Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Sistema Robótico Distribuido CORBA: Caso de Estudio Robot Motoman UP6

por

Ing. Josafat Miguel Mata Hernández

Tesis

Presentada al Programa de Graduados en Ingeniería y Arquitectura

como requisito parcial para obtener el grado académico de

Maestro en Ciencias

especialidad en

Sistemas de Manufactura

Monterrey, N.L., Diciembre de 2005

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

División de Ingeniería y Arquitectura

Programa de Graduados

Los miembros del comité de tesis recomendamos que la presente tesis de Josafat Miguel Mata Hernández sea aceptada como requisito parcial para obtener el grado académico de **Maestro en Ciencias**, especialidad en:

Sistemas de Manufactura

Comité de Tesis:

M.C. Federico Guedea Elizalde
Asesor de la tesis

M.C. Gerardo A. Vallejo Moreno
Sinodal

Dr. Rubén Morales Menéndez
Sinodal

Dr. Federico Viramontes Brown
Director del Programa de Graduados

Diciembre de 2005

A mis padres,

Sr. Juan Agustín Mata E. y Sra. Guadalupe Hernández de Mata.

Por ser un ejemplo de amor incondicional, dedicación y trabajo arduo.

A mi hermana,

Gabriela del R. Mata Hernández.

Por todos tus consejos, motivación y ayuda.

Reconocimientos

Al M.C. Federico Guedea por toda su confianza, apoyo y paciencia. Por compartir conmigo sus conocimientos, ideas y aún su tiempo libre.

M.C. Gerardo Vallejo por su ayuda desde el inicio de esta investigación, tanto en el apoyo técnico, como en el uso de los recursos tecnológicos necesarios.

Dr. Ruben Morales por sus valiosos comentarios y retroalimentación en la perspectiva científica de este trabajo de investigación.

M.C. Ricardo Jiménez por compartir conmigo sus ideas, por su tiempo y guía en la búsqueda de un tema de tesis que llenara mis expectativas.

M.C. Miguel de Jesus Ramírez y al M.C. Manuel Cabrera, responsables del Laboratorio de Mecatrónica, asi como a Saúl, Gilberto, David y Miriam asistentes de investigación del mismo, por todo el soporte.

A todos los integrantes de IDTec Automatización.

A los compañeros y nuevos amigos a quienes tuve la oportunidad de conocer y compartir buenos momentos.

Josafat Miguel Mata Hernández

Instituto Tecnológico y de Estudios Superiores de Monterrey
Diciembre 2005

Resumen

Se presenta el caso de un robot industrial operado remotamente bajo una arquitectura distribuida. Este se basa en el *middleware* estándar CORBA para controlar un brazo manipulador MOTOMAN UP6 de seis grados de libertad. Este robot está basado en el controlador XRC2001. La propuesta se puede extender a cualquier otro controlador Yaskawa (e.g. ERC, ERCII, MRC, MRCII) sin grandes cambios en la configuración final propuesta. Igualmente el sistema puede adaptarse fácilmente para el control de movimientos de otros brazos manipuladores de mayores capacidades, como podrían ser un UP20, UP50, entre otros.

La idea principal es definir un grupo genérico de interfases IDL que puedan ser usados para integrar librerías comerciales que oculten lo intrincado de los componentes de bajo nivel. Con esto los tiempos de desarrollo en las aplicaciones y la interoperabilidad entre equipos independientemente del proveedor de los componentes robóticos se ven mejorados.

El reto final en el caso de estudio es crear una aplicación remota distribuida cliente-servidor, el cual facilite la integración de uno o varios brazos manipuladores basados en los controladores anteriormente mencionados o de otros fabricantes, independientemente del sistema computacional o las plataformas computacionales utilizadas en el desarrollo de las aplicaciones.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

INTRODUCCIÓN

El número de robots industriales instalados alrededor del mundo está alcanzando la cifra de 1,000,000 con casi la mitad de este número instalados en Japón, un 17 por ciento instalado en Estados Unidos de América y el restante repartido entre la Unión Europea y el resto del mundo. En la actualidad un aproximado del 50 por ciento de estos robots son utilizados en aplicaciones del área automotriz y el restante en diversas ramas como fábricas manufactureras, laboratorios, plantas de energía, entre otras industrias [43].

El uso de robots industriales corresponde a aplicaciones de manejo de materiales, soldadura de punto y arco, dosificado, ensamble, inspección, así como otras aplicaciones específicas. Ver figura 1.1 para una descripción más detallada.

En el país, actualmente se encuentran instalados aproximadamente 4 mil 500 robots industriales. De estos el 40 por ciento se destinan al armado de automóviles y el 45 por ciento a la industria de auto partes. El restante 15 por ciento se divide de manera diversa entre la industria electrónica, de alimentos, vidriera, farmacéutica, etc., las cuales todavía no adoptan plenamente el uso extensivo de robots [14].

En México, invertir en un proyecto de robótica es normalmente más alto entre un 27 y un 170 por ciento, con respecto a la mano de obra [2]. La mayor parte de las implementaciones robóticas corresponden a equipo importado o empresas de inversión extranjera que colocan este tipo de sistemas para el soporte en sus líneas de producción. Por lo que se ha impedido el desarrollo y transferencia de investigación y tecnología nacional en la industrial en general. Surge de aquí la necesidad de desarrollar nuevos

paradigmas en aplicaciones robóticas que busquen una mayor flexibilidad y rentabilidad.

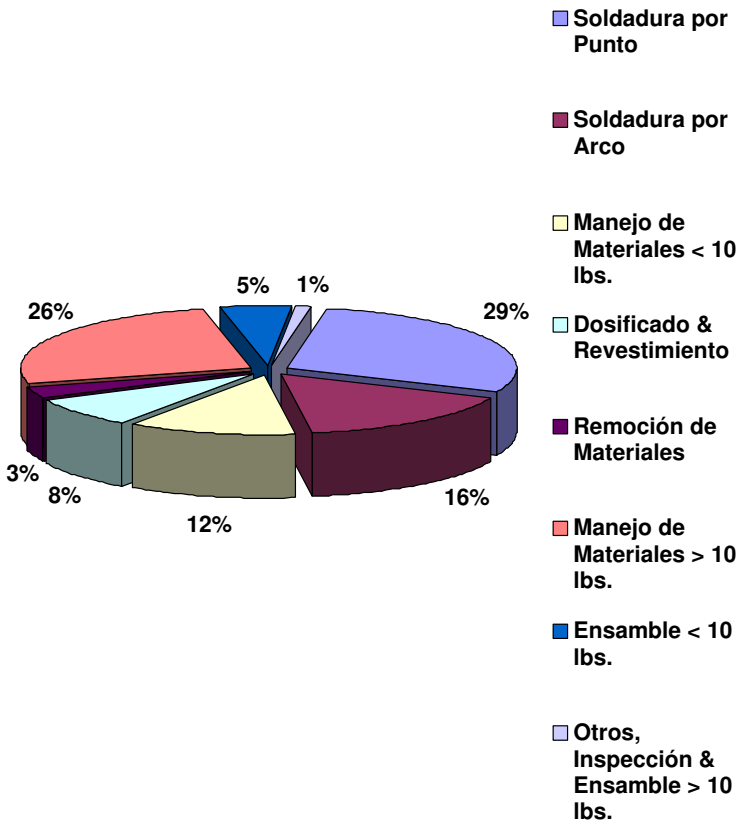


Figura 1.1: Aplicaciones de robots industriales a nivel mundial en el año 2004 [5].

1.1. Descripción del problema

Los sistemas distribuidos han sido introducidos en el área de manufactura debido a su flexibilidad, confiabilidad y su mejor proporción en precio/desempeño. Sin embargo, un problema inherente es la comunicación entre elementos heterogéneos que los componen. Esto debido a la diversidad de proveedores y soluciones características tales como robots con arquitecturas de control propietarios. Representa un reto integrar este tipo de sistemas con otros componentes, sistemas de visión o la integración heterogénea

entre brazos manipuladores robóticos [4].

El implementar sistemas robóticos distribuidos no es una tarea fácil, debido a la diversidad de áreas de especialización interrelacionadas. Esto incrementa el grado de complejidad, conocimiento requerido y los tiempos de desarrollo e implementación de este tipo de sistemas.

En los últimos años, han surgido diversidad de paradigmas y arquitecturas basadas en cómputo distribuido que buscan resolver los problemas inherentes de interoperabilidad entre dispositivos con arquitecturas cerradas (e.g. CORBA de OMG, Java de SUN, .NET de Microsoft y FIPA [1], por mencionar los más recientes). Igualmente el uso extensivo de Internet en todos los ámbitos, representa actualmente un desafío en el área de investigación y desarrollo para encontrar mejores y más eficientes técnicas, modelos, metodologías, estándares, y aún soluciones comerciales que permitan reducir las islas de automatización que se producen al utilizar productos o soluciones propietarias o tecnologías que no permiten una adecuada interoperabilidad entre componentes *hardware* y/o *software* distintos.

En realidad aún se requiere mucho esfuerzo en la investigación y desarrollo de estos sistemas. Por lo que este trabajo se concentra en proponer una solución particular en el área de sistemas róbóticos distribuidos, partiendo de una estructura para el control de un robot previamente introducida en [20]. La estructura de control robótico distribuido, desarrollado por Guedea et al., mediante un convenio entre el ITESM, la Universidad de Waterloo en Canadá y el CONACYT permite crear, integrar y mejorar sistemas distribuidos, combinando herramientas de visión computacional, sistemas de planeación y computación distribuida mediante el uso del *middleware* estándar CORBA (*Common Object Request Broker Architecture*).

La finalidad es crear componentes o módulos de *software* de programación orientado a objetos, que permitan la conectividad y comunicación de los componentes de una manera más fácil [20].

En este contexto, algunas de las conclusiones y posibles mejoras que señalan en sus trabajos de investigación en robótica distribuida por parte de Guedea et al., es la

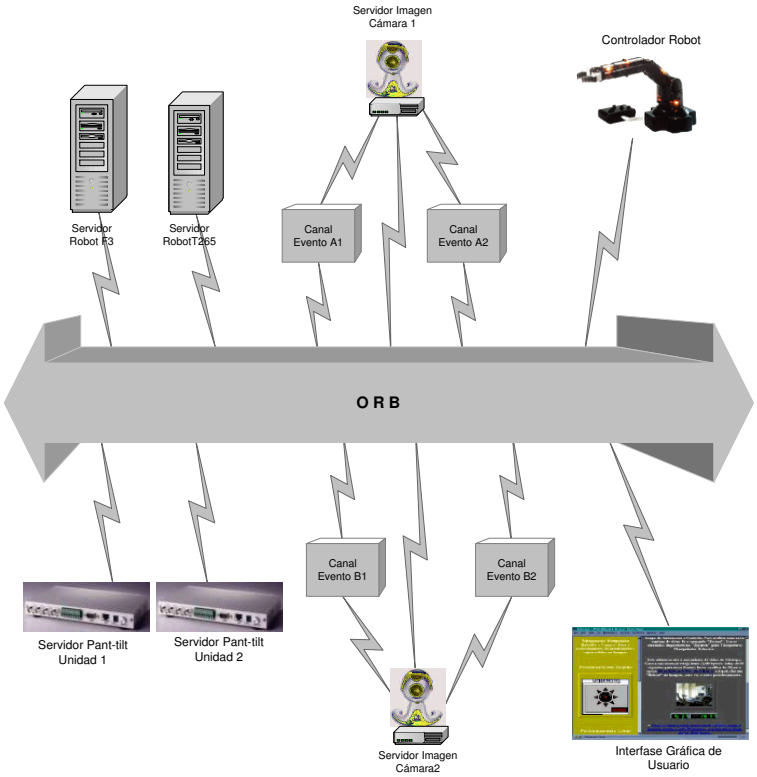


Figura 1.2: Componentes cliente - servidor para un sistema distribuido basado en CORBA.

posibilidad de integrar nuevos componentes, tales como brazos manipuladores robóticos en la mejora del trabajo colaborativo [29].

Un ejemplo de esta solución puede ser visualizada en la figura 1.2, donde se presenta la estructura de sistema distribuido basado en CORBA previamente propuesto. Así como la oportunidad de integrar de una manera menos compleja, brazos robóticos u otros tipos de componentes (sistemas de visión, sensores táctiles, etc.), en una arquitectura abierta.

Desarrollar un componente basado en programación orientado a objetos (definido como componente *envolvente* por Guedea et al.,) del controlador del robot para el código intérprete, es el elemento que requiere un mayor esfuerzo al implementarlo. Este componente debe ser capaz de llevar a cabo la transformación e interpretación de todos los datos. Por lo que estos pasos requieren igualar los tipos y las estructuras de datos de una interfase IDL (*Interface Definition Language*) y la implementación tipo objeto

del *hardware/software* del robot y viceversa [20].

Lo anterior, es una de las problemáticas principales a resolver y forma parte de la propuesta de solución y aportación en este trabajo de investigación. Generar este componente, establecerá un modo de comunicación, cooperación e integración de robots industriales marca Motoman. Extendiendo la aplicación de la metodología propuesta más allá de una sola marca de robot (solución actual), con una gran flexibilidad para su utilización en diversos ambientes de manufactura debido a su arquitectura abierta.

1.1.1. Justificación

En la actualidad aún cuando un gran porcentaje de las aplicaciones industriales existentes son mediante el uso de un solo robot. Diversas aplicaciones en manufactura requieren el trabajo colaborativo de dos o más brazos robóticos y sistemas periféricos externos. Las cuales permiten reducir tiempos de ciclo, incrementar la flexibilidad y la calidad final de un producto.

Un ejemplo del uso de este tipo de sistemas es en celdas de manufactura modulares donde el trabajo colaborativo entre brazos robóticos múltiples permite que el posicionamiento y manejo de materiales sea más flexible. Típicamente un robot se utiliza para sostener la parte, mientras que los otros se encargan de procesar la tarea en cuestión. Debido a la naturaleza del trabajo colaborativo permiten procesar una parte dentro de la misma estación o celda, con lo que se acortan tiempos de ciclo por la no transferencia del material entre estaciones. El hecho que cada proceso es controlado por sistemas de sensores con retroalimentación y procesados a través de controladores de robots distribuidos; la inspección automática y la conformidad con estándares de calidad asegura un costo adicional virtualmente nulo. Como un beneficio adicional, los robots en tales celdas pueden ser equipados con cambiadores automáticos de herramientas, con lo que se incrementa su flexibilidad y reusabilidad. Otros aplicaciones típicas en manufactura con brazos robóticos colaborativos pueden encontrarse en [46] y [10].

Por otro lado, adelantarse a los requerimientos de producción en la actualidad simplemente no es tan factible, por lo cual la integración e implementación constante

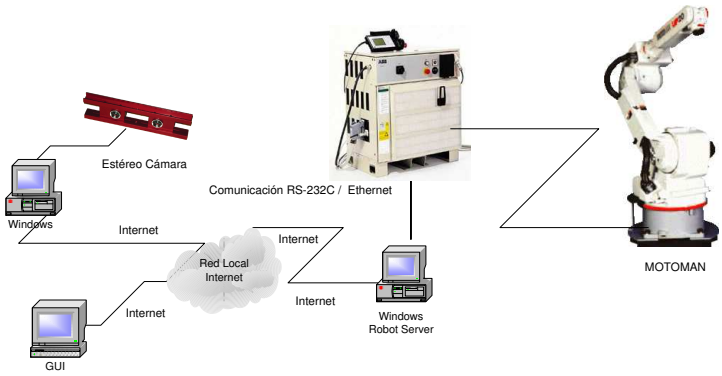


Figura 1.3: Configuración propuesta para sistema robótico distribuido basado en CORBA.

de nuevos equipos y sistemas es algo común hoy día. El deshacerse de equipo costoso tal como los brazos manipuladores robóticos por cambios en la tecnología o en el proveedor, conlleva buscar nuevos métodos para integrar y adaptarlos en una arquitectura abierta, que permita la comunicación entre controladores y periféricos propietarios de una manera transparente.

En manufactura, integrar brazos controladores robóticos, así como otro tipos de dispositivos tales como CNC, PLC, DCS, etc., donde se evita el uso de aplicaciones particulares con *hardware* y *software* propietarios, bajo ambientes restringidos o mediante el uso de redes industriales específicas, es una tendencia sin regresión. El uso de arquitecturas cerradas o limitadas no son eficientes, por lo que la investigación y aplicación de sistemas distribuidos en arquitecturas abiertas es un paradigma actual y con un futuro todavía más prominente. La integración y desarrollo de metodologías para la interconexión de componentes que mejoren la flexibilidad del sistema completo, tal como el caso de estudio que se propone, es un paso próximo que permite aprovechar los recursos disponibles y establecer pautas en la integración de modelos y marcas de robots heterogéneos mediante un solo sistema (ejemplos sean Motoman, ABB, Fanuc, Kuka, etc). Aunado al uso de tecnologías orientadas a objetos, programación basada en componentes comerciales (COTS), utilización de un *middleware* estándar ampliamente consolidado, soportado por asociaciones, compañías internacionales y en continuo

desarrollo tal como CORBA.

Las razones por la cual utilizar CORBA como *middleware* base, se desarrollan en el capítulo 2. En el que se presentan ventajas y desventajas de este con respecto a otros paradigmas de cómputo distribuido.

Un punto importante a mencionar es la factibilidad y bajo costo en el desarrollo del trabajo propuesto, debido a la disponibilidad y aprovechamiento de los recursos existentes dentro del instituto, tales como celdas de manufacturas didácticas, robots CRS y Motoman entre otros, sistemas de visión (DVT y Videre Design), así como licencias de *software* tales como Visual Studio (C++), Rational Rose (UML), ORBacus (ORB CORBA), entre otros.

Objetivos

Desde un principio, se espera como resultado de este trabajo de investigación, el desarrollar una aplicación base lo suficientemente operativa como para ser utilizada en un futuro en aplicaciones industriales y/o comerciales.

De manera más puntual se definen los siguientes objetivos:

- Establecer una documentación completa del estado del arte de los sistemas robóticos distribuidos en aplicaciones de manufactura.
- Desarrollar un comparativo de las diversas tecnologías existentes utilizados en la integración de este tipo de sistemas heterogéneos.
- Proponer una metodología de desarrollo de componentes *envolventes* para brazos robóticos industriales marca Motoman.
- Modelar el sistema de *software* desarrollado usando herramientas tales como UML.
- Implementar un sistema robótico distribuido basado en *middleware* estándar CORBA, que permita la integración de brazos manipuladores Motoman (controladores de robot XRC, MRC, MRCII, ERC y ERCII).

- Desarrollar un cliente CORBA (Interfaz gráfica de usuario, GUI), que permita la comunicación y operación remota entre uno y/o más robots de marcas propietarias distintas.

- Desarrollar un servidor de robot CORBA que establezca la comunicación y control adecuado de uno o más brazos manipuladores robóticos Motoman.

1.1.2. Hipótesis

1. El desarrollo de componentes de comunicación de alto nivel mediante programación orientada a objetos, y el uso de modelos de arquitectura distribuida tales como cliente/servidor o basado en *brokers*, sin la necesidad de programación de bajo nivel (controlador del robot) genera una mayor eficiencia en el resultado final.

2. El desarrollo de componentes *envolventes* basado en tecnologías orientadas a objetos, permite implementar de una manera más simple y con un tiempo de desarrollo menor, otras marcas de brazos manipuladores robóticos de proveedores diversos.

3. La modelación UML del sistema propuesto permite un mejor entendimiento de la arquitectura de *software* desarrollada, además de capturar los requerimientos de una manera metódica, y entender las interrelaciones del sistema con respecto a otras aproximaciones.

1.1.3. Alcances y Limitaciones

Los alcances y limitaciones tienen que ver con aspectos relacionados con el tiempo requerido en el desarrollo de las aplicaciones. Los conocimientos y experiencia necesarios en distintos campos de la ingeniería como son la operación, control y teleoperación de brazos manipuladores robóticos. Comunicaciones basadas en RS-232C y Ethernet, modelos de arquitecturas distribuidas, *middleware* CORBA basado en ORB comerciales, programación C++ basada en MFC, modelación UML de sistemas distribuidos,

entre otros.

Estas áreas de estudio, por ser tan diversas y extensas por sí mismas permiten establecer varias aproximaciones de estudio para cada una de ellas. Sin embargo este trabajo de investigación se concentra en establecer un medio de comunicación entre el controlador del robot y uno o varios clientes, así como en tareas de control de movimientos del robot mediante operación remota. Estos puntos serán debidamente tratados y desarrollados en los capítulos siguientes.

Aspectos igualmente importantes y críticos en aplicaciones con brazos robóticos múltiples tales como la planeación de trayectorias, el balanceo de las operaciones, la programación simultánea, las colisiones, entre otros [46]. Más los desafíos que presenta la telerobótica mediante el uso de Internet, como son las limitantes en el ancho de banda, retardos inciertos en el tiempo, la perdida de datos y la seguridad en la transmisión de los mismos. No son profundamente analizados. Para esto existen diversos métodos que se concentran en reducir estos problemas. Muchos de ellos propuestos por expertos y que pueden encontrarse en la literatura tales como en [37], [26], [3] y [32] por mencionar algunos. Sin embargo en el desarrollo del caso de estudio, se documenta y presentan los resultados y observaciones generados en la etapa de experimentación del proyecto para estas preocupaciones actuales.

Con respecto al servidor del robot a desarrollar. Este se encuentra limitado por las librerías comerciales de MotoCom SDK, las cuales están desarrolladas para la plataforma Windows 2000, NT o XP. Por lo que el servidor de robot para la comunicación con el controlador requiere que éste sea implementado bajo esta plataforma.

Una limitación más, es el requerimiento de una llave electrónica proporcionada por la compañía Motoman Yaskawa, que se instala en el puerto paralelo de la computadora que se utilice como servidor de robot. En caso de establecer más de un servidor de robot para varios controladores XRC, MRC, etc. es necesario instalar una llave por cada servidor robot - controlador Motoman.

La modelación UML del sistema desarrollado, está limitado a diagramas de clase. Estos diagramas son los más recurridos por los desarrolladores de *software*, ya que

permiten visualizar de una manera estática los requerimientos e interacción entre los objetos que componen el sistema. Sin embargo tal como se trata en el capítulo 3, modelar un sistema distribuido es una tarea compleja que en ocasiones aún con las herramientas de modelación actuales, no producen una solución del todo eficiente y única.

1.2. Estructura de los capítulos

Los seis capítulos que componen este trabajo de investigación, incluyendo este capítulo introductorio están organizados de la siguiente manera:

Capítulo 2 Establece una descripción del estado del arte de los sistemas robóticos distribuidos (DRS) con un énfasis en aplicaciones de Manufactura.

