desarrolladores de aplicaciones que necesitan reutilizar componentes ya hechos y probados para construir sus aplicaciones de forma más rápida y robusta.

Sin embargo, el nacimiento de esta clase de forma de trabajo tuvo otras importantes causas. En [Sommerville, 2005] se hace referencia a la CBSE de la siguiente manera:

La ingeniería del software basada en componentes (CBSE) surgió a finales de los 90 como una aproximación basada en reutilización al desarrollo de sistemas software. Su creación fue motivada por la frustración de los diseñadores de que el desarrollo orientado a objetos no había conducido a una reutilización extensiva, tal y como se sugirió originalmente. Las clases de objetos simples eran demasiado detalladas y específicas, y a menudo tenían que ser enlazadas con aplicaciones en tiempo de compilación. Había que tener un conocimiento detallado de las clases para usarlas, lo cual generalmente significaba que había que tener acceso al código fuente del componente. Esto hizo que fuera difícil el marketing de objetos como componentes reutilizables. A pesar de las predicciones optimistas iniciales, nunca se desarrolló un mercado significativo para objetos individuales.

Su objetivo principal es la **reutilización** de componentes, pero llevando este concepto hasta ideas de mercadotecnia como distribución, adquisición e incorporación de los mismos a un sistema:

Así, la Programación Orientada a Objetos (POO) ha sido el sustento de la ingeniería del software para los sistemas cerrados. Sin embargo, se ha mostrado insuficiente al tratar de aplicar sus técnicas para el desarrollo de aplicaciones en entornos abiertos. En particular, se ha observado que no permite expresar claramente la distinción entre los aspectos computacionales y meramente composicionales de la aplicación, y que hace prevalecer la visión de objeto sobre la de componente, estos últimos como unidades de composición independientes de las aplicaciones. Asimismo, tampoco tiene en cuenta los factores de mercadotecnia necesarios en un mundo real, como la distribución, adquisición e incorporación de componentes a los sistemas.

Cuando se habla de un *entorno abierto* se hace referencia a un sistema cuyo tiempo de vida es superior al tiempo de vida de los componentes que lo constituyen. Esto quiere decir que se

trata de un sistema evolutivo, donde los componentes son agregados, reemplazados, y eliminados durante la vida del mismo.

4

JAUS

JAUS es un estándar que define tanto una arquitectura de componentes, como sus interfaces, basadas en mensajes. Todo el estándar hace foco en la intercomunicación entre sistemas robóticos no tripulados, así como entre los componentes constituyentes de un mismo sistema. Este trabajo considera a JAUS el principal estándar a evaluar para lograr los objetivos propuestos, principalmente por las características que posee. Estas, serán enunciadas a lo largo del capítulo, y además será incluida una comparación con otros estándares de similar alcance.

Es importante destacar que JAUS ha sido desarrollado por un grupo de trabajo conformado por organizaciones militares y comerciales, todas voluntarias. JAUS ya ha sido adoptado como un estándar de SAE¹, lo cual muestra cuán prometedor es.

4.1. Breve reseña histórica

En 1998 la Oficina de la Subsecretaría de Defensa de los EEUU (*OUSD* por sus siglas en inglés) impulsó un grupo de trabajo para que cree un estándar, cuyo fin sea la interoperabilidad de vehículos no tripulados terrestres (llamados *UGV*, por sus siglas en inglés). El estándar fue creado y llamado *JAUGS (Joint Architecture for Unmanned Ground Systems)*. Este fue luego generalizado, y ya no sólo contempló vehículos terrestres, sino que abarcó todos los tipos de sistemas no tripulados.

En 2005 el estándar fue adoptado por SAE. A partir de ese momento es formalmente llamado AS-4/JAUS, aunque habitualmente se lo cita como JAUS. La versión del estándar tenida en cuenta para este trabajo es la 3.3.

JAUS provee una perspectiva con independencia de tecnología y de plataforma para sistemas

¹SAE International (siglas de *Society of Automotive Engineers*) es la más grande sociedad de membresía a nivel mundial, dedicada fundamentalmente a la ingeniería en el área de vehículos.

no tripulados, para su comunicación e interacción. Este es el corazón del estándar. JAUS otorga el método común para las comunicaciones con sistemas no tripulados creados hoy y mañana.

4.2. Documentación

La documentación de JAUS está dividida en varias partes.

1. **Domain Model (DM)**. Este documento representa la motivación y los objetivos de JAUS. No está enfocado en especificaciones técnicas del estándar, sino en aquello que lo hace importante.
2. **Reference Architecture (RA)**. Detallado más adelante.
3. **Standard Operating Procedures (SOP)**. Establece el procedimiento mediante el cual el *JAUS Working Group* conduce su negocio.
4. **Document Control Plan (DCP)**. Establece el procedimiento para controlar la configuración de las bases de JAUS.
5. **Strategic Plan (SP)**. Fija las declaraciones de perspectiva y misión de JAUS; planea la educación, expansión y aceptación de la arquitectura.
6. **Compliance Plan (CP)**. Establece políticas, procedimientos, pruebas, y reportes para determinar compatibilidad con el estándar JAUS.

Para el proyecto son particularmente importantes los documentos del *Reference Architecture* (RA).

4.2.1. Reference Architecture (RA)

En el documento RA (también *Reference Architecture Specification*) se especifica el estándar, la arquitectura de sus componentes, la definición de servicios, la definición del formato de los mensajes. También define todos los mensajes, y sus campos: significado, unidades, interpretación, etc.

Posee 3 partes: *Architecture Framework* [RA-3.3 P1, 2007], *Message Definition* [RA-3.3 P2, 2007] y *Message Set* [RA-3.3 P3, 2007].

Architecture Framework

Provee una descripción de la estructura de sistemas basados en JAUS. Sirve para una primera aproximación desde los requerimientos planteados en DM hacia el conjunto de mensajes JAUS.

También constituye una guía para lograr con el uso de mensajes las capacidades especificadas en DM.

Plantea las bases sobre los cuales está construido JAUS:

- Independencia de plataforma.
- Aislamiento de misiones.
- Independencia del hardware de la computadora.
- Independencia de la tecnología.

También desarrolla la topología de un sistema JAUS, basada en **subsistema, nodo, componente e instancia**.

Debe tratarse con cuidado el término **componente**. A lo largo de este texto puede ser utilizado para referirse tanto al elemento de la jerarquía JAUS, como a una instancia particular de cierto componente.

Su contenido incluye la definición de cada uno de los componentes/servicios de JAUS. Estos son tratados con más profundidad en la Sección [4.3.1 Componentes JAUS](#).

Documento *Message Definition*

Define los *tipos* utilizados en los campos de los mensajes JAUS.

Especifica la cabecera general de los mensajes JAUS y establece una forma de clasificación de los diferentes tipos de mensajes del estándar, aunque no los define.

Hace hincapié en las reglas de comunicación, incluyendo el mecanismo *Service Connection*.

Documento *Message Set*

Especifica uno a uno los mensajes, sus campos, su forma de interpretación (incluyendo unidades de medida), y su razón de ser.

También presenta una lista de los servicios soportados por cada componente JAUS.

4.3. Jerarquía de componentes

Se recomienda al lector familiarizarse con los términos de esta sección, leyendo para ello el Glosario.

Como se ha mencionado, JAUS es una arquitectura de **componentes**. Cada componente posee una funcionalidad determinada.

Un **sistema** puede ser pensado como un conjunto de robots interactuando entre sí. El **subsistema** es entonces un único robot del mismo, es una unidad autocontenida del sistema general. Más específicamente, un subsistema puede ser tanto un robot, como una Unidad de Control del Operador (*Operator Control Unit*, **OCU**).

Cada subsistema posee varios componentes que lo constituyen. Estos componentes no necesariamente son puestos en marcha dentro de una misma computadora. Cada computadora que constituye al subsistema es generalmente tratada como un **nodo**[Rowe, n.d.]. En otras palabras, los componentes de un subsistema están dispersos en varios nodos. Esto es mostrado en la Figura 4.1.

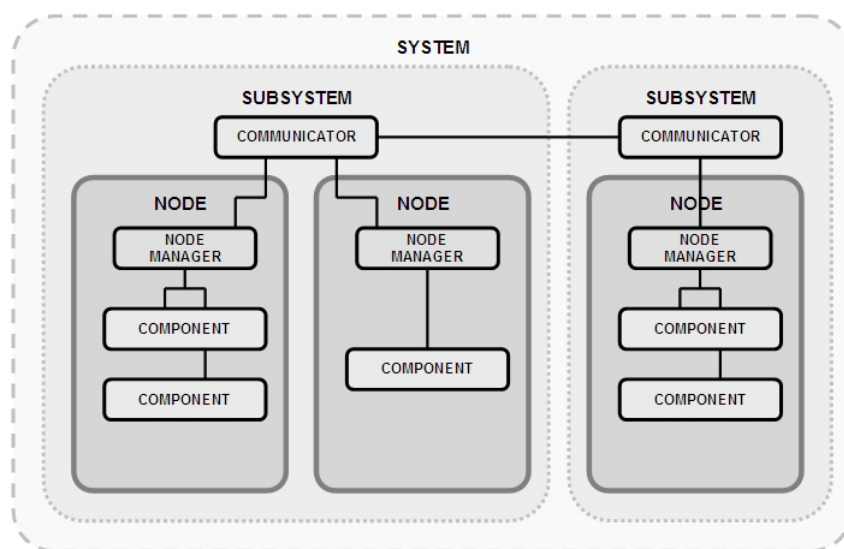


Figura 4.1: Jerarquía de elementos de JAUS.

Un componente cumple cierta funcionalidad del subsistema en el que se encuentra. Cada componente puede poseer varias **instancias** de sí mismo dentro de un nodo, según se crea conveniente.

J AUS deja la distribución física de los componentes en los nodos a criterio del diseñador.

4.3.1. Componentes JAUS

Existen 6 grupos de componentes. Estos son:

1. *Command and Control Components* (2 componentes).
2. *Communications Components* (2 componentes).
3. *Platform Components* (11 componentes).
4. *Manipulator Components* (10 componentes).
5. *Environment Sensor Components* (3 componentes).
6. *Planning Components* (2 componentes).

Este trabajo hace especial hincapié en el Grupo de *Manipulator Components*. Sin embargo, existen algunos componentes fuera de dicho grupo que son importantes para cualquier sistema JAUS compatible. Todos estos, se explicarán en las siguientes secciones.

4.3.2. Componentes de Comando y Control

Los componentes de comando y control son:

- *System Commander* (SC, Administrador de Sistema, ID JAUS 40).
- *Subsystem Commander* (SSC, Administrador de Subsistema, ID JAUS 32).

Ambos componentes ejecutan las tareas necesarias para coordinar la actividad en todo el sistema/subsistema.

Realizan planeamiento de misión, entrega de comandos, y verificación de estado del resto de los componentes mediante consultas. Estas tareas pueden ser hechas por una computadora, o por una persona, a través de una interfaz gráfica. En este último caso se estaría hablando de un OCU.

Pueden enviar y recibir cualquier tipo de mensaje JAUS.

4.3.3. Componentes de Comunicaciones

Este grupo es de especial importancia. Está compuesto por los componentes *Communicator* y *Node Manager*[Rowe, n.d.]. El propósito de ambos está relacionado principalmente al ruteo de mensajes JAUS. Se los describirá a continuación.

Componente *Communicator* (ID JAUS 35)

En cada subsistema, existe un componente *Communicator*. Es el componente destinado a lograr la comunicación entre los nodos de un subsistema.

Una primera regla de enrutamiento de mensajes JAUS[RA-3.3 P2, 2007] establece:

- *Los mensajes desde y hasta un subsistema deben ir a través del comunicador.*

Esta regla le da carácter de indispensable al *Communicator* dada la múltiple existencia de subsistemas.

Componente *Node Manager* (ID JAUS 1)

Asimismo, dentro de un mismo nodo, el encargado de enviar y recibir los mensajes a los componentes, es el *Node Manager*.

Las reglas de enrutamiento segunda y tercera son:

- *Los mensajes desde y hasta un nodo deben ir a través del Node Manager.*
- *Los mensajes internos de un nodo (de componente a componente) deben ir a través del Node Manager.*

Estas reglas le dan carácter de indispensable, y único por nodo, al *Node Manager*.

4.3.4. *Manipulator Components*

Este grupo de componentes fue diseñado especialmente para el trabajo con *manipuladores* montados en las plataformas JAUS. Un manipulador puede pensarse como un brazo robótico. Disponer de uno le permite a la plataforma móvil incrementar el conocimiento y control sobre el contexto físico que lo rodea. Esto ocurre mediante el uso de alguna herramienta, cámara, etc.

De acuerdo al alcance de este trabajo, los componentes manipuladores constituyen el conjunto sobre el que se hace foco. Esto ocurre a causa de las necesidades del GRSI, como puede verse en la Sección [2.3 Replanteo de requerimientos](#).

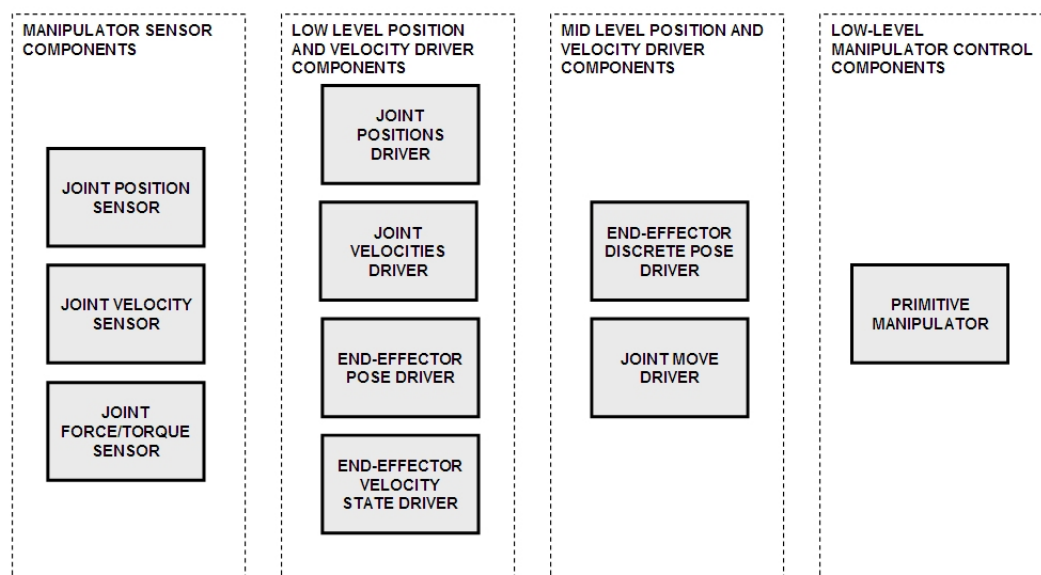


Figura 4.2: Componentes Manipulator.

A continuación se realizará una breve descripción respecto de cada uno de ellos.

- **Primitive Manipulator (PM, Manipulador Primitivo, ID JAUS 49).** Este componente provee el servicio de manejo de los actuadores del manipulador. Cada articulación posee un actuador que recibe un porcentaje de esfuerzo de la máxima potencia aplicable, sea en un sentido o en otro (-100 % a 100 %).

Adicionalmente este componente brinda información indispensable sobre el manipulador, especificaciones técnicas del mismo: para cada articulación brinda un tipo, velocidad máxima de desplazamiento/giro², límites de movimiento, etc.

- **Joint Position Sensor (JPS, Sensor de Posición de Articulación, ID JAUS 51).** Este componente brinda como servicio los datos instantáneos asociados a la posición³ de cada una de las articulaciones del manipulador. Algún componente le solicita la información, y él se encarga de entregarla.

Para que esto sea posible, este componente debe interactuar con algún dispositivo de medición de posiciones dispuesto en el manipulador.

- **Joint Velocity Sensor (JVS, Sensor de Velocidad de Articulación, ID JAUS 52).** Similar al anterior, este componente entrega información asociada a la velocidad instantánea de cada

²Dependiendo del tipo de articulación, de revolución o prismática, el movimiento se mide en radianes o en metros, respectivamente.

³Entiéndase por posición el ángulo de giro (para articulaciones de revolución) o el desplazamiento (para articulaciones prismáticas).

articulación.

- **Joint Force/Torque Sensor (JFTS, Sensor de Fuerza/Torque de Articulación, ID JAUS 53).**
Similar a los dos anteriores, este componente brinda información asociada a la fuerza/torque instantáneo de cada articulación, según sea su tipo⁴.
- **Joint Positions Driver (JPD, Controlador de Posiciones de Articulación, ID JAUS 54).**
Este componente realiza control a lazo cerrado de la posición de las articulaciones del manipulador. Posee como entradas: un valor de posición deseada para cada articulación (generalmente del componente OCU), las posiciones actuales, y las especificaciones del manipulador. Como salida se tiene un nivel de esfuerzo para cada articulación.
- **End-Effector Pose Driver (EEPD, Controlador de Pose del Extremo Efecto, ID JAUS 55).**
Este componente realiza control a lazo cerrado de la posición y orientación del extremo efector.
Como entrada posee la posición y orientación deseadas, las posiciones de las articulaciones, y las especificaciones del manipulador. La salida es el porcentaje de esfuerzo para cada articulación.
- **Joint Velocities Driver (JVD, Controlador de Velocidades de Articulación, ID JAUS 56).**
Este componente realiza el control a lazo cerrado de la velocidad de cada articulación.
Las entradas son la velocidad deseada de cada articulación, la velocidad instantánea de cada una y las especificaciones del manipulador. La salida es el porcentaje de esfuerzo de cada articulación.
- **End-Effector Velocity State Driver (EEVSD, Controlador de Estado de Velocidad del Extremo Efecto, ID JAUS 57).** Este componente realiza control a lazo cerrado de la velocidad del extremo efector.
Las entradas son el estado de velocidad deseado del extremo efector, las posiciones y velocidades actuales de las articulaciones, y la información de la especificación del manipulador. La salida es el porcentaje de esfuerzo de cada articulación.
- **Joint Move Driver (JMD, Controlador de Movimiento de Articulación, ID JAUS 58).** Este componente realiza control a lazo cerrado de la posición de cada articulación, especificadas en ciertos lapsos de tiempo. Adicionalmente los parámetros de movimiento para cada de las posiciones están restringidos. Por parámetros entiéndase: velocidad máxima, aceleración máxima, desaceleración máxima.
- **End-Effector Discrete Pose Driver (EEDPD, Controlador de Pose Discreta del Extremo Efecto, ID JAUS 59).** Este componente realiza control a lazo cerrado de las poses del

⁴Dependiendo del tipo de articulación, prismática o de revolución, se entregará información de fuerza o torque, respectivamente.

extremo efector, a través de una serie de posiciones y orientaciones especificadas en etapas discretas de tiempo.

Para más información respecto de estos componentes, dirigirse a [RA-3.3 P1, 2007]. En [SOSA, 2004] puede encontrarse una aplicación práctica de varios de los componentes manipuladores en un brazo robótico industrial.

4.4. JAUS y otros estándares

La siguiente tabla muestra las ventajas y desventajas de JAUS respecto de los demás estándares de interoperabilidad y control, según [Rowe, n.d.].

Estándar	Aceptación	Aplicabilidad general	Campo de uso
JAUS	✓✓	✓✓✓	Muy amplio (todo tipo de sistema no tripulado).
NATO STANAG 4586	✓✓✓	✓	Orientado a UAV's específicamente.
NATO STANAG 7085	✓✓✓	✓	Interoperabilidad relacionada a datos de imágenes.
MIL-STD 1760	✓✓✓	✓	Interoperabilidad de sistemas de armamento.
AAS F-41	✓✓✓	✓	Control de vehículos submarinos.

Tabla 4.1: Comparación entre algunos estándares de interoperabilidad.

El estándar con el que más se solapa JAUS, es el NATO STANAG 4586, que está específicamente enfocado hacia UAV's ⁵. Es ampliamente aceptado y posee una gran madurez, pero su aplicabilidad está restringida a un rubro muy específico de la robótica.

JAUS es el único estándar aplicable a diversos sistemas robóticos, sean submarinos, terrestres o aéreos; tripulados o no, con o sin manipuladores. Más aún, sus actualizaciones hacen que se ajuste cada vez mejor a las necesidades de los robots actuales. De la mano de SAE, JAUS garantiza un futuro con mucha atención por parte de nuevos desarrolladores, situación que es muy favorable para un estándar. Actualmente su aceptación se ve reflejada en la existencia de varias implementaciones libres (véase la Sección 4.5 Implementaciones y herramientas de JAUS).

Haciendo referencia al proyecto actual, son varios los elementos que se tienen en cuenta para hacer la evaluación de los estándares. En primer lugar, este trabajo está enfocado en los manipuladores. JAUS soporta esta rama de la robótica, y como punto adicional a favor, no está

⁵UAV's: Unmanned Aerial Vehicles, o vehículos aéreos no tripulados.

limitado a ella. Esto significa que si el usuario en el futuro se encontrase restringido por la herramienta, podría ser capaz de extenderla, y aún mantenerse en el estándar. Además existen implementaciones libres del mismo, y su documentación se encuentra disponible. JAUS también se muestra prometedor, y abierto a las sugerencias de sus usuarios a través del [JWG](#).

Estos han sido los motivos principales por los cuales se ha optado por llevar adelante el proyecto con el estándar JAUS.

4.5. Implementaciones y herramientas de JAUS

Para realizar un sistema JAUS ya no es necesario partir desde cero. Existe una variedad de implementaciones que facilitan este cometido, y que eliminan varios riesgos planteados en la Sección [2.5.2 Riesgos de Tecnología](#).

4.5.1. Alternativas

A continuación se enunciarán las alternativas de implementación de JAUS más importantes.

OpenJAUS

OpenJAUS consiste en una serie de herramientas que facilitan el desarrollo e implementación de sistemas JAUS compatibles, en la versión especificada por *JAUS Reference Architecture 3.3*. Es *Open Source* (licencia BSD). Incluye librerías implementadas en lenguaje C/C++, y ejemplos. La interfaz entre los componentes es JUDP⁶. Tiene como soporte varios tutoriales, y un foro que se encuentra actualmente en actividad.

Cada componente OpenJAUS posee:

- Una funcionalidad.
- Un conjunto definido de [servicios](#).
- Una máquina de estados.
- Una función por cada estado, que se ejecuta periódicamente.
- Actividades a realizar ante la llegada de cada mensaje entrante soportado.

A partir de estos elementos es fácil realizar un componente JAUS.

⁶JUDP es un protocolo sobre UDP desarrollado para las comunicaciones de JAUS.

Jaus++

Implementación *Open Source* (licencia BSD) en lenguaje C++ para múltiples plataformas (Windows, Linux).