Capítulo 3 Presenta el marco teórico del *middleware* estándar CORBA.

Capítulo 4 Modelación UML del sistema propuesto cliente - servidor CORBA para un brazo robótico Motoman UP6.

Capítulo 5 Presenta la metodología de desarrollo de componentes *envolventes* y ejemplifica un caso de estudio basado en la propuesta.

Capítulo 6 Concluye el trabajo realizando un análisis de los resultados, generando las conclusiones generales. Se dan recomendaciones para futuras investigaciones.

Capítulo 2

SISTEMAS ROBÓTICOS DISTRIBUIDOS

Debido al continuo desarrollo en operaciones de automatización y la necesidad de compartir recursos, se han desarrollado múltiples soluciones para conectar sistemas robóticos a redes computacionales. Paradójicamente uno de los principales problemas dentro de las redes robóticas es que muchas veces no pueden comunicarse entre sí, aún cuando estén conectadas a la misma red. Esto debido a que las diferencias entre protocolos de comunicación (estructura de datos y/o el número de puertos para comunicación) son íntrinsecos a ellos.

2.1. Antecedentes

2.1.1. Conceptos de Sistemas Distribuidos

Un sistema distribuido (ver fig. 2.1) es una colección de computadoras independientes que para el usuario final se presenta como un solo sistema coherente. Está definición tiene dos aspectos, la primera tiene que ver con *hardware*: máquinas autónomas. La segunda con *software*: el usuario piensa que está ante un solo sistema. Una característica importante son las diferentes computadoras que forman el sistema y la forma que se comunican es transparente para el usuario. Otra característica importante es que las aplicaciones y los usuarios pueden interactuar con un sistema distribuido de una manera consistente y uniforme, independientemente de dónde y cuándo la interacción se lleve a cabo. Los sistemas distribuidos debieran ser fáciles de escalar y expandir [58].

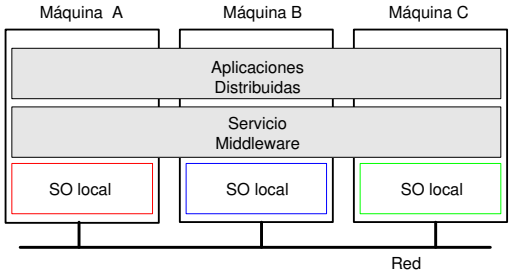


Figura 2.1: Sistema distribuido organizado como *middleware*. Adaptado de [58].

Programación Orientada a Objetos

En la programación orientada a objetos, el programador debe dividir las funciones y datos requeridos en bloques apropiados, distribuirlos en computadoras asignadas e intercomunicarse entre ellas. Por ello se han desarrollado una serie de éstandares para sistemas distribuidos, entre ellos se encuentran OSF/DCE (*Distributed Computing Enviroment From The Open Software Foundation*), DCOM(*Distributed Computing Object Model*), RMI (*Remot Method Invocation*) y SOAP (*XML-based protocol*) [51].

La tecnología orientada a objetos tiene tres características:

1. abstracción de datos.
2. herencia.
3. asociación dinámica.

Por tanto una aplicación que integra objetos corriendo en diferentes redes computacionales se le llama *aplicación distribuida*. Así pues la tecnología orientada a objetos tiene la ventaja de facilitar el desarrollo de grandes sistemas modulares.

Estructuras Orientadas a Objetos

Una *estructura* es un grupo de clases integradas y relaciones que describen uniones de objetos cooperativos. Una *estructura* es más que un librería de componentes de

software: define la arquitectura común debajo de una aplicación concreta construida sobre él. Consiste tanto en código reusable (librería de componentes) y diseño reusable (arquitectura).

D. Brugali [11] muestra una clasificación de *estructuras* de acuerdo a su alcance:

Infraestructura del sistema. Utilizado solo por organizaciones de *software* para el desarrollo de *estructuras* de sistemas operativos y comunicaciones.

Integración de *middleware*. Simplifica el desarrollo de aplicaciones interoperables y es comúnmente utilizado en el desarrollo de sistemas distribuidos robóticos y de automatización. Consiste en una serie de servicios que permiten a los sistemas distribuidos operar en conjunto.

Aplicaciones empresariales. Desarrollado para dominios específicos, comparados con los dos anteriores son muy caros, pero soportan tanto aplicaciones de usuario final como de productos.

Información global. En la actualidad es más utilizado en el desarrollo de organizaciones virtuales multinacionales.

El usar *middleware* consiste en desarrollar aplicaciones de usuario final delegando la ejecución de funcionalidades comunes a los servicios *middleware*. Entre estos se encuentran:

Administración de eventos distribuidos. cSoporta notificación dinámica de eventos realizados por objetos remotos.

Localización de objetos remotos. Permite a los objetos distribuidos cooperar entre sí, sin importar la ubicación de su red, plataforma utilizada e inclusive lenguaje de programación.

Localización de objetos distribuidos. Permite a los clientes objetos identicar al servidor objeto que ofrezca la funcionalidad que necesitan.

Administración de Seguridad en red. Admite firmas digitales y encriptación de datos.

Administración de persistencia y transacción. Permite almacenamiento persistente, acceso a datos distribuidos, replicación de datos y consistencia de información.

Conceptos de Sistemas Robóticos

Los sistemas robóticos múltiples autónomos han sido estudiados para trabajos cooperativos/competitivos en un marco de robots agrupados (robots en *Web*), redes robóticas, sistemas robóticos inteligentes [18].

Los sistemas telerobóticos se refieren a sistemas los cuales usan comunicación lineal exclusivamente y tienen una tarea específica en un ambiente particular.

Desde este enfoque se cuentan con muchos servidores *Web* a los que se conecta los robots y son controlados por operaciones remotas. Se necesita solamente una PC, un servidor *Web* y *software*.

Sistemas Robóticos Distribuidos

Consolidar ambos enfoques permite poder tratar a las redes robóticas como objetos distribuidos. Los sistemas roboticos distribuidos (DRS) son sistemas compuestos de agentes múltiples.

En este contexto surge el concepto de redes robóticas distribuidas (DRS) [24] en un esfuerzo de mejorar la interoperabilidad y disponibilidad de los mismos.

Se puede considerar a un robot remoto como objeto porque el controlador del robot oculta la estructura del robot del operario y no es necesario conocer acerca de las características internas del robot, solo su interfaz de control y a la red robótica como una red distribuida.

Esto se logra definiendo una interfaz común para controlar la red robótica y obteniendo como resultado que todos los sistemas robóticos sean interoperables entre si.

2.2. ¿Por qué utilizar CORBA?

Gopalan S. Raj lleva a cabo un estudio comparativo entre CORBA, DCOM y Java/RMI, desde el punto de vista de la arquitectura computacional de estos modelos, y aspectos relacionados a la programación. Finaliza con un cuadro comparativo entre estas tecnologías en el cual se proveen mecanismos de invocación transparente y el acceso a objetos remotos distribuidos de una manera más o menos similar [45].

D. Brugali et al., presentan un excelente trabajo de investigación en cómputo distribuido en las áreas de robótica y automatización. Revisando arquitecturas y modelos de estructuras orientadas a objetos. Tales como los modelos cliente/servidor, *three tier*, *broker* y multiagentes. Igualmente establece conclusiones en el ámbito de paradigmas de cómputo distribuido tales como Java, CORBA, Microsoft .NET y FIPA en aplicaciones robóticas y de automatización [11].

P. Young et al., evalúan diferentes arquitecturas de *middleware* con respecto a lograr la interoperabilidad entre sistemas heterogéneos. Además de las arquitecturas previamente comentadas, anexa en su estudio las aproximaciones comerciales de SeeBeyond®, HLA (*High Level Architecture*) y W3C-XML (*World Wide Web Consortium, eXtensible Markup Language*) [42].

Existen un buen número de trabajos de investigación que presentan ventajas y desventajas de cada uno de estas arquitecturas de cómputo distribuido sin embargo en su mayoría coinciden con los siguientes hechos.

Existen al menos dos razones por la cual Java RMI no ha sido utilizado de manera extensa en el área de robótica y automatización. El desempeño es todavía una preocupación principal debido a que Java es solo parcialmente compilado. Y el código *byte* no puede accesar directamente todos los recursos computacionales en la máquina anfitriona, sino a través del JVM (*Java Virtual Machine*) [11]. Igualmente es un esquema específico al lenguaje Java, por lo tanto no es muy recomendable su uso en aplicaciones de manufactura heterogéneos [6].

Por su parte .NET es todavía una nueva tecnología. No existen ejemplos relevantes de aplicaciones .NET es el área de robótica y automatización documentados. [11]. De-

bido a su herencia bajo la plataforma Windows y su limitación a solo cuatro lenguajes de programación para el desarrollo de aplicaciones, .NET no puede considerarse como una solución lo suficientemente amplia para la interoperabilidad [42].

Previamente COM/DCOM era utilizado como la estructura base sin embargo este presenta desventajas tales como que DCOM es solamente un protocolo para LAN, especialmente diseñado para ambientes Microsoft.

Las aplicaciones de manufactura requieren un nivel razonable de estandarización para ser aceptados. Esto garantiza una fácil configuración y más importante capacidad de reconfiguración después de un período de tiempo. Un ejemplo significante de la falta de estandarización es el número de *drivers* específicos que los vendedores de *software* tienen que desarrollar. Por otro lado algunos estándares universales no son aceptados porque son caros, difíciles de instalar o muy lejanos de las practicas comunes de programación. Tal es el caso de ISO MMS (*Manufacturing Message Specification*), el cual excedía en su dependencia con su arquitectura base, lo que representaba estar fuera del alcance de las compañías pequeñas [6], [12].

Otras aproximaciones se presentan con OPC (*OLE for Process Control*) el cual presentan soluciones listas para usar, pero propietarias en su núcleo. Como una estrategia de implementación, OPC requiere que los servidores sean compatibles con OLE/DCOM y escritos en C++. Ofrecen una interfase de automatización lo cual permite a los desarrolladores o usuarios finales escribir clientes en Visual Basic e importar datos en aplicaciones Windows estándar. Proyectos alternativos recientes se presentan con Java OPC, un paquete Java para implementar OPC bajo Java, y probando varios esquemas de interacción entre RMI, CORBA y XML [6].

En el caso de agentes inteligentes en robótica y automatización, FIPA no ha definido estándares específicos para estos dominios. En la literatura aún no se documentan sistemas robóticos o automatizados que den fuerza a estándares especificados por FIPA. Sin embargo el paradigma de agentes inteligentes ha llegado a ser muy popular en este tipo de aplicaciones, debido a su disponibilidad para modelar sistemas cooperativos autónomos. Ejemplos de sistemas multiagentes pueden encontrarse en las áreas de

automatización de fábricas, robótica móvil y diseño colaborativo [11].

Se decidió utilizar CORBA como arquitectura *middleware* ya que es una tecnología comprobada y madura, con la capacidad de usar componentes *software* que pueden ser implementados usando diferentes lenguajes de programación y que corren sobre diferentes plataformas *hardware*. Además de hacer uso de protocolos y servicios definidos por un estándar internacional que facilitan la comunicación entre componentes. Aunado a que se tenía ya una estructura de control robótico desarrollado en CORBA, por lo que el ciclo de desarrollo del *software* para la integración del brazo manipulador robótico Motoman sería más corto.

En la figura 2.2 se muestra un comparativo de los *middlewares* más populares y su nivel de impacto.

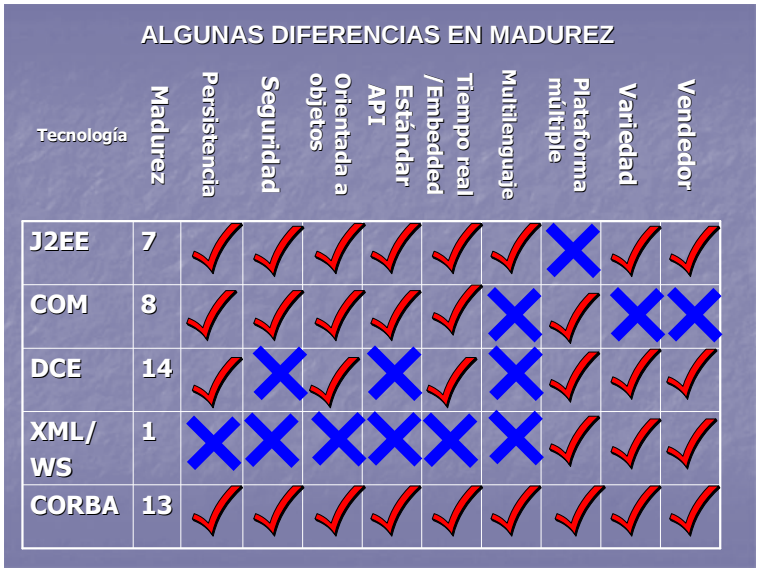


Figura 2.2: Comparativo de tecnologías *middleware*.

En el capítulo siguiente se detalla el estándar CORBA como parte de la Arquitectura de Administración de Objetos (OMA) de la OMG que consiste de un ORB, Servicios Objeto, Facilidades Comunes, Interfases Dominio y Objetos Aplicación.

Capítulo 3

EL CONTEXTO DE CORBA

3.1. Introducción

Debido a la necesidad de permitir la interoperatividad de aplicaciones distribuidas en ambientes heterogéneos y aprovechar la programación orientada a objetos en 1989 se funda la *Object Management Group* (OMG), organización que agrupa en la actualidad alrededor de 800 compañías. ¿El resultado?, la especificación de una arquitectura genérica: OMA (*Object Management Architecture*) [Fig 3.1]. Donde CORBA es la tecnología asociada a esta arquitectura, cuya filosofía de trabajo es un ambiente de sistema abierto.

El auge en la descentralización de los sistemas, de los servicios que brindan los mismos a terceros y en especial de las personas, hacen de los entornos distribuidos una gran opción en el desarrollo de sistemas compactos y versátiles que de cara a la tecnología dan solución a problemas de diferente índole y en diferentes campos, como son la medicina, las telecomunicaciones, la banca, la manufactura, la telerobótica, etc. Existen muchos proyectos de investigación que utilizan uno de los componentes más importantes de OMA: CORBA [19].

En el transcurso de este apartado introduciremos características más específicas de CORBA, sin embargo de una manera generalizada CORBA usa un ORB (Object Request Broker) como *middleware* que establece relaciones cliente - servidor entre objetos, lo cual permite a componentes u objetos distribuidos sobre redes, comunicarse unos con otros, sin tener que preocuparse en el lugar donde reside el servidor, el lenguaje de

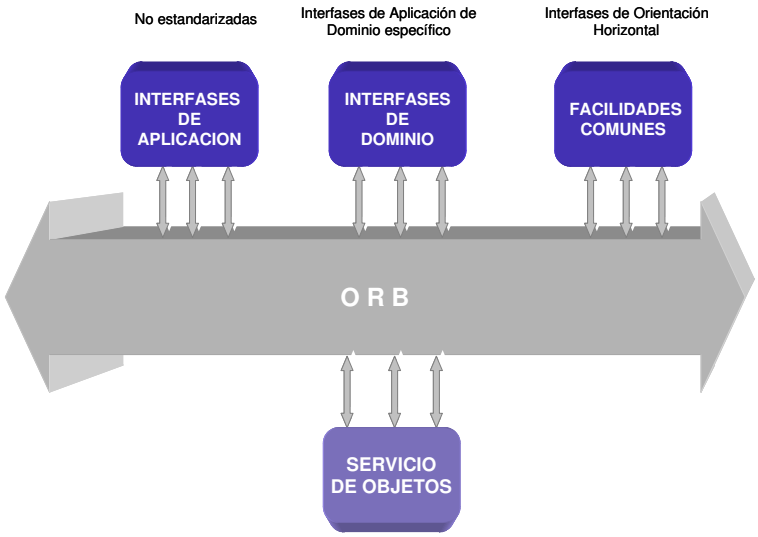


Figura 3.1: Arquitectura Genérica de Administración de Objetos.

programación y el sistema operativo residente.

Ventajas y Desventajas de CORBA

Ventajas

Entre sus múltiples ventajas se enumeran algunas:

- Abstrae a la aplicación de todo lo referente a las comunicaciones, el programa ni siquiera sabe donde esta el objeto al que llama.
- Permite reutilizar programas anteriores simplemente añadiendo algo de código.
- Tiene todas las ventajas de la programación orientado a objetos.
- Se puede programar en varios lenguajes. Se puede crear un programa con varios lenguajes distintos ya que cada una de las partes solo verá el interfaz CORBA de la otra.
- Abstrae el Sistema Operativo.
- Es un estándar para todas las plataformas.

Desventajas

Sin embargo, algunos de sus inconvenientes son:

- Más capas de *software*, más carga.
- Un objeto se ejecuta en el sistema que le alberga.
- Falta de mecanismos de sincronización y prioridad.
- Falta de herencia en las excepciones.
- Algunos ORB's no soportan en ocasiones todos los tipos de datos proveídos por el IDL.
- Modelo de hilos (*threads*) no muy robusto.

3.2. Arquitectura de administración de objetos

OMA pretende definir en un alto nivel de abstracción las reglas necesarias para la distribución de la computación orientada a objetos en entornos heterogéneos. Se compone de dos modelos, el Modelo de Objetos y el Modelo de Referencia, el primer modelo define como se deben describir los objetos distribuidos en un entorno heterogéneo [27] y el segundo modelo caracteriza las interacciones entre dichos objetos.

En el Modelo de Objetos de OMA, un *objeto* es un ente encapsulado con una única identidad y por intermedio de una interfaz bien definida es posible acceder a sus servicios (cuando un cliente solicita los servicios de un objeto, la localización y la implementación de dicho objeto son transparentes al cliente. El cliente no necesita saberlo, simplemente solicita el servicio a través de la interfaz antes mencionada). Los componentes del Modelo de Referencia de OMA lo podemos apreciar en la figura 3.2, su núcleo es el ORB (*Object Request Broker*- Mediador de Peticiones de Objeto), además de dar transparencia en la localización y activación de objetos, se encarga de facilitar la comunicación entre clientes y dichos objetos. A continuación se describe el conjunto de interfases que utilizan el ORB en el modelo de referencia en cuestión.

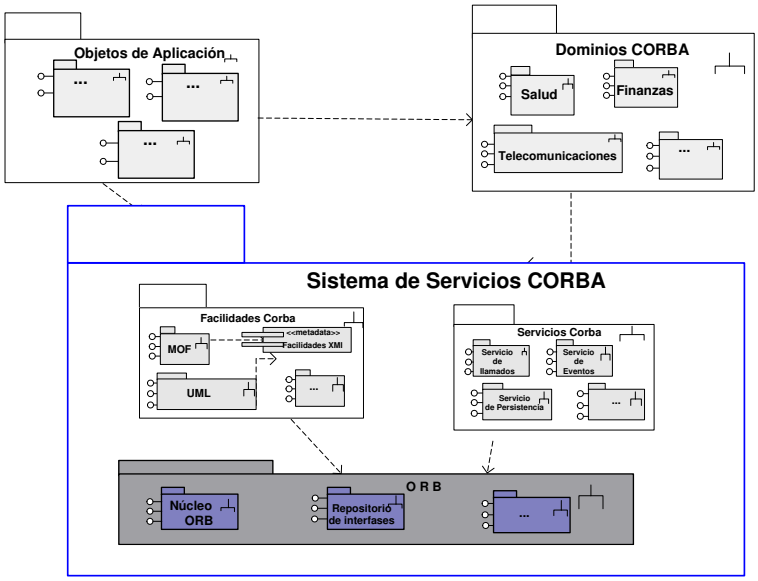


Figura 3.2: Arquitectura de Administración de Objetos.

3.2.1. Interfases de Aplicación

Desarrolladas específicamente para una determinada aplicación. No han sido estandarizadas por OMG, sin embargo si alguna de éstas comienzan a aparecer en varias aplicaciones distintas se convierten en candidatas para estandarización en una de las categorías existentes.

3.2.2. Dominios CORBA

Estas interfases están orientadas a aplicaciones de dominio específico (verticalmente orientadas). Por ejemplo, interfases destinadas a aplicaciones financieras, o a aplicaciones de telecomunicaciones o de manufactura. Es por ello que en la figura 3.2 existen diferentes gráficas de dominios de interfases, que representa un campo o un dominio de aplicaciones (en telecomunicación, salud, entre otros).

3.2.3. Facilidades CORBA

Estas interfases son de dominio independiente (orientación horizontal). Es decir que pueden ser usadas en cualquier campo, por ejemplo en medicina, telecomunicacio-

nes, tele-ingeniería, entre otras y están orientadas a aplicaciones de usuario final, como un ejemplo el *Distributed Document Component Facility* (DDCF) de OMG, basado en *OpenDoc* de *Apple Computer*, que permite la composición, presentación e intercambio de objetos basados en un modelo de documento.

3.2.4. Servicios de objeto

Es una colección de interfases de orientación horizontal, que aumentan y complementan la funcionalidad del ORB, estas son usadas por muchos programas que distribuyen sus objetos (*Distributed Object Programs*). Por ejemplo, servicios que descubren otros servicios disponibles en el dominio de las aplicaciones, entre ellos están:

Servicio de Nombres

. Permite a un cliente encontrar un objeto por su nombre. La función básica del Servicio de Nombres es la asociación de estos nombres con referencias de objetos. Un servidor crea asociaciones entre los nombres y las referencias para los objetos CORBA que están destinados a servir como puntos iniciales de contacto. Un cliente que conoce el nombre de un objeto puede entonces acceder al objeto por medio del servicio de nombres sin preocuparse por su valor.

Haciendo una analogía diremos Servicios de Nombres de OMG tiene la misma función de DNS de Internet que traduce dominios (ejem. www.itesm.com.mx) en direcciones IP (como 192.168.140.112).

El Servicio de Nombres se encuentra disponible en dos versiones:

- El Servicio de Nombres CORBA original [39].
- El Servicio de Nombres CORBA interoperable [8].

Entre las ventajas de utilizarlo se encuentra:

- Los clientes pueden usar nombres con significado para los objetos, en lugar de tratar con referencias como cadenas de caracteres.