No posee implementación de mensajes del subgrupo de manipulador.

De manera similar a lo ya presentado sobre OpenJaus, cada componente Jaus++ cuenta con:

- Una funcionalidad específica.
- Servicios y sus mensajes de entrada/salida.
- Funciones asociadas a cada tipo de mensaje de entrada.
- Estados y funciones asociadas a los mismos.

Re squared

RE^2 es una API en C/C++ para múltiples plataformas, incluyendo Linux, Windows, QNX, LynxOS, etc. que corre tanto sobre UDP/IP como sobre innovadores servicios de distribución de datos de tiempo real.

Es comercial.

Rep Invariant JAUS

El kit de desarrollo de software (SDK) *Rep Invariant JAUS* (RI-JAUS) es una librería (C++) que implementa JAUS para sistemas autónomos.

Hace hincapié en sistemas embebidos con recursos limitados, incluyendo aquellos con procesadores *ARM*.

Existe una versión *Open Source*, pero está disponible para ciudadanos estadounidenses, y que trabajen en EEUU. La documentación está disponible para todo el mundo.

4.5.2. Comparación

Aquí se presenta un cuadro comparativo de las diferentes alternativas. Luego se presenta una evaluación, para lo cual muchos de los riesgos planteados en la Sección [2.5.2 Riesgos de Tecnología](#) fueron tenidos en cuenta.

	OpenJAUS	Jaus++	RE ²	RI-JAUS
Costo	0 u\$s	0 u\$s	5000 u\$s (versión estudiantil)	4999 u\$s (licencia anual para desarrollo)
Licencia	BSD (<i>Open Source</i>).	BSD (<i>Open Source</i>).	Software privativo.	Dos licencias: <i>GNU Public License</i> y licencia de uso comercial.
Documentación	Poca. Tutoriales. Comentarios en código.	Muy buena.	No accesible sin pago previo.	Buena.
Soporte	Foro activo. E-mail.	Autores (e-mail).	E-mail y teléfono.	E-mail y teléfono.
Estado de desarrollo	Iniciado en el 2007. Versión 3.3.0a (tercera versión). Implementados todos los mensajes de la versión 3.3 de JAUS. Presenta un ejemplo funcionando asociado a un simulador de vehículo no tripulado. Implementa el subgrupo de manipulador.	Iniciado en el 2007. Versión 1.0 (primera). Soporta algunos mensajes de la versión 3.3 de JAUS. No posee implementación para mensajes del subgrupo de manipulador.	Versión 2.0.	Versión 0.9.0 Beta.
S.O.	Windows, Linux.	Windows, Linux.	Windows, Linux, QNX, LynxOS, Mac OS X.	Windows, Linux/POSIX, QNX, Mac OS X.
Lenguaje	C	C++	C++	C++
Interfaces	JUDP.	JUDP y JSerial.	JUDP y JSerial.	JUDP (JSerial próximamente).
Grupo de desarrollo	Tom Galluzzo, Danny Kent, y Bob Touchton. <i>Center for Intelligent Machines and Robotics (CIMAR). University of Florida (USA).</i>	Daniel Barber. <i>Applied Cognition and Training in Immersive Virtual Environments (ACTIVE) Lab. University of Central Florida (USA).</i>	RE ² Inc.	Rep Invariant Systems, Inc.
Enlace	www.openjaus.com	active-ist.sourceforge.net	www.resquared.com	www.repinvariant.com
Notas	-	-	Soporte para Sistemas Operativos <i>Real Time</i> .	Soporte para Sistemas Operativos <i>Real Time</i> . La versión libre está disponible sólo para ciudadanos de EEUU.

Tabla 4.2: Comparación entre algunas implementaciones de JAUS.

En primer lugar es necesario decir que el precio de RE² no está al alcance del proyecto, por lo cual no se lo tendrá en cuenta. La versión libre de RI-JAUS no está disponible para el grupo, pues para acceder a ella hay que ser ciudadano estadounidense (y con trabajo en EEUU). Estas dos implementaciones quedan entonces descartadas.

Uno de los riesgos planteados en la Sección 2.5.2 **Riesgos de Tecnología** estaba asociado al soporte de las SDK. Mientras Jaus++ posee una completa documentación y abundantes ejemplos, OpenJAUS se presenta con algunos tutoriales básicos y ningún documento fuerte que explique las reglas de programación adoptadas. Para entender a fondo a OpenJAUS es necesario introducirse en el código mismo. Sin embargo, se muestra estructurado, con nombres inconfundibles, tanto para funciones como variables, etc. OpenJAUS presenta además un foro activo, que es conducido por los propios autores.

Actualmente el lenguaje C es más utilizado que C++ (según [TIOBE, 2009] posee alrededor del 17% de aceptación, contra el 10% para C++). Sin embargo la posibilidad de trabajar con objetos es siempre una buena idea. Por otro lado, C permitiría que los nuevos desarrolladores se integren más fácilmente al proyecto, siempre y cuando la programación estructurada sea respetada y correctamente documentada.

Elección de OpenJAUS

Una investigación más profunda ha dejado al descubierto que Jaus++ no posee implementación de mensajes del subgrupo *Manipulator*. Esta implementación se considera **indispensable**.

Asimismo, el ritmo de crecimiento de OpenJAUS es un buen indicador. Su grupo ha lanzado 3 versiones en un tiempo muy corto. El foro se muestra en actividad. La comunidad aporta valiosa información, y tener un lugar donde encontrarla no es un punto menor.

Entre algunos de los planes que tiene el grupo de OpenJAUS, se encuentran[OJDC, 2009]:

- *Wireshark*⁷ para JAUS.
- Depuración remota.
- Interfaz gráfica: diagramas de secuencia, árbol del sistema, estadísticas en tiempo real.

En este proyecto se utilizará **OpenJAUS**. La implementación de los mensajes del subgrupo *Manipulator*, el foro activo, el crecimiento, y el lenguaje C, han sido los factores determinantes.

4.6. OpenJAUS

Como se ha enunciado antes, OpenJAUS consiste en un proyecto llevado adelante por Tom Galluzzo, Danny Kent, y Bob Touchton.

El fuente que se obtiene de la página[OpenJAUS, 2009] es válido tanto para Linux como para Windows (con formato de proyecto para *Microsoft Visual Studio 2005/2008*). En su versión 3.3.0b consta de 4 directorios principales:

- **libjaus**. Aquí se puede encontrar el código que da a origen a una librería, cuyo objetivo es brindar el formato de los mensajes JAUS, la definición de los tipos del estándar, y una implementación básica de los elementos de JAUS.

⁷Software analizador de protocolos de red. Posee soporte para TCP/IP, DHCP, etc. La información que brinda es útil para solucionar problemas de red.

- **libopenJaus.** Contiene el fuente de una librería que incluye aspectos concretos de la implementación, como ser la del componente, la del *Node Manager*, así como utilidades que permiten la comunicación entre componentes mediante UDP, etc.
- **ojNodeManager.** Implementación de los componentes *Node Manager* y *Communicator*. Puesto que el comportamiento de ambos es muy similar, una medida fue integrar a ambos en un mismo proyecto.
- **ojVehicleSim.** Entregado como ejemplo.

En la página también se puede acceder a tutoriales.

4.6.1. Aspectos de implementación

Según la implementación OpenJAUS, cada componente tiene los elementos que se muestran en la figura siguiente.

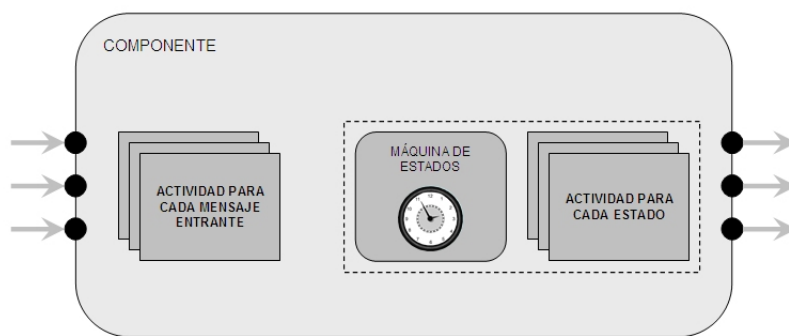


Figura 4.3: Componente JAUS según implementación OpenJAUS.

Existen dos disparadores de eventos que pueden provocar cambios internos: la máquina de estados (de ejecución periódica), y la llegada de un mensaje entrante.

La máquina de estados ejecuta periódicamente una de diferentes funciones, según el estado en el que se encuentre. Esto se puede observar en la Figura 4.3. El cambio de estado se puede llevar a cabo tanto en dichas funciones o en las funciones que atienden los mensajes entrantes, simplemente con una llamada.

La llegada de un mensaje también genera un evento. Para esto OpenJAUS utiliza una única función, que trata todos los mensajes entrantes.

Las funciones a ejecutar en cada caso (asociadas a la máquina de estados y al arribo de mensajes) no tienen un nombre específico, aunque sí deben respetar cierto formato de prototipo (en

términos de C). No poseer nombre fijo es lógico, puesto que es el desarrollador quien establece los estados posibles de la máquina de estados. Cuanto más estados se deseen, más funciones deberán ser definidas (y más nombres también). Cada función podrá tener el nombre que el programador desee. Como consecuencia, deben ser indicadas a la librería de OpenJAUS una por una. Esto se hace mediante el uso de una práctica denominada *callback*, donde una función es entregada como parámetro a otra función, para que esta última la ejecute posteriormente.

Inicialización de un componente

Poner en marcha un componente JAUS desde OpenJAUS requiere la ejecución de varias líneas de código. He aquí un ejemplo para un *Primitive Manipulator*.

```
/* Creación del componente. Se indica nombre, identificador, y
   frecuencia. */
cmpt = ojCmptCreate("PrimManip", JAUS_PRIMITIVE_MANIPULATOR, 1.0);
```

El código de arriba constituye el primer paso. El primer parámetro es el nombre de la instancia, es utilizado como una forma de identificación más clara para el usuario. El segundo parámetro es el *ID JAUS* del componente a instanciar. Es importante puesto que la dirección JAUS de la instancia de este componente depende de este dato. El último parámetro es la frecuencia (en Hz) con la cual se debe ejecutar la máquina de estados.

```
// Servicio añadido al componente. Se indica el tipo de componente.
ojCmptAddService(cmpt, JAUS_PRIMITIVE_MANIPULATOR);
```

Luego se indica a OpenJAUS qué [servicios](#) ha de brindar esta instancia.

```
/* Mensajes acordes al servicio añadido.
   Componente, servicio, mensaje y vector de presencia. */
ojCmptAddServiceOutputMessage(cmpt, JAUS_PRIMITIVE_MANIPULATOR, J..., 0xFF);
ojCmptAddServiceInputMessage(cmpt, JAUS_P..., JAUS_SET_JOINT_EFFORT, 0xFF);
```

De acuerdo al servicio añadido, se deben registrar los mensajes de entrada y salida

correspondientes.

```
/* Establecimiento de la función para el tratamiento
   de mensajes entrantes. */
ojCmptSetMessageProcessorCallback(cmpt, pmProcessMessage);

// Funciones para ejecución por parte de la maquina de estados.
ojCmptSetStateCallback(cmpt, JAUS_STANDBY_STATE, pmStandbyState);

// Inicialización de la maquina de estados (en estado StandBy).
ojCmptSetState(cmpt, JAUS_STANDBY_STATE);

// Ejecución de la maquina de estados del componente.
ojCmptRun(cmpt);
```

Es establecida la función de procesamiento de mensajes entrantes, y también las funciones asociadas a la máquina de estados (para el caso sólo una, el estado *standby*).

La ejecución de la última línea pone en marcha la instancia. A partir de ese momento recibirá mensajes y ejecutará su máquina de estados periódicamente. La instancia está implementada como un proceso liviano, más específicamente como un *POSIX Thread* (o *pthread*).

Estructura de usuario

El componente de OpenJAUS brinda una herramienta útil. En la misma estructura del componente (cuyo puntero es citado en los ejemplos como *cmpt*) el programador puede ligar una estructura específica. Esta nueva estructura podrá contener los datos específicos de la implementación del usuario. Variables de estado, datos históricos, los últimos mensajes recibidos y otros datos son buenos candidatos.

Para usar esta estructura de datos basta con hacer lo siguiente.

```
// Definición de la estructura PdData.

PdData* data;
data = (PdData*)malloc(sizeof(PdData)); /* Creación de una estructura de
                                         datos del componente (definida
                                         arriba). */
ojCmptSetUserData(cmpt, (void *)data); /* Asocia la estructura del usuario
                                         con el componente (el componente
                                         tiene un puntero para estos
                                         fines). */

// ...
data = (PdData*)ojCmptGetUserData(pd); // Uso de la estructura.
```

Envío de mensajes

OpenJAUS utiliza dos formatos de mensajes: uno específico y otro genérico (también llamado *JausMessage*). Basta con saber que siempre que un mensaje esté a punto de ser enviado, o haya sido recientemente recibido, tendrá un formato genérico.

El mensaje genérico es un mensaje del cual no se puede extraer más que los datos de la cabecera. OpenJAUS no ha procesado este mensaje aún, y por ello no sabe cómo interpretar su cuerpo. Leyendo el campo *commandCode*, el programador puede identificar el mensaje y proceder a tratarlo para su uso extendido.

Si se quisiera enviar por ejemplo el mensaje *Set Wrench Effort* desde un *Primitive Driver* se podría hacer lo siguiente.

```

SetWrenchEffortMessage swe; // Mensaje específico.
JausMessage jausMessage;   // Mensaje genérico.
JausAddress destino;        // Dirección del destino.

PdData* data; // Puntero a la estructura del componente.
data = (PdData*)ojCmptGetUserData(pd); // Obtención de la estructura.

swe = data->setWrenchEffort; /* Se obtiene un mensaje (de la
                             estructura) pre-inicializado. */

/* Establecimiento de los campos del mensaje
   específico (en orden de aparición). */

// Campos de la cabecera.
destino = jausAddressCreate(); // Destino.
destino->subsystem = 1;
destino->node = 1;
destino->component = JAUS_PRIMITIVE_DRIVER;
destino->instance = 1;
jausAddressCopy(swe->destination, destino);
jausAddressDestroy(destino);

// Campos del cuerpo.
swe->presenceVector = 0xFF;
dato = datoAEnviar ;
swe->propulsiveLinearEffortXPercent = dato;

// Conversión del mensaje específico a uno genérico.
jausMessage = setWrenchEffortMessageToJausMessage(swe);
ojCmptSendMessage(pd, jausMessage); // Envío del mensaje.

// Liberación de auxiliares.
jausMessageDestroy(jausMessage);

```

Mensajes entrantes

Los mensajes entrantes son procesados por una única función. Generalmente se guarda una copia del último mensaje de cada tipo (el más actualizado). Dicha copia suele ser referenciada por un puntero. Este se encuentra en la estructura de datos del usuario, citada anteriormente.

```

void pdProcessMessage(OjCmpt pd, JausMessage message)
{
    // Declaración de referencias a subestructuras del componente.
    // ...
    SetWrenchEffortMessage setWrenchEffort; // Mensaje específico.

    // Declaracion de auxiliares.
    JausAddress address;
    JausMessage txMessage;
    PdData *data;

    data = (PdData*)ojCmptGetUserData(pd); // Estructura de usuario.

    // Identificación y procesamiento del mensaje recibido.
    switch(message->commandCode)
    {
        // ...
        case JAUS_SET_WRENCH_EFFORT: // En caso de Set Wrench Effort...
            setWrenchEffort = setWrenchEffortMessageFromJausMessage(message);
            if(setWrenchEffort)
            { // Destruye el mensaje viejo, y lo reemplaza por el actualizado.
                setWrenchEffortMessageDestroy(data->setWrenchEffort);
                data->setWrenchEffort = setWrenchEffort;
            }
            break;
        // ...

        default: // En caso de ser un mensaje que se desee atender...
            ojCmptDefaultMessageProcessor(pd, message);
            break;
    }
}

```

4.6.2. OpenJAUS como plataforma

Los kit de desarrollo JAUS (tratados en la Sección [4.5 Implementaciones y herramientas de JAUS](#)) son medios a través de los cuales un emprendimiento se acerca al estándar, con un mínimo esfuerzo.

Generalmente esto ocurre, aunque pueden existir una serie de elementos que dificultan su uso. A continuación se presentará una serie de ventajas y desventajas que presenta OpenJAUS al perseguir los objetivos del proyecto.

Ventajas de OpenJAUS

Herramientas como OpenJAUS tienen éxito, puesto que la implementación de JAUS no es una tarea sencilla. Cuando se afirma esto, no se hace referencia a la complejidad de los algoritmos, sino principalmente a la cantidad abrumadora de información que tiene que ser puesta en cuenta al intentar, por ejemplo, enviar un mensaje JAUS a un componente ubicado en otro subsistema. Implementar JAUS en sus niveles más bajos requiere principalmente mucho estudio, tiempo y madurez de conocimientos sobre la norma.

Otra buena razón para utilizar estas herramientas, tiene que ver con las actualizaciones de JAUS. El equipo de trabajo de JAUS está buscando permanentemente alternativas que pudieran ajustarse mejor a las necesidades de las implementaciones. Esas alternativas son tratadas en el comité JAUS, y en base a la profundidad de cada cambio, y a la cantidad de los mismos, se presentan actualizaciones. Herramientas como OpenJAUS siempre hacen el intento por mantener la interfaz de usuario intacta, a menos que sea realmente necesario un cambio. Esto amortigua los efectos de las actualizaciones en el software desarrollado.

Existe otra importante ventaja que se desprende del uso de OpenJAUS. Radica en la herencia de las pruebas sobre la herramienta. El proyecto OpenJAUS ha sido utilizado numerosamente en competencias como *DARPA Urban Challenge*⁸, mostrando la fiabilidad de la cual es merecedor. Por otro lado, las pruebas realizadas por el CIMAR entre componentes OpenJAUS y sistemas JAUS garantizan que, haciendo uso de OpenJAUS, el proyecto actual será perfectamente reconocido en dichos marcos.

Desventajas de OpenJAUS

Migrar a JAUS con [niveles de compatibilidad](#) II o III (según [\[Rowe, n.d.\]](#) y [\[Faruque, 2006\]](#)) usualmente requiere un replanteo de la arquitectura de software del robot. Esto es una consecuencia del estándar. Si se habla de un desarrollo basado en herramientas de alto nivel, es necesaria además la transcripción hacia C ó C++: el trabajo se torna extenso y contraproducente.

Por otro lado, el conjunto de llamadas de OpenJAUS posee tanta flexibilidad, que es posible transgredir al mismo JAUS. Un ejemplo de esto ocurre en el siguiente caso: el componente X se registra ante el *Node Manager* como tipo X, pero es capaz de enviar mensajes de un componente Y. Es responsabilidad del desarrollador cumplir con el estándar.

⁸Defense Advanced Research Projects Agency (DARPA).

4.6.3. Manipulador con OpenJAUS

Si bien la mayoría de los robots que utilizan JAUS son vehículos, existen excepciones. En [SOSA, 2004], un robot industrial *PUMA 762* de 6 DOF ha sido adaptado al estándar haciendo uso de OpenJAUS.



Desarrollo

5

Interfaz JAUS para HDR

El proceso de desarrollo de software en robótica involucra el trabajo de diversos grupos. Cada uno está enfocado en distintas funcionalidades, aquellas que requiere el sistema final para lograr el comportamiento esperado de un robot. Tal como se presentó en la Sección [3.2.1 Herramientas de desarrollo de un componente](#), existe un amplio conjunto de herramientas de alto nivel destinadas a facilitar el diseño e implementación de prototipos. Estos son útiles puesto que constituyen una prueba de concepto que brinda experiencia al desarrollador, y genera realimentación de información. El presente capítulo, presenta las características fundamentales de las herramientas MATLAB, Simulink y LabVIEW. No se debe perder de vista que la interoperabilidad es una característica del diseño de software basado en componentes, tal y como es propuesto por JAUS. Por último, se especifica por qué es importante el diseño e implementación de una Interfaz JAUS.

5.1. Herramientas de desarrollo de robots

5.1.1. LabVIEW

LabVIEW es el acrónimo de *Laboratory Virtual Instrument Engineering Workbench*. Consiste tanto en un lenguaje de programación gráfico, como un entorno especialmente diseñado para este paradigma. Nació orientado a aplicaciones de control de instrumentos electrónicos, orientación por la cual recibió su nombre.

Su empresa propietaria es *National Instruments* (NI). LabVIEW es el medio mediante el cual NI brinda interesantes puntos de acceso a su hardware y sus servicios. Su mercado se centra en la adquisición de datos, control de instrumentos e instrumentación virtual.

Pensando en la robótica, sus propuestas se muestran cautivantes, principalmente por el gran

soporte para el manejo de hardware NI. Esto contribuye a una mejor abstracción por parte de los desarrolladores.

Por otro lado, la robótica se presenta como un campo indiscutiblemente interdisciplinario: coexisten dentro de cada robot conceptos de mecánica, electrónica, y de software, entre otros. Es común así encontrarse con especialistas que poco entienden de programación. Principalmente por ello LabVIEW es una buena opción. Su aprendizaje es más sencillo que el de los lenguajes de programación convencionales. Su paradigma es más intuitivo: para utilizar un instrumento basta con añadir su bloque en el entorno gráfico, e interconectarlo con el resto de los bloques. Existen osciloscopios virtuales, generadores, multiplicadores, etc.

Actualmente la herramienta cuenta con soporte para una amplia gama de equipos de hardware, como son PDA's, FPGA's, DSP's y embebidos en general.

Para mencionar, según [TIOBE, 2009], LabVIEW tienen una popularidad del 0.17% de los programadores. Su posición en el *ranking* es la 39, uno por debajo del conocido lenguaje *Smalltalk*. Si bien este porcentaje puede parecer bajo, es necesario tener en cuenta que son considerados también lenguajes como C, Perl, PHP, Python y Java, los cuales no poseen la orientación que persigue este trabajo.

5.1.2. MATLAB

MATLAB es el acrónimo de *MATrix LABoratory*, consiste en una herramienta matemática ideal para el común de las simulaciones. Su empresa propietaria se llama *MathWorks*.

Provee gran cantidad de funcionalidades distribuidas en *toolbox*, dedicadas a cálculos en áreas como ser bioinformática, adquisición de datos, computación en paralelo, etc.

Por si fuera poco, posee un lenguaje de programación propio denominado *M*. Esto permite a los usuarios implementar algoritmos a medida, en caso de que la gran variedad de funcionalidades provistas no sea suficiente.

Sin lugar a dudas, la robótica implica muchos conceptos asociados al control. Este área basa sus desarrollos en modelos representativos de la realidad, traducidos en ecuaciones matemáticas. Es por esto que en fases tempranas de diseño y estudio, los grupos de robótica vuelcan sus proyectos en simulaciones en MATLAB, más precisamente en una herramienta llamada Simulink, descripta luego.