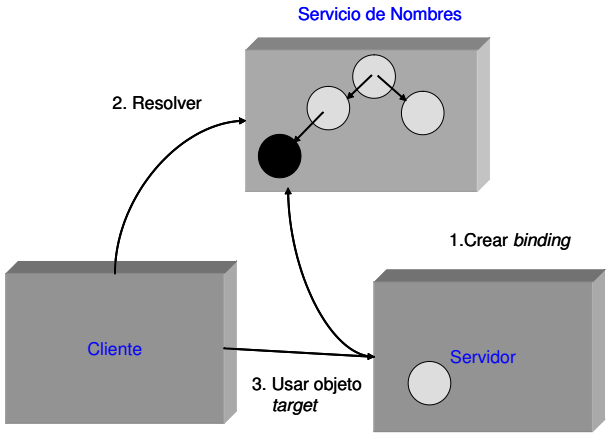


Figura 3.3: Uso básico del Servicio De Nombres.

- Si cambia el valor de una referencia publicada con un nombre, se puede obtener clientes que usen una implementación diferente de una interfaz sin tener que cambiar el código fuente. Los clientes usan el mismo nombre, pero obtienen referencias distintas.

- El Servicio de Nombres se puede utilizar para resolver el problema de cómo los componentes de una aplicación acceden a las referencias iniciales para una aplicación. Publicar estas referencias en el Servicio de Nombres elimina la necesidad de almacenarlas como cadenas de caracteres en archivos.

Antes que el cliente pueda buscar un objeto, debe crearse el enlace entre la localización del objeto y su nombre. Este enlace se conoce como **asociación** (*object binding*) y normalmente es realizado por un servidor CORBA.

La figura 3.3 ilustra los pasos utilizado en un Sevicio de Nombres. Este es implementado típicamente como un proceso que corre independientemente del cliente y servidor. Los pasos son los siguientes:

1. Crear una asociación (*object binding*). Cuando un servidor inicia, crea un número de objetos CORBA, los cuales servirán como puntos de contacto inicial para los clientes. Estos objetos son asignados a los clientes creando objetos asociados en el servicio de nombres. La correspondencia consiste en una serie de asociaciones entre nombre/referencia objeto.

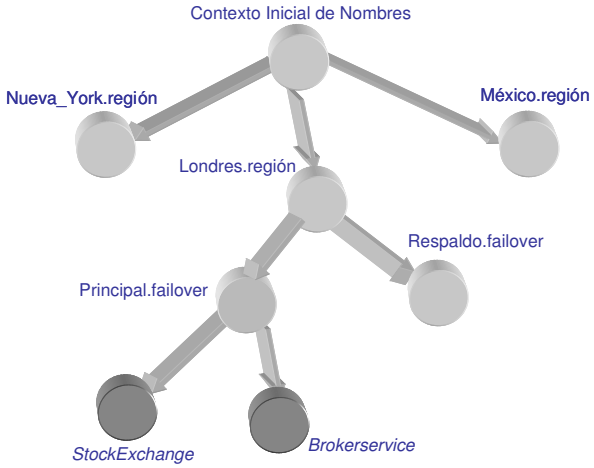


Figura 3.4: Representación de un grafo de nombrado.

2. Acordar un nombre. Un cliente puede accesar uno de los objetos asignados en el paso anterior. Esto es buscar un objeto referencia usando su nombre como clave.

3. Utilizar un objeto destino. Ya que el objeto referencia es todo lo que el cliente necesita para accesar al objeto, el cliente puede continuar y usar el objeto.

Este es la principal función del servicio de nombres, la cual es esencialmente, una base de datos de objetos asociados (*object binding*). Esta colección de asociaciones normalmente son arregladas en una jerarquía a la que se conoce como *grafo de nombrado* (fig. 3.4). Hay dos clases de asociaciones en la jerarquía:

- Contexto (*Context binding*): Una asociación entre un nombre y un contexto de nombrado (*naming context*)

- Objeto (*object binding*): Una asociación entre un nombre y un referencia de objeto (*object reference*)

De alguna manera se puede ver como un directorio en un sistema de archivos. Los círculos claros representan los contextos de nombrado lo que sería como los directorios en un sistema de archivos. Los círculos oscuros representan las referencias (*object references*) quienes análogamente serían los archivos del sistema.

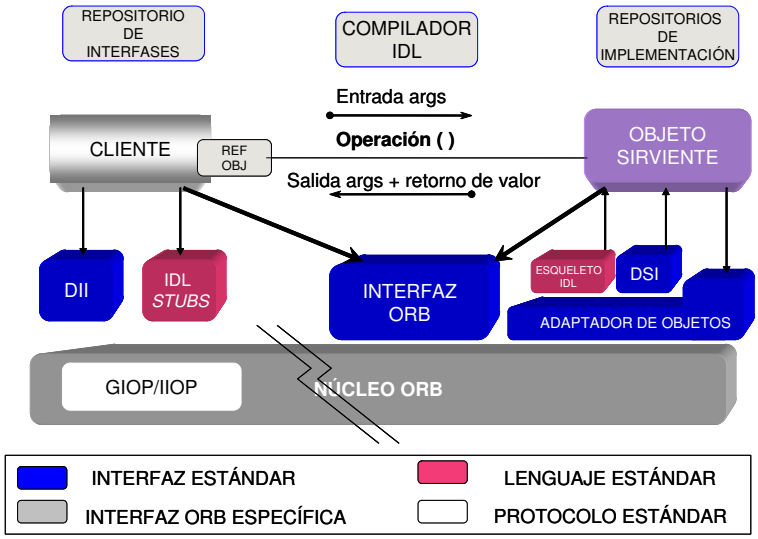


Figura 3.5: Arquitectura del ORB de CORBA.

Otros servicios

Sin embargo existen otros tipos de servicios como: El **servicio de eventos** (permite a los objetos registrarse dinámicamente ante el interés de un evento particular). El **servicio de *"traiding"*** (donde un cliente puede encontrar objetos basándose en sus propiedades). El **servicio de persistencia** (a un cliente le es posible almacenar objetos constantemente), el **servicio de ciclo de vida** (define operaciones para crear, copiar, mover y borrar objetos de un ORB), el **servicio de seguridad** (da entre muchas más, soporte de autenticaciones, listas de control de acceso a objetos), y más como el **servicio de transacciones**, **de control de concurrencia**, **de búsqueda** y **de inicialización** entre otros [23].

3.3. Componentes de la especificación CORBA

CORBA es una infraestructura computacional abierta de objetos distribuidos, que está siendo especificada por la OMG, con el ánimo de describir todas las características del ORB de OMA. En la figura 3.5 podemos apreciar cada uno de los componentes de CORBA.

3.3.1. Cliente

Es la entidad que invoca operaciones sobre un objeto de la implementación de objetos. Los servicios que brinda dicho objeto son transparentes al llamante, bastaría simplemente con invocar un método sobre un objeto; de tal forma que un objeto remoto para una entidad cliente (local y llamante) se comporta como si fuese un objeto local (ver flechas con terminación circular de la figura 3.5).

3.3.2. ORB

El ORB es el encargado de dar transparencia en la comunicación a los clientes, en lo que se refiere al envío de requerimientos y al retorno de respuestas, cuando dichos clientes, solicitan los servicios de un objeto. El objeto que un cliente desea, y al cual el ORB envía sus requerimientos, es llamado el *objeto destino*.

El ORB se encarga de la localización de los objetos ya que el cliente no la conoce, ni la necesita (un objeto puede estar en un proceso corriendo en otra máquina dentro de la red o sobre la misma maquina, en diferente o en el mismo proceso), el cliente tampoco conoce la implementación de los objetos con los se desea interactuar, ni el lenguaje de programación en que están escritos, ni el sistema operativo, ni el *hardware* sobre el cual están corriendo; el cliente tampoco se preocupa de la activación de los objetos requeridos, ya que el ORB es el encargado de activar los objetos si fuese necesario, además, el cliente no necesitará conocer los mecanismos de comunicación (TCP/IP, llamada de métodos locales, etc.) que sé utilizan, simplemente el ORB pasa los requerimientos de los clientes a los objetos y envía una respuesta a quien hizo el requerimiento. Por otra parte, la transparencia del ORB permite que los desarrolladores se preocupen más de sus aplicaciones y menos de los asuntos que tengan que ver con programación de sistemas distribuidos a bajo nivel.

3.3.3. Interfaz ORB

Es un conjunto de librerías o APIs (*Access Point Interfaces*) que definen un conjunto de funciones del ORB y que pueden ser accedidas directamente por el código cliente, entre ellas están las de convertir las referencias de objetos (cuando se solicita un servicio a un *objeto destino* el servidor envía una referencia de dicho objeto, que en realidad es la información necesaria que un cliente necesita para interoperar con el ORB y dicho *objeto destino*) en *strings* o viceversa y las que sirven para crear listas de argumentos de requerimientos, hechos a través de una invocación dinámica, la cual se describirá en la sección 3.3.7.

3.3.4. Lenguaje de Definición de Interfases IDL

Cuando un cliente solicita los servicios de un objeto, este debe conocer las operaciones soportadas por dicho objeto, las interfases de un objeto simplemente describen dichas operaciones. IDL (*Interface Definition Language*) es un lenguaje de "especificación", que en la actualidad es un éstandar ISO [50], parecido en estructura a C++, que permite declarar el "contacto" de un objeto con el mundo exterior. Una de las ventajas de describir interfases de esta forma, es separar los puntos de acceso a un objeto (sus interfases) de su propia implementación, lo que permite que los objetos sean implementados en diferentes lenguajes de programación (C, C++, Java, Ada 95, SmallTalk, Cobol) e interactúen entre sí en forma transparente (aspecto importante en un sistema heterogéneo), como se representa en la figura 3.6.

El IDL ofrece tipos básicos predefinidos como enteros con y sin signo, caracteres, booleanos, *strings* y del tipo complejos, como enumerados, estructuras, uniones, secuencias (arreglos unidimensionales) y excepciones. Se usan para definir los tipos de los parámetros y los tipos devueltos por las operaciones, los cuales a su vez se definen en las interfases. El IDL también proporciona un módulo constructor para poder definir ámbitos de nombrado.

Los argumentos de las operaciones del IDL deben declarar su dirección, de forma que el ORB sepa si sus valores deberían ser enviados desde el cliente al objeto destino,

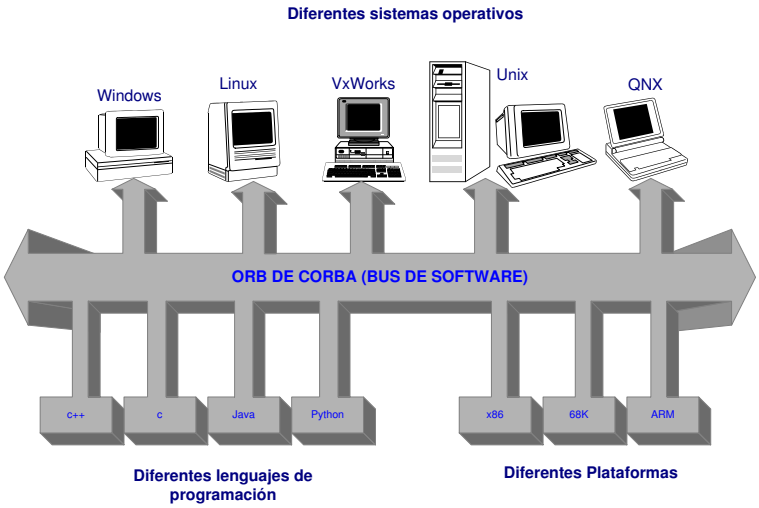


Figura 3.6: Interoperabilidad de objetos implementados en diferentes lenguajes.

viceversa o ambos.

Una característica fundamental de las interfases IDL es que pueden heredarse de una o más interfases distintas. Este arreglo permite definir otras interfases a partir de las existentes, así como objetos que implementan una nueva interfaz derivada de interfases base [23].

Estructura del IDL del Servicio de Nombres.

Las definiciones IDL para el Servicio de Nombres se proporcionan en una archivo llamado `Cosnaming.idl`. Este archivo contiene un módulo denominado `Cosnaming`. Este módulo contiene varias definiciones de tipo y dos interfases: `NamingContext` y `BindingInterator` [23].

3.3.5. "Stubs" y "Skeletons"

Un *Stub* (normalmente llamado *Proxy*) es un ente encargado de enviar los requerimientos de un cliente a un servidor a través del ORB (comúnmente llamado *marshaling*, que consiste en convertir los requerimientos de un cliente implementado en un algún lenguaje de programación en una representación adecuada para el envío de información a través del ORB); el *Skeleton* (en el servidor) es el encargado de colaborar con la recepción de dichos requerimientos desde el ORB y enviarlos a la Implementación de

Objetos de CORBA (llamado *unmarshaling*, que es simplemente hacer una conversión de un formato de transmisión a un formato en un lenguaje de programación dado), visto de atrás hacia delante, a través del *Skeleton* se envía alguna respuesta a través del ORB y es recibida por el cliente por medio del *Stub*, normalmente al conjunto de los envíos a través del *Stub* y el *Skeleton* es llamado *invocación estática* (tanto el cliente como el objeto de implementación, tienen pleno conocimiento de las interfases IDL que están siendo invocadas).

3.3.6. Repositorio de Interfases (IR)

El IR (*Interface Repository*) es una base de datos distribuida que contiene información de las interfases IDL definidas para los objetos que cooperarán en un entorno distribuido y que puede ser accedida o sobre escrita en tiempo de ejecución; podemos pensar en el IR como un objeto CORBA, con una base de datos asociada y que tiene un conjunto de operaciones que pueden ser utilizadas como si fuese un objeto cualquiera, entre los servicios que ofrece dicho objeto CORBA, es el permitir navegar sobre la jerarquía de interfases almacenadas en la base de datos, de tal forma que se pueda conocer si fuera necesario, la descripción de todas las operaciones que un objeto soporta [23]. Una forma muy interesante y de mucha utilidad es usar el IR para descubrir interfases de objetos en tiempo de ejecución, empleando invocación dinámica, que se verá a continuación.

3.3.7. Interfaz de Invocación Dinámica y DSI.

El otro tipo de invocación que existe en CORBA es la Invocación Dinámica, que permite en tiempo de ejecución (*run-time*), descubrir las operaciones de un objeto, sin tener un conocimiento previo de sus interfases (sin un *stub*), para dicha invocación dinámica existen dos tipos de interfases, una es la Interfaz de Invocación Dinámica DII (*Dynamic Invocation Interface*) y la otra es llamada el DSI (*Dynamic Skeleton Interface*); el DII en una aplicación cliente que se encarga de hacer peticiones de algún objeto del que no se conocen sus interfases, dicha petición se hace a través de un pseudo

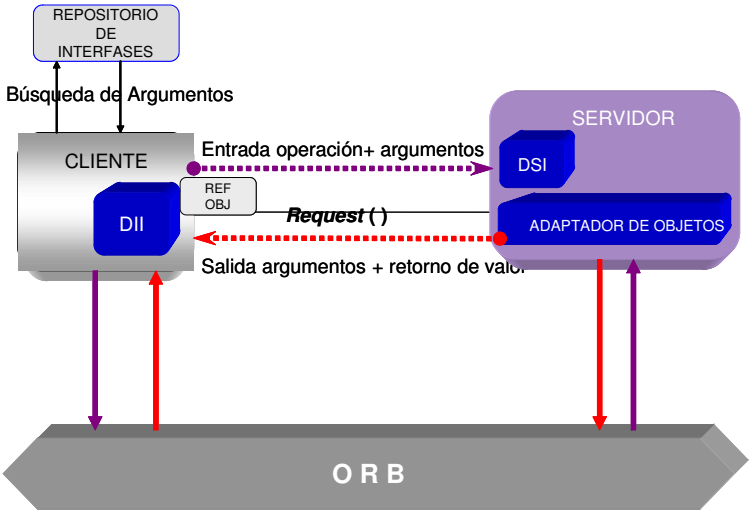


Figura 3.7: Invocación Dinámica con DII y DSI.

objeto llamado *request*, sobre el cual el cliente especifíca el nombre de la operación y sus argumentos que pueden ser obtenidos del Repositorio de Interfases(IR). Cuando la petición está completo se le envía al servidor, dicho envío puede hacerse de tres formas, en la primera la petición se envía y todos los procesos se bloquean hasta que el servidor emita una respuesta (Invocación Sincrónica), en la segunda cuando la petición se envía, el cliente sigue procesando y más tarde recoge la respuesta (Invocación Sincrónica Aplazada), en la última forma cuando se envía la petición, el cliente sigue procesando y la respuesta del servidor se recoge por algún otro medio (por ejemplo un proceso separado que la recoja).

En el lado del servidor, cuando un *request* (pseudo objeto) es recibido, el DSI es quien lo toma y envía alguna respuesta al cliente ante la petición solicitada (lo que es llamado el *Dispatching*). En la figura 3.7 se puede apreciar de una manera gráfica una forma de hacer invocación dinámica.

3.3.8. Adaptador de objetos

Es el ente de contacto entre el ORB y los objetos de implementación y es quien acepta requerimientos en nombre de los objetos servidores, dicho adaptador se encarga en tiempo de ejecución de activar, hacer peticiones, pasar requerimientos y generar

referencias de dichos objetos. También, de colaborar con el ORB para que todos los requerimientos que se hagan de múltiples conexiones, sean recibidos sin ningún tipo de bloqueo. El adaptador de objetos tiene tres interfases asociadas, una al DSI, una al IDL *skeleton* y otra a la implementación de objetos (Ver figura 3.5) siendo las dos primeras privadas y la última pública, todo esto con el ánimo de aislar la implementación de los objetos del ORB tanto como sea posible.

La OMG estandarizó un adaptador de objetos llamado BOA (*Basic Object Adaptator*), dicho BOA y los servidores, tiene la posibilidad de soportar más de un adaptador de objetos. OMG actualmente ha lanzado una especificación que mejora algunos defectos de portabilidad del adaptador de objetos BOA y es llamada POA (*Portable Object Adaptator*).

3.3.9. Repositorio de Implementación

Es una base de datos que en tiempo de ejecución, da información acerca de las clases que un servidor soporta, los objetos que están requeridos, sus identificadores IDs (es un número único que asigna el adaptador de objetos a cada instancia de un objeto) y una serie de datos administrativos de los objetos, como trazas de información e información de seguridad entre otros.

3.4. Implementación de CORBA en el mercado

En el mercado existen compañías que han adoptado a CORBA como opción para el desarrollo de productos y otras como una buena opción de abrirse paso en el mercado en diferentes campos, telecomunicaciones, medicina, manufactura, banca, etc. Hay diferentes compañías que implementan el ORB de OMG entre ellos están Orbacus, Orbix y OrbixWeb de IONA (Para aplicaciones C++ y Java respectivamente), NEO y JOE de SunSoft, HP ORB de HP, SOM de IBM, VisiBroker de Visigenic, ILU de Xerox Parc, Object Broker de Digital, Netscape´s ONE ORB, Corbus de BBN, Chorus/COOL ORB de Chorus, Dais de ICL Soft, CorbaPlus de Expersoft, OmniBroker de Object-Oriented

Concepts, OmniORB de Olivette Research Labs y Distribute Smalltalk de ParcPlace entre otros.

Capítulo 4

MODELACIÓN *UML* DEL SISTEMA ROBÓTICO DISTRIBUIDO

4.1. Introducción

UML (*Unified Modeling Language*). Es un lenguaje de modelado de propósito general que es utilizado para especificar, visualizar, construir y documentar los componentes de un sistema de *software*. Captura las decisiones y entendimiento acerca de los sistemas que deben ser construidos y es utilizado para entender, diseñar, configurar, mantener y controlar la información de dichos sistemas. Está proyectado para su uso con todos los métodos de desarrollo, etapas de ciclo de vida, dominios de aplicaciones y medios. Soporta la mayoría de los procesos de desarrollo orientados a objetos y además captura información estática y el comportamiento dinámico de un sistema. Contiene estructuras de organización para ordenar modelos en paquetes que permite particionar sistemas complejos en fragmentos mejor manejables [47].

Ventajas de usar UML

En el caso donde el tipo de nivel de modelación es importante y se requiere reuso y capacidades de extensibilidad. Un lenguaje de modelación orientado a objetos debe ser considerado. La notación UML ofrece una vista clara de las clases que componen el sistema, haciendo más fácil entender las relaciones existentes entre las entidades del sistema. Además usando UML significa usar un lenguaje de modelación mucho

más utilizado, con las ventajas incuestionables que esto representa como: mejor gente entrenada, una diversidad de herramientas que lo soportan, etc. Otra de las razones por la que UML es muy popular es su simplicidad. Es muy cercano a los conceptos principales encontrados en la mayoría de los lenguajes de programación. Además de ser lo suficientemente general para hasta cierto punto para abarcar muchos tipos de dominios.

Desventajas de UML

Las principales áreas donde UML presenta problemas son: el comportamiento y la especificación de la comunicación, así como la representación del tiempo. No ofrece los medios para precisamente especificar el comportamiento, ya que la semántica de sus máquinas de estados no están precisamente definidas y actualmente las acciones en una transición o la descripción del cuerpo de una operación son especificadas informalmente. En UML se pueden solo especificar las señales que una clase puede recibir, pero no se pueden describir las rutas de comunicación. Finalmente el concepto de tiempo no tiene semántica y no es claro el cómo realmente es usado [31].

Sin embargo, a pesar de las desventajas que pudiera tener este lenguaje de modelación, se seleccionó como una primera aproximación para la modelación del sistema robótico distribuido basado en CORBA para el Motoman UP6, ya que este es el lenguaje de modelación de mayor popularidad entre la comunidad de tecnologías orientadas a objetos, además de que satisface bien las particularidades de sistemas complejos, grandes y distribuidos. La finalidad de esto, es establecer una manera más didáctica de entender las interrelaciones entre las clases que componen el sistema. Una de los mayores inconvenientes de un sistema distribuido es que normalmente se pasa directamente al desarrollo sin previamente modelar el sistema. Al final se puede llegar a tener un código *espagueti*, en el cuál nuevos desarrolladores requerirán tiempo y de un programador con la experiencia previa en el sistema para entender las interrelaciones ocurrentes en el mismo. Información más detallada de este lenguaje se discuten en varios libros y trabajos de investigación como [28, 36, 47].

Se usarán diagramas de clase UML para modelar el sistema. Estos se utilizan para describir la estructura estática del modelo. Incluyen los atributos y las operaciones para proveer acceso a las clases [38].

4.2. Modelación de aplicaciones CORBA con UML

La arquitectura CORBA provee un *estructura* que se puede extender para construir aplicaciones distribuidas robustas. Esta arquitectura prefabricada evita los detalles de la comunicación entre procesos y los servicios distribuidos del sistema operativo.