Según [TIOBE, 2009] MATLAB tiene una popularidad del 0.56%¹ de la comunidad. Ocupa la posición número 20.

¹"The TIOBE Programming Community index gives an indication of the popularity of programming languages".

5.1.3. Simulink

Simulink es una herramienta para simulaciones de múltiples áreas. Consiste en un entorno gráfico interactivo, y en un conjunto de bloques que le permiten al usuario diseñar, simular, implementar y probar una amplia variedad de sistemas dinámicos. Esto incluye comunicaciones, control, procesamiento de señales, video e imágenes².

Gracias a estas características, Simulink es la elección de muchos equipos de trabajo dedicados al control.

Si bien Simulink viene provisto con bloques específicos para cada área, pueden ser creados nuevos bloques a partir de funciones en código **M** (de MATLAB). Esto permite un grado de reutilización importante.

5.2. JAUS y Herramientas de Desarrollo en Robótica

Las HDR previamente detalladas, no están preparadas para la implementación de componentes que puedan comunicarse entre sí mediante el uso de un estándar independiente de la plataforma. En otras palabras: hoy por hoy, para desarrollar un sistema con bloques implementados en LabVIEW y con bloques en MATLAB, es necesario utilizar protocolos específicos entre ambas herramientas. Este trabajo pretende extender las posibilidades de implementación de un componente a estas HDR, y no restringirlas a las mismas.

Claro está que obligar a un grupo de trabajo a decidirse por una u otra herramienta puede ser contraproducente, ya sea porque participan profesionales con diferentes formaciones y conocimientos, o porque las ventajas que pueden encontrarse en LabVIEW tal vez no estén presentes en MATLAB y viceversa.

Por otra parte el desarrollo basado en componentes propuesto por JAUS presenta la mejor alternativa para grupos de trabajo que desarrollan diferentes piezas de software simultáneamente. Estas ventajas, no están disponibles en las herramientas de desarrollo antes presentadas. Por tal motivo, un grupo de robótica deberá optar por una u otra opción:

- Abandonar las herramientas como Simulink, MATLAB o LabVIEW, y migrar a la programación en C para hacer uso de OpenJAUS;
- Ó implementar JAUS en los lenguajes ofrecidos por las herramientas de desarrollo (M o gráfico).

Considerando que, tanto desde MATLAB como desde LabVIEW se puede invocar código

²Obtenido de <http://www.mathworks.com/products/simulink/>.

implementado en C, es fructífero implementar una librería en dicho lenguaje, que actúe como interfaz entre MATLAB/Simulink/LabVIEW y OpenJAUS. Así, se permite a un sistema implementado bajo alguna HDR comportarse como un componente JAUS, pero sin migrar.

La Figura 5.1 presenta la idea.

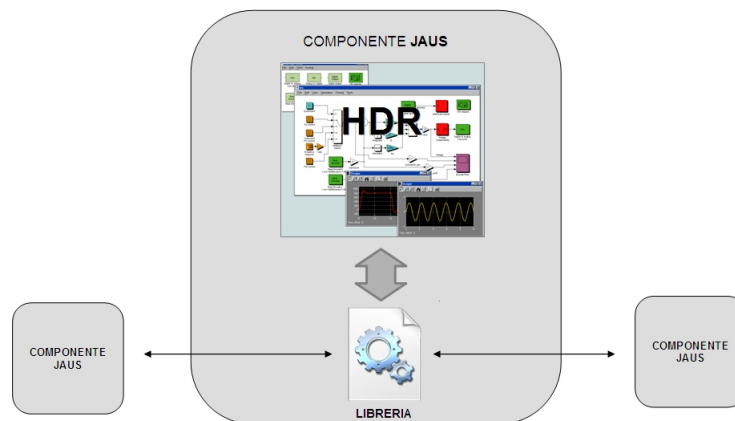


Figura 5.1: Componente JAUS implementado en HDR.

Esta solución permite mantener el entorno de trabajo utilizado por un grupo de robótica, e introducir lentamente los conceptos propios de JAUS y las ventajas inherentes de la programación por componentes. A medida que un proyecto se muestra viable, la implementación puede posteriormente llevarse a cabo con otros lenguajes en busca de eficiencia (ver Figura 5.2).

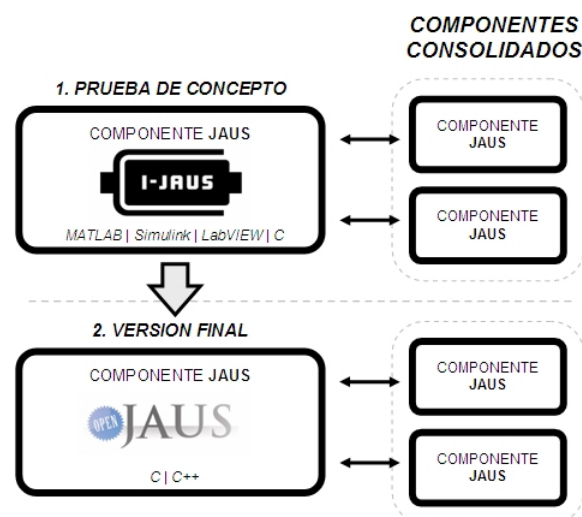


Figura 5.2: Ejemplo de evolución de la implementación de un componente.

En el Capítulo 6 se presentarán puntos importantes del uso y la implementación de la librería de Interfaz JAUS, denominada I-JAUS.

6

Diseño y Uso de I-JAUS

I-JAUS es la implementación fundamental de este trabajo. Constituye la herramienta que da solución a las necesidades ya planteadas del grupo *cliente*.

En lo que sigue del documento se presentarán cuestiones básicas relacionadas al uso de I-JAUS, así como aquellos elementos del diseño que sean necesarios para un entendimiento más profundo del funcionamiento de la librería.

6.1. Documentación

I-JAUS dispone de documentación técnica dedicada a sus usuarios, acorde a lo planteado en la Sección [2.3 Replanteo de requerimientos](#). Estos documentos son el *Manual de Referencia de I-JAUS* (disponible en versión *HTML* y *PDF*, consultar Apéndice [A Manual de Referencia de I-JAUS](#)) y el *Manual de referencia I-JAUS para MATLAB* (en versión *HTML*).

En dichos documentos el usuario podrá encontrarse con descripciones y explicaciones del uso de I-JAUS. También pueden encontrarse ejemplos en el Apéndice [B Ejemplos de I-JAUS](#), acorde a los requerimientos planteados en la Sección [2.3 Replanteo de requerimientos](#).

6.2. Estilo de codificación

6.2.1. Fuentes en C

La bibliografía [[Cannon, 1997](#)] brinda un conjunto de reglas que son ampliamente aceptadas para la codificación en el lenguaje C. Estas, combinadas con reglas que permiten documentación en código (mediante *Doxygen*, al estilo *Javadoc*), hacen que I-JAUS posea fuentes ordenadas, y

con abundantes comentarios. Todo esto permite a cualquier programador continuar con el hilo de desarrollo del proyecto, incluso generando la documentación que mejor se ajuste a sus necesidades a partir de los fuentes.

6.2.2. Fuentes en MATLAB

I-JAUS posee documentación específica en el propio código de MATLAB.

Gracias al proyecto *M2HTML*, se permite al usuario acceder a la documentación de cada función de MATLAB de dos maneras: tanto desde consola (con el comando *help*), como desde el HTML generado con la herramienta.

6.3. Estructura de usuario

Como se ha presentado en la Sección [4.6.1 Estructura de usuario](#), OpenJAUS brinda al usuario facilidades para que mantenga su propio conjunto de datos, donde guardará toda la información que se quiera asociada al componente. La Interfaz utiliza esta misma estructura, pero de una forma diferente.

Cuando el usuario de OpenJAUS implementa un componente, establece de manera estática su estructura de datos. Esto ocurre porque conoce los mensajes y variables que su componente ha de usar durante la ejecución. Sin embargo, la librería desconoce los datos que el componente ha de usar, sino hasta el momento de ejecución. Por ejemplo, se puede elegir para el componente tanto un servicio [JPS](#) como un [JPD](#), o ambos. Esta elección ocurre en tiempo de ejecución, e implica el uso de estructuras de datos completamente diferentes. Para hacer frente a este problema, se ha agregado un grado de indirección a la estructura de datos planteada para OpenJAUS. En la Figura [6.1](#) se puede observar el agregado: la Estructura genérica. El usuario de OpenJAUS partiría habitualmente de la Estructura específica.

Un componente requiere simultáneamente varios [servicios](#). En principio, siempre cuenta con el servicio *Core* y adicionalmente puede incorporar otros servicios, de acuerdo a la necesidad del usuario.

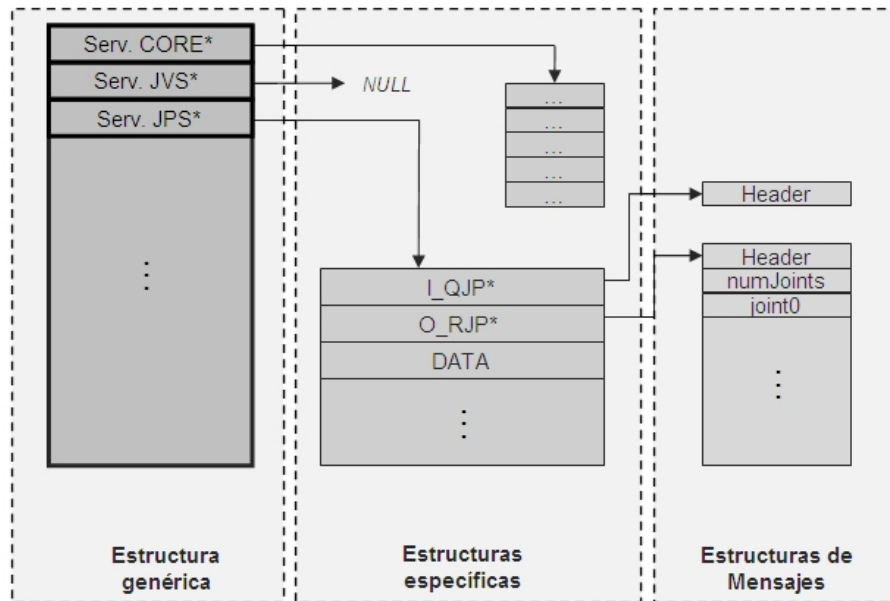


Figura 6.1: Disposición de los datos en estructuras ante el servicio JPS.

Como se puede ver en la Figura 6.1, I-JAUS guarda en la Estructura de usuario de OpenJAUS referencias a las estructuras específicas de cada servicio. Este primer nivel del conjunto de datos ha sido denominado *Estructura genérica*. Dado que durante la inicialización, un servicio puede ser añadido o no, estas referencias pueden variar, y cambiar su valor nulo para hacer referencia a una estructura específica existente.

En el segundo nivel, se encuentran las denominadas *Estructuras específicas* (habituales en OpenJAUS). Estas contienen información relacionada a un servicio en particular. En caso de estar activo un servicio, su estructura específica correspondiente es creada, y será usada por el componente. En la figura se muestra la estructura de un servicio particular, el servicio *Joint Positions Sensor (JPS)*. Este servicio soporta mensajes *Query Joint Positions* (entrante) y *Report Joint Positions* (saliente). De cada mensaje (saliente o entrante) se tiene una copia. Algunos datos particulares asociados al servicio también son guardados en esta estructura.

En el tercer y último nivel, existen las denominadas *Estructuras de mensajes*. Estas provienen de la librería OpenJAUS, y contienen todos los campos necesarios para leer un mensaje entrante, o para actualizar uno saliente.

6.4. Arquitectura de Componentes

El estándar JAUS es quien define la arquitectura de componentes así como los mecanismos de comunicación entre los mismos. Por tal motivo, en este trabajo se respeta dicha definición y no se plantean requerimientos a los componentes.

Como se mencionó en la Sección [5.2 JAUS y Herramientas de Desarrollo en Robótica](#), la Interfaz permite el desarrollo de un componente JAUS pero sin migrar de una HDR particular. La arquitectura básica de dicho componente se presenta en la Figura [6.2](#).

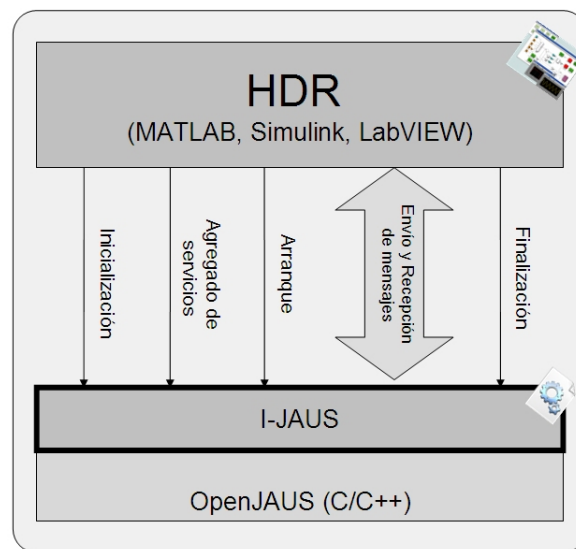


Figura 6.2: Llamadas primitivas de I-JAUS.

Internamente, I-JAUS funciona como mediador entre las solicitudes de la HDR y el componente OpenJAUS. En la Figura [6.3](#) se puede observar la arquitectura básica de I-JAUS.

Existen algunas llamadas que utilizan de forma casi directa funcionalidades de OpenJAUS. Sin embargo no es el común de los casos. Las llamadas *yUpdateXMessageHeader* e *yUpdateXMessage* modifican en la *Estructura de Usuario* (o *Principal*) mensajes asociados a cierto servicio, para luego poder enviarlos a cierto destino mediante llamadas *ySendXMessage* e *ySendXMessageTo*.

La llegada de un mensaje al componente provoca cambios no sólo en la *Estructura de Usuario*, sino también en la *cola de preservación de orden*, que será explicada luego.

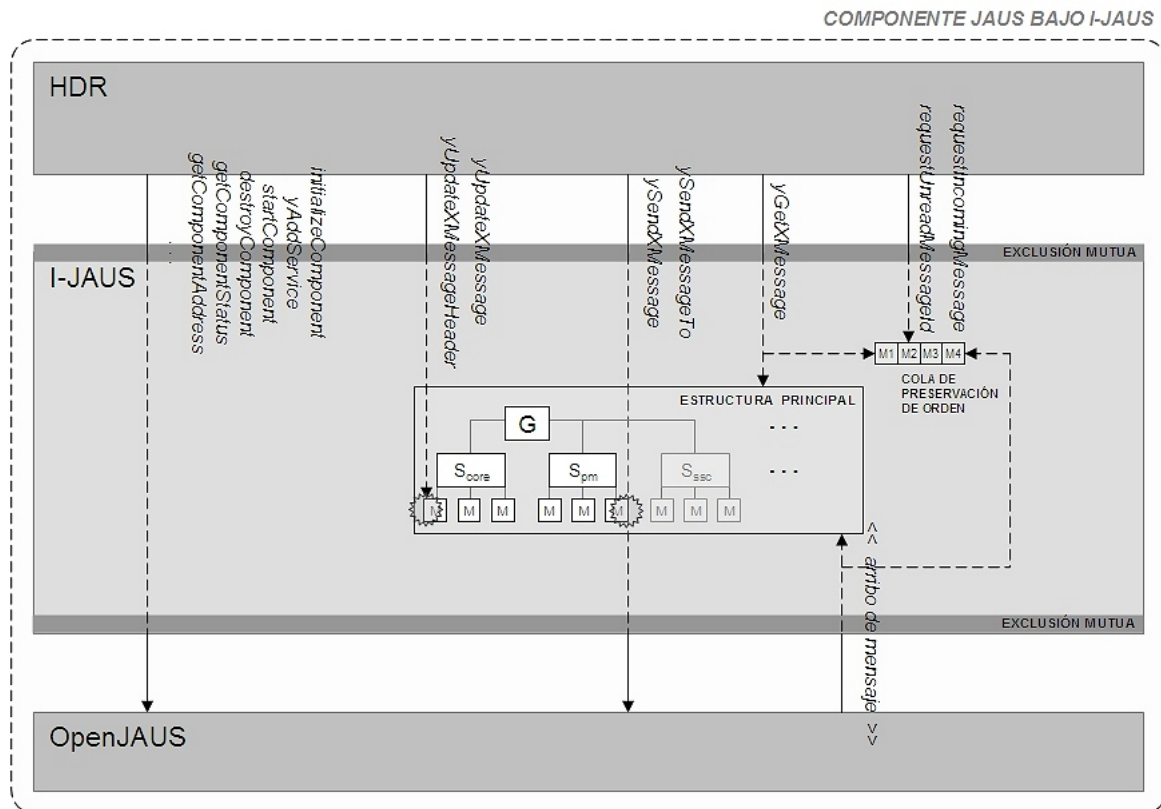


Figura 6.3: Arquitectura de I-JAUS.

Es de especial importancia el mecanismo de *exclusión mutua* utilizado para garantizar la integridad de los datos en la *Estructura de Usuario*. Esto se detalla a continuación.

6.5. Integridad de Datos

Las siguientes secciones describen el tratamiento que OpenJAUS hace sobre la estructura de datos del usuario (Sección 4.6.1 *Estructura de usuario*), y cómo es que al hacer la adaptación con la HDR es posible la pérdida de integridad de datos.

6.5.1. OpenJAUS

La librería de OpenJAUS está implementada de forma tal que, al poner en marcha un componente, es iniciado un hilo diferente de aquel que hizo la instanciación. Esto permite separar la ejecución del componente, de la ejecución del código que lo crea, lo inicializa, lo destruye, etc.

Aquellos espacios que el usuario utiliza para salvar los mensajes salientes, los mensajes entrantes, y demás información, son leídos y/o escritos **sólo** durante la ejecución del código de usuario. En OpenJAUS, el código del usuario puede ser ejecutado ante dos posibles eventos: ante la llegada de un mensaje (función *messageProcessor*), y ante la ejecución de la máquina de estados (función correspondiente al estado). Ambos eventos son atendidos por el mismo hilo del componente. Esto puede verse en la Figura 6.4.

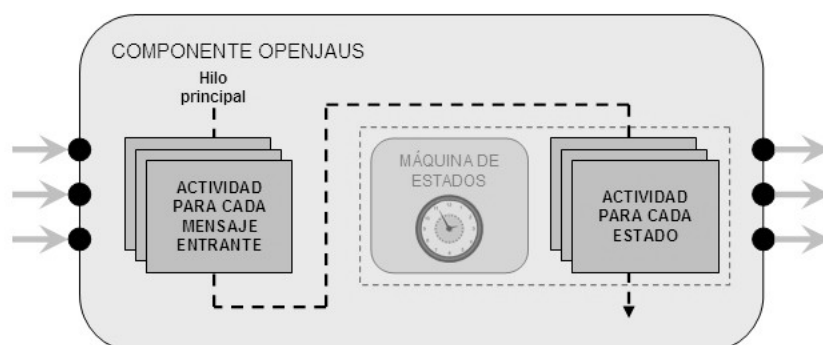


Figura 6.4: Hilo de ejecución en un componente OpenJAUS.

Dado que existe un único hilo que accede a los datos del usuario, los problemas de concurrencia relacionados a su integridad están resueltos. En cualquier instante, el hilo está destinado a leer o está destinado a escribir estos espacios.

6.5.2. OpenJAUS y HDR

Hacer uso de OpenJAUS desde una HDR es equivalente a incluir un segundo hilo al proceso.

La ejecución de un componente con dos hilos tiene varias implicancias positivas, principalmente al hablar de la atención de los eventos asincrónicos (como la llegada de mensajes). Algunas de ellas son:

- La librería puede recibir mensajes gracias a OpenJAUS, mientras el proyecto se encuentra ejecutando código de la HDR, sin depender del momento en el que la HDR realiza llamadas y cede el control a la librería¹.
- La llegada de mensajes JAUS provoca en la librería una serie de cambios. Muchos de estos mensajes son especialmente procesados por OpenJAUS, y se muestran al usuario final con un

¹De otra manera, la recepción de mensajes se realizaría únicamente en los momentos en los que el control es entregado a la librería, es decir en aquellos momentos en que se realizan llamadas.

nivel de complejidad inferior. Todo este procesamiento no se realiza durante cada llamada, sino que el hilo de OpenJAUS lo realiza asincrónicamente, ni bien recibe el mensaje. Esto hace que las llamadas a la librería sean ágiles.

- La librería puede manejar internamente llamadas bloqueantes, sin que esto bloquee a la HDR. Asimismo, la HDR podría bloquearse sin que esto bloquee al hilo de OpenJAUS (y no se reciban mensajes por ello).
- De acuerdo al estándar JAUS, un componente se mantiene registrado contra el *Node Manager* gracias al envío de un mensaje periódico denominado *heartbeat*. No disponer de un hilo adicional para el componente cargaría al usuario de responsabilidades como esta.

Sin embargo, añadir un nuevo hilo requiere cuidados especiales: existe la posibilidad de que un dato sea modificado y leído *simultáneamente* (Figura 6.5, caso a). Esto crea condiciones propensas para la pérdida de integridad de datos. Dicho problema es resuelto mediante un simple mecanismo de exclusión mutua (Figura 6.5, caso b).

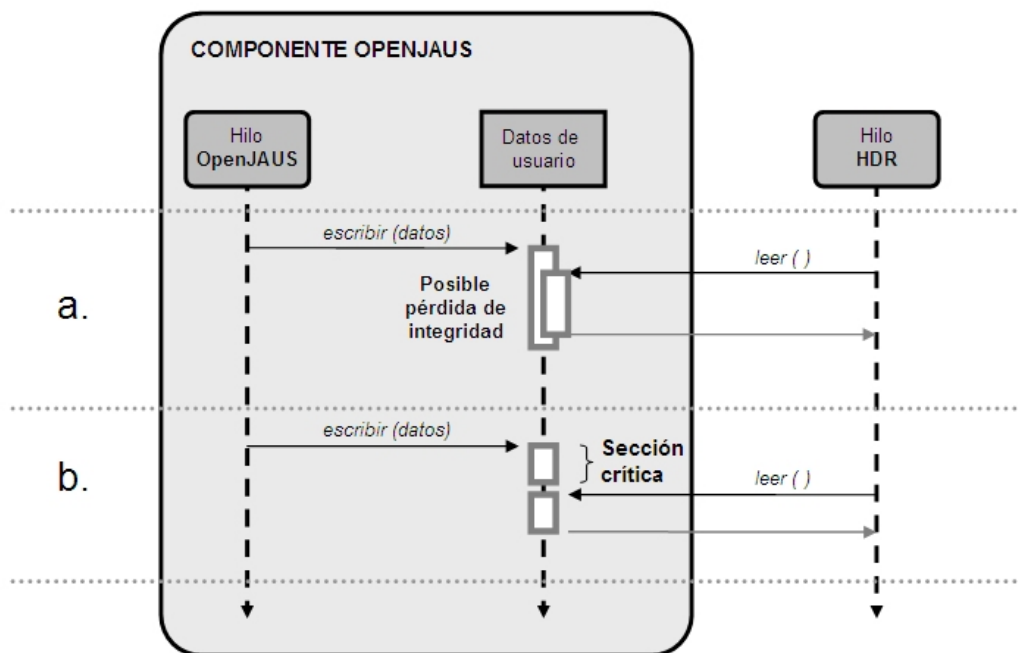


Figura 6.5: Productor/Consumidor como consecuencia de un segundo hilo de ejecución.