Es posible utilizar OMG IDL para especificar objetos y componentes de negocio. IDL es un lenguaje de especificación puro. Permite definir las interfases a los objetos sin restricciones en términos de su implementación. Por lo tanto, se puede usar IDL para definir la estructura de la aplicación y separar las definiciones de los objetos del negocio de los detalles de su implementación. Al separar las especificaciones objeto de su implementación, se obtienen beneficios de ocultamiento de información, neutralidad en el desarrollo e independencia en la plataforma de uso.

IDL tiene algunas desventajas. Primero, no permite especificar el comportamiento del objeto o las relaciones de las clases, a no ser el de generalización. Consecuentemente, se pueden especificar las operaciones asociadas con una interfase, pero no se pueden definir métodos, casos de uso, colaboraciones, máquinas de estado, flujos de trabajo o varias relaciones típicamente asociadas con objetos de negocios reales. Segundo, IDL es un lenguaje sin representación gráfica. Mientras que esto puede ser satisfactorio para especificar estructuras simples, es una limitación indeseada para definir relaciones estructurales complejas y de comportamiento [52].

4.3. Especificación OMG UML

UML provee un extenso conjuntos de notaciones que pueden usarse para describir diferentes aspectos del *software* bajo desarrollo, incluyendo concurrencia compleja y distribución. Contiene mecanismos de extensión y un lenguaje de restricción llamado

lenguaje de restricción de objetos (OCL), el cual puede usarse para colocar restricciones adicionales en los modelos y describir condiciones previas y posteriores de operaciones. Desde el punto de vista semántico, UML provee un metamodelo (el cual es de hecho una instancia del meta-metamodelo (MOF)) que define modelos bien formados, y posibles relaciones entre modelos y elementos de los modelos. De cualquier manera, UML es solo un lenguaje, su proceso es independiente y por lo tanto no prescribe como sus notaciones deben usarse. Por lo que usar UML para modelar sistemas distribuidos, aún requiere un método - una opción de modelos y procesos para su elaboración.

La especificación del *Meta-Object Facility* (MOF) define un lenguaje abstracto y una *estructura* para especificar, construir y manejar metamodelos neutrales en tecnología. Un metamodelo no es más que un lenguaje abstracto para algún tipo de información de metadato. El UML y el MOF están basados en una arquitectura conceptual de metamodelo por capas, donde los elementos en una capa conceptual dada describen los elementos en la proxima capa [53]. Por ejemplo:

El MOF meta-metamodelo es el lenguaje usado para definir un metamodelo UML. El metamodelo UML es el lenguaje usado para definir modelos UML, y el modelo UML es un lenguaje que define aspectos de un sistema computacional. Ver figura 4.1.

Aunque el núcleo de la especificación UML es la definición de la sintaxis y semántica del lenguaje, también incluye definiciones relacionadas con extensiones del lenguaje, restricciones e intercambios de modelo. Las secciones principales de la especificación se describen a continuación:

Semántica UML.

Define la semántica del lenguaje usando un metamodelo. El lenguaje es organizado en paquetes, y las metaclases son descritas en un estilo "semiformal" que combina notación gráfica, lenguaje restrictivo y un lenguaje natural.

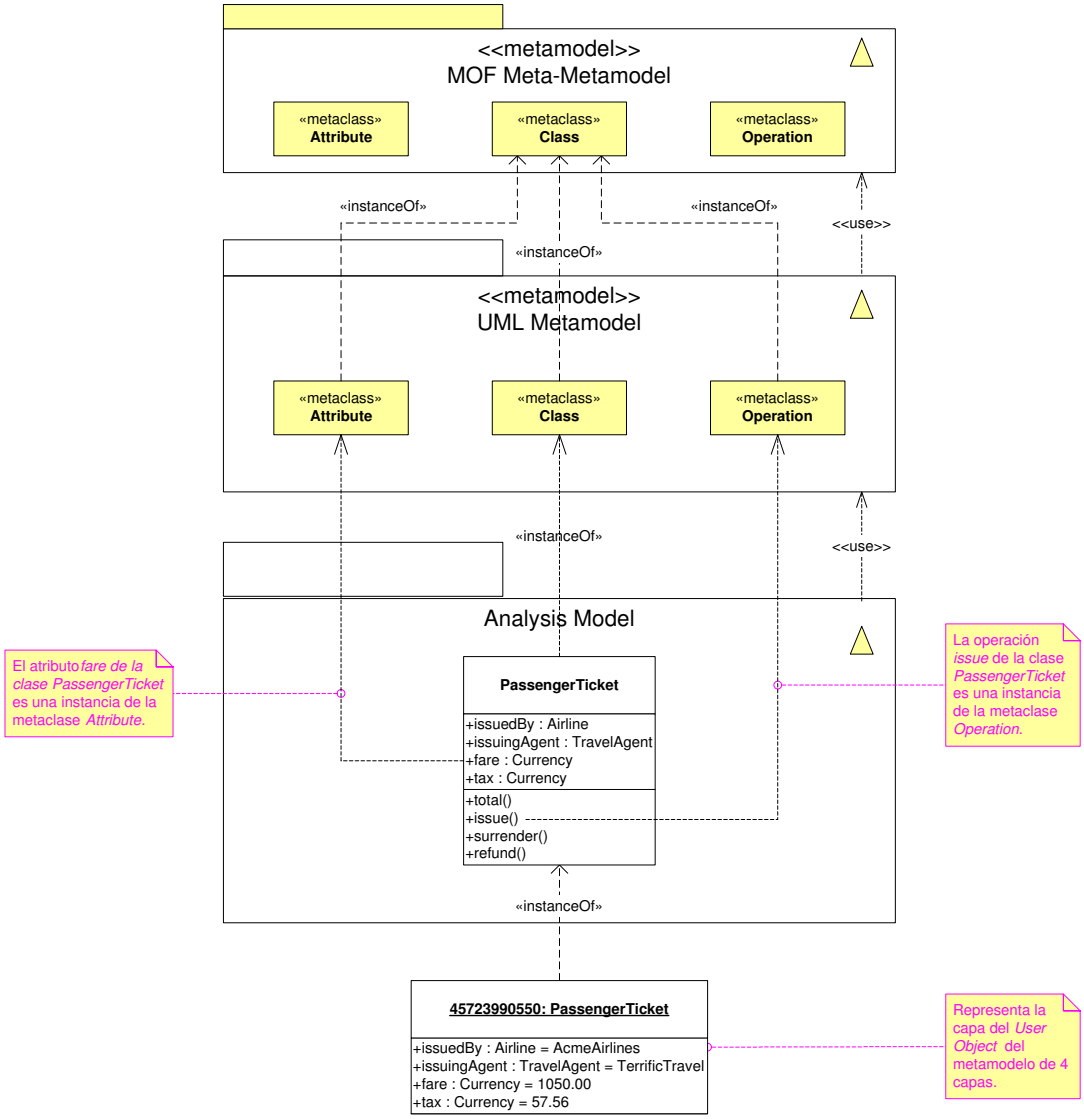


Figura 4.1: Arquitectura del Metamodelo de 4 capas de OMG. Adaptado de [52].

Guía de Notación UML.

Define la sintaxis gráfica para expresar la semántica descrita en la especificación de semántica UML.

Perfiles estándar UML.

Define extensiones del lenguaje para procesos de desarrollo de *software* y modelación de negocios.

Definición de Facilidades CORBA.

Define repositorios para modelos definidos usando UML. La facilidad habilita la creación, almacenamiento y cambio en la administración de los modelos UML.

Lenguaje de Restricción de Objeto (OCL).

Define la sintaxis y semántica del OCL, un lenguaje declarativo para especificar restricciones de objetos.

4.4. Aproximaciones de diseño y modelación de aplicaciones distribuidas

Las aproximaciones actuales para el diseño de aplicaciones distribuidas se basan en notaciones de diseño y análisis orientados a objetos. Ejemplos de estas, son el *Rational Unified Process* (RUP), *Enterprise Distributed Object Computing* (EDOC) y el CORBA *Profile* para UML. RUP es un método de diseño muy general, el cual no se enfoca en problemas específicos que ocurren dentro de sistemas que son distribuidos. EDOC es muy específico para la modelación de procesos de negocio, en este sentido no es propiamente un método de diseño de *software*. El perfil de CORBA para UML es una reflexión de conceptos del lenguaje de definición de interfase (IDL) CORBA en UML, de aquí que se enfoque puramente en la estructura y la definición de la signatura, pero

no en aspectos de comportamiento y cuestiones específicas de interacción entre objetos [9].

Para seleccionar una notación apropiada, el conjunto de requerimientos que deben ser contemplados son:

- Aceptación de la notación y disponibilidad de herramientas (CASE).
- Habilidad para limitar o extender la notación.
- Notación gráfica y conceptos de orientación a objetos como fundamento.

Tres aproximaciones actualmente dominan la manera los programadores usan UML para modelar y crear aplicaciones.

- Modelar el sistema en UML, posteriormente utilizar un lenguaje de programación para manualmente implementarlo. En este caso, los programadores usualmente descartan el modelo una vez el código real existe.
- Modelar el sistema en UML, utilizar un lenguaje de programación para generar el código *stub*. Las herramientas usualmente proveen mecanismos para sincronizar el modelo con el código si cambios son llevados a cabo - llamado *round trip engineering*. Sino se utiliza sincronización, el modelo se almacena para propósitos de documentación, pero pudiera descartarse el modelo, después de generar el código *stub*.
- Modelar el sistema en UML, y usar abstracciones especializadas tales como el envío de señales para proveer la implementación, usualmente en C++ directamente. Ya que existen relaciones íntimas entre el código y el modelo, este último es útil aún después que la aplicación a sido completada [7].

El perfil UML para la especificación CORBA fue diseñado para proveer un medio estándar para expresar la semántica de IDL CORBA usando notación UML, por lo tanto poder soportar la expresión de estas semánticas con herramientas UML.

Si se quiere representar un tipo CORBA con notación UML, la aproximación usual es modelarlo como un clasificador y estereotiparlo para indicar si representa una interfase, un tipo valor, una estructura o una union, etc. Esta es una aproximación legítima, ya que el estereotipo es uno de los mecanismos oficiales de extensión legítimos para UML [57].

En nuestro caso la interfase IDL parar el sistema robótico distribuido puede representarse tal como se indica en la figura 4.2.

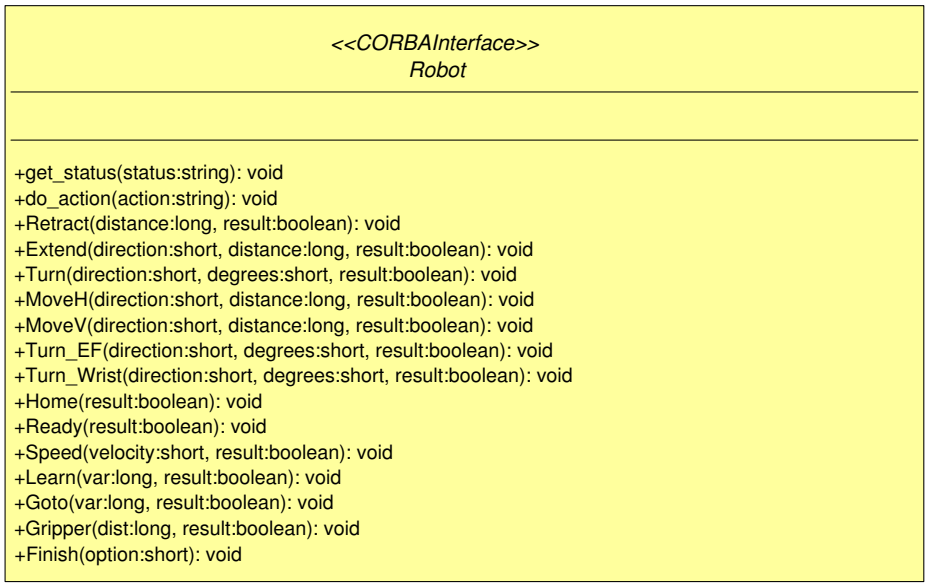


Figura 4.2: Representación UML de la interfase CORBA IDL del sistema robótico UP6.

Más clases que comprenden el sistema robótico distribuido para el UP6 se presentan en el apéndice A.

Capítulo 5

CASO DE ESTUDIO

5.1. Antecedentes

Tal como presentamos anteriormente, CORBA provee la oportunidad de usar componentes de *software* que puedan ser implementados usando diferentes lenguajes de programación y bajo diferentes plataformas. El ORB es tanto un *middleware* como una tecnología enfocada a componentes. Como *middleware* soporta la transparencia local/remota entre los clientes y servidores. Como tecnología de componentes, soporta la definición de APIs y la activación en tiempo de ejecución de módulos ejecutables. El modelo de programación de CORBA permite extender las interfases existentes polimórficamente, reemplazar instancias de objetos, y anexar nuevos objetos a un sistema existente [44].

El *software* de los sistemas de manufactura se construye frecuentemente mediante la integración pre-existente de componentes. Una especificación precisa de las interacciones de los componentes en estos sistemas es requerido para asegurar el mantenimiento y las capacidades de prueba de los mismos. Además, los sistemas utilizados en manufactura deben ser de preferencia estándares que especifiquen como lograr la interoperabilidad y sustitución de componentes. El uso de componentes a través de las interfases explícitas entre diferentes lenguajes, permite incrementar el reuso, la calidad del resultado y las capacidades de delegar tareas externas [16].

Establecidas las razones por las cuales se consideró utilizar el *middleware* estándar CORBA en capítulos anteriores. Revisemos algunas de las características básicas del

ORB comercial seleccionado en este caso de estudio. Se seleccionó ORBacus versión 4.22 de la empresa IONA®. El utilizar un ORB comercial, ayuda a crear aplicaciones flexibles, capaces de ser utilizadas en redes heterogéneas. Además de considerar aspectos relacionados a los lenguajes de programación soportados, herramientas de depuración, calidad de la documentación, soporte técnico y bajo costo. Algunas de sus características generales se mencionan a continuación:

- Facilidad de uso.
- Los servidores comienzan automáticamente por medio del repositorio de implementación.
- Referencias de objetos estilo URL.
- Balanceo de Carga - balance en las peticiones de los clientes entre un conjunto de objetos replicados y servidores ocupados.
- *Qualities of Service* tales como tolerancia a fallas, administración de la conexión activa, seguridad, módulos de carga dinámica, flexibilidad a través de protocolos de transporte tipo *plug-in*.
- IDL a *Hypertext Markup Language* (HTML).
- IDL a *Rich Text Format* (RTF).
- Los servicios de Nombres, Eventos y Propiedades vienen integrados en el producto.
- Puede interoperar con los Servicios de Notificación de Orbix, el *Orbix Trader* y los servicios *Orbix Telecom Logging*
- etc.

Una configuración básica para instalar este ORB comercial se anexa en el apéndice 4.

Se consideró tomar como caso de estudio un brazo manipulador robótico Motoman UP6, ya que se contaba con dos robots industriales instalados en el Laboratorio de Mecatrónica del Instituto, con la infraestructura adecuada para llevar pruebas de experimentación. Estos robots forman parte integral de las celdas didácticas de manufactura en este laboratorio.

El robot consiste básicamente de un brazo manipulador UP6, un controlador de robot modelo XRC2001, y un programador manual o *teach pendant*. Ver figura 5.1.

Figura 5.1: Brazo manipulador robótico Motoman UP6.

El controlador XRC2001 ofrece capacidades de comunicación mediante redes industriales tales como DeviceNet, ControlNet, Profibus-DP, e Interbus-S. Sin embargo la desventajas que este tipo de redes presentan, es que requieren adquirir múltiples módulos de *software* y *hardware* para cada tipo de controlador, además de que cada uno de estos componentes normalmente solo trabajan bajo determinadas y limitadas especificaciones relacionadas al proveedor o red específica. Al final, esto produce soluciones con un alto costo y una baja interoperabilidad entre dispositivos.

Igualmente, aunque Motoman provee *software* comercial para sus controladores tanto para aspectos de calibración, programación fuera de línea y tareas de comunicación e intercambio de archivos. Estos no proporcionan el grado de personalización e

interoperabilidad cuando se requieren ejecutar tareas o movimientos específicos o controlar el robot de manera remota, como en nuestro caso. Como un ejemplo podemos mencionar el *software* VDE (*Visual Data Exchange*) el cual provee comunicación RS-232C o Ethernet para los controladores MRC y XRC, pero con la restricción de que es una aplicación ejecutable sin posibilidades de modificación de acuerdo a necesidades específicas del cliente. Aunado a que la plataforma de trabajo del mismo está limitado a Windows. El programa está limitado a copiar archivos entre el controlador XRC y una PC, borrar archivos y expandir el área de almacenamiento de los trabajos del robot en la PC.

Probablemente el *software* de comunicación más avanzado para el controlador XRC sea MotoView®. Este programa provee una interfase *Web* interactiva que permite monitorear el *status*, los datos y las entradas/salidas del robot de manera remota, e igualmente provee al usuario final con capacidades de manipulación de archivos del robot. Utiliza Internet Explorer como navegador y está basado en Java. Esta aproximación presenta muy buenas ventajas, pero no provee todas que un sistema distribuido debe satisfacer.

5.2. Librería MotoCom SDK de Yaskawa®

Al iniciar este trabajo de investigación, se evaluaron diferentes aproximaciones con respecto a la manera de comunicarse con los controladores XRC2001. Previamente existía un trabajo de tesis desarrollado dentro del campus Monterrey, que trataba la comunicación con un robot manipulador Motoman Yasnac K3 usando un controlador MRC, mediante RS-232C. A partir de la experiencia y conclusiones de este trabajo se detectaron varios puntos adversos para la aproximación que se deseaba seguir.

El código fue desarrollado en Visual Basic 6.0. Este es un lenguaje de programación sencillo en su nivel básico, sin embargo en el manejo de comunicaciones, se vuelve más complicado, y no tiene la flexibilidad de otros lenguajes de programación como C++ [48].

Se requería conocer bien el protocolo de comunicación de Motoman que es el BSC (*Binary Synchronous Code*) donde en el envío de comandos se debe atener a las restricciones del protocolo. El enviar programas al controlador representaba igualmente una dificultad, ya que se requieren desarrollar librerías para el intercambio de diálogo variable entre el robot y la computadora [48].

Desde un principio esta aproximación no era la adecuada a seguir, para nuestro propósito final.

En el departamento de diseño de sistemas computacionales de la Universidad Tecnológica de Sydney (UTS), se generó una interfase de librerías capaz de proveer el *software* necesario para controlar un Motoman UPJ-3 conectado a un controlador JRC2001, vía RS-232 y utilizando el lenguaje de programación C++. El objetivo final era permitir a los desarrolladores construir aplicaciones en C++ y controlar el robot desde una PC de escritorio, sin la necesidad de aprender y desarrollar en el lenguaje nativo PAC del robot.

Esta interfase de librerías bajo ambiente Windows es requerida para compilar y cargar programas PAC al JRC2001. Entre las mejoras mencionadas al finalizar el proyecto era desarrollar una clase más portable que permitiera desarrollar aplicaciones en otros ambientes como pudiera ser GNU/Linux, UNIX, entre otros.

Esta aproximación era más cercana a la idea original a desarrollar. Se tenían librerías de comunicación desarrolladas en C++. Presentaban la ventaja de que ORBacus soporta la compilación de aplicaciones desarrolladas en este lenguaje. Sin embargo, estas librerías estaban limitadas al funcionamiento con un controlador JRC2001 y el lenguaje nativo PAC. Este formato es distinto al que utilizan los controladores de última generación de Motoman como es el INFORM II para los controladores XRC, MRC, ERC, etc.

Finalmente, se decidió utilizar una librería comercial, que permitiera minimizar tiempos de desarrollo y además cumpliera con una gran flexibilidad en su programación.

MotoCom SDK

Esta librería proporciona una manera de transmitir datos entre una PC y controladores de robots industriales Motoman (XRC, MRC, MRCII, ERC y ERCII). La librería está compuesta en la forma de un DLL (*Dynamic Link Library*) de Windows.

Esta librería de transmisión tiene las siguientes funciones generales:

- Función de transmisión de archivos.
- Función de control de movimientos del robot.
- Función DCI (*Data Communication Interface*).
- Función de lectura/escritura de señales de I/O.
- Otras funciones especializadas.

En los cuadros 5.2, 5.3, 5,4, 5.5 y 5.6 se describen estás funciones que son la base del desarrollo de las funciones *envolventes* para el desarrollo del cliente - servidor CORBA.

Todas estas funciones proporcionan la flexibilidad y robustez requerida para desarrollar una aplicación lo suficientemente operativa en cuanto al control de movimientos del robot en una una aplicación comercial o industrial.

Más información con respecto a las definiciones de estas funciones, la forma en que se encapsularon en componentes denominados *envolventes* se verán conforme se avance en esta documentación.

Cuadro 5.1: Descripción de las funciones MotoCom disponibles en alto nivel.

Función	Nombre de la función	Descripción
Funciones de transmisión de archivos de datos	BscDownload	Envía un archivo específico al controlador del robot.
	BscUpload	Recibe un archivo específico desde el controlador del robot.
Funciones de Control del Robot	BscFindFirst	Lee el nombre del primer trabajo de toda la lista registrada al momento.
	BscFindFirstMaster	Lee el primer nombre del trabajo de la lista que pertenecen al trabajo maestro.
Estatus de Lectura	BscFindNext	Lee el próximo nombre de trabajo registrado en el tiempo presente.
	BscGetCtrlGroup	Lee el control de grupo y la información de la tarea.
	BscDownLoad	Envía un archivo específico al controlador del robot.
	BscGetError	Lee un código de error o código de alarma.
	BscGetFirstAlarm	Lee un código de alarma y retorna el código del mismo.
	BscGetStatus	Lee la información de status.
	BscGetUFrame	Lee los datos del marco de usuario especificado.
	BscGetVarData	Lee variables.
	BscIsAlarm	Lee status de la alarma.
	BscIsCtrlGroup	Lee la información del grupo de control.
	BscIsCycle	Lee información en el modo playback.
	BscIsError	Lee status de errores.
	BscIsErrorCode	Obtiene el código de error.
	BscIsHold	Lee el status de *"hold"*.
	BscIsJobLine	Lee el número de línea del trabajo actual.
	BscIsJobName	Lee el nombre del trabajo actual.
	BscIsJobStep	Lee el número de paso del trabajo actual.
	BscIsLoc	Lee la posición actual del robot en pulsos o en el sistema XYZ.
	BscIsPlaymode	Lee el modo de operación.

Cuadro 5.2: Descripción de las funciones MotoCom disponibles... continuación 2.