6.6. Llamadas a la librería

La librería I-JAUS es una *librería de enlace dinámico* (o *DLL* por sus siglas en inglés). Permite satisfacer los requerimientos de funcionamiento de un componente (Sección [2.3 Replanteo de requerimientos](#)). Posee la implementación de un conjunto de funciones que permiten hacer del proyecto del usuario (en la HDR) un componente JAUS. Para lograr esto es necesario comunicarse con la misma, e inicializar un componente a partir de sus llamadas.

El conjunto de funciones disponibles puede ser encontrado tanto en la documentación de I-JAUS como en los archivos .h de sus fuentes. Allí, cada llamada en particular se encuentra precedida por la palabra clave *EXPORTAR*, caso contrario se trataría de una función interna, que no debe importar al usuario común (pues no es visible desde la HDR).

A continuación, y a modo de ejemplo, se muestra un fragmento del archivo *generic.h* (ver Apéndice [A Manual de Referencia de I-JAUS](#)), el cual contiene algunas llamadas comunes a todos los componentes, indistintamente de los [servicios](#) que presten.

```
...
/* Inicializa componente genérico. Primera llamada a usar. */
EXPORTAR IJInt initializeComponent(IJChar name[], IJUChar id);
/* Arranca hilo del componente genérico. Ultima llamada de inicialización. */
EXPORTAR IJInt startComponent(void);
/* Finaliza ejecución del hilo del comp. genérico, y libera sus estructuras.*/
EXPORTAR IJInt destroyComponent(void);
/* Solicita ID de algún mensaje no leído. */
EXPORTAR IJInt requestUnreadMessageId(void);
...
```

Los tipos de datos que son retornados, fueron definidos especialmente por I-JAUS. Su definición puede encontrarse en la cabecera *ijtypes.h* (ver Apéndice [A Manual de Referencia de I-JAUS](#)). Los tipos más usados son mostrados en la Tabla [6.1](#).

Tipo	Equivalente en C	Descripción
IJChar	char	Byte (caracter con signo). 8 bits.
IJUChar	unsigned char	Byte (caracter sin signo). 8 bits.
IJShort	short	Entero (con signo). 16 bits.
IJUShort	unsigned short	Entero (sin signo). 16 bits.
IJInt	int	Entero (con signo). 32 bits.
IJUInt	unsigned int	Entero (sin signo). 32 bits.
IJDouble	double	Punto flotante (precisión doble). 64 bits.

Tabla 6.1: Tipos de uso más frecuente en I-JAUS.

Como puede verse en el fragmento de *generic.h*, son también utilizados punteros a estos tipos primitivos.

En lo que sigue de esta Sección 6.6, se profundizará en las características de I-JAUS, y cómo debe utilizarse.

6.6.1. Nombre de las llamadas

Se tienen algunas reglas que permiten darse cuenta qué nombre tendrá cierta función de I-JAUS.

Las letras **y** y **X** serán utilizadas a lo largo del texto para denotar respectivamente al nombre del servicio (con iniciales en minúscula), y al nombre del mensaje (nombre completo, capitalizado).

Existen llamadas para los diversos servicios. Algunas de estas persiguen un objetivo similar: obtener cierto mensaje entrante de ese servicio, escribir cierto mensaje saliente, enviarlo, etc. Todas estas llamadas presentan un formato similar. En la Tabla 6.2 se muestran esos grupos de llamadas, y sus formatos.

Llamada/Formato	Descripción	Ejemplo
<i>yAddService(...)</i>	Agregado del servicio y al componente.	<i>jpdAddService(...)</i>
<i>yGetXMessage(...)</i>	Obtención del mensaje X , relacionado al servicio y .	<i>jpdGetSetJointPositionsMessage(...)</i>
<i>yUpdateHeaderXMessage(...)</i>	Actualización de los campos de la cabecera del mensaje X (servicio y).	<i>jpdUpdateHeaderSetJointEffortsMessage(...)</i>
<i>yUpdateXMessage(...)</i>	Actualización de los campos del cuerpo del mensaje X , (servicio y).	<i>jpdUpdateSetJointEffortsMessage(...)</i>
<i>ySendXMessage()</i>	Envío del mensaje X , (servicio y). El destino será el presente en la cabecera del mismo mensaje.	<i>jpdSendSetJointEffortsMessage()</i>
<i>ySendXMessageTo(...)</i>	Envío del mensaje X , (servicio y). El destino será el entregado como parámetro a la llamada.	<i>jpdSendSetJointEffortsMessageTo(...)</i>

Tabla 6.2: *Formato de algunas llamadas I-JAUS.*

Sin embargo, existe un conjunto de llamadas que comunes a todos los componentes (independientemente de sus servicios). Naturalmente, los nombres de las mismas no presentan iniciales que identifiquen a un servicio en particular. En lo que sigue de la sección serán tratadas dichas llamadas.

6.6.2. Puesta en funcionamiento de un componente

Para que un componente pueda comunicarse con el resto del sistema JAUS, es necesario ponerlo en marcha. Esto implica 3 etapas, descritas a continuación (ver Figura 6.2).

Inicialización de un componente

La inicialización representa el primer paso para poner en funcionamiento un componente desde una HDR mediante I-JAUS. La función puesta a disposición se denomina *initializeComponent()* (ver Apéndice A Manual de Referencia de I-JAUS). Cuenta con dos atributos que permiten indicar el nombre y el *ID* que recibe el componente.

Iniciar un componente es indispensable puesto que lo prepara para comportarse tal como lo especifica JAUS. En particular, se inicializa la *estructura genérica* definida por I-JAUS y luego se la asocia a la *estructura de usuario* de OpenJAUS.

En el Apéndice B Ejemplos de I-JAUS se tienen ejemplos de esta y otras llamadas.

Agregado de servicios

Este es el paso posterior a la inicialización. Permite que un componente pueda alojar el último ejemplar de cada mensaje de entrada y el último ejemplar de cada mensaje de salida, del servicio agregado. Para lograrlo, la Interfaz debe inicializar la *estructura específica* del servicio en cuestión y luego incorporarla a la *estructura genérica* del componente. La Figura 6.1 muestra el resultado en la estructura al agregar el servicio *Joint Position Sensor*.

Algunas cuestiones merecen ser resaltadas aquí. En primer lugar, la *estructura genérica* desarrollada por I-JAUS está preparada para contener la *estructura específica* de todos los servicios *Manipulator*. I-JAUS realiza el mismo procedimiento (antes descrito) para cada uno de los servicios agregados en esta etapa. En segundo lugar, JAUS posee un *ID* para cada uno de los componentes/servicios que define. El estándar indica que cada componente puede poseer únicamente dos servicios: el servicio *Core*, y aquel servicio asociado al *ID* del componente. Esto implica que no se debería agregar un servicio cuyo *ID* (ver [RA-3.3 P3, 2007]) es diferente del *ID* del componente.

En caso de ser necesario agregar más de dos servicios (incluido el servicio *Core*) el estándar propone utilizar para el componente un *ID* no definido. Está claro que este *ID* debe ser utilizado en la etapa de inicialización antes descrita.

I-JAUS permite agregar cada servicio de manera sencilla. Simplemente, se debe invocar la función *yAddService()*. Por ejemplo, si se desea agregar el servicio *Joint Velocity Sensor*, la función

se denomina *jvsAddService()*.

Esta etapa finaliza en el momento en el que el componente es arrancado. Hecho esto, no será posible agregar más servicios.

Arranque del componente

El componente, a nivel de OpenJAUS, cobra vida a partir de la ejecución de un hilo. Éste será el responsable de la recepción de mensajes, entre otras cosas.

Para arrancar un componente el único requisito es que el mismo haya sido inicializado. Claro está, su funcionamiento toma real sentido si se le han agregado servicios, pero I-JAUS no restringe a que sea un paso obligado. En este punto se agrega el servicio *Core*, obligatorio para todo componente JAUS.

Para realizar este paso, la interfaz pone a disposición la función *startComponent()*. Es indispensable realizar esta llamada para poner en funcionamiento el componente I-JAUS.

A partir de este momento, se encuentran en ejecución el *hilo HDR* y el *hilo OpenJAUS*, en el mismo proceso. El código implementado en la HDR es capaz de enviar y recibir mensajes JAUS compatibles mediante funciones puestas a disposición por I-JAUS.

Resumen de procedimiento

A continuación se presentan brevemente los tres pasos explicados hasta aquí.

- Inicializar componente mediante la función *initializeComponente()*.
- Agregar servicio/s al componente mediante funciones de formato *yAddService()*.
- Arrancar el componente mediante la función *startComponent()*.

JAUS utiliza el mensaje *heartbeat* para determinar si un componente está aún presente en el sistema. Esta señal es solicitada periódicamente a un componente a partir su inicialización (primer paso). Si el mismo no responde, se lo considera fuera del sistema. La respuesta de este tipo de solicitudes es hecha por OpenJAUS, por ello es necesario que el hilo del componente se encuentre en ejecución. **Un componente debe ser inicializado y arrancado de forma relativamente rápida**, para evitar que el *Node Manager* (quien solicita el *heartbeat*) considere muerto al componente.

Esto debe ser tenido en cuenta principalmente en entornos con intérprete de comandos, donde el usuario no es lo suficientemente veloz para realizar todas estas llamadas en un tiempo corto. Cuando la ejecución de estas llamadas se realiza de forma automática no existe problema alguno.

En el Apéndice [B Ejemplos de I-JAUS](#), se presentan ejemplos para utilizar I-JAUS desde MATLAB, en los cuales se realizan los tres pasos explicados anteriormente.

6.6.3. Envío de mensajes desde I-JAUS

Todo mensaje posee campos que determinan cómo este debe ser enviado, y con qué información. I-JAUS distingue dos grandes conjuntos de campos: los de la cabecera, y los del cuerpo. Tanto un grupo como el otro pueden ser escritos antes de realizar el envío del mensaje.

Las funciones asociadas al envío de mensajes fueron citadas en la Tabla [6.2](#). Como resumen se tiene la siguiente tabla.

Llamada	Descripción
<code>yUpdateHeaderXMessage(...)</code>	Actualización de los campos de la cabecera del mensaje X (servicio y).
<code>yUpdateXMessage(...)</code>	Actualización de los campos del cuerpo del mensaje X, (servicio y).
<code>ySendXMessageTo(...)</code>	Envío del mensaje X, (servicio y). El destino será el entregado como parámetro a la llamada.
<code>ySendXMessage()</code>	Envío del mensaje X, (servicio y). El destino será el presente en la cabecera del mismo mensaje.

Tabla 6.3: Resumen de las llamadas para el envío de mensajes.

Los campos de la cabecera son iguales para cualquier tipo de mensaje. Su modificación es poco habitual una vez hecha la primera escritura. Por esta razón existe una llamada de escritura de campos de cabecera, y otra para los campos del cuerpo.

La cabecera involucra el destino. Al diseñar las llamadas, los desarrolladores vieron importante dar una facilidad extra al usuario: indicar el destino en la misma llamada de envío. Por ello existe la llamada `ySendXMessageTo(...)`. Con ella el usuario puede prescindir de la actualización de la cabecera.

En caso de ser necesario escribir la cabecera, el usuario debe ingresar el destino. Si este no varía a lo largo de la vida del componente, es útil la llamada `ySendXMessage()` (el destino se encuentra implícito en la cabecera del mensaje).

6.6.4. Recepción de mensajes

OpenJAUS se encarga de muchas de las cuestiones básicas que permiten la comunicación entre componentes. Lo importante de mencionar en este punto, es que pone a disposición un mecanismo para indicar qué función será la encargada del procesamiento de mensajes entrantes. De esta manera, todos los mensajes recibidos requieren una misma estructura² y son procesados por dicha

²La estructura se llama `JausMessage`, y es definida por OpenJAUS

función. La misma, ha sido denominada *genProcessMessage()* en la Interfaz y será invocada cada vez que arribe un mensaje al componente.

De acuerdo con lo explicado en secciones anteriores, el servicio *Joint Positions Driver* estaría en condiciones de preservar simultáneamente:

- El último mensaje *Report Joint Efforts* recibido.
- El último mensaje *Report Joint Positions* recibido.
- El último mensaje *Report Manipulator Specifications* recibido.
- El último mensaje *Set Joint Positions* recibido.

Sin embargo, existen componentes definidos por JAUS que cuentan con algunos mensajes en común. ¿Qué ocurre cuando más de un servicio activo es capaz de recibir un mensaje X?

Cada servicio cuenta con una función dedicada al procesamiento de mensajes entrantes. Sencillamente, la función *genProcessMessage()* entrega el mensaje a dichas funciones, delegando la responsabilidad a cada servicio de verificar si se trata de un mensaje capaz de recibir. Para un servicio *Joint Position Driver*, la función específica se denomina *jpdProcessMessage()*.

Internamente, estas funciones preservan aquellos mensajes de entrada correspondientes al servicio en la estructura específica del mismo. Finalizada esta tarea, se puede afirmar que la **recepción ha sido correcta**.

Debe quedar claro: si un componente recibe un mensaje que no es soportado por los servicios activos, lo descartará.

Cada vez que llega un mensaje se utiliza el mecanismo de exclusión mutua. Esto fue presentado en la Sección [6.5.2 OpenJAUS y HDR](#).

6.6.5. Campos de tamaño variable

Muchas de las funciones de I-JAUS tienen como objetivo principal la obtención de datos, el ejemplo principal de esto son las llamadas *yGetXMessage*. Una gran cantidad de las mismas están destinadas a obtener mensajes que poseen campos de tamaño variable, como por ejemplo el mensaje *Report Joint Positions*, cuyo campo *Joint N* posee una longitud determinada por el campo *Number of joints* (1° campo del cuerpo).

Dado que I-JAUS es utilizado por Herramientas de Desarrollo cuya implementación no es de conocimiento público (al menos en principio), es riesgoso hacer presunciones sobre cómo las HDR

administran el espacio de memoria utilizado para hacer el traspaso de la información. Por esta razón, I-JAUS ha sido construido con políticas conservadoras.

La librería es la primera que conoce la longitud de los campos de tamaño variable, pues cada uno de sus campos son accesibles desde el momento en que llega el mensaje. Dicho esto, existen dos alternativas principales para informar a la HDR del mensaje.

La primera (Figura 6.6) consiste en dejar a la librería la responsabilidad de reservar espacio de memoria, poner en dicho bloque los datos correspondientes, y entregar una referencia a la HDR. Este mecanismo se muestra inseguro por varias razones. En primer lugar, no es sencillo afirmar en qué momento ese espacio de memoria es utilizado por la HDR. Esto depende de su implementación. Por tanto no existe un momento indicado (y seguro) para liberar el espacio, o para modificar sus datos. En segundo lugar, existen varios entornos que, si bien soportan el uso de librerías de enlace dinámico, no soportan completamente el uso de punteros. LabVIEW por ejemplo, no tiene soporte para el uso de punteros por referencia (o punteros a punteros).

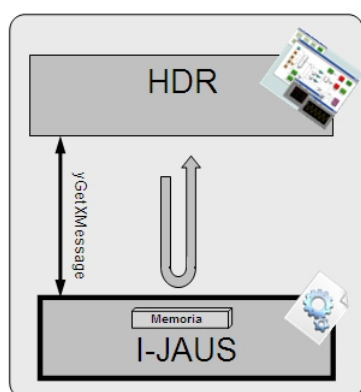


Figura 6.6: Memoria reservada por I-JAUS.
Forma insegura.

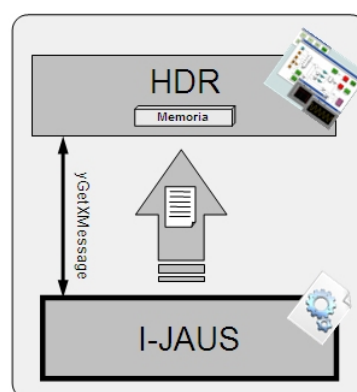


Figura 6.7: Memoria reservada por HDR.
Forma segura.

La segunda alternativa, mostrada en la Figura 6.7 consiste en dejar a la HDR la responsabilidad de la reserva de memoria para el traspaso de datos. Al momento de hacer la llamada, la librería copia los datos correspondientes en la zona de memoria reservada por la HDR. De esta manera, el momento en el cual se libera, o se lee esa zona de memoria, es determinado por la misma HDR. Esta es la política adoptada por I-JAUS.

Sin embargo, esta solución tiene un problema. El tamaño del espacio de memoria requerido es desconocido para la HDR. Entonces, ¿qué cantidad de espacio reservar? Existen dos soluciones que se describirán a continuación.

Antes de abordarlas es necesario tener en cuenta algo. El argumento que indica el tamaño

(variable) del campo, es usado también como parámetro de entrada, indicando la cantidad de elementos que la HDR ha reservado. En caso de ser un espacio insuficiente, el código de retorno lo informará mediante un error, y pondrá en este mismo campo la cantidad correcta de elementos requeridos.

Para más información ver el Apéndice [A Manual de Referencia de I-JAUS](#).

Procedimiento formal de reserva de memoria

Este mecanismo es sencillo, aunque largo: consiste en hacer una falsa llamada a la función de obtención, pero forzando al error de espacio insuficiente (mediante el campo de tamaño puesto a 0). De esta manera se averigua el espacio de memoria requerido por I-JAUS, se hace la reserva, y luego se repite la llamada.

Cada mensaje con campo de tamaño variable viene acompañado de un campo de tamaño fijo, que informa la longitud del primero. Siendo *Report Joint Positions* el mensaje (y el servicio *Joint Position Driver*), el campo de tamaño variable es *jointPosition* y el campo de longitud es *numJoints*.

Procedimiento rápido de reserva de memoria

Este procedimiento se basa en el conocimiento del diseñador, y en la presunción de un tamaño máximo de los campos de tamaño variable.

1. Realizar la reserva de memoria basada en la presunción del diseñador.
2. Hacer la llamada , poniendo en *numJoints* el valor supuesto, y pasando la referencia al espacio de memoria en el argumento *jointPosition[]*.
3. Verificar que el código de error sea de éxito.

En caso de observar que los retornos son errores relacionados al espacio reservado, basta con aumentar el espacio y volver a intentar.

6.6.6. Solicitud de mensaje en espera

¿Qué mensaje debe leerse primero?

I-JAUS dispone de un mecanismo de *solicitud de mensajes en espera* que permite al usuario conocer qué *tipo de mensaje*³ lleva más tiempo esperando ser atendido. El nombre de la llamada asociada es *requestUnreadMessageld*. Para entender su funcionamiento es necesario tener presente estos hechos:

³El tipo de un mensaje está dado por su *ID* (campo *Command Code*).

- Un componente sólo posee **la última copia** de cada tipo de mensaje recibido. No hay cola de *instancias de mensaje*. Esto quiere decir que, por ejemplo, no habrán dos instancias⁴ *Set Joint Efforts* en un *Primitive Manipulator*.
- La *solicitud del mensaje en espera* entregará el ID del mensaje cuyo tipo lleva más tiempo en espera.

La justificación del primer ítem es sencilla: la utilización de colas introduce un retardo en el conocimiento de ciertos eventos, lo cual es muy contraproducente en el planteo de JAUS⁵. Para aquellos mensajes que requieren más fiabilidad, el estándar brinda el mecanismo especial *ACK* que podrá ser utilizado.

El segundo ítem está asociado al concepto de *inanición*. Únicamente para preservar el orden de llegada de los tipos de mensajes, I-JAUS dispone de una *cola de preservación de orden*. Como se ha citado, I-JAUS dispone de una única copia de un *tipo de mensaje*. Si la cola mantuviese el orden de llegada de *instancias de mensaje* (y no de *tipos de mensaje*), existiría un problema de inanición.

La Figura 6.8 presenta un ejemplo.

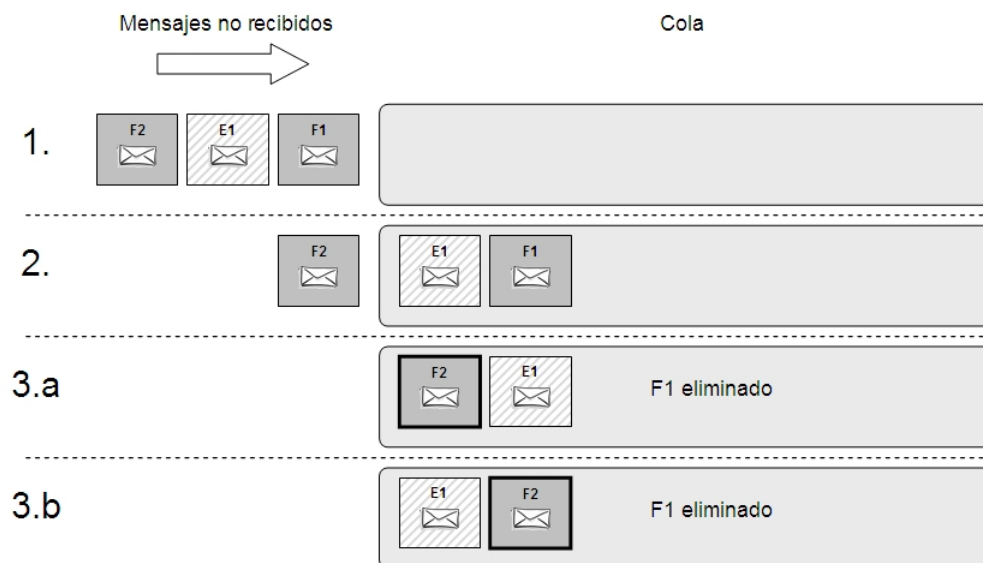


Figura 6.8: Comportamiento de las diferentes colas de preservación de orden de mensajes recibidos.

⁴Instancias de mensaje son dos mensajes del mismo tipo, pero enviados en diferentes momentos. Por ejemplo, dos mensajes *Set Joint Efforts* con diferentes esfuerzos cada uno.