Función	Nombre de la función	Descripción
	BscIsRemoteMode	Lee el comando.
	BscIsRobotPos	Lee la posición actual del robot en un sistema de coordenadas específico.
	BscIsTaskInf	Lee información de la tarea.
	BscIsTeachMode	Lee si está en el modo *"teach"* o en el modo *"play"*.
	BscJobWait	Espera la completación de un trabajo hasta que el movimiento del robot para o un tiempo específico expira.
	BscCancel	Cancela un error.
	BscChangeTask	Cambia una tarea.
	BscContinueJob	Comienza un trabajo La ejecución comienza desde la línea actual del trabajo en curso.
	BscConvertJobP2R	Convierte un trabajo en pulsos a un trabajo relativo en un sistema de coordenadas específico.
	BscConvertJobR2P	Convierte un trabajo relativo en un sistema de coordenadas especifico o trabajo por pulsos.
	BscDeleteJob	Borra un trabajo.
	BscHoldOff	Fija *"hold"* en apagado.
	BscHoldOn	Fija *"hold"* en encendido.
	BscImov	Mueve el robot con un movimiento lineal desde la posición actual en un valor incremental en un sistema de coordenadas especificado.
Funciones de transmisión de archivos de datos	BscMDSP	Envía un mensaje de datos.
	BscMov	Mueve el robot con un movimiento lineal desde la posición actual en valores incrementales en un sistema de coordenadas específico.

Cuadro 5.3: Descripción de las funciones MotoCom disponibles... continuación 3.

Función	Nombre de la función	Descripción
Funciones de Control del Robot	BscMovj	Mueve el robot con movimiento joint a una posición destino en un sistema de coordenadas específico.
	BscMovl	Mueve el robot con un movimiento lineal a una posición destino en un sistema de coordenadas específico.
Estatus de Lectura	BscOPLock	*"Interlocks"* el robot.
	BscOPUnLock	Libera el robot del estado *"Interlocks"*.
	BscPMov	Mueve el robot a una posición de pulsos específicos.
	BscPMovj	Mueve el robot a una posición de pulsos específicos con movimiento *joint*.
	BscPMovl	Mueve el robot a una posición de pulsos específicos con movimiento lineal.
	BscPutUFrame	Establece un marco de datos de usuario específico.
	BscPutVarData	Establece datos de variables.
	BscStartJob	Comienza un trabajo. Un trabajo a comenzar debe tener el nombre del último seleccionado.
	BscSelectJob	Selecciona un trabajo.
	BscSelectMode	Selecciona un modo *"teach"* o *"play"*.
	BscSelLoopCycle	Cambia el modo de ciclo a modo automático.
	BscSelOneCycle	Cambia el modo de ciclo a uno solo.
	BscSelStepCycle	Cambia el modo de ciclo, por pasos.
	BscSetLineNumber	Establece un número de línea del trabajo actual.
	BscSetMasterJob	Fija un trabajo como el trabajo maestro.

Cuadro 5.4: Descripción de las funciones MotoCom disponibles... continuación 4.

Función	Nombre de la función	Descripción
Soporte de la función DCI	BscSetCtrlGroup	Restablecer una alarma del robot.
	BscReset	Fija un grupo de control.
	BscServoOff	Establece la fuente de suministro del servo OFF.
	BscServoOn	Establece la fuente de suministro del servo ON.
	BscUpload	Recibe un archivo específico desde el controlador del robot.
	BscDCILoadSave	Carga o salva un trabajo con la instrucción DCI.
	BscDCILoadSaveOnce	Carga o salva un trabajo con la instrucción DCI.
	BscDCIGetPos	Obtiene una variable con la función DCI.
	BscDCIPutPos	Establece una variable con la función DCI.
Lectura/Escritura de las señales de I/O	BscReadIO	Lee una cuenta específica de status de las bobinas.
	BscWriteIO	Escribe una cuenta específica de estatus de las bobinas.
Otras Funciones	BscClose	Libera un manejador de comunicaciones.
	BscCommand	Envía un comando de comunicación.
	BscConnect	Conecta una línea de comunicación.
	BscDisconnect	Desconecta una línea de comunicación.
	BscDiskFreeSizeGet	Retorna la capacidad libre en una unidad de disco específico.
	BscGets	Envía una cadena de caracteres mediante transmisión en nivel TTY.
	BscInBytes	Retorna el número de caracteres los cuales son recibidos mediante transmisión TTY.

Cuadro 5.5: Descripción de las funciones MotoCom disponibles... continuación 5.

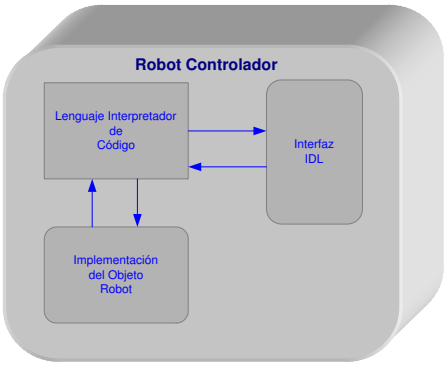
Función	Nombre de la función	Descripción
	BscIsErrorCode	Obtiene un código de error.
	BscOpen	Obtiene un manejador de comunicaciones.
	BscOutBytes	Retorna el número restante de caracteres los cuales son enviados mediante transmisión TTY.
	BscPuts	Envía una cadena de caracteres mediante transmisión en nivel TTY.
	BscSetBreak	Cancela la transmisión.
	BscSetComm	Establece un parámetro de comunicación.
	BscSetCondBSC	Establece un temporizador de control de comunicación.
	BscStatus	Lee el status.
Cuadros adaptados de [35].		

5.3. Estructura del Sistema Robótico Distribuido basado en CORBA

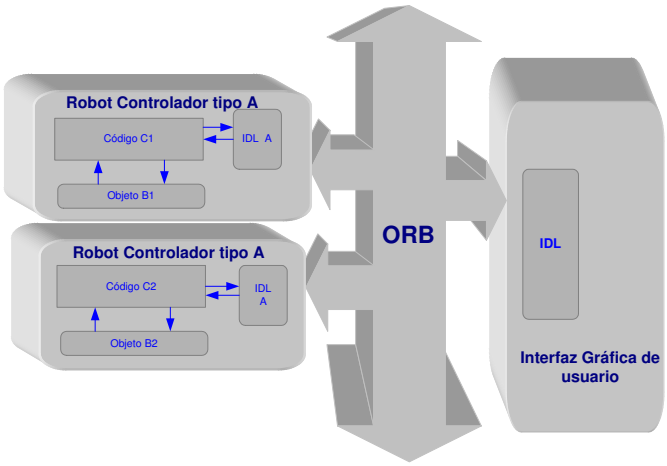
Tal como se mencionó en la introducción, se tomaron como base algunas de las recomendaciones y propuestas de trabajos establecidos por Guedea et al., en las publicaciones de [21], [55] y [49]. Se adopta una estructura para el desarrollo de sistemas robóticos distribuidos basados en el estándar CORBA, establecido en [22].

El esquema de la fig. 5.2 muestra algunas de las oportunidades que está arquitectura presenta, para integrar robots industriales u otros tipos de componentes (sistemas de visión, sensores táctiles, actuadores finales, etc.), en un sistema abierto. Desarrollar el componente *envolvente* del controlador del robot para el código intérprete, es el elemento que requiere un mayor esfuerzo al implementarlo. El desarrollo de este componente es la parte central de la aportación en este proyecto de investigación.

En este trabajo se usa el concepto de encapsulamiento de funciones para crear un conjunto de objetos genéricos que faciliten el desarrollo e integración de sistemas



(a)



(b)

Figura 5.2: (a) Elementos básicos en un componente *envolvente*, (b) Dos componentes del tipo A de dos controladores de robot accesados a través del mismo tipo de interfase en el módulo GUI.

distribuidos usando la especificación y servicios estándar CORBA [10]. A partir de las funciones proporcionadas por MotoCom SDK, éstas deben encapsularse lo suficiente para igualarse a los tipos de datos manejados en las interfases IDL.

El trabajo es una primera etapa para proveer una estructura integral para sistemas robóticos distribuidos cooperativos y más autónomos. Por lo que la aproximación inicial es crear un sistema robótico operado de manera remota, usando componentes de objetos genéricos o "bloques de construcción" que puedan ser usados en diferentes tipos de equipos.

Esta meta se puede lograr usando funciones que traten con los detalles específicos de cada equipo pero que dan una funcionalidad más simple y poderosa a otros componentes del sistema.

El concepto de componentes *envolventes* es un concepto basado en una especificación de *middleware* estándar. Estos son módulos orientados a objetos que crean una interfaz abstracta para una clase especifica de componentes *software* y/o *hardware* [22].

5.4. Componentes *Envolventes*

Soo Kim et al., introduce el concepto de *objeto envolvente* el cual está relacionado con nuevos paradigmas en integración de sistemas que incluyen *layering*, migración, reingeniería, ingeniería inversa e ingeniería directa y describe como pueden ser aplicados [30].

Las técnicas para producir *componentes envolventes* proveen un medio natural para integrar sistemas heterogéneos. En el presente trabajo de investigación se específica la siguiente configuración básica de un *componente envolvente*. Este consiste de tres bloques principales: *la interfaz IDL, el código de Interpretación/Transformación y la librería de implementación del objeto software y/o hardware.*

La figura 5.2 muestra estos bloques, donde un controlador de robot es considerado para la implementación objeto del *software/hardware*.

5.4.1. La interfase IDL

Es una definición de interfase para una clase particular de componentes, su definición es un problema importante para construir componentes reusables y de fácil conexión. Se definen tres funcionalidades básicas para esta interfase: *abstracción, monitoreo y configuración.*

Por *abstracción* se refiere al conjunto de funciones particulares (métodos) que un componente de tipo clase específico debe tener. Esto sin tomar en cuenta los detalles de la implementación.

Por *monitoreo*, se refiere a las funciones generales que todo componente debe tener para consultar sus estados internos.

Finalmente, la *configuración* representa la capacidad de cambio de los atributos internos del componente acorde a los requerimientos externos.

5.4.2. Código de Interpretación/Transformación

Este elemento es el que requiere mayor esfuerzo cuando se implementa en un componente *envolvente*. Este requiere que se hagan todas las transformaciones e interpretaciones de los datos, es decir, envolver todas las diferencias entre la interfase y la implementación del objeto. Por ejemplo, si la implementación del objeto maneja la posición de un eje del robot usando un dato del tipo *long integer*, pero la interfase está declarada como un dato del tipo flotante, el código de transformación hace los cambios en los tipos de datos en ambos lados.

5.4.3. Implementación del objeto Hardware/Software

La integración dentro del sistema total se requiere para este elemento. Usualmente, está definido para un *hardware* específico o es proveído por el fabricante. La mayoría de las veces, este componente se entrega como un API. Sin embargo, es difícil para propósitos específicos y algunas veces imposibles acceder al código de bajo nivel. Ejemplos de estos elementos puede ser rutinas para el control del movimiento del robot,

librerías de procesamiento de imágenes o de bases de datos, entre otras.

5.5. Características de los componentes *envolventes*

Las características estándar de estos componentes son reusabilidad, conectividad, generalización y flexibilidad, aunado a la abstracción y manipulación.

Usando la especificación CORBA permite lograr la conectividad al menos en tres aspectos:

- Plataforma
- Sistemas Operativos
- Lenguajes de Programación

La abstracción y la reusabilidad son propiedades relacionadas. La reusabilidad busca el obtener módulos intercambiables, mientras que la abstracción ayuda a crear estos módulos. Entre más abstracto sea el módulo, más general es la definición. Por otro lado, si se logra manipular un componente en el momento, entonces se puede lograr cierto grado de flexibilidad.

5.6. Metodología de Desarrollo e Integración de Componentes *Envolventes*

El propósito de la metodología es proveer una guía de desarrollo e integración de *componentes envolventes* para brazos robóticos articulados Motoman. Se considera que la metodología propuesta es lo suficientemente general para extender su aplicación a diversas marcas de robots industriales y aún periféricos externos. En la literatura varios autores han establecidos varios métodos y aproximaciones para desarrollar *componentes envolventes* en ambientes distribuidos, sin embargo estos se concentran en aspectos

generales relacionados con seguridad entre *componentes envolventes* [17], [56]. La migración de sistemas software heredados (*legacy systems*) a ambientes distribuidos CORBA como en [30], [54] y [15]. Así como Hughes et al., presenta una metodología general para migrar aplicaciones heredadas a sistemas administrados por objetos distribuidos bajo el concepto de *componentes envolventes* [25].

No es objetivo de este trabajo establecer comparativos experimentales con estos métodos existentes, solo establecer referencias con respecto a nuestra aproximación.

La metodología propuesta consiste de ocho pasos que se concentran en establecer sistemas con la suficiente flexibilidad e heterogeneidad, así como reducir tiempos de desarrollo en aplicaciones futuras.

1. *Identificar las funciones y servicios operativos más importantes del robot.*

Esto es, establecer los movimientos articulados, lineales o rotativos que se requieren implementar. Los componentes *envolventes* deben permitir cierto nivel de manipulación y reconfiguración con el fin de ser flexibles. Esto se logra por medio de funciones de configuración definidos en la interfase IDL.

2. *Establecer funciones abstractas, con una aproximación orientada al usuario.*

Una función abstracta se refiere a la manera como el componente es concebido. Por ejemplo, el brazo robótico puede ser visualizado como un conjunto de ejes, donde cada uno de ellos tiene un conjunto de propiedades, tales como un rango máximo y mínimo de movimiento, tipo de movimiento (articulado vs. lineal), etc. Esta es una aproximación genérica que aplica a casi todos los tipos de brazos manipuladores robóticos, pero el esfuerzo para desarrollar una aplicación pasa a otro nivel superior de diseño. Por otro lado, un brazo manipulador puede ser visualizado como un equipo que puede realizar un conjunto de funciones tales como, extender/retraer el brazo, girar la muñeca o la base del robot, moverse arriba/abajo, etc. Esta es una aproximación genérica también, donde el esfuerzo principal está en el desarrollo del componente pero la complejidad en el nivel más alto se ve reducida.

3. *Establecer un número de funciones base.*

Entre menor número de funciones, más fácil será la conexión. Definiendo una cantidad pequeña pero poderosa de funciones entre las interfases de los diferentes componentes da mayor oportunidad de establecer una buena comunicación. Sin embargo, el esfuerzo requerido para conseguir esta buena comunicación es una tarea que requiere tiempo.

4. *Establecer el ambiente de desarrollo de las aplicaciones.*

Identificar el ambiente de desarrollo y las herramientas requeridas para desarrollar las aplicaciones (cliente - servidor). Es aquí donde se selecciona el ORB, los lenguajes de programación, plataformas, etc.

5. *Programación de componentes* envolventes.

Utilizar principios de programación orientados a objetos. Donde se establecen etapas de diseño, desarrollo, instalación, depuración y ejecución de cada una de los componentes. La figura 5.3 representa estas diferentes etapas.

6. *Integración de componentes* envolventes *en arquitectura robótica distribuida.*

Los diversos servicios disponibles en CORBA, permiten seguir aproximaciones distintas. Esto hace posible ir extendiendo las capacidades del sistema robótico a desarrollar. Por lo que una primera solución base puede consistir en desarrollar un cliente GUI y un servidor robot. Estos comunicados mediante una interfaz IDL estándar y utilizando un servidor de nombres.

7. *Evaluación del impacto de la arquitectura utilizada y aplicaciones desarrolladas.*

Identificar y describir el impacto de las diferencias entre la ejecución de la aplicación desarrollada con respecto a una programa del tipo *stand alone*, u operación manual mediante el *teach pendant* por mencionar un ejemplo.

8. *Documentar el sistema robótico distribuido.*

Esto es describir las definiciones de los objetos, las interacciones, clases, etc. Para un mejor entendimiento, mantenimiento y expansiones futuras.

Se presenta el desarrollo de esta metodología utilizando como caso de estudio el brazo manipulador robótico Motoman UP6. Los primeros cuatro pasos son generales y de alguna manera se desarrollan intrínsecamente durante la documentación de este trabajo de investigación. Por lo que se considera más importante los pasos cinco a siete. En donde se explica la forma en programar los componentes *envolventes* y las etapas de integración y experimentación de estos componentes *envolventes* en el caso de estudio.

5.7. Programación de Componentes *Envolventes* del Sistema Robótico Distribuido

La figura 5.3 muestra de manera gráfica las distintas etapas requeridas para desarrollar e implementar el sistema robótico distribuido.

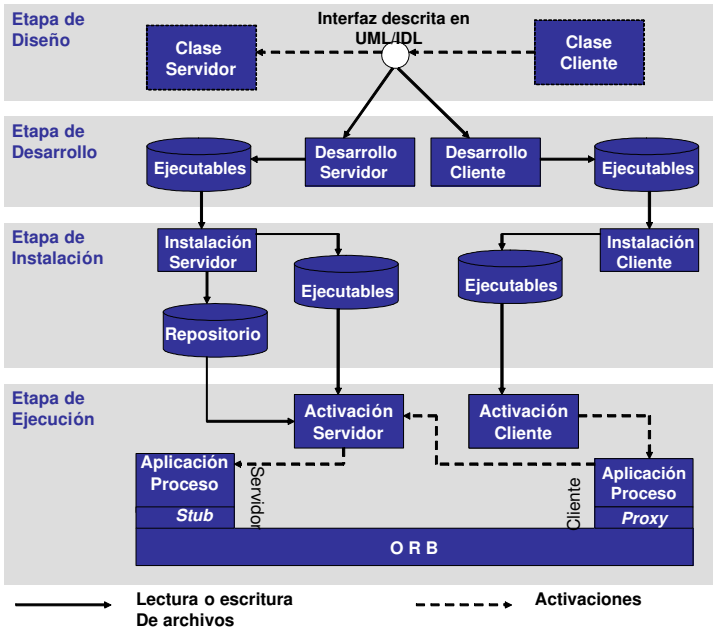


Figura 5.3: Etapas de desarrollo de los componentes del sistema distribuido. Adaptado de [41].

Etapa de Diseño

La etapa de diseño donde se describen los modelos que componen las distintas clases del cliente y servidor CORBA, se trataron en el capítulo 4. Ahí se muestran los diagramas UML básicos del sistema propuesto y una descripción más detallada de sus atributos y métodos se muestran en el apéndice 1. Con respecto a la estructura de control del sistema robótico propuesto por Guedea et al., este ya fue introducido previamente.

Etapa de Desarrollo

Una de las metas principales del proyecto es muy simple: mover el robot UP6 de manera remota. Para esto el usuario final requiere un medio específico para manipular el robot. La manera tradicional de manipular y programar un robot es mediante el *teach pendant*, sin embargo pueden existir tareas y/o aplicaciones, en los cuales esto no sea recomendable ó posible; ya sea por cuestiones de complejidad o actividades que puedan producir algún peligro para el usuario final. Esto es solo una generalización de la necesidad de operar de manera remota un dispositivo robótico.

Ahora bien, el medio más utilizado en la actualidad debido al uso extensivo de las tecnologías computacionales y el Internet en todos los ámbitos y dominios, son las interfases gráficas de usuario (GUI). Estas deben ser desarrolladas de manera que permitan a un operador, manipular remotamente un dispositivo robótico de una manera simple y amigable, pero al mismo tiempo robusta.

Se desarrolló un cliente GUI que agrupa en su interfaz las funciones *envolventes* CORBA definidas mediante IDL. Y en el servidor robot se llevó a cabo la implementación de cada una de estas funciones. Una cuadro descriptivo de estas funciones *envolventes* se muestra en 5.7.

Sin embargo por cuestiones de comodidad, programación, experimentación y depuración del sistema propuesto. Es recomendable generar primeramente una aplicación del tipo *stand alone* que agrupe el cliente-servidor en uno sólo.

Cuadro 5.6: Funciones *envolventes* principales para el control del Motoman UP6.

Funciones	Argumentos	Descripción
Extend	Distancia	Extiende el brazo una distancia específica.
Retract	Distancia	Retrae el brazo una distancia específica.
Move-H	Distancia, Dirección	Mueve el brazo sobre un eje una distancia específica.
Move-V	Distancia, Dirección	Mueve el brazo hacia arriba o hacia abajo, una distancia específica.
Turn	Grados, Dirección	Gira la base del brazo manipulador un ángulo específico.
Turn-G	Grados, Dirección	Gira el actuador final un ángulo específico.
Turn-W	Grados, Dirección	Gira la muñeca un ángulo específico.
MoveTo	Posición	Se mueve a una posición previamente definida.
Learn	Posición	Guarda la posición actual del brazo en una variable.
Ready	NA	Mueve el brazo a una posición inicial.
Home	NA	Mueve el brazo a una posición de *Home*.
Gripper	Posición	Abre o cierra el *"Gripper"*.
Speed	Velocidad	Fija la velocidad de todos los movimientos.

Se utilizó como punto de partida, el código base anexado en la documentación de la librería MotoCom SDK. Este código contenía algunas funciones implementadas para la comunicación con el controlador XRC mediante lenguaje C++; así como para la transferencia de archivos de trabajos (.JBI) del robot a una computadora personal y viceversa.

La primera parte del desarrollo experimental, consistió en establecer una comunicación exitosa mediante RS-232C y Ethernet con el controlador XRC; utilizando este código base. Esta etapa, probablemente fue una de las cuales, mayor tiempo consumió para lograr un resultado adecuado.

La configuración de las comunicaciones tanto serial como por Ethernet, son funciones adicionales que requieren activación por parte del proveedor del brazo robótico. Los manuales de usuario no engloban toda la información requerida para establecer una configuración adecuada. Por lo que se requirió una búsqueda bibliográfica, consulta de manuales de usuario y de programación, de configuración de *hardware*, y aún de soporte técnico externo para finalmente producir una comunicación adecuada.

Dependiendo del controlador del robot a utilizar (ERC, MRC or XRC) los parámetros del puerto serial interno o del *teach pendant* deben ser correctamente inicializados. Estos parámetros difieren y deben de configurarse correctamente, dependiendo si se usa el puerto serial interno del controlador o el externo mediante el *teach pendant*. En caso de que se utilice Ethernet como medio de comunicación, se debe instalar una tarjeta Ethernet dentro del controlador de tal manera que los dispositivos se comuniquen mediante el protocolo TCP/IP. Los parámetros del controlador deben ser inicializados con una dirección IP, una máscara de subred, y en caso de usar Internet, debe especificarse el *"default gateway"* y la dirección del servidor. Después de llevar a cabo la configuración adecuada, el servidor del robot solo requiere la dirección IP del controlador para establecer una correcta comunicación. Una guía para la configuración básica mediante Ethernet se presenta en el apéndice C. Se recomienda consultar la información contenida en el manual Motocom SDK y en el suite de comunicación VDE®, ya que contienen información importante acerca de la configuración *hardware* y de los

parámetros a modificar del robot para las comunicaciones.