⁵De los 10 componentes *Manipulator*, 6 están destinados al control a lazo cerrado.

Se tienen un mensaje F (frecuente), y un mensaje E (esporádico). Si la cola mantiene el orden de *instancias de mensaje*, se tiene la siguiente situación: llega el mensaje F (1 en la figura), y luego el E (2 en la figura). Hasta este punto el retorno de la llamada sería el tipo del mensaje F, pues es el mensaje que más tiempo posee en espera. Al llegar nuevamente el mensaje F, por el hecho de que hay una sola copia de cada tipo, queda eliminado *F1* (y se pone al final de la cola *F2*, situación 3.a en la figura). Por su gran frecuencia, el mensaje F se encuentra habitualmente al final de la cola. Esto quiere decir que la llamada retornaría siempre mensajes de tipo E (situación de inanición del tipo F).

Mantener una cola de *tipos de mensaje* evita este problema. La situación anterior tiene otro resultado: llega el mensaje *F1*, y luego el *E1*. Al llegar el mensaje *F2* (3.b en la figura) se elimina *F1*, pero el tipo F mantiene su primer lugar en la cola.

6.6.7. Acceso a mensajes

¿Cómo se notifica a la HDR que el componente ha recibido un mensaje? ¿Cómo se indica cuál ha sido?

I-JAUS pone a disposición un conjunto de funciones que brindan respuesta a las preguntas formuladas.

Consulta por mensaje entrante

La consulta esta destinada a verificar la llegada de un nuevo mensaje.

La Interfaz ofrece dos versiones diferentes de consulta, cuyas características se presentan a continuación. En ambos casos se debe acceder a los datos de usuario, motivo por el cual es utilizado el mecanismo de exclusión mutua.

Consulta no bloqueante

La primer variante, simplemente permite consultar por la llegada de un nuevo mensaje. Si la cola de preservación de orden **no** se encuentra vacía, retorna el *ID* del primer mensaje en espera. La función asociada se denomina *requestUnreadMessageId()*.

Consulta bloqueante

La segunda variante, un tanto más compleja, permite que el usuario indique el *ID* del mensaje por el cual consulta. En principio, se accede a la cola de preservación de orden para verificar si se encuentra el mensaje requerido. En caso afirmativo, retorna exitosamente. Caso contrario, se bloquea condicionalmente el hilo *HDR* por un cierto lapso de tiempo (configurable).

En este escenario el hilo se desbloquea cuando:

- Se cumple el lapso de tiempo sin que el mensaje sea recibido (*timeout*).
- El mensaje esperado es recibido.

La función asociada se denomina *requestIncomingMessage()*.

Además, esta última variante presenta un caso particular. I-JAUS define un valor de *ID* que funciona como *wildcard*. En ese caso, el hilo se bloquea si no hay ningún mensaje nuevo y se desbloquea ante el primer mensaje que es recibido.

Obtención de un mensaje entrante

La obtención de un mensaje entrante, implica acceder a la *estructura específica* del servicio al cual corresponde. Por supuesto, se necesita del mecanismo de exclusión mutua para prevenir que durante su transcurso, se pueda producir la escritura de un nuevo mensaje.

La reserva de memoria para el paso de parámetros, es necesaria antes de invocar una función dedicada a la obtención de un mensaje. Ni bien llega uno es almacenado en la *estructura específica* del servicio correspondiente. Sin embargo, la HDR accede a estos datos luego de que I-JAUS los copie al espacio de memoria proporcionado por ella misma (durante la llamada de obtención). Por esto el usuario debe tener en cuenta esta reserva (ver Sección [6.6.5 Campos de tamaño variable](#)) previo a realizar este tipo de llamadas.

JAUS define un gran variedad de mensajes, los cuales difieren no solo en el significado de sus campos, sino también en la cantidad de los mismos. Esto ha derivado en la implementación por parte de I-JAUS, de diversas funciones para acceder a cada tipo de mensaje. Cada función respeta el formato *yGetXMessage()*. Por ejemplo, la función *jpdGetReportJointPositionsMessage* permite la obtención del mensaje *Report Joint Positions* correspondiente al servicio *Joint Positions Driver*.

Una vez que un mensaje es obtenido por la HDR, I-JAUS lo descarta. Esto implica, eliminar aquel nodo que preserva su *ID* en la cola de preservación de orden.

6.6.8. Otras llamadas

Existen otras llamadas importantes. Algunas de ellas son:

Llamada	Descripción
coreEstablishIncomingSC(...)	Establece un <i>Service Connection</i> entrante.
coreTerminateIncomingSC(...)	Finaliza un <i>Service Connection</i> entrante.
coreAddSupportedOutgoingSC(...)	Agrega soporte para un <i>Service Connection</i> saliente.
coreIsOutgoingSCActive(...)	Verifica si un <i>Service Connection</i> saliente posee usuarios activos.
coreGetOutgoingSCInfo(...)	Retorna información sobre usuarios activos de un <i>Service Connection</i> saliente.
coreTerminateOutgoingSC(...)	Elimina el soporte para un <i>Service Connection</i> saliente.
setRequestorTimeout(...)	Establece el tiempo de expiración para llamadas bloqueantes.
getComponentAuthority(...)	Obtiene el nivel de autoridad asignado.
setComponentAuthority(...)	Asigna el nivel de autoridad.
getComponentState(...)	Obtiene el estado del componente.
setComponentState(...)	Establece el estado del componente.
getComponentAddress(...)	Obtiene la dirección/ID JAUS del componente.
getComponentController(...)	Obtiene la dirección/ID JAUS del componente controlador (si lo hay).

Tabla 6.4: Llamadas particulares de I-JAUS.

Para más información, referirse al *Manual de Referencia de I-JAUS* (Apéndice [A Manual de Referencia de I-JAUS](#)).

6.6.9. Finalización

La forma correcta de quitar un componente del sistema es mediante su destrucción, desde la HDR. La llamada asociada es *destroyComponent()*. La misma detiene el hilo de OpenJAUS, libera las estructuras asociadas, e indica al *Node Manager* que éste componente va a abandonar el sistema.

6.7. Retorno de errores

I-JAUS dispone de una forma de informar al usuario de cada situación anómala que ocurra durante una llamada. Si bien se ha llamado *código de error* al valor retornado por las llamadas, es necesario aclarar que no siempre dicho código corresponde con un error, sino que a veces informa al usuario de acontecimientos importantes, como la transgresión del estándar JAUS.

Entre algunos acontecimientos se encuentran:

- Forma incorrecta de poner en funcionamiento un componente.
- Parámetros incorrectos.
- Llamada que no es válida de acuerdo a la configuración del componente (por ejemplo respecto de los servicios activos).

- Tiempo de espera agotado (algunas llamadas son bloqueantes).

Para más información sobre cada código de error, vea el *Manual de Referencia de I-JAUS* (Apéndice [A Manual de Referencia de I-JAUS](#)).

6.8. Uso de I-JAUS en HDR

Es necesario aclarar que antes de la ejecución de cualquier componente, debe existir en el nodo (PC o embebido) un *Node Manager* en ejecución. La Figura 6.9 presenta un posible escenario a modo de ejemplo.

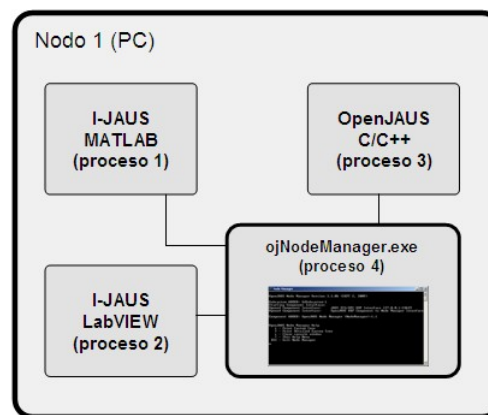


Figura 6.9: Procesos en ejecución dentro de un nodo JAUS.

Cada HDR requiere de una serie de pasos para poner en marcha un componente I-JAUS. Las secciones siguientes los presentan.

6.8.1. LabVIEW

En la versión 8.5 este programa permite hacer uso de Librerías de Enlace Dinámico (o *dll*), que es el modo en que I-JAUS es provisto. En la Figura 6.10 se puede ver la configuración realizada para acceder a una llamada de la librería desde este entorno.

Resumidamente, el procedimiento para disponer de los bloques con las llamadas preconfiguradas (de forma automática) es:

Tools -> Import -> Shared Library (.dll)...

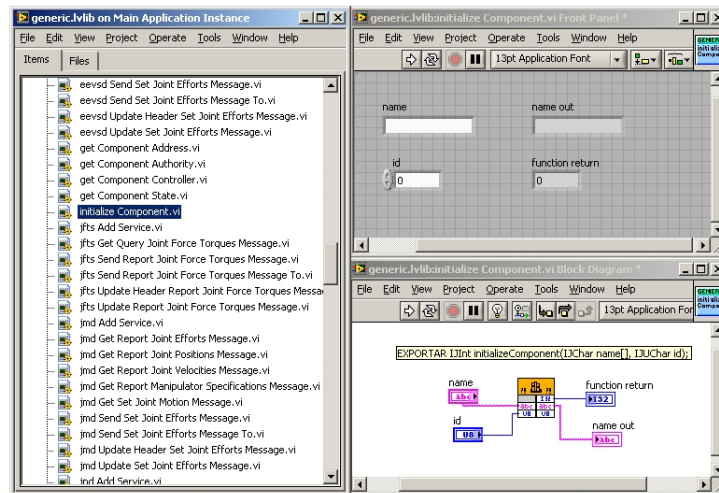


Figura 6.10: Configuración desde LabVIEW de una llamada a la librería de I-JAUS.

Los resultados luego de un poco de trabajo pueden tener una apariencia como la de la Figura 6.11. Allí se muestran algunos comandos asociados al componente *End-Effector Pose Driver*, junto al envío y la recepción de mensajes. Las funcionalidades del servicio *Core* son igualmente fáciles de implementar.

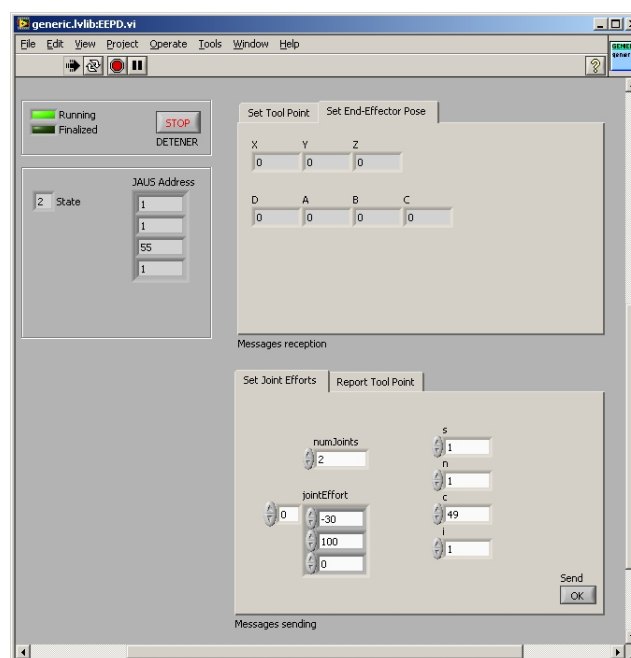


Figura 6.11: Componente JAUS EEPD corriendo en LabVIEW (al estilo panel de control).

Para más información referirse al *Manual de Referencia de I-JAUS* (Apéndice [A Manual de Referencia de I-JAUS](#)).

Pueden encontrarse ejemplos en el Apéndice [B Ejemplos de I-JAUS](#).

6.8.2. MATLAB

Para esta HDR se ha construido una capa adicional: el usuario de MATLAB está provisto con más facilidades respecto del usuario de LabVIEW, así como de un Manual de Referencia exclusivo. Esto fue realizado puesto que se requería brindar facilidades incluso a nivel de Simulink (ver Sección [2.3 Replanteo de requerimientos](#)).

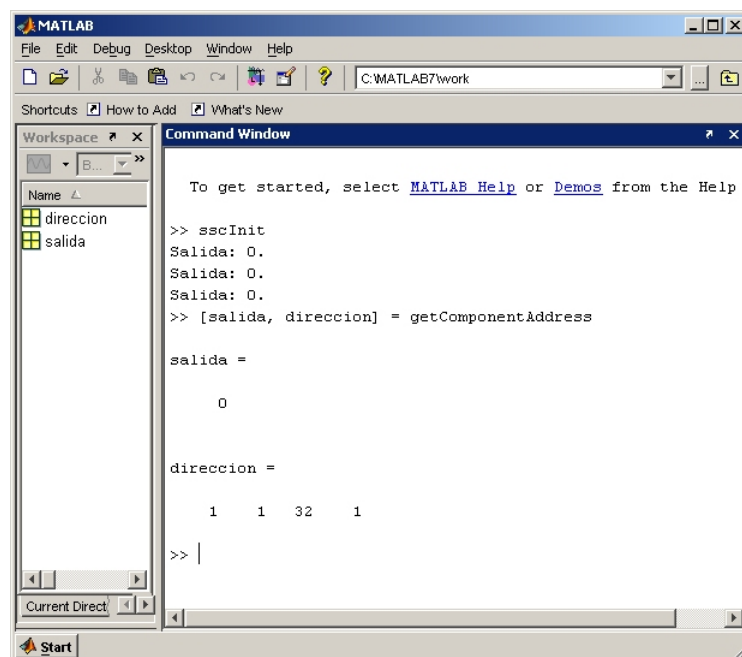


Figura 6.12: Componente *Subsystem Commander* corriendo en MATLAB (consola).

El componente puede ser ejecutado por el mismo usuario, desde el intérprete de comandos de MATLAB, sin compilación alguna. Una vez arrancado el componente, este queda a la espera de las llamadas del usuario. Esto brinda enormes posibilidades para aprender el estándar, explorar las posibles llamadas, interpretarlas y realizar pruebas con un mínimo esfuerzo⁶.

Para más información referirse al *Manual de Referencia de I-JAUS para MATLAB*.

⁶MATLAB soporta *autocomplete*. Significa que el usuario puede prescindir de la escritura del nombre completo de cada llamada. Esto se logra tipeando las primeras letras de la misma (en el intérprete), y luego presionando la tecla TAB.

Pueden encontrarse ejemplos en el Apéndice [B Ejemplos de I-JAUS](#).

6.8.3. Simulink

Gracias a que Simulink permite hacer uso de las funciones de MATLAB, e I-JAUS brinda soporte especial para MATLAB, el proyecto I-JAUS es perfectamente utilizable desde este reconocido entorno de simulación.

Una de las vías mediante las cuales se puede reutilizar código *M* en Simulink es a través de *S-Functions*.

Pueden encontrarse ejemplos en el Apéndice [B Ejemplos de I-JAUS](#).

7

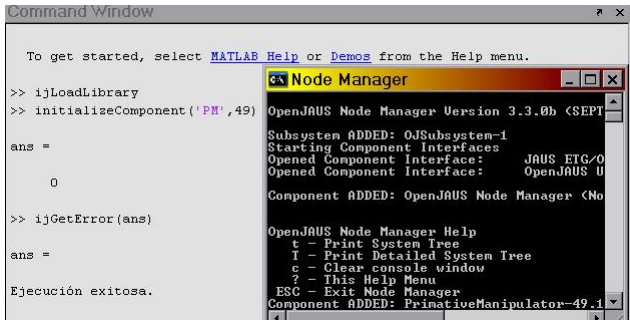
Validación y Verificación

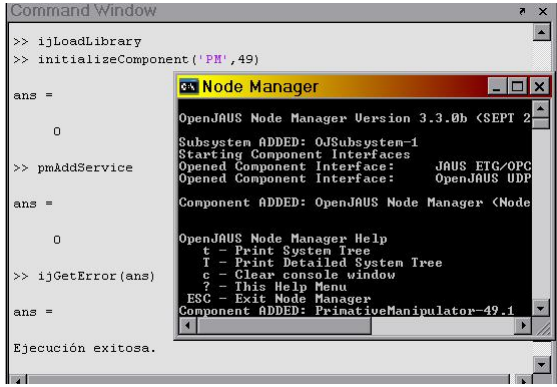
Durante el desarrollo de la Interfaz se hicieron numerosas pruebas destinadas tanto a verificar el correcto funcionamiento de cada una de las partes del código, como a validar los requerimientos de la implementación. De todas ellas se han documentado las más significativas.

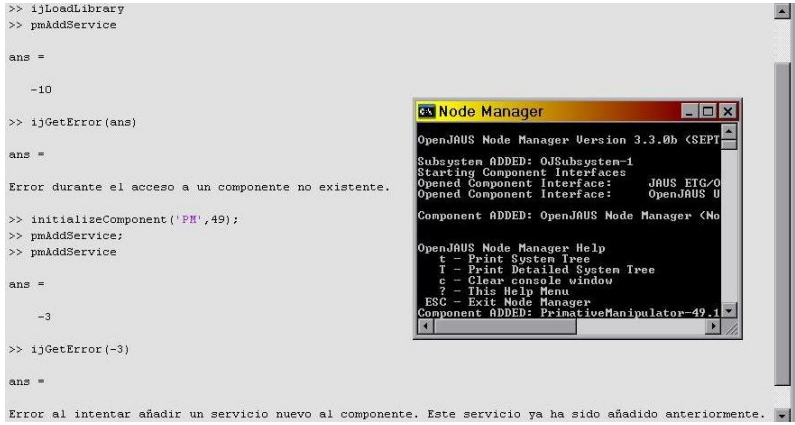
Algunas pruebas están enfocadas en funcionalidades básicas de los componentes y otras están enfocadas en cuestiones que hacen a la interacción entre los mismos. Para estas últimas, se plantearon escenarios hipotéticos que se acercan al uso recomendado de la Interfaz.

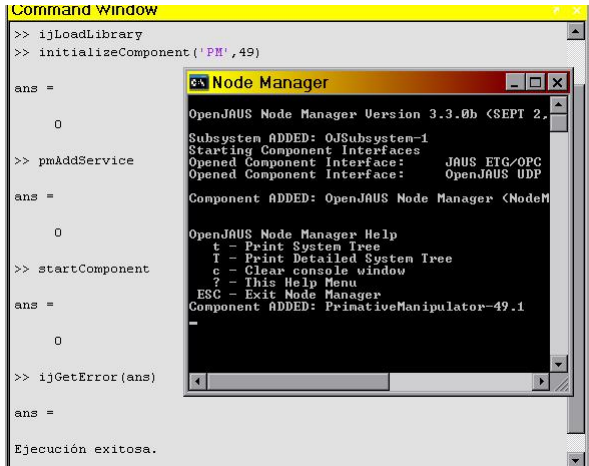
Las pruebas fueron realizadas en etapas: inicialmente en lenguaje C y, a medida que la implementación de la Interfaz maduró, las mismas se realizaron en MATLAB. Con esto se verificó el correcto funcionamiento en uno de los entornos en que se espera que I-JAUS sea utilizada comúnmente.

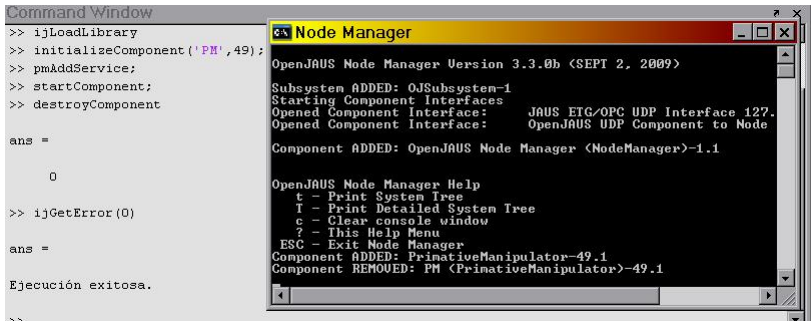
El hecho de iniciar un componente e integrarlo a un sistema JAUS (Requerimiento 1.1 en Sección [2.3 Replanteo de requerimientos](#)) requiere tres etapas.

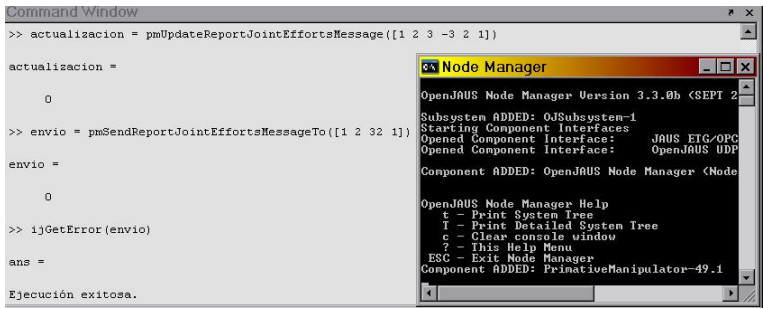
CASO DE PRUEBA	
Nombre	Inicialización de componente I-JAUS.
Identificador	1. Requerimiento 1.1 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verifica la inicialización correcta de un componente.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1 * NodeId: 2 * Subsystem_Identification: OJSubsystem * Node_Identification: OJNode Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	Ejecutar Node Manager. Cargar librería.
Acciones	<ol style="list-style-type: none"> Crear componente OpenJAUS. Crear la <i>estructura genérica</i> del componente. Inicializar la <i>estructura genérica</i> del componente. Inicializar <i>tokens</i> utilizados para exclusión mutua. Inicializar la <i>cola de preservación de orden</i>.
Datos	<ol style="list-style-type: none"> Nombre del componente. Valor: "PM". ID del componente. Valor: 49.
RESULTADOS	
Esperados	Creación e inicialización correcta de la Estructura Genérica del componente. Retorno de parámetro 0 (GEN_SUCCESS).
Reales	 <p>The screenshot shows two windows. The 'Command Window' on the left displays the following MATLAB commands and output:</p> <pre>>> iLoadLibrary >> initializeComponent('PM',49) ans = 0 >> iGetError(ans) ans = Ejecución exitosa.</pre> <p>The 'Node Manager' window on the right shows the status of the OpenJAUS Node Manager Version 3.3.0b. It lists the added subsystem (OJSubsystem-1) and interfaces (JAUS ETG/O and OpenJAUS U). It also shows the help menu options: t - Print System Tree, T - Print Detailed System Tree, c - Clear console window, ? - This Help Menu, and ESC - Exit Node Manager. The component added is PrimitiveManipulator-49.1.</p>

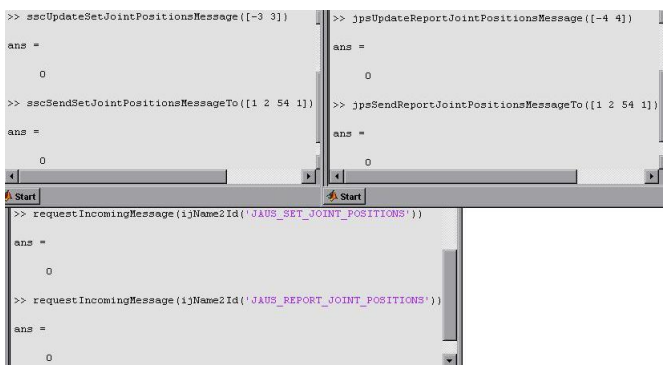
CASO DE PRUEBA	
Nombre	Agregar servicio. Requerimiento 1.1 (Sección 2.3 Replanteo de requerimientos)
Identificador	2
Versión	1
Propósito	Verificar incorporación de servicio en forma correcta.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1 * NodeId: 2 * Subsystem_Identification: OJSubsystem * Node_Identification: OJNode Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	Ejecutar Node Manager, cargar librería e inicializar componente.
Acciones	<ol style="list-style-type: none"> Agregar servicio Primitive Manipulator en OpenJAUS. Configurar la interfaz del componente para soportar el servicio. <ul style="list-style-type: none"> * Agregar mensajes de entrada del servicio. * Agregar mensajes de salida del servicio. Crear la estructura específica del servicio. Inicializar la estructura específica del servicio. Crear copia de cada mensaje asociado al servicio. Agregar estructura específica del servicio a la estructura general del componente.
Datos	La inicialización de componente requiere Nombre y ID. Valores "PM" y 49 respectivamente.
RESULTADOS	
Esperados	Inicialización y agregación correcta del servicio. Retorno de parámetro 0 (GEN_SUCCESS).
Reales	 <pre> Command Window >> ijLoadLibrary >> initializeComponent('PM', 49) ans = 0 >> pmAddService ans = 0 >> ijGetError(ans) ans = Ejecución exitosa. </pre> <p>Node Manager</p> <p>OpenJAUS Node Manager Version 3.3.0b <SEPT 2</p> <p>Subsystem ADDED: OJSubsystem-1</p> <p>Starting Component Interfaces</p> <p>Opened Component Interface: JAUS ETG/OPC</p> <p>Opened Component Interface: OpenJAUS UDP</p> <p>Component ADDED: OpenJAUS Node Manager <Node</p> <p>OpenJAUS Node Manager Help</p> <p>t - Print System Tree</p> <p>T - Print Detailed System Tree</p> <p>c - Clear console window</p> <p>? - This Help Menu</p> <p>ESC - Exit Node Manager</p> <p>Component ADDED: PrimitiveManipulator-49.1</p>


CASO DE PRUEBA	
Nombre	Imposibilitar agregado de servicio.
Identificador	3. Requerimiento 1.1 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar que no se puede agregar un servicio agregar previo a inicializar el componente y verificar que no se puede agregar más de una vez un servicio.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1 * NodeId: 2 * Subsystem_Identification: OJSubsystem * Node_Identification: OJNode Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	Ejecutar Node Manager. Cargar librería.
Acciones	<ol style="list-style-type: none"> Agregar servicio Primitive Manipulator en OpenJAUS. Inicializar componente. Agregar servicio Primitive Manipulator en OpenJAUS. Agregar nuevamente servicio Primitive Manipulator en OpenJAUS.
Datos	La inicialización de componente requiere Nombre y ID. Valores "PM" y 49 respectivamente.
RESULTADOS	
Esperados	<ol style="list-style-type: none"> Imposibilidad de agregar servicio por falta de inicialización. Retorno de parámetro -10 (GEN_COMPONENT_NOT_INITIALIZED). Inicialización de componente. Retorno de parámetro 0 (GEN_SUCCESS). Agregación de servicio Primitive Manipulator. Retorno de parámetro 0 (GEN_SUCCESS). Imposibilidad de agregar nuevamente el servicio Primitive Manipulator. Retorno de parámetro -3 (GEN_SERVICE_IS_ALREADY_PRESENT_ERROR).
Reales	 <pre> >> iLoadLibrary >> pmAddService ans = -10 >> iGetError(ans) ans = Error durante el acceso a un componente no existente. >> initializeComponent('PM',49); >> pmAddService; >> pmAddService ans = -3 >> iGetError(-3) ans = Error al intentar añadir un servicio nuevo al componente. Este servicio ya ha sido añadido anteriormente. </pre> <p>The screenshot shows the OpenJAUS Node Manager interface with the following text:</p> <pre> OpenJAUS Node Manager Version 3.3.0b <SEPT> Subsystem ADDED: OJSubsystem-1 Starting Component Interfaces Opened Component Interface: JAUS ETG/O Opened Component Interface: OpenJAUS U Component ADDED: OpenJAUS Node Manager <No OpenJAUS Node Manager Help t - Print System Tree T - Print Detailed System Tree c - Clear console window ? - This Help Menu ESC - Exit Node Manager Component ADDED: PrimitiveManipulator-49.1 </pre>

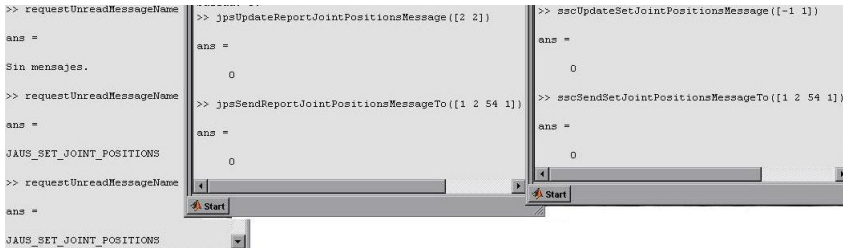
CASO DE PRUEBA	
Nombre	Arranque de componente.
Identificador	4. Requerimiento 1.1 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar la correcta puesta en ejecución del componente.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1 * NodeId: 2 * Subsystem_Identification: OJSubsystem * Node_Identification: OJNode Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Cargar librería. Iniciar componente. Agregar servicio Primitive Manipulator.
Acciones	<ol style="list-style-type: none"> Agregar servicio Core (obligatorio en cualquier componente). Establecer función responsable de procesamiento de mensajes entrantes. Ejecución del <i>hilo OpenJAUS</i>.
Datos	La inicialización de componente requiere Nombre y ID. Valores "PM" y 49 respectivamente.
RESULTADOS	
Esperados	Visualización en pantalla de Node Manager (el componente responde el <i>heartbeat</i>). Retorno de parámetro 0 (GEN_SUCCESS).
Reales	 <p>Command Window</p> <pre> >> iLoadLibrary >> initializeComponent('PM', 49) ans = 0 >> pmAddService ans = 0 >> startComponent ans = 0 >> iGetError(ans) ans = Ejecución exitosa. </pre> <p>Node Manager</p> <pre> OpenJAUS Node Manager Version 3.3.0b <SEPT 2, 2005> Subsystem ADDED: OJSubsystem-1 Starting Component Interfaces Opened Component Interface: JAUS EIG/OPC Opened Component Interface: OpenJAUS UDP Component ADDED: OpenJAUS Node Manager <NodeM OpenJAUS Node Manager Help t - Print System Tree I - Print Detailed System Tree c - Clear console window ? - This Help Menu ESC - Exit Node Manager Component ADDED: PrinitiveManipulator-49.1 </pre>

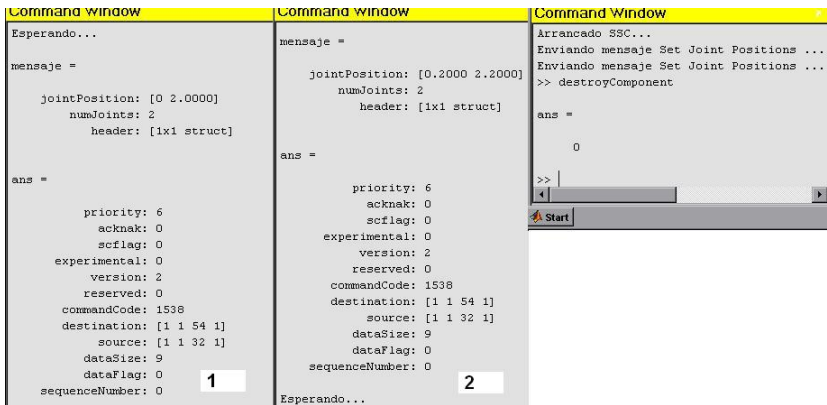
CASO DE PRUEBA	
Nombre	Dstrucción de componente.
Identificador	5. Requerimiento 1.1 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar la correcta destrucción de un componente.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1 * NodeId: 2 * Subsystem_Identification: OJSubsystem * Node_Identification: OJNode Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Cargar librería. Inicializar componente. Agregar un servicio. Arrancar componente.
Acciones	<ol style="list-style-type: none"> Destruir componente. <ul style="list-style-type: none"> * Liberar la estructura específica del servicio activo. * Liberar la estructura específica del servicio Core. * Liberar la estructura genérica del componente. * Detener ejecución del hilo <i>OpenJAUS</i>. * Liberar <i>tokens</i> utilizados para exclusión mutua.
Datos	La inicialización de componente requiere Nombre y ID. Valores "PM" y 49 respectivamente.
RESULTADOS	
Esperados	Finalización de la ejecución del hilo <i>OpenJAUS</i> . Visualización en pantalla de Node Manager. Retorno de parámetro 0 (GEN_SUCCESS).
Reales	 <p>The screenshot shows two windows. On the left is a 'Command Window' with the following text:</p> <pre> >> iLoadLibrary >> initializeComponent('PM',49); >> pmAddService; >> startComponent; >> destroyComponent ans = 0 >> iGetError(0) ans = Ejecución exitosa. </pre> <p>On the right is the 'Node Manager' window, which displays the status of the OpenJAUS system. It shows that the subsystem 'OJSubsystem-1' is started, and the component 'OpenJAUS Node Manager (NodeManager)-1.1' is added. It also shows a list of services and their status, including 'JAUS ETG/OPC UDP Interface 127.' and 'OpenJAUS UDP Component to Node'.</p>

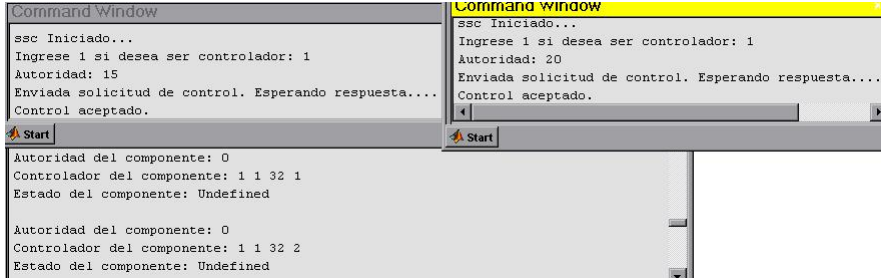
CASO DE PRUEBA	
Nombre	Envío de mensaje.
Identificador	6. Requerimiento 1.3 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar la correcta actualización de los campos de un mensaje saliente en la estructura específica y el posterior envío.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1 * NodeId: 2 * Subsystem_Identification: OJSubsystem * Node_Identification: OJNode Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Cargar librería. Inicializar componente. Agregar servicio Primitive Manipulator. Arrancar componente.
Acciones	<ol style="list-style-type: none"> Actualizar campos del mensaje Report Joint Efforts en la estructura específica del servicio. Envío de mensaje Report Joint Efforts.
Datos	La actualización requiere de un arreglo de esfuerzos jointEffort. Valores [1 2 3 -3 2 1]. El envío requiere un arreglo con la dirección destino. Valor [1 2 32 1].
RESULTADOS	
Esperados	<p>Secuencia esperada:</p> <ol style="list-style-type: none"> Retorno de la actualización de campos: 0 (GEN_SUCCESS). Retorno del envío de mensaje: 0 (GEN_SUCCESS).
Reales	 <p>The screenshot shows two windows. On the left, the MATLAB Command Window displays the following code and output:</p> <pre>>> actualizacion = pmUpdateReportJointEffortsMessage([1 2 3 -3 2 1]) actualizacion = 0 >> envio = pmSendReportJointEffortsMessageTo([1 2 32 1]) envio = 0 >> iJGetError(envio) ans = Ejecución exitosa.</pre> <p>On the right, the Node Manager window shows the status of the OpenJAUS Node Manager. It indicates that Subsystem 1 (OJSubsystem-1) is started, and the component PrimitiveManipulator-49.1 is added. The status is 'Component ADDED: PrimitiveManipulator-49.1'.</p>

CASO DE PRUEBA	
Nombre	Procesamiento de mensaje entrante.
Identificador	7. Requerimiento 1.3 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar el correcto funcionamiento del mecanismo de recepción de mensajes. Un servicio recibe dos mensajes diferentes.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1. NodeId: 2. Subsystem_Identification: OJSubsystem. Node_Identification: OJNode. Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Puesta en funcionamiento de componente Joint Positions Driver. Puesta en funcionamiento de componente Subsystem Commander. Puesta en funcionamiento de componente Joint Position Sensor.
Acciones	<ol style="list-style-type: none"> Actualizar y enviar mensaje Set Joint Positions desde componente Subsystem Commander a componente Joint Position Driver. Actualizar y enviar mensaje Report Joint Positions desde componente Joint Position Sensor a componente Joint Position Driver. En componente Joint Positions Driver <ul style="list-style-type: none"> * Procesar mensaje entrante mediante la función específica del servicio activo. <ul style="list-style-type: none"> - De existir, liberar copia previa del mensaje recibido. - Almacenar mensaje en la estructura específica del servicio al cual corresponde. * Consultar por la llegada de mensaje Set Joint Positions. * Consultar por la llegada de mensaje Report Joint Positions.
Datos	Set Joint Positions: en el cuerpo Joint Position = [-3 3]. Destino [1 2 54 1]. Report Joint Positions: en el cuerpo Joint Position = [-4 4]. Destino [1 2 54 1].
RESULTADOS	
Esperados	Recepción de cada mensaje por la función correcta: jpdProcessMessage() (es aquella asociada al servicio Joint Position Driver).
Reales	

CASO DE PRUEBA	
Nombre	Procesamiento de mensaje entrante.
Identificador	7. Requerimiento 1.3 (Sección 2.3 Replanteo de requerimientos)
Versión	2
Propósito	Verificar el correcto funcionamiento del mecanismo de recepción de mensajes. Dos servicios activos de un mismo componente reciben mensajes independientes.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1. NodeId: 2. Subsystem_Identification: OJSubsystem. Node_Identification: OJNode. Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Puesta en funcionamiento de componente con Nombre JVD-JPD, ID 77, servicios Joint Velocities Driver y Joint Positions Driver. Puesta en funcionamiento de componente Joint Velocity Sensor. Puesta en funcionamiento de componente Joint Position Sensor.
Acciones	<ol style="list-style-type: none"> Actualizar y enviar mensaje Report Joint Velocities desde componente Joint Velocity Sensor a componente JVD-JPD. Actualizar y enviar mensaje Report Joint Positions desde componente Joint Position Sensor a componente JVD-JPD. En componente JVD-JPD <ul style="list-style-type: none"> * Procesar mensaje entrante mediante la función específica de cada servicio activo. <ul style="list-style-type: none"> - De existir, liberar copia previa del mensaje recibido. - Almacenar mensaje en la estructura específica del servicio al cual corresponde. * Consultar por la llegada de mensaje. * Consultar por la llegada de mensaje.
Datos	Report Joint Velocities: en el cuerpo Joint Velocity = [-1 1]. Destino [1 2 77 1]. Report Joint Positions: en el cuerpo Joint Position = [-2 2]. Destino [1 2 77 1].
RESULTADOS	
Esperados	Recepción de cada mensaje por la función correcta: jvsProcessMessage() debe recibir Report Joint Velocities y jpsProcessMessage() debe recibir Report Joint Positions.
Reales	

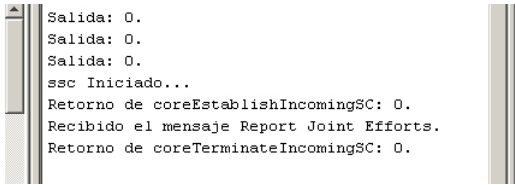
CASO DE PRUEBA	
Nombre	Procesamiento de mensaje entrante.
Identificador	7. Requerimiento 1.3 (Sección 2.3 Replanteo de requerimientos)
Versión	3
Propósito	Verificar el correcto funcionamiento del mecanismo de recepción de mensajes, habiéndose incorporado la <i>cola de preservación de orden</i> .
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1. NodeId: 2. Subsystem_Identification: OJSubsystem. Node_Identification: OJNode. Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Puesta en funcionamiento de componente Joint Positions Driver. Puesta en funcionamiento de componente Subsystem Commander. Puesta en funcionamiento de componente Joint Position Sensor.
Acciones	<ol style="list-style-type: none"> Actualizar y enviar mensajes Set Joint Positions desde componente Subsystem Commander a componente Joint Positions Driver. Actualizar y enviar mensajes Report Joint Positions desde componente Joint Position Sensor a componente Joint Positions Driver. En componente Joint Positions Driver <ul style="list-style-type: none"> * Procesar mensaje entrante mediante la función específica del servicio activo. <ul style="list-style-type: none"> - De existir, liberar copia previa del mensaje recibido. - Almacenar mensaje en la estructura específica del servicio al cual corresponde. - Verificar si existía en cola una instancia previa del mensaje recibido. De ser negativo, preservar ID del mensaje en nodo y agregarlo al final de la cola.
Datos	Set Joint Positions: en el cuerpo Joint Position = [-1 1]. Destino [1 2 54 1]. Report Joint Positions: en el cuerpo Joint Position = [2 2]. Destino [1 2 54 1].
RESULTADOS	
Esperados	<p>Recepción de cada mensaje por la función correcta: jpdProcessMessage() (es aquella asociada al servicio Joint Position Driver).</p> <p>El primer mensaje recibido no debe perder su lugar en la <i>cola de preservación de orden</i>.</p>
Reales	

CASO DE PRUEBA	
Nombre	Obtención de mensaje entrante.
Identificador	8. Requerimiento 1.3 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar el correcto funcionamiento del mecanismo de obtención de mensajes.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1. NodeId: 2. Subsystem_Identification: OJSubsystem. Node_Identification: OJNode. Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Puesta en funcionamiento de componente Joint Positions Driver. Puesta en funcionamiento de componente Subsystem Commander.
Acciones	<ol style="list-style-type: none"> Enviar dos mensajes Set Joint Positions desde componente Subsystem Commander a componente Joint Positions Driver. En componente Joint Position Driver, consultar por la llegada de nuevo mensaje mediante función requestUnreadMessageId(). <ul style="list-style-type: none"> * Obtener primer nodo de la cola de preservación de orden. * Obtener mensajes Set Joint Position (saltado si la consulta indica que no hay mensaje por obtener). <ul style="list-style-type: none"> - Reservar memoria en HDR para almacenar mensaje. - Copiar datos de mensaje en espacio de memoria reservado por HDR. - Presentar datos del mensaje.
Datos	Set Joint Positions: en cuerpo de mensaje Joint Position [a b], a inicia en 0, b inicia en 2, ambos incrementan 0.2 con cada envío.
RESULTADOS	
Esperados	Obtención de cada mensaje entrante.
Reales	 <p>The figure displays three sequential screenshots of the MATLAB Command Window, illustrating the test execution process. The first window shows the initial state with 'Esperando...' and a message structure. The second window shows the received message details. The third window shows the execution of 'destroyComponent' and the final state.</p>

CASO DE PRUEBA	
Nombre	Obtención de Control Exclusivo.
Identificador	9. Requerimiento 1.2 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar el correcto funcionamiento del mecanismo de obtención de control exclusivo.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1. NodeId: 1. Subsystem_Identification: OJSubsystem. Node_Identification: OJNode. Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Puesta en funcionamiento de componente Primitive Manipulator. Puesta en funcionamiento de componente Subsystem Commander (A). Puesta en funcionamiento de componente Subsystem Commander (B).
Acciones	<ol style="list-style-type: none"> Subsystem Commander A <ul style="list-style-type: none"> * Actualizar Autoridad del componente. * Obtener control exclusivo del Primitive Manipulator. Subsystem Commander B <ul style="list-style-type: none"> * Actualizar Autoridad del componente. * Obtener control exclusivo del Primitive Manipulator.
Datos	Autoridad de Subsystem Commander A: 15. Autoridad de Subsystem Commander B: 20.
RESULTADOS	
Esperados	Subsystem Commander A obtiene control exclusivo de Primitive Manipulator. Luego Subsystem Commander B obtiene control exclusivo de Primitive Manipulator, por tener mayor autoridad.
Reales	 <p>The image shows two side-by-side screenshots of a 'Command Window' from a MATLAB environment. Both windows display the same sequence of text: 'ssc Iniciado...', 'Ingrese 1 si desea ser controlador: 1', 'Autoridad: 15', 'Enviada solicitud de control. Esperando respuesta...', and 'Control aceptado.'. Below this text is a 'Start' button. The left window shows the initial state: 'Autoridad del componente: 0', 'Controlador del componente: 1 1 32 1', and 'Estado del componente: Undefined'. The right window shows the state after the 'Start' button is clicked: 'Autoridad del componente: 0', 'Controlador del componente: 1 1 32 2', and 'Estado del componente: Undefined'.</p>

CASO DE PRUEBA	
Nombre	Recepción de mensajes del Controlador.
Identificador	10. Requerimiento 1.2 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar el correcto funcionamiento de la recepción de mensajes cuando el componente receptor tiene un controlador.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo nodeManager.conf se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * SubsystemId: 1. NodeId: 1. Subsystem_Identification: OJSubsystem. Node_Identification: OJNode. Comunicación de nodos. En sección: [Node_Communications] <ul style="list-style-type: none"> * Enabled: false * #JUDP_Interface: true * #JUDP_IP_Address: * #JAUS_OPC_UDP_Interface: true * #JAUS_OPC_UDP_IP_Address: <p>Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).</p>
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Puesta en funcionamiento de componente Primitive Manipulator. Puesta en funcionamiento de componente Subsystem Commander (A). Puesta en funcionamiento de componente Subsystem Commander (B).
Acciones	<ol style="list-style-type: none"> Subsystem Commander A (SCA) <ul style="list-style-type: none"> * Actualizar Autoridad del componente. * Obtener control exclusivo del Primitive Manipulator. * Enviar periódicamente mensaje Set Joint Efforts al Primitive Manipulator. Subsystem Commander B (SCB) <ul style="list-style-type: none"> * Actualizar Autoridad del componente. * Obtener control exclusivo del Primitive Manipulator. * Enviar periódicamente mensaje Set Joint Efforts al Primitive Manipulator.
Datos	
RESULTADOS	
Esperados	El Subsystem Commander A obtiene control exclusivo de Primitive Manipulator; este último recibe solo mensajes del primero. Luego el Subsystem Commander B obtiene control exclusivo de Primitive Manipulator; este último descarta mensajes del SCA y solo recibe mensajes del SCB.
Reales	<div> <pre> Autoridad del componente: 0 Controlador del componente: 1 1 32 1 Estado del componente: Undefined mensaje = jointEffort: -0.8011 numJoints: 1 header: [1x1 struct] ans = priority: 6 acknak: 0 scflag: 0 experimental: 0 version: 2 reserved: 0 commandCode: 1537 destination: [1 1 49 1] source: [1 1 32 1] dataSize: 3 dataFlag: 0 sequenceNumber: 0 1 </pre> </div> <div> <pre> Autoridad del componente: 0 Controlador del componente: 1 1 32 2 Estado del componente: Undefined mensaje = jointEffort: -1.7990 numJoints: 1 header: [1x1 struct] ans = priority: 6 acknak: 0 scflag: 0 experimental: 0 version: 2 reserved: 0 commandCode: 1537 destination: [1 1 49 1] source: [1 1 32 2] dataSize: 3 dataFlag: 0 sequenceNumber: 0 2 </pre> </div>