Finalmente en este aspecto, hay que mencionar que para llevar a cabo una comunicación serial RS-232C entre el controlador XRC2001 y la PC. Es necesario deshabilitar la tarjeta Ethernet interna del controlador. En la experimentación que se llevó a cabo no se logró establecer una comunicación serial mientras la tarjeta Ethernet estuviera activada.

En el cuadro 5.7 se presenta la configuración del cable serial RS-232C utilizado para fines prácticos.

Una vez solventado estos problemas, se desarrolló la programación de las funciones de toma de control del XRC2001, las funciones básicas para activar y desactivar los modos *teach/play*, *servo on/off*, el control de las alarmas, *hold on/off*, etc. En este aspecto es importante mencionar que para controlar de manera remota la función de *servo on/off*, se requiere instalar un *jumper* via *hardware* en el controlador en cuestión, esto de acuerdo a información contenida en el manual de operación. Sin embargo es posible controlar el robot remotamente, si se activa el modo *play* y se encienden los servos al inicializar la sesión por primera vez, esto por medio del panel de control principal en el controlador. Una vez prendidos y establecido el modo remoto, pueden ser activados y desactivados por *software*.

El cuadro 5.8 muestra las funciones C++ disponibles en las librerías MotoCom SDK y la manera en que muchas de estas fueron encapsuladas mediante las funciones *envolventes* en CORBA. Esto se lleva a cabo con la finalidad de implementar la interfase IDL previamente definida. Algunas de estas funciones son agrupadas en más de un componente para lograr el estatus o movimiento requerido. Igualmente, hay varias aproximaciones para producir un movimiento específico, cual usar depende de la tarea específica a realizar. Por lo que el uso de una determinada función de MotoCom a utilizar depende de la mejor y más eficiente aproximación.

Cuadro 5.7: Configuración Null Modem RS-232C para conexión XRC2001-PC.

Descripción	9 Pin (Macho)	9 Pin (Hembra)
FG (Tierra de Chasis)	-	-
TD (Transmitir Datos)	3	2
RD (Recibir Datos)	2	3
RTS (Petición de envío)	7	8
CTS (Listo Para Envío)	8	7
SG (Señal de Tierra)	5	5
DSR (Listo para Fijar Datos)	1, 6	4
DTR (Terminal de Datos Lista)	4	6, 1

Cuadro 5.8: Funciones MotoCom implementadas como funciones *envolventes* CORBA.

Función	Nombre de la función en MotoCom SDK	Componente *Envolvente* CORBA
Funciones de transmisión de archivos de datos	BscDownload	-
	BscUpload	-
Funciones de Control del Robot Estatus de Lectura	BscFindFirst	-
	BscFindFirstMaster	-
	BscFindNext	-
	BscGetCtrlGroup	-
	BscDownLoad	-
	BscGetError	Usadas en Todas
	BscGetFirstAlarm	Usadas en Todas
	BscGetStatus	Learn
	BscGetUFrame	-
	BscGetVarData	Learn
	BscIsAlarm	Ready
	BscIsCtrlGroup	-
	BscIsCycle	-
	BscIsError	Usadas en Todas
	BscIsErrorCode	Usadas en Todas
	BscIsHold	Ready
	BscIsJobLine	-
	BscIsJobName	-
	BscIsJobStep	-
	BscIsLoc	Usadas en Todas
	BscIsPlaymode	Ready
	BscIsRemoteMode	Ready
	BscIsRobotPos	Usadas en Todas
	BscIsTaskInf	-
	BscIsTeachMode	Ready
	BscJobWait	-

Cuadro 5.9: Funciones MotoCom como funciones *envolventes*... continuación 2.

Función	Nombre de la función en Motocom	Componente *Envolvente* CORBA
Control del Sistema	BscCancel	-
	BscChangeTask	-
	BscContinueJob	-
	BscConvertJobP2R	-
	BscConvertJobR2P	-
	BscDeleteJob	-
	BscHoldOff	Ready
	BscHoldOn	-
	BscImov	Extend, Retract, Speed
	BscMDSP	-
	BscMov	Move H, Move V, Speed
	BscMovj	Home, MoveTo, Move H, Move V, Speed
	BscMovl	Home, MoveTo, Speed
	BscOPLock	-
	BscOPUnLock	-
	BscPMov	Turn, Turn-G, Turn-W, Speed
	BscPMovj	Turn, Turn-G, Turn-W, Speed
	BscPMovl	Turn, Turn-G, Turn-W, Speed
	BscPutUFrame	-
	BscPutVarData	Learn
	BscStartJob	Gripper
	BscSelectJob	Gripper
	BscSelectMode	Ready
	BscSelLoopCycle	-
	BscSelOneCycle	-
	BscSelStepCycle	Ready
	BscSetLineNumber	-
	BscSetMasterJob	-
	BscSetCtrlGroup	-
	BscServoOff	Ready
	BscServoOn	Ready
	BscUpload	-
Soporte de la función DCI	BscDCILoadSave	-
	BscDCILoadSaveOnce	-
	BscDCIGetPos	MoveTo
	BscDCIPutPos	Learn
Lectura/Escritura de las señales de I/O	BscReadIO	-
	BscWriteIO	-

Cuadro 5.10: Funciones MotoCom como funciones *envolventes*... continuación 3.

Función	Nombre de la función en Motocom	Componente *Envolvente* CORBA
Otras Funciones	BscClose	Ready
	BscCommand	Ready
	BscConnect	Ready
	BscDisconnect	Ready
	BscDiskFreeSizeGet	-
	BscGets	-
	BscInBytes	-
	BscIsErrorCode	Ready
	BscOpen	Ready
	BscOutBytes	-
	BscPuts	-
	BscSetBreak	Ready
	BscSetComm	Ready
	BscSetCondBSC	Ready
	BscStatus	-

Ahora bien es importante presentar el formato, tipos de argumentos de entrada y retorno requeridos por las funciones de MotoCom, para realizar la conversión al tipo de dato adecuado en su implementación como componente *envolvente* en CORBA.

Se presentan dos funciones básicas MotoCom implementadas en el proyecto. Esto por cuestiones de espacio y de derechos reservados por Yaskawa®. La documentación completa de las mismas se puede consultar en [35].

La función BscIsLoc, es una de las más utilizadas dentro de la implementación de las funciones *envolventes* en CORBA, prácticamente está presente en todas las funciones que tienen que ver con el control de movimientos del robot tanto lineales como por articulaciones. Ësta regresa la posición del brazo robótico en valores de pulsos como en coordenadas cartesianas. El valor de retorno es un puntero que almacena el valor de la posición actual.

Cuadro 5.11: Función MotoCom: BscIsLoc.

Función	Lee la posición actual del robot en pulsos o en el sistema de coordenadas XYZ.	
Formato	declspec dllexport short APIENTRY BscIsLoc short nCid, short ispulse, short *rconf, double *p	
Argumentos	IN (Transferencia)	
	nCid	Número ID del manejador de comunicación.
	ispulse	0: Sistema de coordenadas cartesianas; 1: Sistema de coordenadas angulares.
	*rconf	Puntero de almacenamiento de la forma.
	*p	Puntero de almacenamiento de la posición actual.
	OUT (Retorno)	
	*rconf	Puntero de almacenamiento de la forma.
	*p	Puntero de almacenamiento de la posición actual.
	Valor de retorno.	
	-1: 0:	Falla obtenida. Completada Normalmente.

Cuadro 5.12: Función MotoCom: BscPMovj.

Función	Mueve el robot una posición en pulsos específica con movimiento articulado.	
Formato	declspec dllexport short APIENTRY BscPMovj short nCid, double sped, short toolno, double *p	
Argumentos	IN (Transferencia)	
	nCid spd toolno *p	Número ID del manejador de comunicación. Mover a velocidad de 0.1 a ???.? mm/seg, 0.1 a ???.? °/seg. Número de herramienta. Puntero de almacenamiento de la posición destino.
	OUT (Retorno)	
	Ninguno	
	Valor de retorno	
	0: Otros:	Completada Normalmente. Código de Error.
Comentarios	Posición Destino	
	Los datos de la posición destino se representan como sigue: P[0] P[1] P[2] P[3] P[4] P[5] P[6] P[7] P[8] P[9]	 Números de pulsos, eje S. Números de pulsos, eje L. Números de pulsos, eje U. Números de pulsos, eje R. Números de pulsos, eje B. Números de pulsos, eje T. Números de pulsos, eje 7. Números de pulsos, eje 8. Números de pulsos, eje 9. Números de pulsos, eje 10.

Cuadro 5.13: Rango de Pulsos Máximos y Mínimos para el UP6.

Eje	Número Máx. Positivos de Pulsos	Número Máx Negativos de Pulsos
L	203,813	-121,000
S	264,444	-264,444
B	181,818	-149,610
T	149,610	-121,000
R	160,000	-160,000
U	134,444	-107,555

En el cuadro anterior se listan un aproximado de los rangos de pulsos máximos y mínimos para el robot UP6, para cada eje articulado. Estos valores se obtuvieron de manera práctica, moviendo a sus posiciones máximas y mínimas para cada uno de los ejes.

Es recomendable verificar la posición cero para cada eje del robot, ya que a partir de esto y de acuerdo a los rangos de trabajo dados en grados, puede calcularse el valor teórico exacto de pulsaciones. Esto será de utilidad para posteriormente realizar los cálculos de las funciones *envolventes* CORBA de extender y retraer.

La función en C++, BscPMovj es también una de las funciones más utilizadas en nuestro caso de estudio. Este permite mover el robot una posición de pulsos específicos para cualquier eje articulado a una velocidad determinada por el usuario.

5.8. Desarrollo de funciones *envolventes Extender* y *Retraer*

Se presentan a continuación el desarrollo de la funciones *envolventes* Extender y Retraer. Estas permiten el movimiento simultáneo de dos ejes articulados, mediante los cuales se pueden implementar movimientos que de otra manera no podrían realizarse de manera manual, o la programación de este tipo de movimientos mediante la pro-

gramación INFORM II, requiere implementar subrutinas más elaboradas para llevar a cabo este tipo de movimientos.

La característica importante de estas dos funciones, son los cálculos geométricos que se implementan para realizar estos dos tipos de movimientos. Esto permite que a partir de la longitud dada de los ejes L y U para el robot Motoman UP6, así como la orientación de giro determinado para estos ejes. Permita realizar una extensión y/o retracción del brazo en una distancia x. Manteniendo una misma posición del actuador final, a una altura fija o manteniendo un ángulo dado.

Este cálculo geométrico, es genérico independientemente de la marca o proveedor del brazo robótico. Solo requerirá definir las nuevas longitudes de las articulaciones y la orientación de giro de estas, para producir el nuevo resultado.

5.8.1. Cálculos para comandos de movimientos *Extender* y *Retraer*

El cálculo se basa en el movimiento de dos ejes móviles, además de un eje fijo para el caso del UP6 tal como se indica en las figuras 5.4, 5.5 y 5.6. Donde a y n son las longitudes de los segmentos móviles y m es el estático. θ_2 y θ_u es la posición angular de los ejes móviles; mientras que m y n forman un ángulo recto . La altura h se define desde la posición final del segmento 3 con respecto a la línea imaginaria del centro de referencia que empieza en el eje 1. La distancia d indica el valor de la posición final del segmento 3 con respecto al origen.

Los valores de θ_1 y θ_2 están relacionados con el valor angular de las uniones, cuyo valor inicial se especifica en el manual de referencia del UP6 [13].

Dada las referencias anteriores tenemos que los valores de los ángulos θ_1 y θ_2 están dados por las fórmulas siguientes:

$$\theta_1 = 90 - \theta_l \tag{5.1}$$

$$\theta_2 = 90 - \theta_l + \theta_u + \theta_x \tag{5.2}$$

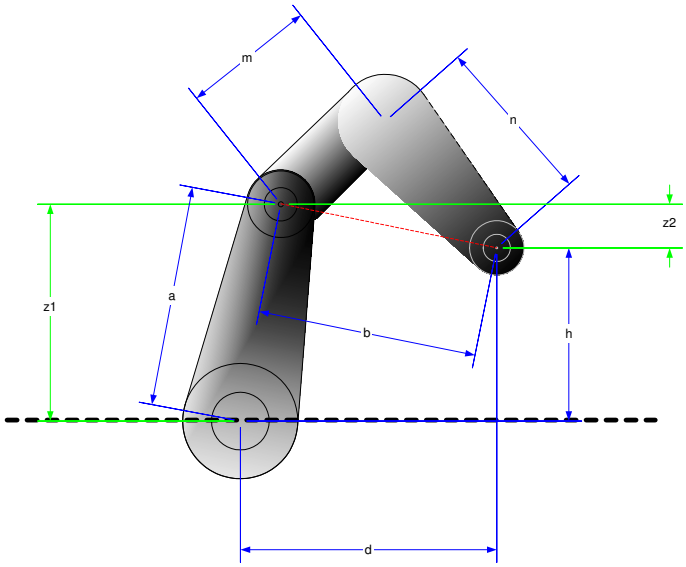


Figura 5.4: Esquema de una articulación de dos ejes.

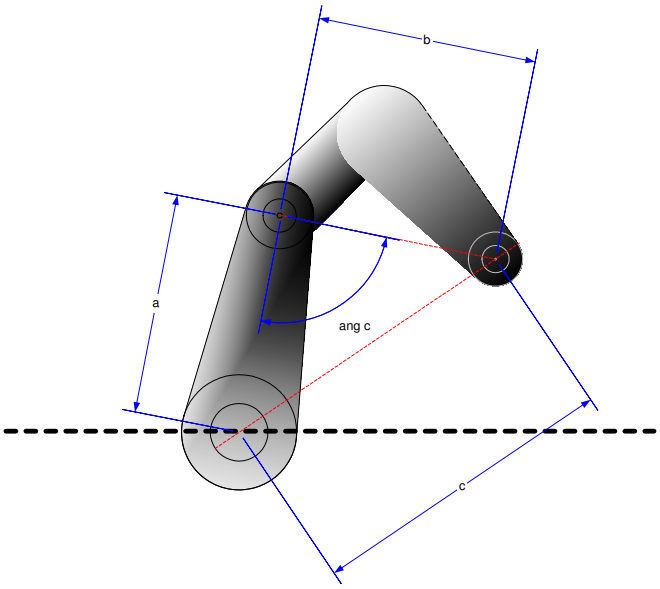


Figura 5.5: Esquema de los segmentos de dos articulaciones.

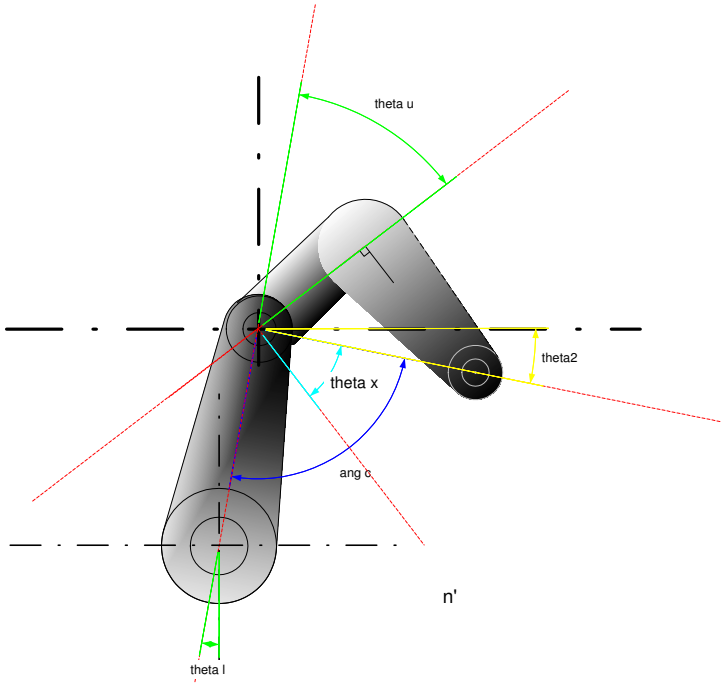


Figura 5.6: Representación de los ángulos en las articulaciones.

Los valores que se conocen son las longitudes a, m y n así como los ángulos θ_1 y θ_2. Dada esta información, los valores que se calculan son el lado c, el ángulo c y la altura h de acuerdo a la siguiente formulación.

Valor del ángulo C

$$\angle_c = \theta_x + \theta_z \tag{5.3}$$

$$\angle_c = \theta_x + 90 + \theta_u \tag{5.4}$$

Valor del lado c se encuentra dado por:

$$c^2 = a^2 + b^2 - 2ab\cos(\angle_C) \tag{5.5}$$

Valor de la altura h

$$h = z_1 h = z_1 + z_2 z_1 = a * \sin(\theta_1) z_2 = b * \sin(\theta_2) h = a * \sin(\theta_1) + b * \sin(\theta_2) \tag{5.6}$$

Valor de la distancia d

Puede ser encontrado usando dos formulaciones:

$$d = \sqrt{c^2 + h^2} d = a * \cos(\theta_1) + b * \sin(\theta_2) \tag{5.7}$$

Valor de los ángulos θ_B *y* α

$$\angle B = \arccos \frac{b^2 - a^2 - c^2}{-2ac} \tag{5.8}$$

$$\angle \alpha = \theta_1 - \angle_B \tag{5.9}$$

Una vez que se tienen estos valores se pueden calcular los nuevos ángulos de acuerdo a los incrementos o decrementos deseados en altura (movimiento vertical) o en extensión (movimiento de extender o contraer el brazo).

5.9. Caso: Cálculo cuando ocurre un cambio en la extensión

$$C_n = c + \triangle\ c \tag{5.10}$$

Cuando se pide un cambio en la extensión, se puede efectuar dicha extensión en tres formas:

- Una extensión manteniendo el ángulo del lado c.
- Una extensión manteniendo la altura actual h.

- Una extensión manteniendo la distancia actual d.

Esto se puede visualizar en las figuras 5.7, 5.8 y 5.9.

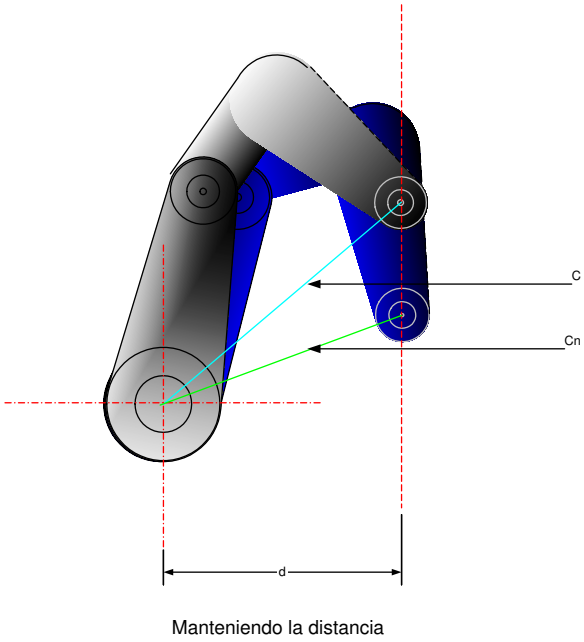


Figura 5.7: Manteniendo la distancia.

5.9.1. Cambio en extensión manteniendo ángulo

En esta opción tanto el valor de h como el valor de d se ven afectados de acuerdo a la siguiente formulación: (realmente es irrelevante para el movimiento del robot).

$$h_n = c_n \sin \alpha \tag{5.11}$$

$$d_n = c_n \cos \alpha \tag{5.12}$$

El valor del ángulo C se calcula de acuerdo a

$$\theta_{CN} = \arccos \frac{{c_n}^2 - b^2 - a^2}{2ab} \tag{5.13}$$

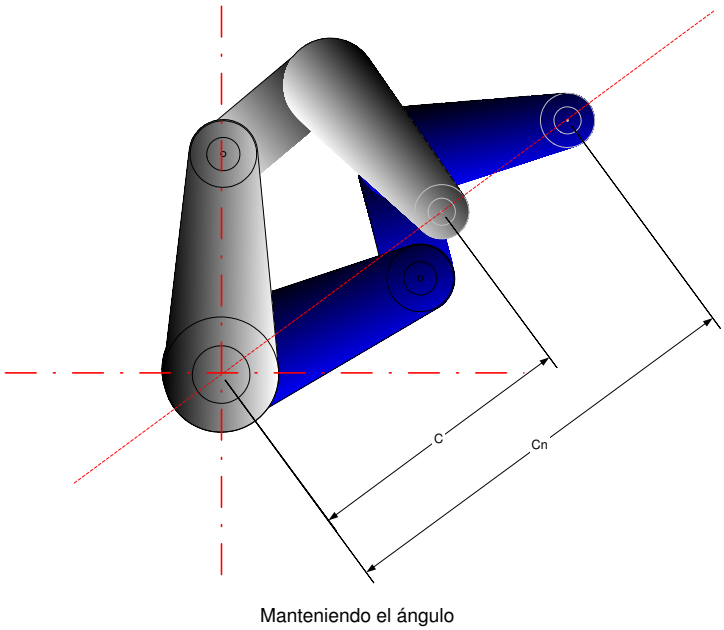


Figura 5.8: Manteniendo el ángulo.

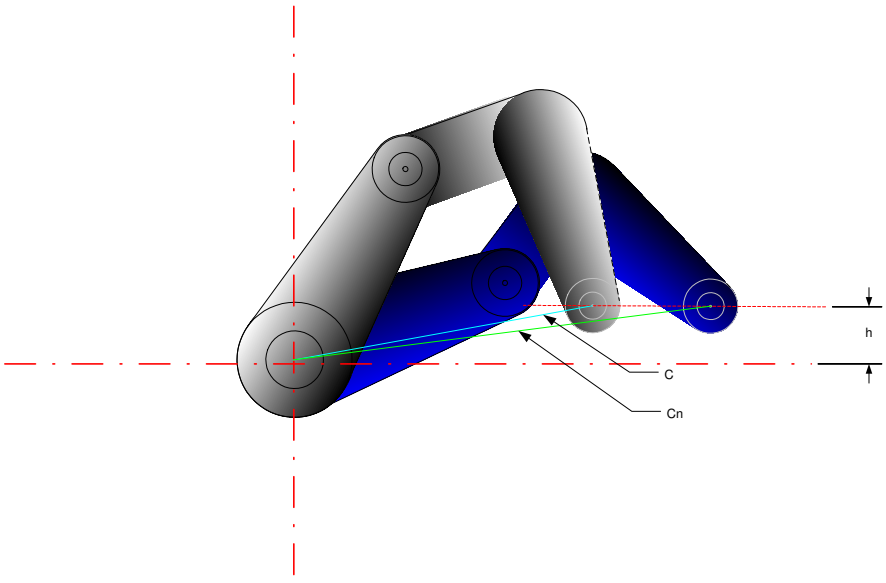


Figura 5.9: Manteniendo la altura.