CASO DE PRUEBA	
Nombre	Uso de <i>Service Connection</i> saliente.
Identificador	11. Requerimiento 1.4 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar el correcto funcionamiento del mecanismo de <i>Service Connection</i> saliente.
DEPENDENCIAS	
Entorno	<p>Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2).</p> <p>Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).</p>
Configuración	<p>En archivo <code>nodeManager.conf</code> se debe configurar:</p> <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * <code>SubsystemId</code>: 1 * <code>NodeId</code>: 1 * <code>Subsystem_Identification</code>: OJSubsystem * <code>Node_Identification</code>: OJNode Comunicación de nodos. En sección: [Node_Communications]. Similar a Caso de Prueba 6.1. Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Cargar librería <i>i_jaus.dll</i> en MATLAB. Puesta en funcionamiento del componente solicitante del SC. <ul style="list-style-type: none"> * Cargar librería, inicializar, agregar servicio <i>Subsystem Commander</i> y arrancar componente. * Solicitud de SC del mensaje <i>Report Joint Efforts</i>, con <i>Presence Vector</i> 255, con frecuencia Solicitud periódica hasta lograda la conexión. <ul style="list-style-type: none"> * Cuando lograda la conexión, recibe y muestra el mensaje. Puesta en funcionamiento del componente que entrega el SC. <ul style="list-style-type: none"> * Cargar librería, inicializar, agregar servicio <i>Primitive Manipulator</i> y arrancar componente.
Acciones	<ol style="list-style-type: none"> Desde el componente en cuestión (PM) realizar la llamada <i>coreAddSupportedOutgoingSC(rjeld)</i>, donde <i>rjeld</i> es el ID del mensaje <i>Report Joint Efforts</i>. Realizar periódicamente la llamada <i>coreGetOutgoingSCInfo(rjeld)</i> para averiguar si hay mensajes de SC pendientes de envío. El SC antes creado solicita este SC hasta que logra conexión. <ul style="list-style-type: none"> * La llamada puede retornar <code>GEN_OUTGOING_SC_INACTIVE</code> indicando que no hay usuarios a los que se deba enviar el mensaje en el momento. * Si la llamada fue exitosa (<code>GEN_SUCCESS</code>), los valores retornados poseen toda la información necesaria para realizar el envío del mensaje correspondiente. El envío de mensajes está fuera de esta prueba.
Datos	El valor de <i>rjeld</i> es 17921.
RESULTADOS	
Esperados	La llamada <i>coreGetOutgoingSCInfo(rjeld)</i> debe entregar 10 veces por segundo datos asociados al usuario del SC. Cada vez que se obtienen los datos de la llamada, se debe observar que <i>info.destination(i,:)</i> indique el solicitante del SC. El dato <i>info.sequenceNumber(i,:)</i> debe aumentar en cada retorno de datos.
Reales (imagen)	 <pre> me cont info numUs rjeld salida 1 1 SC con salida pendiente (destino [1 1 32 1], sn=21). 2 1 Sin SC. 3 1 SC con salida pendiente (destino [1 1 32 1], sn=22). 4 1 Sin SC. 5 1 SC con salida pendiente (destino [1 1 32 1], sn=23) </pre>
Reales	El componente <i>Primitive Manipulator</i> obtiene periódicamente los datos necesarios para entregar el SC solicitado.

CASO DE PRUEBA	
Nombre	Uso de <i>Service Connection</i> entrante.
Identificador	12. Requerimiento 1.4 (Sección 2.3 Replanteo de requerimientos)
Versión	1
Propósito	Verificar el correcto funcionamiento del mecanismo de <i>Service Connection</i> entrante.
DEPENDENCIAS	
Entorno	Librería compilada con Microsoft Visual Studio 2005 Versión 8.0.50727.42 (Microsoft .NET Framework Versión 2.0.50727 SP2). Prueba ejecutada en MATLAB Versión 7.0.0.19920 (R14).
Configuración	En archivo <code>nodeManager.conf</code> se debe configurar: <ol style="list-style-type: none"> Dirección. En sección: [JAUS]. <ul style="list-style-type: none"> * <code>SubsystemId</code>: 1 * <code>NodeId</code>: 1 * <code>Subsystem_Identification</code>: <code>OJSubsystem</code> * <code>Node_Identification</code>: <code>OJNode</code> Comunicación de nodos. En sección: [Node_Communications]. Similar a Caso de Prueba 6.1. Agregar al <i>path</i> de MATLAB aquella ruta donde se encuentran los archivos .m utilizados para la prueba. También agregar un directorio llamado <i>i_jaus</i> con todos los archivos .h del proyecto (que están en <i>/generic/include</i>), y los .dll (en <i>/generic/lib</i>).
Inicialización	<ol style="list-style-type: none"> Ejecutar Node Manager. Cargar librería <i>i_jaus.dll</i> en MATLAB. Puesta en funcionamiento del componente que entrega el SC. <ul style="list-style-type: none"> * Cargar librería, inicializar, agregar servicio <i>Primitive Manipulator</i> y arrancar componente. * Poner a disposición el SC del mensaje <i>Report Joint Efforts</i>. Puesta en funcionamiento del componente solicitante del SC. <ul style="list-style-type: none"> * Cargar librería, inicializar, agregar servicio <i>Subsystem Commander</i> y arrancar componente.
Acciones	<ol style="list-style-type: none"> Desde el componente en cuestión (SSC) realizar la llamada <i>coreEstablishIncomingSC(rje, pv, pmDestino, frec)</i>, donde <i>rjeld</i> es el ID del mensaje <i>Report Joint Efforts</i> solicitado. <i>pv</i> es el vector de presencia con el cual se espera el mismo. <i>pmDestino</i> es el servidor de SC. <i>frec</i> es la frecuencia a la que se solicita el mensaje. Se debe recibir el mensaje periódicamente. Finalizar el SC (llamada <i>coreTerminateIncomingSC(handle)</i>).
Datos	El valor de <i>rje</i> es 17921. <i>pv</i> es 255. <i>pmDestino</i> es [1 1 49 1]. <i>frec</i> es 10.
RESULTADOS	
Esperados	Ambas llamadas deben entregar resultados <i>GEN_SUCCESS</i> .
Reales (imagen)	
Reales	El componente SSC recibe el mensaje luego de solicitar el SC entrante.

7.1. Matriz de trazabilidad

Esta matriz representa la relación entre los requerimientos planteados en la Sección [2.3](#) y los casos de prueba recién presentados.

	T1	T2	T3	T4	T5	T6	T7.1	T7.2	T7.3	T8	T9	T10	T11	T12
R1.1	✓	✓	✓	✓	✓									
R1.2											✓	✓		
R1.3						✓	✓	✓	✓	✓				
R1.4													✓	✓
R2														
R3														

Tabla 7.1: *Matriz de trazabilidad.*

Debe quedar claro que, tanto R2 como R3, son requerimientos que no poseen casos de prueba por deberse exclusivamente a documentación.

8

Demostraciones

En este capítulo se presentarán los diferentes escenarios en los que I-JAUS fue utilizado para demostrar su funcionamiento, y para poner en evidencia sus ventajas.

8.1. PICOLEBOT

El PICOLEBOT es un prototipo de brazo robótico de bajo costo, de movimiento planar. Posee una articulación extra que brinda al robot la posibilidad de mover el extremo efector en el eje Z, aunque de modo reducido. Fue construido exclusivamente para realizar demostraciones sobre I-JAUS.

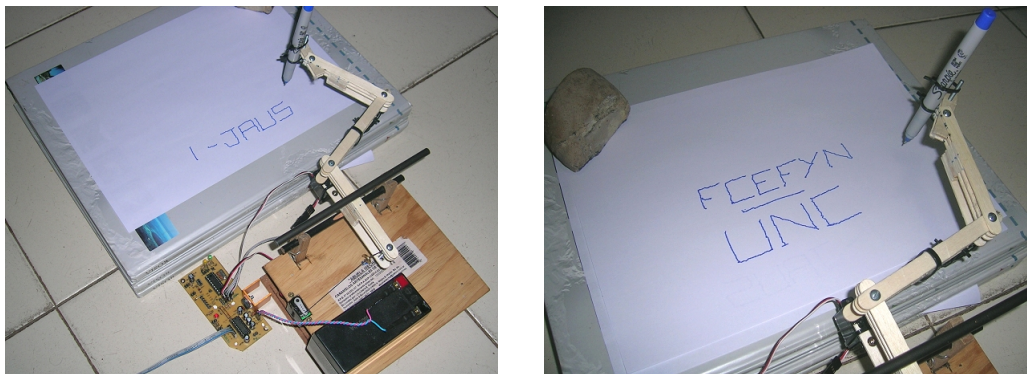


Figura 8.1: *PICOLEBOT* escribiendo.

Su extremo efector es recambiable. Uno de sus suplementos consiste en una pequeña fibra que realiza trazos conforme se mueve sobre un papel.

Representa un bloque de salida, y puede ser *encapsulado* por varios componentes JAUS,

como los componentes *End-Effector Pose Driver* (control de la posición del extremo efector, con coordenadas cartesianas), *Joint Positions Driver* (control de cada articulación) o *End-Effector Discrete Pose Driver* (control de poses logradas en el tiempo).

A continuación se presenta una serie de demostraciones haciendo uso del PICOLEBOT. Para lograr las mismas, el robot fue *envuelto* en un componente implementado con I-JAUS, sobre MATLAB. El manejo fue realizado desde componentes SSC, cuyas implementaciones y puntos de entrada se muestran diversos.

8.1.1. Utilizando Joystick (OpenJAUS - C)

En este caso, la unidad OCU fue implementada haciendo uso de OpenJAUS, en lenguaje C. Esta decisión fue tomada a causa de las facilidades del manejo de dispositivos como el *joystick* en dicho lenguaje. El robot PICOLEBOT fue *envuelto* en un componente *End-Effector Pose Driver*, implementado en I-JAUS bajo MATLAB.

El OCU es muy sencillo, simplemente envía coordenadas del extremo efector al EEPD mediante el mensaje JAUS *Set Tool Point*.

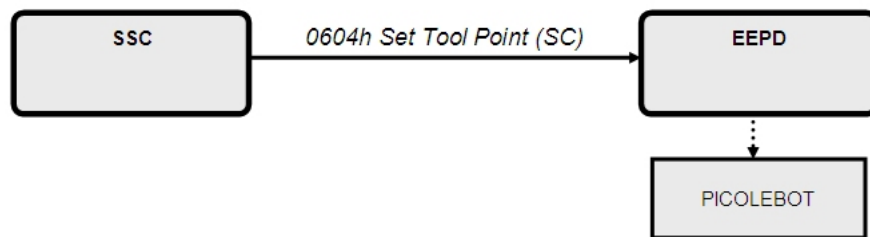


Figura 8.2: Diagrama del subsistema: SSC y EEPD.

Este ejemplo sirve a los efectos de demostrar que un componente puede ser implementado en diferentes lenguajes, y haciendo uso de diferentes herramientas. Basta con que el estándar sea respetado.

8.1.2. Utilizando Mouse (MATLAB)

En este caso, el SSC constituye un OCU. La GUI está basada en MATLAB. La unidad OCU muestra el campo de acción del robot (Figura 8.3). Esto es logrado gracias al mensaje *Report Manipulator Specifications* que recibe del componente EEDPD/PM (del servicio *Primitive Manipulator*). Los puntos que no están en dicho campo son indicados en la leyenda. El usuario establece la posición del extremo efector mediante el ratón. Cada vez que se indica el trazado de

una línea, el componente SSC envía un mensaje JAUS *Set End-Effector Path Motion* al componente EEDPD/PM que *envuelve* al PICOLEBOT (Figura 8.4). Este último componente se ejecuta en otra instancia de MATLAB, que puede correr en una computadora diferente.

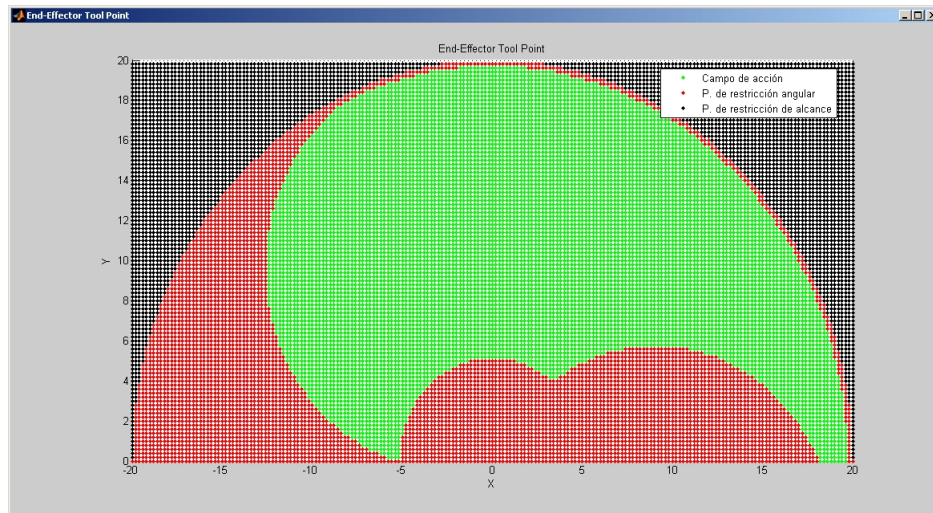


Figura 8.3: SSC en MATLAB donde la posición del puntero determina la del extremo efector.

Basado en [Faruque, 2006], se tiene el siguiente diagrama de esta demostración:

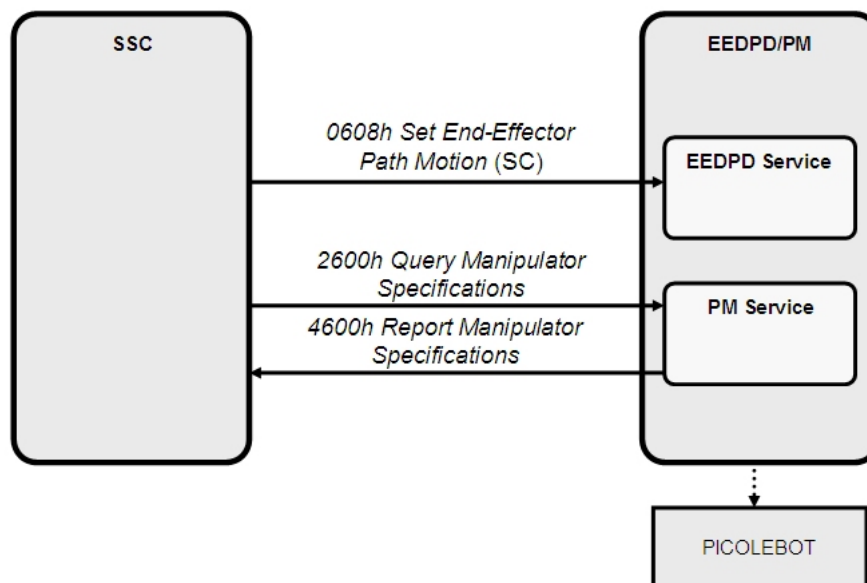


Figura 8.4: Diagrama del subsistema: SSC y EEDPD/PM.

A continuación se pueden ver los resultados de la prueba.

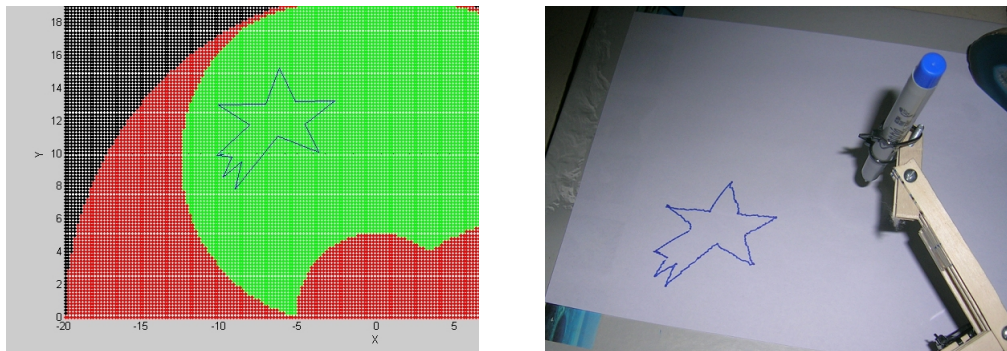


Figura 8.5: Resultados de la prueba.

8.1.3. Utilizando voz (MATLAB)

Esta demostración consiste en un componente SSC capaz de identificar patrones de sonido, tales como letras pronunciadas por el usuario, y traducirlos en comandos JAUS. Se utilizó como base un proyecto existente llamado *Speech Recognition*¹. Este fue adaptado y *envuelto* en un componente con servicio *End-Effector Pose Driver*.

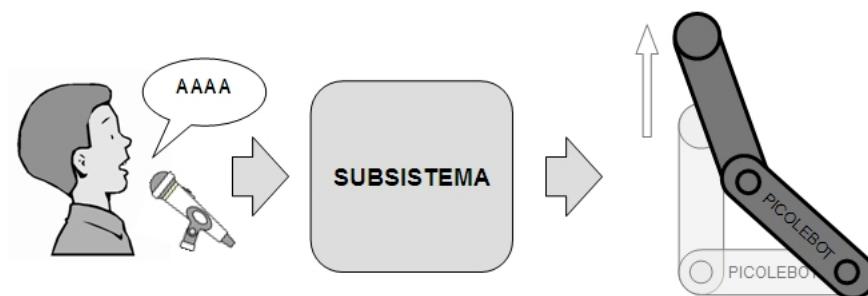


Figura 8.6: Manipulación del extremo efector mediante la voz.

La Figura 8.7 representa este diseño.

¹Este se encuentra disponible en <http://www.speech-recognition.de/matlab-examples.html>.

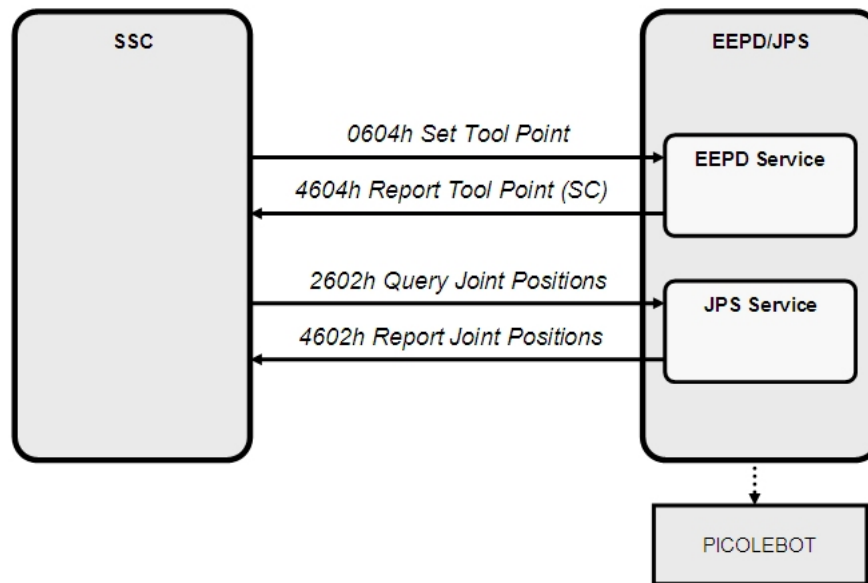


Figura 8.7: Diagrama del subsistema: SSC y EEPD/JPS.

8.1.4. Utilizando *seguimiento ocular (MATLAB)*

Esta demostración utiliza el proyecto *Fast Eyetracking*² para lograr reconocimiento facial bajo MATLAB. Una vez hecho un reconocimiento general de las facciones del rostro, se obtiene información de la posición de las pupilas. Luego se identifica el cuadrante al cual pertenece la posición de una de ellas (ver Figura 8.8) respecto de la posición inicial. En base a esto, se entrega una posición al extremo efector.

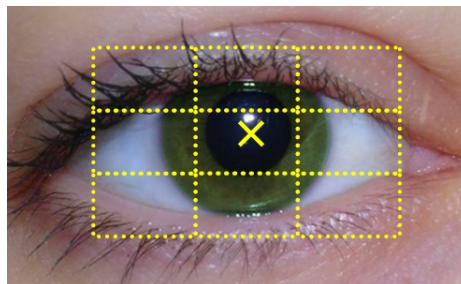


Figura 8.8: Cuadrantes de reconocimiento de pupila.

En las siguientes imágenes se observa el seguimiento para dos personas.

²Proyecto disponible en <http://www.mathworks.com/matlabcentral/fileexchange/25056-fast-eyetracking>.

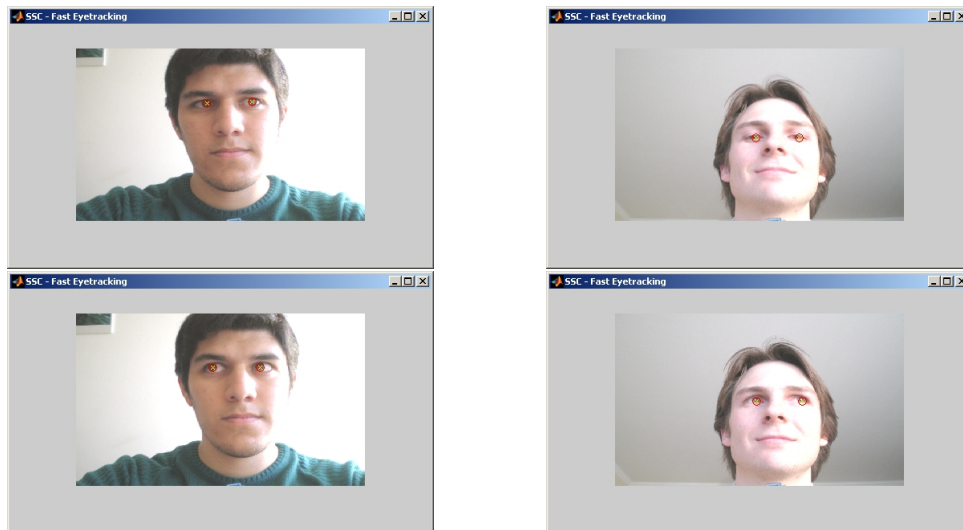


Figura 8.9: Resultados del reconocimiento de ojos.

La unidad OCU envía el mensaje *Set Tool Point*, indicando una posición particular. Al igual que en demostraciones previas, se utiliza el componente *End-Effector Pose Driver* para envolver al PICOLEBOT.

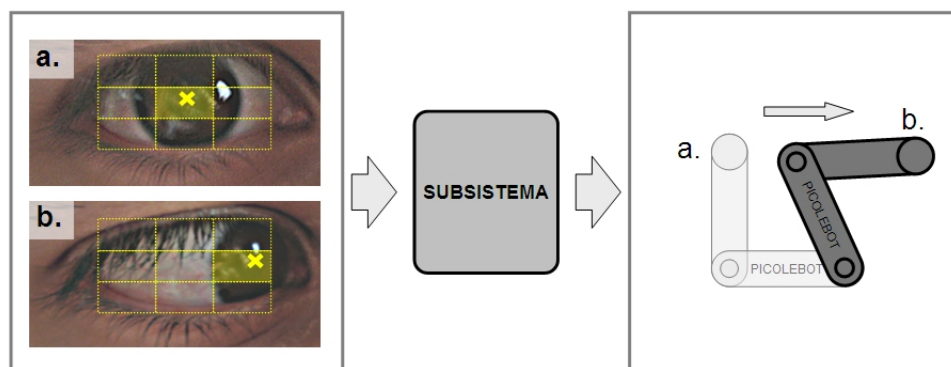


Figura 8.10: Comportamiento del PICOLEBOT ante un movimiento ocular.

Esta demostración tiene mucha importancia: así como es posible identificar patrones para realizar el seguimiento de ojos, también es posible reconocer patrones de otros elementos, tales como herramientas, manos, o algún objeto que se desee tener en cuenta a la hora de dar posición a un brazo robótico.

8.2. Robot Laparoscópico

El Robot Laparoscópico es descrito en la Sección [2.1.1 Robot Laparoscópico](#).

Esta demostración hace hincapié en dos cuestiones:

1. La importancia del soporte que I-JAUS posee para MATLAB.
2. La sencillez con la que es posible integrar el modelo del Robot Laparoscópico con un componente I-JAUS, y por ello con un sistema JAUS.

El primer ítem es de particular importancia, puesto que permite desarrollar componentes I-JAUS mediante *S-Functions* para Simulink. El segundo ítem es una consecuencia del primero.

El componente EEPD fue desarrollado en un bloque *S-Function*, e integrado al modelo del Robot Laparoscópico en Simulink. Fue reutilizado el componente SSC implementado en OpenJAUS, aquel que soporta el uso de joystick. Con esto se envían comandos al extremo efector del brazo robótico.

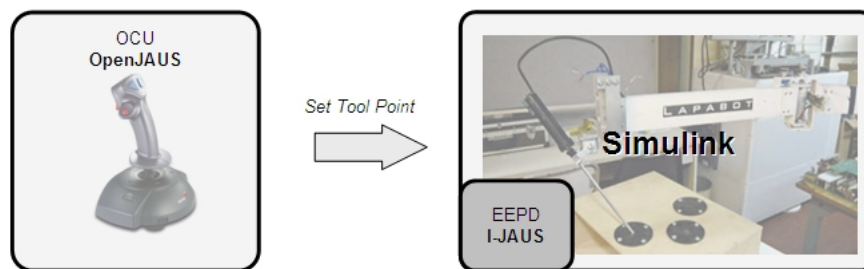


Figura 8.11: Demostración haciendo uso del Robot Laparoscópico.

Esta demostración representa el primer paso hacia la integración del Robot Laparoscópico a los proyectos de diversas partes del mundo que soporten el estándar JAUS.

8.3. Modelo PUMA 762 3D

Este escenario tiene por propósito demostrar que es posible adaptar modelos de simulación de brazos robóticos al estándar, haciendo uso de I-JAUS.

Se hace uso de un modelo 3D del brazo robótico industrial PUMA 762, para MATLAB. Con pequeñas modificaciones este proyecto fue transformado en un componente JAUS EEPD bajo I-JAUS.

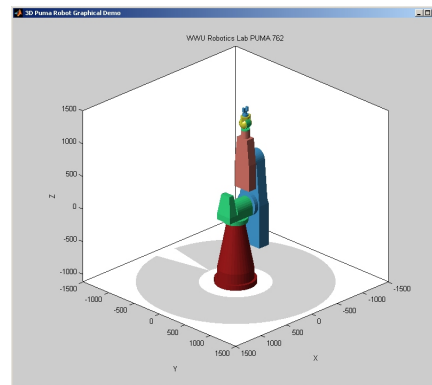


Figura 8.12: Modelo 3D del robot industrial PUMA 762.

8.3.1. Utilizando GUI (LabVIEW)

Este modelo puede ser controlado mediante un OCU implementado en LabVIEW, como es mostrado en la Figura 8.13.

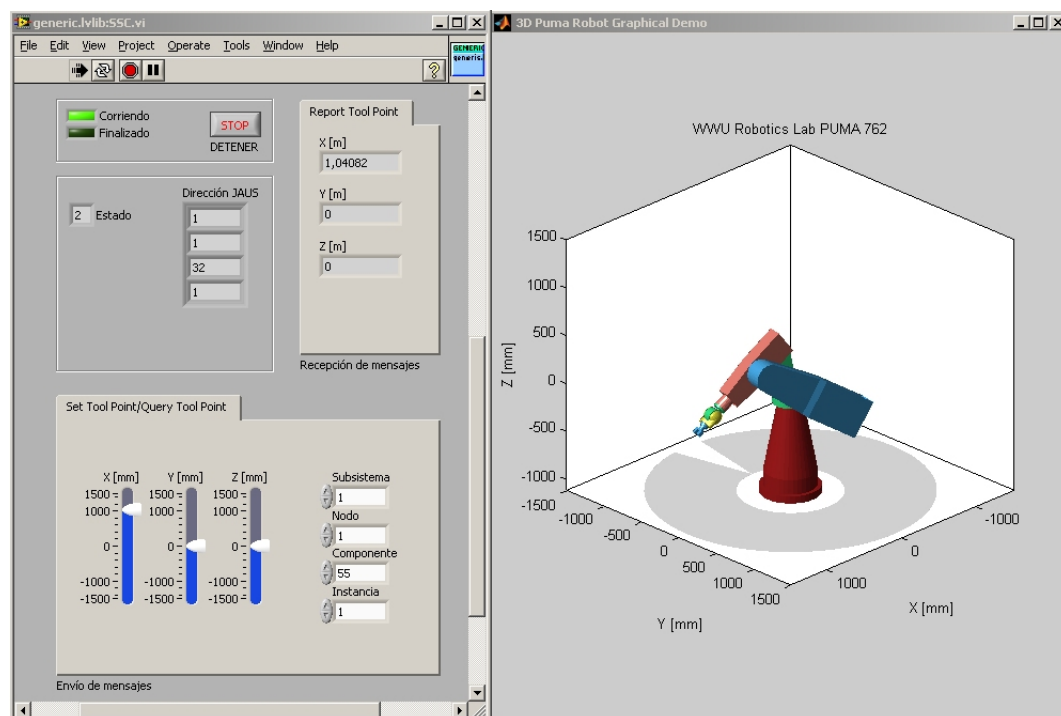


Figura 8.13: Componentes de LabVIEW (panel SSC) y MATLAB (PUMA3D en EEPD) interactuando.

9

Conclusiones

El estándar JAUS establece bases para lograr interoperabilidad entre sus componentes. OpenJAUS hace concreto este atributo, pero para componentes implementados en el lenguaje C. Lo novedoso del presente trabajo consiste en ampliar las alternativas de implementación de un componente (en entornos como MATLAB, Simulink y LabVIEW), y así favorecer las etapas tempranas de desarrollo de robots.

Es importante que el lector entienda que la herramienta desarrollada es un mero medio para llegar a un fin: lograr interoperabilidad entre proyectos de implementación variada. Sin embargo, es necesario considerar algunos puntos adicionales a los efectos de lograr este cometido. A continuación se presentan conclusiones y recomendaciones que el lector debe tener en consideración para ello.

Consideraciones del estándar

En la evaluación desarrollada de los estándares, JAUS se mostró el más propicio para lograr los objetivos del proyecto. Esto se debe a que define un conjunto de componentes que permite construir el software de un brazo robótico y darle al mismo un alto grado de reusabilidad.

Sin embargo, se considera que es necesario un estudio profundo del estándar para diseñar un sistema basado en JAUS adecuado. Esto significa tener en consideración los siguientes puntos:

- **Interoperabilidad limitada.** Es necesario distinguir entre un componente bien diseñado y un cuasi-componente, cuya implementación posee dependencia o presunciones sobre la implementación de otros componentes. JAUS provee los mecanismos suficientes para lograr que *las implementaciones de los componentes sean completamente desligadas entre sí.*

- **Jerarquía de componentes.** Así como un componente constituye un bloque dentro de un subsistema (un robot autónomo), este último constituye un bloque dentro de un sistema. Un robot bien implementando tiene un alto grado de integrabilidad con otros, pues así ocurre a nivel de sus componentes. El desarrollador deberá tener en claro estas cuestiones para definir los límites del subsistema.
- **Evitar solapamiento en el desarrollo.** El uso de componentes definidos por JAUS conduce la forma de trabajo de un grupo y delimita la labor de cada uno de sus integrantes, facilitando mediante ello la cooperación. Todo esto optimiza la etapa de desarrollo.

Tener en cuenta estas consideraciones hace la diferencia entre las formas de uso correcta e incorrecta de la norma.

Entornos de desarrollo

En el ambiente de desarrollo en el que se ha trabajado se observaron numerosos proyectos de bioingeniería y robótica, implementados en LabVIEW los primeros y en MATLAB y Simulink los últimos. Dichas herramientas poseen mucha aceptación en laboratorios donde se realizan este tipo de implementaciones. Las entrevistas realizadas al grupo *cliente* concuerdan con esta afirmación, y justifican la elección de dichos entornos para este trabajo.

Durante las sucesivas etapas de prueba de I-JAUS se obtuvo una conclusión interesante: el tiempo de desarrollo de una misma prueba menguaba notablemente utilizando MATLAB en lugar de C. El intérprete de comandos del primero permitía realizar cada *test* de forma dinámica, y luego almacenar los comandos utilizados en *scripts*, obteniendo así pruebas reutilizables con un mínimo esfuerzo. Un hecho deja en evidencia las ventajas de esta forma de trabajo: sólo para el envío y recepción de mensajes de los doce servicios implementados se tienen más de cien llamadas, y absolutamente todas estas fueron puestas a prueba en menos de una semana. Se puede afirmar entonces que, además de ser el entorno más utilizado en el ambiente estudiado, brinda facilidades que justifican aún más su elección.

Pruebas de concepto

I-JAUS da la posibilidad de integrar proyectos existentes (sobre las citadas Herramientas de Desarrollo) con el estándar JAUS, en un tiempo reducido: las demostraciones que involucraron mando por voz, por seguimiento ocular, por ratón, por joystick, mando por interfaz gráfica, uso del PICOLEBOT, uso del Brazo Laparoscópico y uso del modelo 3D del PUMA 762 fueron realizadas en un tiempo inferior a dos semanas. Esta agilidad es consecuencia de dos puntos:

- La facilidad que presenta I-JAUS para ser integrado a un proyecto existente.
- El medio facilitado por I-JAUS, es decir el estándar JAUS, para que los proyectos se comuniquen entre sí.

Cada una de las demostraciones constituye una prueba de concepto que apunta a los objetivos del GRSI, y que demuestra la utilidad de I-JAUS.

Impacto

I-JAUS posee beneficios inmediatos para sus usuarios. De los observados, los más importantes son:

- División de un proyecto en bloques. Esto moviliza al grupo a finalizar componentes que han sido rigurosamente probados. Así es posible abstraerse de la implementación de partes consideradas fiables, para seguir experimentando.
- Facilitación del aprendizaje de JAUS. Esto ocurre gracias a que MATLAB permite uso interactivo de un componente JAUS (desarrollado en I-JAUS) con su intérprete de comandos.
- Aumento de proyectos tenidos en cuenta. Un componente puede hacer uso de cualquier implementación (ya no limitada a sólo C), basta que sea hecha en una Herramienta de Desarrollo soportada. LabVIEW y MATLAB son cunas de desarrollos innovadores, ahora integrables a cualquier proyecto del GRSI.

Futuro

Una de las posibilidades en vista es seguir de cerca la evolución de JAUS. Los cambios del estándar son llevados adelante de la mano de un importante organismo como es SAE. Aunque I-JAUS está actualmente basado en la versión 3.3 de los documentos *Reference Architecture*, su actualización es posible a efectos de acoplarse a las mejoras del estándar.

En vista de que el proyecto ha cumplido con todos los objetivos planteados, toma valor el hecho de extender las funcionalidades de I-JAUS, y así soportar el resto de los grupos de componentes del estándar. Esto eliminaría las limitaciones que posee actualmente la herramienta, y permitiría su uso para implementar sistemas robóticos basados en plataformas móviles, utilización de cámaras, etc.

Además de I-JAUS, este trabajo deja un conjunto de componentes parcialmente implementados. Esto abre las puertas a cada usuario para que mejore dicha colección, y por qué no, cree nuevos componentes y los comparta con la comunidad.

Glosario

CBSE – Component Based Software Engineering Aquella rama de la Ingeniería del Software que hace foco en los componentes para el desarrollo de sistemas de software. [32](#), [34](#)

Componente Es la entidad JAUS que provee una única capacidad funcional al sistema no tripulado. Un componente JAUS reside enteramente dentro de un nodo JAUS. [39](#)

DOF – Degrees Of Freedom Hace referencia a los grados de libertad de un brazo robótico. Cuando más DOF presente una implementación, más articulaciones poseerá. [56](#)

EEDPD Hace referencia al componente JAUS *End-Effector Discrete Pose Driver*, o Controlador de Pose Discreta del Extremo Efecto. [43](#)

EEPD Hace referencia al componente JAUS *End-Effector Pose Driver*, o Controlador de Pose del Extremo Efecto. [43](#)

EEVSD Hace referencia al componente JAUS *End-Effector Velocity State Driver*, o Controlador de Estado de Velocidad del Extremo Efecto. [43](#)

GUI *Graphical User Interface*, o Interfaz Gráfica de Usuario. [103](#)

HDR – Herramienta de Desarrollo en el campo de la Robótica Hace referencia a aquellas herramientas que brindan ventajas especiales a los desarrollos de software de robótica estudiados en este proyecto. Puede pensarse tanto en MATLAB como en LabVIEW, así como en Simulink. [29](#), [30](#)

Instancia de componente JAUS La multiplicidad de copias y la redundancia de componentes JAUS son provistas por las instancias de componentes. Todos los componentes son identificables por medio de sus identificadores de subsistema, nodo, componente e instancia. [39](#)

JFTS Hace referencia al componente JAUS *Joint Force/Torque Sensor*, o Sensor de Fuerza/Torque de Articulación. [43](#)

JMD Hace referencia al componente JAUS *Joint Move Driver*, o Controlador de Movimiento de Articulación. [43](#)

JPD Hace referencia al componente JAUS *Joint Positions Driver*, o Controlador de Posiciones de Articulación. [43](#), [64](#)

JPS Hace referencia al componente JAUS *Joint Position Sensor*, o Sensor de Posición de Articulación. [42](#), [64](#)

JVD Hace referencia al componente JAUS *Joint Velocities Driver*, o Controlador de Velocidades de Articulación. [43](#)

JVS Hace referencia al componente JAUS *Joint Velocity Sensor*, o Sensor de Velocidad de Articulación. [42](#)

JWG - Jaus Working Group Es el grupo de trabajo que lleva adelante el proyecto JAUS.. [45](#)

Nivel de compatibilidad/interoperabilidad JAUS Hace referencia al grado de reusabilidad que permite un diseño basado en JAUS. En [[Faruque, 2006](#)] y [[Rowe, n.d.](#)] se plantean 3 niveles. *Level I Interoperability* implica interoperabilidad a nivel de subsistemas. *Level II Interoperability* implica interoperabilidad a nivel de nodos. *Level III Interoperability* implica interoperabilidad a nivel de componentes. [55](#)

Nodo Define una capacidad de procesamiento diferenciable dentro de un subsistema. Un nodo posee un conjunto de funciones coherentes, y debe proveer un componente *Node manager* para administrar el flujo y controlar el tráfico de mensajes JAUS. [39](#)

OCU - Operator Control Unit Unidad de control mediante la cual un operador controla el sistema/subsistema JAUS. [39](#), [40](#), [103](#)

PM Hace referencia al componente JAUS *Primitive Manipulator*, o Manipulador Primitivo. [42](#)

Scrum Es una metodología ágil de desarrollo de software que toma su nombre y principios de los estudios realizados sobre nuevas prácticas de producción. [11](#)

SDK Siglas de *Software Development Kit*. Representa una herramienta que facilita un implementación específica, JAUS para el caso. Ejemplos son OpenJAUS, JAUS++, etc.. [21](#)

Servicio Representa un conjunto de mensajes de entrada y salida que son soportados por un componente. Constituye adicionalmente un mecanismo para permitir a otros componentes o usuarios determinar qué mensajes se asocian a qué componentes. Un diccionario de servicios JAUS puede ser encontrado en RA V3.3. [45](#), [50](#), [64](#), [70](#)

Sistema Es un conjunto lógico de subsistemas. Incluye todos los subsistemas de interfaz humana, y subsistemas no tripulados comunes en aplicaciones robóticas. [39](#)

Sprint Consiste en una iteración de desarrollo bajo Scrum. Es el núcleo central que proporciona la base de desarrollo iterativo e incremental para dicha metodología. [11](#)

SSC Hace referencia al componente JAUS *Subsystem Commander*, o Administrador de Subsistema. [40](#), [103](#)

Subsistema Es la entidad que realiza una o más funciones del sistema no tripulado. Un subsistema debe proveer uno o más comandos de comunicaciones, y capacidad de control. Un subsistema móvil debe ejecutar comandos de movilidad como una única unidad, y mantener un centro de gravedad definido, relativo a todas las articulaciones y cargas. [39](#)

Bibliografía

- [Sommerville, 2005] Sommerville, I. (2005). *Ingeniería del software*. Madrid: Addison-Wesley.
- [Kniberg, 2007] Kniberg, H. (2007). *Scrum y XP desde las trincheras*. EEUU: C4Media.
- [Palacio, 2006] Palacio, J. (2006). *El modelo Scrum*. Navegapolis.net.
http://www.navegapolis.net/files/s/NST-010_01.pdf
- [UPalermo, n.d.] *Las citas bibliográficas y otras normas de estilo en los trabajos de investigación, las tesis y trabajos de integración final*.
https://wwws.palermo.edu/homer/Intranet/biblioteca/Archivos/biblioteca_guia_de_citas.doc
- [Rowe, n.d.] Rowe, S. Wagner, C. (n.d.). *An Introduction to the Joint Architecture for Unmanned Systems*. EEUU: Cybernet Systems Corporation.
- [DM-3.1, 2004] *The Joint Architecture for Unmanned Systems. Domain Model. Volume 1. Version 3.1* (2004). EEUU: Jaus Working Group.
- [RA-3.3 P1, 2007] *The Joint Architecture for Unmanned Systems. Reference Architecture Specification. Volume II, Part 1. Architecture Framework. Version 3.3.* (2007). EEUU: Jaus Working Group.
- [RA-3.3 P2, 2007] *The Joint Architecture for Unmanned Systems. Reference Architecture Specification. Volume II, Part 2. Message Definition. Version 3.3.* (2007). EEUU: Jaus Working Group.
- [RA-3.3 P3, 2007] *The Joint Architecture for Unmanned Systems. Reference Architecture Specification. Volume II, Part 3. Message Set. Version 3.3.* (2007). EEUU: Jaus Working Group.

- [JAUSWG, 2009] *JAUS Working Group*. Julio, 11, 2009.
<http://www.jauswg.org/>
No disponible desde *Septiembre del 2009*.
- [OpenJAUS, 2009] *OpenJAUS*. Mayo, 30, 2009.
<http://openjaus.com/trac/openjaus>
- [OJDC, 2009] *OpenJAUS Developer's Conference*. Junio, 10, 2009.
http://www.openjaus.com/documents/OpenJAUS_DevCon_July08.pdf
- [Faruque, 2006] Faruque, R. (2006). *A JAUS Toolkit for LabVIEW, and a Series of Implementation Case Studies with Recommendations to the SAE AS-4 Standards Committee*. EEUU: Virginia Polytechnic Institute and State University.
- [RE2, 2009] *RE²*. Julio, 11, 2009,
<http://www.resquared.com/JAUS-SDK-Features-Benefits.html>
- [JAUS++, 2009] *Jaus++*. Julio, 11, 2009,
<http://active-ist.sourceforge.net/>
- [SOSA, 2004] Sosa, O. (2004). *Design and Implementation of a Modular Manipulator Architecture*. EEUU: University of Florida.
- [Fuentes, n.d.] Fuentes, L. Troya, M. Vallecillo A. (n.d.). *Lección 1. Desarrollo de Software Basado en Componentes*. Málaga: Universidad de Málaga. Disponible en:
<http://www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf>
- [Szyperski, 1998] Szyperski, C. (1998). *Component Software. Beyond Object-Oriented Programming*. Addison-Wesley.
- [TIOBE, 2009] *TIOBE Programming Community Index for July 2009*. Julio, 5, 2009.
<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
- [TIOBE2, 2009] *TIOBE Software: How to use Coding Standards*. Mayo, 5, 2009,
<http://www.tiobe.com/index.php/content/paperinfo/HowToUseCodingStandards.html>
- [Cannon, 1997] Cannon, L. Elliott, R. Kirchhoff, L. Miller, J. Milner, J. Mitze, R. Schan, E. Whittington, N. Spencer, H. Keppel, D. Brader, M. (1997). *Recommended C Style and Coding Standards* (Rev. 6.0). EEUU: AT&T's Indian Hill labs.
<http://www.psgd.org/paul/docs/cstyle/cstyle.htm>

Apéndices



Manual de Referencia de I-JAUS

B

Ejemplos de I-JAUS