El valor del ángulo θ_H no se cambia pero el valor del ángulo θ_B se cambia de acuerdo a

$$\angle B = \arccos \frac{b^2 - a^2 - {c_n}^2}{-2ac_n} \tag{5.14}$$

El ángulo θ_1 es la suma de los ángulos $\angle B$ y α, así que el valor a colocar en la unión 1 es el siguiente:

$$\theta_1 = \theta_B + \angle\alpha \tag{5.15}$$

$$\theta_{l_n} = 90 - \theta_{1n} \tag{5.16}$$

$$\theta_{1_n} = \angle\alpha + \theta_{bn} \tag{5.17}$$

El valor de la unión 2 está dada por el ángulo C.

$$\theta_{J2} = \theta_{CN} - \theta_x - 90 \tag{5.18}$$

Lo anteriormente visto, explica un movimiento para extender el brazo del robot de manera simultánea en los ejes a, m y n, dada una distancia en milímetros por el usuario final. El código de implementación del objeto CORBA para un movimiento de retracción puede observarse en el apéndice B. Los cálculos son prácticamente iguales que en un movimiento de extender, solo que se calculan en un sentido inverso.

Ahora se introduce la implementación de la función *gripper on/off*. Esta función requirió una aproximación distinta a la utilizada en los demás objetos. Se necesitaba activar y desactivar un actuador final, sin embargo se tenía la limitación con respecto a las funciones MotoCom disponibles para la lectura y escritura de las señales de entrada/salida (I/O). Existen dos tipos de funciones para esto: *BscReadIO* y *BscWriteIO*. La primera permite leer una amplio rango de direcciones de señales (desde 0xxx - 9xxx para el XRC2001). Solo que la segunda está limitada a solo escribir en los números de

dirección 9xxx. Esto corresponde a los rangos de señal del *Pseudo Input Signal*. Aunque pudo darse el caso de cambiar la dirección de I/O asignado al gripper. Esto hubiera representado un problema de actualización para los programas previamente desarrollados y grabados en el controlador XRC.

Por lo tanto, se prefirió desarrollar rutinas de activación y desactivación del *gripper* usando la programación nativa básica del robot, guardándolos como archivos de trabajos (.JBI) y mediante funciones de selección de archivos *(BscSelectJob)* y ejecución de trabajos *(BscStartJob)* se obtuvo el resultado deseado de una manera efectiva. Esto permite visualizar la flexibilidad del sistema propuesto. Así como la expansión del sistema distribuido.

En caso de requerirse la ejecución de tareas previamente desarrolladas o trabajos complejos puede seleccionarse y ejecutarse los trabajos en el controlador de manera remota y una vez finalizado esto, regresar el control operativo al usuario final, todo bajo una arquitectura distribuida.

Se muestra como ejemplo una de las subrutinas de trabajo desarrolladas bajo el esquema de programación nativa INFORM II para la activación del *gripper*.

```
0000    NOP
0001    JUMP *NOHERRAM IF B001 = 0
0002    HAND 1 ON
0003    TIMER T = 0.75
0004    'MENSAJE SE TOMO LA PIEZA
0005    DOUT OT#(5) ON
0006    *NOHERRAM
0007    END
```

La figura 5.10, muestra el interfaz cliente CORBA desarrollado. Este es lo suficientemente robusto para llevar a cabo la comunicación con el controlador, además de ejecutar los distintos movimientos definidos en el interfaz IDL. Ejemplos de estos son Ready, Home, Extender, Retraer, Gripper, Mover Horizontalmente, Mover Verticalmente, entre otros.

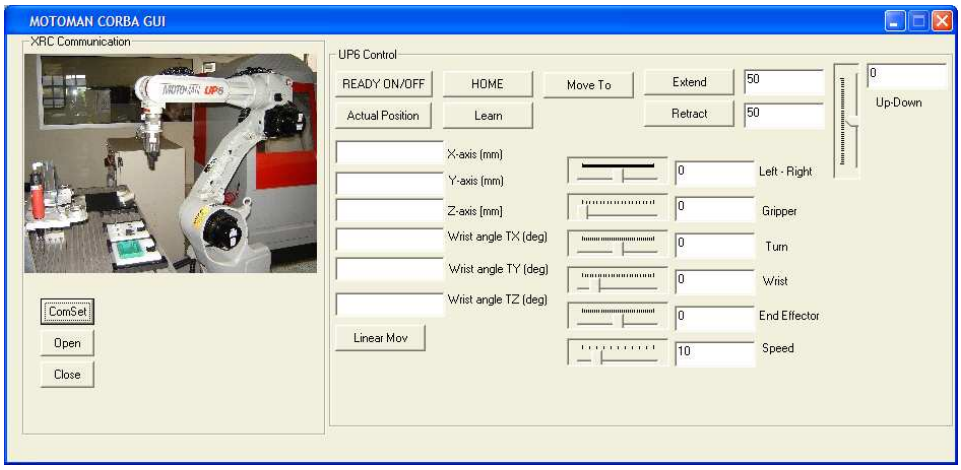


Figura 5.10: Interfaz Gráfica de Usuario para el cliente CORBA.

5.10. Integración del brazo manipulador robótico Motoman UP6 en sistema distribuido CORBA

A continuación presentamos los componentes generales que integran el sistema robótico distribuido para nuestro caso de estudio. Ver Fig. 5.11.

Etapa de Instalación

Estos componentes tienen diferentes roles dependiendo de la tarea que vayan a ejecutar. Se comunican unos con otros utilizando dos aproximaciones: Servicios de Nombres e Interfases IDL.

Servicio de Nombres

Cada vez que un componente comienza su ejecución, se registra dentro del servidor de nombres con un nombre específico. Esta es una característica estándar para todos los componentes. Si algún otro componente aparece, para requerir una comunicación con el componente previo, la única acción a llevar a cabo es "resolver" el nombre [20].

Esta aproximación es útil para conectar o reconectar diferentes módulos a través

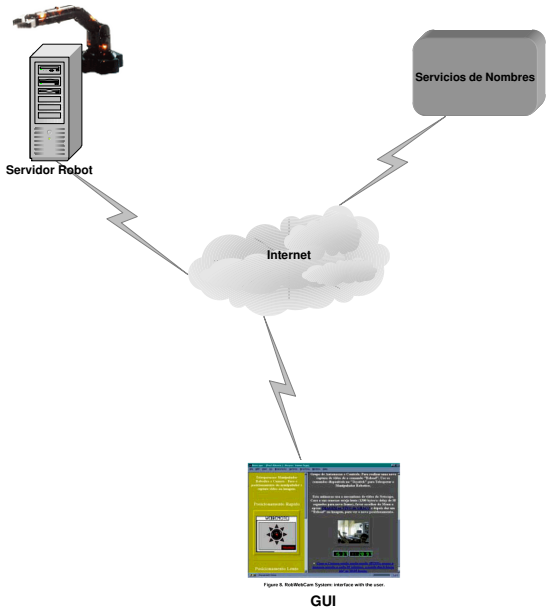


Figura 5.11: Componentes del Sistema Robótico Distribuido Motoman UP6.

de la red, sin preocuparse por resolver las referencias IOR. Los módulos clientes solo requieren el nombre de los servidores y la referencia del Servidor de Nombres, el cual es predefinido en el sistema.

Interfases IDL

La ventajas de estas definiciones, es que estas no están atadas a una estructura o configuración específica de un robot, sino que pueden ser reutilizadas entre varios brazos manipuladores robóticos.

Servidor Robot

Este es el objeto servidor o componente *envolvente*, el cual procesa todas las llamadas al brazo manipulador desde uno o más clientes. Las funciones *envolventes* se listan en el cuadro 5.7. El servidor de robot primero obtiene el control del robot al cual está conectado, y basado en llamadas internas, establece el eje específico a ser controlado [21].

El robot servidor es un componente *software*, el cual se liga con la interfaz gráfica

cliente (GUI). Las operaciones principales son:

- Conectarse con el controlador del robot XRC2001.
- Obtener el control de acceso al brazo manipulador.
- Crear su propia referencia IOR.
- Registrarse en el servicio de nombres CORBA.

Después de este paso, el servidor entra en un lazo infinito esperando por comandos a ejecutarse de parte del cliente. Si varios clientes intentan tener acceso al brazo, una política de FIFO (*First Input, First Output*) se mantiene para dar control solo a un cliente a la vez.

Interfaz Gráfica de Usuario (GUI)

Este componente se comunica con el servidor robot usando las interfases IDL. La interfaz fue desarrollada bajo ambiente de programación C++. En un futuro podría comunicarse con otros tipos de servidores CORBA a través de canales de eventos e interfases IDL definidos por estos.

5.11. Evaluación del Sistema Róbotico Distribuido Motoman UP6

Etapa de Ejecución

Algunos aspectos deben tomarse en consideración. El servicio de nombres debe ser inicializado primero. Posteriormente las implementaciones de los objetos (servidores) para cada componente definidos deben ser inicializados. Una vez que los servidores están listos la interfaz gráfica puede ser inicializada.

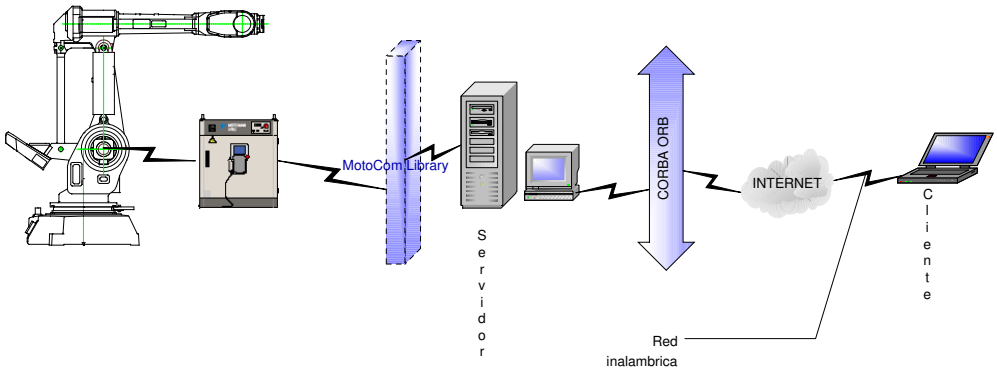


Figura 5.12: Componentes CORBA Cliente - Servidor para el Motoman UP6.

Capítulo 6

CONCLUSIONES

6.1. Análisis de los resultados obtenidos

El trabajo de investigación desde un principio se enfocó en obtener resultados experimentales positivos. Se contaba con un trabajo previo, que se tomó como base para reducir el tiempo de desarrollo de las aplicaciones CORBA. Principalmente en la estructura de la interfase cliente. Por lo que el mayor esfuerzo de programación fue en las implementaciones del código servidor del robot.

Se siguieron los mismos tipos de definiciones en las interfases IDL tal como en los robots F3 de la marca CRS mencionados en [22]. Esto sirve para comprobar que independientemente del tipo de robot utilizado, la estructura robótica distribuida se comporta de manera transparente entre marcas de robots heterogéneos. El utilizar los mismos tipos de definiciones IDL, permitió concentrarse en la generación de las *funciones envolventes* para integrar el brazo robótico Motoman UP6.

Tal como se plantea en el caso de estudio, las aportaciones principales de este trabajo, es la generación de las *funciones envolventes* CORBA, para la comunicación y el control de movimientos de robots Motoman para controladores XRC, MRC y ERC mediante el uso de las funciones disponibles en la librería DLL Motocom SDK. Así como la generación de una metodología de integración de brazos articulados robóticos Motoman en sistemas distribuidos basados en CORBA. La cual puede aplicarse aún para integrar brazos articulados robóticos de marcas heterogéneas. Esto permite justificar el valor científico de nuestra aproximación establecida. Así como su posible utiliza-

ción como una solución confiable en ambientes industriales con los debidas mejoras en aspectos de seguridad y control de acceso.

Ya que no se contaba con una documentación adecuada en la estructura de control robótica base. Al principio fue complicado involucrarse con este sistema. Aunado a la poca experiencia con el modelo cliente - servidor en los sistemas distribuidos basados en CORBA. El presentar una primera aproximación para el modelo UML del sistema robótico distribuido desarrollado, proporciona una ayuda significativa para involucrarse en futuras expansiones y mejoras.

Las pruebas experimentales con la interfaz gráfica de usuario (GUI) desarrollado en C++, en el cual se integran el cliente - servidor en una sola aplicación, via Ethernet dieron los resultados esperados. Se logró un control total de los movimientos del robot, de acuerdo a las funciones definidas en el CORBA IDL. Todos los movimientos de cada uno de las articulaciones, se llevan a cabo dentro de las velocidades de operación permitidas por el robot, y se tiene una buena precisión y exactitud en los movimientos finales deseados. La ejecución de tareas tales como la activación, desactivación del *gripper*, se llevan a cabo adecuadamente. Estableciendo una base para el desarrollo de futuras aplicaciones más complejas, con la confianza de que la solución al utilizar este tipo de aproximación es lo suficientemente robusta.

Una vez generado el código de implementación de las funciones en una aplicación del tipo *stand alone*, el siguiente paso fue integrarlos en la implementación servidor robot CORBA. La integración se llevó sin grandes complicaciones, solo se requirieron cambios mínimos con respecto al código original desarrollado en la aplicación independiente. Tomando en cuenta el impacto entre las conversiones de los tipos de datos utilizados en las definiciones de las funciones en Motocom SDK con respecto a los definidos en la interfase IDL. Así como la utilización de excepciones como medida de seguridad para operaciones no válidas en el lado del servidor.

El utilizar Ethernet como medio de comunicación entre el servidor CORBA y el controlador del robot no dieron los resultados esperados con respecto a los tiempos de respuesta en la ejecución de las tareas. Aunque al final los movimientos articulados se

llevaban a cabo, regresar el control del movimiento al cliente remoto era demasiado lento para propósitos prácticos. Una solución para este problema queda abierto para futuros trabajos.

Sin embargo se logró establecer un control de movimientos estable y dentro de las expectativas iniciales al utilizar RS-232C como medio de comunicación entre el controlador XRC del robot y el servidor CORBA. La experimentación realizada con el sistema robótico distribuido llevado acabo dentro de una red de área local (LAN) y aún utilizando una conexión inalámbrica (IEEE 802.11) entre el cliente y el servidor CORBA, generaron resultados positivos con respecto al movimiento remoto del brazo robótico. Hablando en términos de tiempos de respuesta, control de movimientos articulados y lineales (precisión, exactitud, velocidad), y aún en el aspecto de seguridad mediante el uso de excepciones para la verificación en cuanto a la ejecución de las tareas.

6.2. Conclusiones

Aproximadamente de las setenta y cinco funciones disponibles en la librería DLL del MotoCom SDK se implementaron en este trabajo de investigación un aproximado de cuarenta y cinco por ciento del total. Se establecieron comunicaciones RS-232C y Ethernet. Se desarrolló una aplicación independiente para la operación remota del robot UP6, la cual está implementada en su totalidad en el lenguaje de programación C++.

Se desarrolló el sistema cliente - servidor en arquitectura distribuida mediante CORBA, para obtener un sistema con todas las ventajas que estos tipos de sistemas proporcionan.

Se implementaron diversas funciones *envolventes* para los movimientos lineales y por articulaciones. Se utilizan funciones de transferencias de archivos para llamar rutinas en el controlador, tal como en la función *gripper on-off*. Esto permite sentar bases fundamentadas para seguir generando nuevas mejoras, donde el tiempo de desarrollo deberá ser mucho menor por la estructura de programación ya implementada.

6.3. Propuesta para futuros trabajos

Las oportunidades de desarrollo, son muy variadas. Tal como se menciona en el Capítulo 2, el estándar CORBA es una tecnología ya madura en el mercado. Con una comprobada eficiencia, pero también con áreas de oportunidad que nuevas tecnologías intentan emular y superar, tal como Java de SUN, .NET de Microsoft e infinidad de *middlewares* no estandarizados que atacan nichos especializados tal como las tecnologías *Web*, entre otros.

A pesar de esto, CORBA de OMG, proporciona continuamente nuevos servicios y aproximaciones tales como CORBA para Tiempo Real, CORBA Mínimo, CORBA Tolerante a Fallas, entre otros.

En lo que corresponde al sistema propuesto con el Motoman UP6, el siguiente paso lógico es implementar el servidor de imágenes CORBA, esto permitirá un lazo cerrado para la retroalimentación del usuario final con respecto a los movimientos teleoperados. Además de establecer nuevas aproximaciones en aplicaciones de robot - visión.

Una mejora en el sistema actual, sería desarrollar un cliente Java para la manipulación del robot dentro de navegadores tales como Netscape ó Internet Explorer. Java es una plataforma mucho más simple de integrar en este tipo de tecnologías basadas en Internet con respecto a CORBA.

Adicionalmente algunos investigadores se encuentran trabajando en interfases cliente usando dispositivos tales como PDA (Personal Digital Assistant) y teléfonos móviles en reemplazo de las computadoras personales con las ventajas que esto representa tales como movilidad y conveniencia en su uso [33].

Un aspecto importante es el control de acceso y seguridad. En este trabajo solo se utiliza el manejo de excepciones como medida de seguridad en caso de que las operaciones de comunicación y manejo en las tareas no se lleven adecuadamente entre el cliente y el servidor. Además de que las funciones integradas en el Motocom SDK en caso de no estar bien definidas o al hacer mal uso de los parámetros que las componen, provoca que el robot por seguridad apague sus motores servo y se utilicen automáticamente los frenos de seguridad. Teniendo que reinicializar el sistema por completo. Por supuesto

que esto no es lo ideal en sistemas teleoperados, pero es un punto a mejorar.

En cuanto al control de acceso para usuarios. Actualmente el sistema no descarta entre distintos clientes. Por lo que cualquier cliente con la definición IDL correcta puede conectarse sin problema alguno. Esto en la práctica no es recomendable, por lo que aspectos tales como la seguridad entre canales, autenticación, confidencialidad, etc. Son mecanismos que deberán desarrollarse en aplicaciones futuras si se considera utilizarse el sistema en un ambiente fábril. En este aspecto el ORB comercial utilizado en el desarrollo del sistema robótico distribuido UP6 permite la conexión de módulos tales somo FreeSSL (*Secure Sockets Layers*) [40] para proveer facilidades en encriptación y autenticación.

Capítulo 7

GLOSARIO

API (Application Program Interface): Interfaz Para La Programación De Aplicaciones.

BOA (Basic Object Adaptor): Adaptador básico de Objetos.

Broker Mediador.

CNC (Computer Numerical Control): Control Numérico por Computadora.

CORBA (Common Object Request Broker Architecture): Arquitectura común para un mediador de objetos.

DCE (Distributed Computing Environment): Entorno computacional distribuido.

DCOM (Distributed Computing Object Model): Modelo de Objeto de Cómputo Distribuido.

DCS (Distributed Control System): Sistema de Control Distribuido.

DDCF (Distributed Document Component Facility): Facilidad de distribución de componentes de documentos.

DII (Dinamic Invocation Interface): Interfaz de Invocación Dinámica.

DRS (Distributed Robotic Systems): Sistema Robótico Distribuido.

DSI (Dinamic Skeleton Interface): Esqueleto de la Interfaz Dinámica.

ESIOPs (Environment-Specific Inter-ORB Protocol): Protocolo de entorno específico entre ORBs.

Framework Estructura.

FIPA (Foundation for Intelligent Physical Agents): Fundación para Agentes Físicos Inteligentes.

GIOP (General Inter-ORB Protocol): Protocolo General entre ORBs.

IDL (Interface Definition Language): Lenguaje de Definición de Interfases.

IIOP (Internet Inter-ORB Protocol): Protocolo Internet entre ORBs.

Middleware Sistema de *software* que reside entre aplicaciones y debajo de sistemas operativos, protocolos de red y *hardware*.

MOF (Meta-Object Facility): Facilidad del Meta-Objeto.

OMA (Object Management Architecture): Arquitectura de administración de objetos o Arquitectura de gestión de objetos.

OMG (Object Management Group): Grupo de Gestión de Objetos.

ORB (Object Request Broker): Mediador de Petición de Objetos.

POA (Portable Object Adaptor): Adaptador Portable de Objetos.

PLC (Programmable Logic Controller): Controlador Lógico Programable.

Proxy Instancia que proporciona al cliente una interfaz al objeto remoto.

RMI (Remote Method Invocation): Método de Invocación Remoto.

Skeleton Esqueleto.

SOAP (Simple Object Access Protocol): Protocolo de Acceso a Objetos Simple.

Stub Representante (código intermediario).

UML (Unified Modeling Language): Lenguaje De Modelación Unificado.

URL (Uniform Resource Locator): Localizador Uniforme de Recursos.

Bibliografía

[1] Foundation for intelligent physical agents. http://www.fipa.org, August 2005.

[2] G. Aguirre. Apoyan empresas de base tecnológica. CONACYT, 2004.

[3] R. J. Anderson and M. W. Spong. Bilateral control of teleoperators with time delay. *IEEE Trans. Automat. Contr.*, 34:494–501, 1989.

[4] T. Ariza, F. J. Fernández, and F. R. Rubio. Implementing a virtual manufacturing device for mms/corba. *Emerging Tecnnologies and Factory Automation. Proceedings. 2001 8th IEEE International Conference on*, 2:711–714, October 2001.

[5] Robotic Industries Association. Robotics market remains hot in north america; new orders jump 12 percent in first half of 2004. http://www.roboticsonline.com/public/articles/2Q2004Charts.pdf, May 2004.

[6] J. Barata, L. Camarinha-Matos, R. Boissier, P. Leitão, F. Restivo, and M. Raddadi. Integrated and distributed manufacturing, a multi-agent perspective. *in Proc. of 3rd Workshop on European Scientific and Industrial Collaboration*, 27-29 June 2001.

[7] Morgan Bjorkander. Graphical programming using uml and sdl. *IEEE Computer Magazine*, pages 30–35, December 2000.

[8] Fintan Bolton. Interoperable naming service. www.informit.com/articles, September 2001. Extracto del libro Pure Corba 3.

[9] M. Born, E. Holz, and O. Kath. A method for the design and development of distributed applications using uml. *in proceedings International Conference on*

Technology of Object-Oriented Languages and Systems (TOOLS Pacific), IEEE Computer Society Press, Sydney, Australia, November 2000.

[10] Barbara M. Braun, Gregory P. Starr, John E. Wood, and Ron Lumia. A framework for implementing cooperative motion on industrial controllers. *IEEE Transactions on Robotics and Automation*, 20(3):583–589, June 2004.

[11] Davide Brugali and Mohamed E. Fayad. Distributed computing in robotics and automation. *IEEE Transactions on Robotics and Automation*, 18(4), August 2002.

[12] Luis M. Camarinha-Matos and Hamideh Afsarmanesh. Designing the information technology subsystem for enterprise integration. Technical report, New University of Lisbon, Faculty of Sciences and Technology, Portugal, 2003.

[13] Motoman Co. Up6. Technical report, Yaskawa Co., 2004.

[14] Antimio Cruz. Frena a la robótica crisis en industria. Periódico Reforma, Agosto 2003. Comentario de J. Campos, Director de FANUC Robotics México.

[15] H Fang, Jc Zeng, and Yinz Guo. Corba/java technology based legacy systems reusability research and design. *Proceeding of the First International Conference on Machine Learning and Cybernetics*, pages 481–485, November 2002.

[16] D. Flater. Specification of interactions in integrated manufacturing systems. Technical Report 6484, NISTIR, March 2000.

[17] T. Fraser, L. Badger, and M. Feldman. Hardening cots software with generic software wrappers. *IEEE Symposium on Security and Privacy*, 1999.

[18] Toshio Fukuda, Iso Takagawa, and Yasuhisa Hasegawa. From intelligent robot to multi-agent robotic systems. *Integration of Knowledge Intensive Multi-Agent Systems, 2003. International Conference on*, pages 413–417, October 2003.

[19] Object Management Group. The common object request broker: Architecture and specification. ftp://ftp.omg.org/pub/docs/ptc/96-08-04.pdf, July 1996.

[20] F. Guedea, R. Morales, R. Soto, I. Song, and F. Karray. Wrapper components for distributed robotic systems. *MICAI 2004:Advances in Artificial Intelligence*, 2004.

[21] F. Guedea, R. Soto, I. Song, and F. Karray. Enhancing distributed robotics systems using corba. *Proceedings of the First International Conference on Humanoid, Nanotechnologies, Information Technology, Communication and Control, Environment and Management 2003 (HNICEM'03)*, March 2003.

[22] F. Guedea, R. Soto, I. Song, and R. Morales. Towards autonomous robotic systems using wrapper components. *Proceedings of the International Symposium on Intelligent Control, ISIC 2003*, 2003.

[23] Michi Henning and Steve Vinoski. *Programación Avanzada en CORBA con C++.* Addison Wesley, 2002.

[24] T. Hori, H. Hirukawa, T. Suehiro, , and S. Hirai. Networked robots as distributed objects. *In IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 61–66, September 1999.

[25] E. R. Hughes, R. S. Hyland, Steven D. Litvintchouk, A. S. Rosenthal, A. L. Schafer, and S. L. Surer. A methodology for migration of legacy applications to distributed object management. *Proceedings of the 1st International Conference on Enterprise Distributed Object Computing*, pages 236–244, October 1997.

[26] P. J. Hyeon and C. H. Chul. Sliding-mode controller for bilateral teleoperation with varying time delay. *in Proc. IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics*, pages 311–316, 1999.

[27] IEEE. *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments.* IEEE Communications Magazine, Februar 1997.

[28] Meilir Page Jones. *Fundamentals of Object Oriented Design in UML.* Addison-Wesley, 2001.

[29] F. Karray, F. Guedea, R. Soto, and I. Song. Integration of distributed robotic systems. *Journal of Advanced Computational Intelligence and Intelligence Informatics*, Vol. 8(No. 1), 2004.

[30] Hyeon Soo Kim and James M. Bieman. Migrating legacy software systems to corba based distributed environments through on automatic wrapper generation technique. *in Proc. of The 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, July 2005.

[31] Philippe Leblanc and Ileana Ober. Comparative case study in sdl and uml. Technical report, Institut National Polytechnique de Toulouse, 2000.

[32] R. C. Luo and L.-Y. Chung. Stabilization for linear uncertain system with time latency. *IEEE Trans. Ind. Electron.*, 49:905–910, August 2001.

[33] Ren C. Luo, Kuo L. Su, Shen H. Shen, and Kuo H. Tsai. Networked intelligent robots through the internet: Issues and opportunities. *Proceedings of the IEEE*, 91(3):371–380, March 2003.

[34] Motoman. *Ethernet I/F Board Instruction Manual for UP/SKX-Series Robots*, Mayo15 1999. Part Number 142974-1.

[35] Motoman. *MotoCom SDK Function Manual*. Yaskawa, West Carrollton, OH, part number 147324-1 edition, June 2002.

[36] Pierre-Alain Muller. *INSTANT UML*. Wrox Press Ltd., 1997.

[37] R. Oboe and P. Fiorini. A design and control environment for internet-based telerobotic. *Int. J. Robot. Res.*, 17:443–449, 1998.

[38] OMG. *Unified Modeling Language Specification*, version 1.3 edition, June 1999.

[39] OMG Object Magnament Group, www.omg.org. *Naming Service Specification V1.2*, Septiembre 2002.

[40] ORBacus. Freessl user guide version 2.1. Technical report, Iona Technologies, 2003.

[41] Lars-Ola Osterlund. Component technology. *Proceedings of the IEEE Computer Applications in Power*, 13(1):17–25, January 2000.

[42] Young P., Chaki N., Berzins V., and Luqi. Evaluation of middleware architectures in achieving system interoperability. *In Proceedings of 14th IEEE International Workshop on Rapid Systems Prototyping*, pages 108–116, 2003.

[43] J. Pinto. Analysis sports robotics technology trends. www.automationtechies.com, Diciembre 2003.

[44] J.Ñ. Pires and J. M. G. S da Costa. Object-oriented and distributed approach for programming robotic manufacturing cells. *IFAC Journal Robotics and Computer Integrated Manufacturing*, 16(1):29–42, March 2000.

[45] Gopalan Suresh Raj. A detailed comparison of corba, dcom an java/rmi. http://my.execpc.com/ gopalan/misc/compare.html, September 1998.

[46] Paul G. Ranky. Collaborative synchronous robots serving machines and cells. *Industrial Robot: An International Journal*, 30(3):213–217, May 2003.

[47] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.

[48] Enrique Guajardo Santos. Interfaz de usuario para el robot motoman yasnac k3. Master's thesis, ITESM Campus Monterrey, 2003.

[49] R. Sanz, M. Alonso, I. López, and C. A. García. Enhancing control architectures using corba. *Proceedings of the 2001 IEEE International Symposium on Intelligent Control*, 3(189-194), September 2001.

[50] Ricardo Sanz. Corba for control systems. paper, Universidad Politécnica de Madrid, 2000. IFAC.

[51] Richard Schantz and Douglas C. Schmidt. Middleware for distributed systems. Technical report, University of California, bbn technologies, 2002.

[52] Jon Siegel. *CORBA 3: Fundamentals and Programming.* OMG Press/Wiley, 2000.

[53] Raul Silaghi. State of the art. Technical report, école polytechnique féderale de lausanne, 2004.

[54] Harry M. Sneed. Encapsulating legacy software for use in client/server systems. *Proceedings of the Third IEEE Working Conference on Reverse Engineering*, pages 104–119, 1996.

[55] I. Song, F. Guedea, and F. Karray. Distributed control framework design and implementation for multirobotic systems: a case study on block manipulation. *Proceedings of the International Symposium on Intelligent Control, ISIC 2004*, September 2004.

[56] T. Souder and S. Mancoridis. A tool for securely integrating legacy systems into a distributed environment. *in Proc. of Sixth Working Conference on Reverse Engineering*, pages 47–55, 1999.

[57] Joint Revised Submission. Uml profile for corba. Technical Report ad/00-05-07, OMG, June 2000.

[58] A. Tanenbaum and M. Van-Steen. *Distributed Systems Principles and Paradigms.* Ed. Prentice Hall, www.prenhall.com/tanenbaum/, 2002.

Apéndice A

MODELACIÓN UML

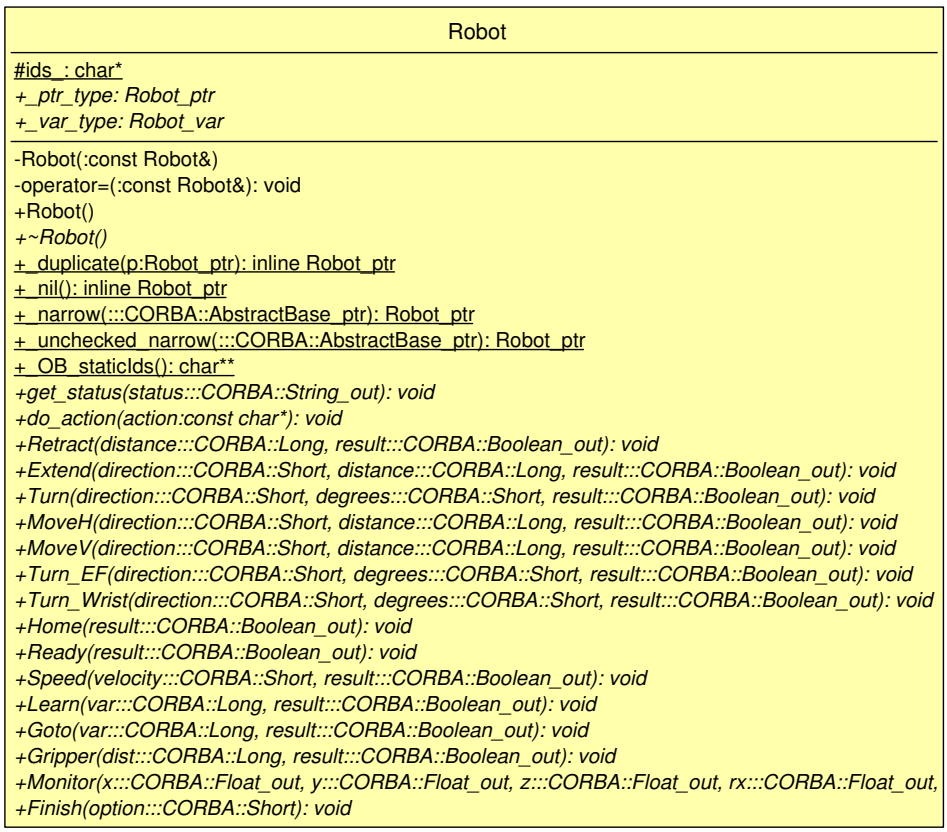


Figura A.1: Clase Interfase Robot

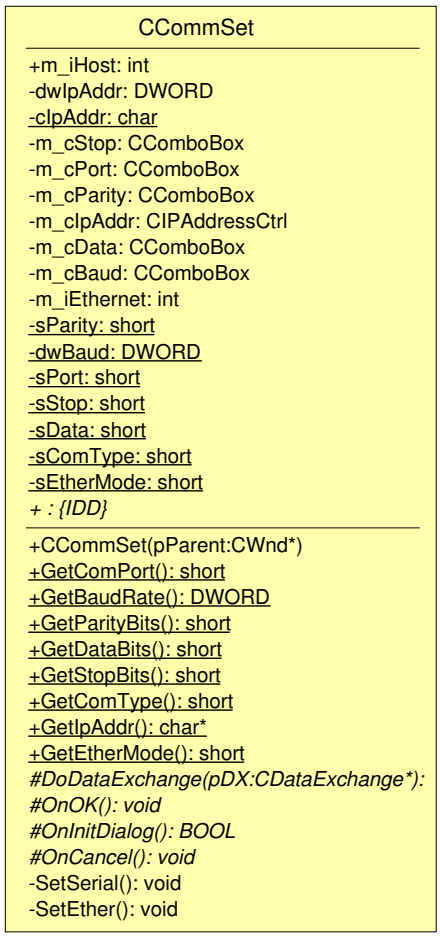


Figura A.2: Clase CCommSet

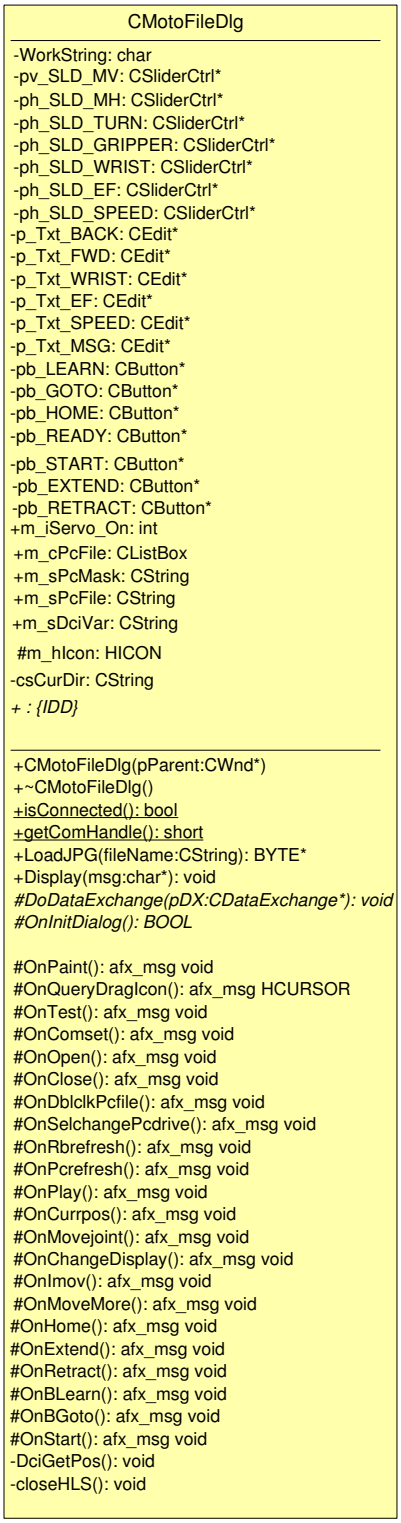


Figura A.3: Clase CMotoFileDlg

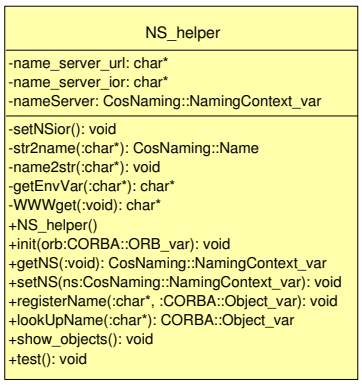


Figura A.4: Clase Servicio de Nombres

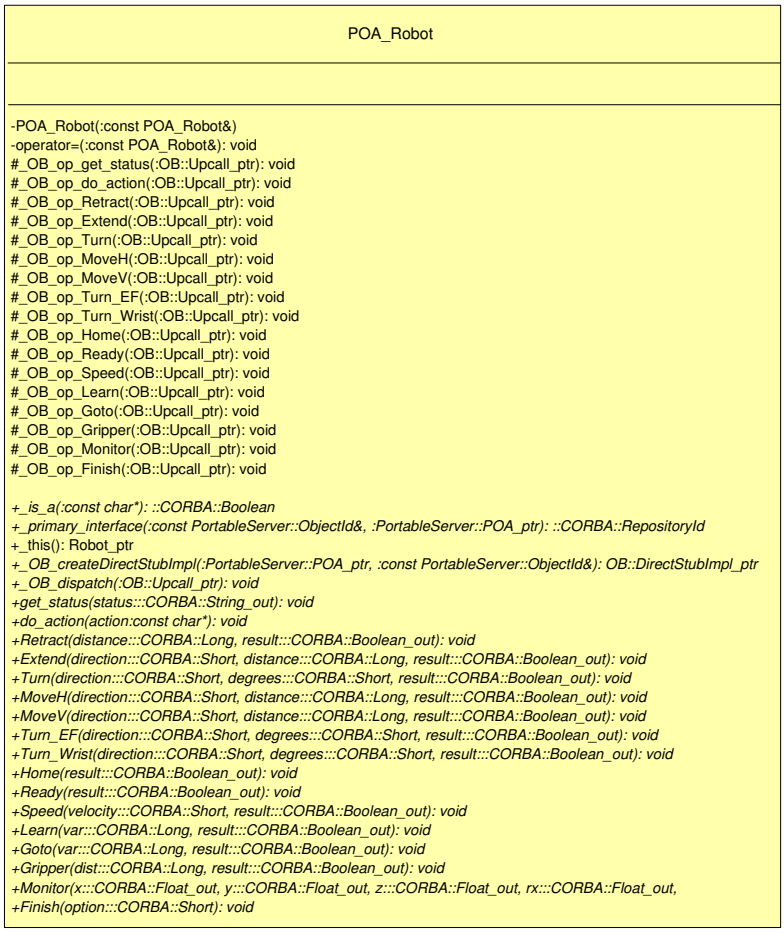


Figura A.5: Clase POA Robot

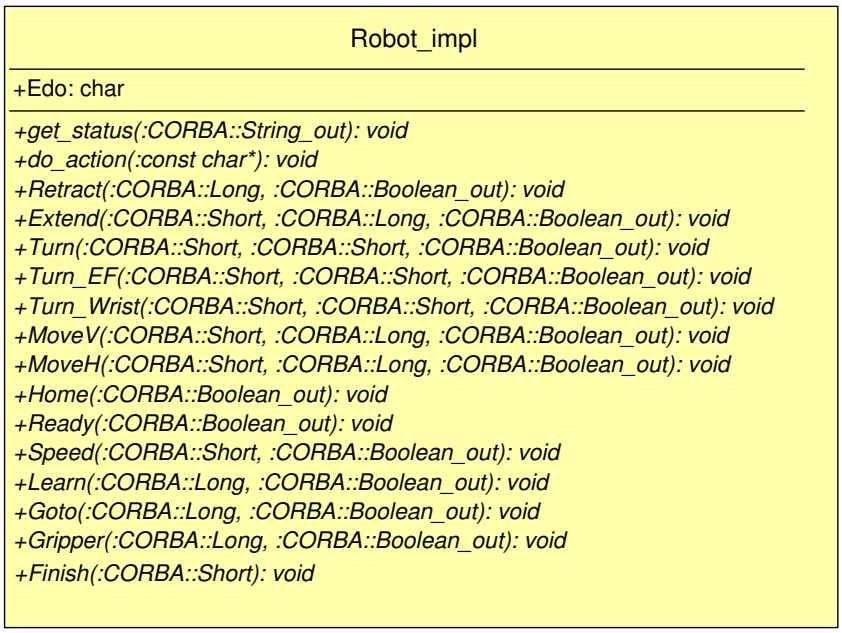


Figura A.6: Clase Implementación Robot

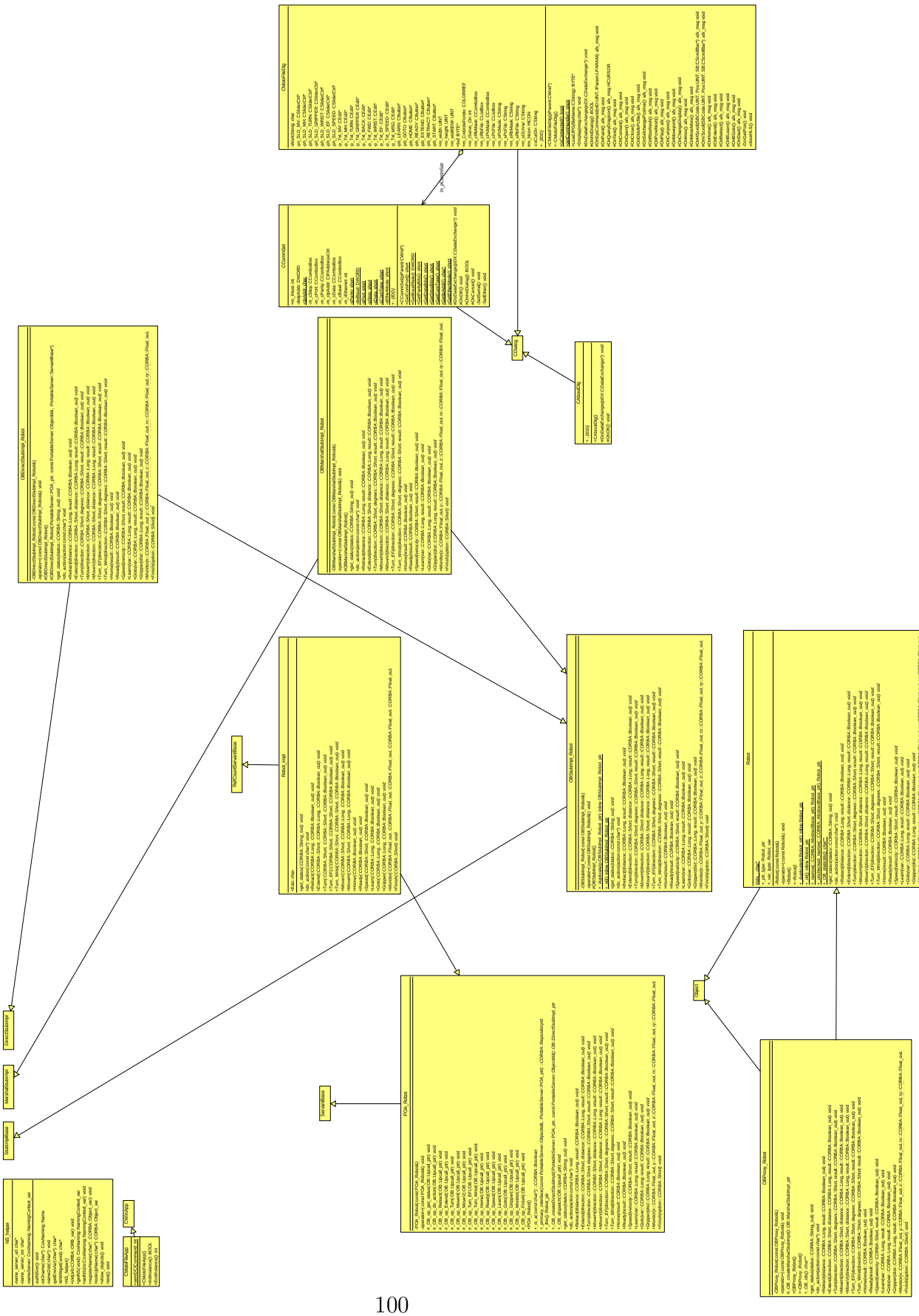


Figura A.7: Diagrama de clases UML para sistema robótico distribuido CORBA - Motoman UP6.

Apéndice B

CÓDIGO C++ - CORBA

```
// Se presentan las funciones envolventes CORBA desarrolladas en el
lado del servidor.

#include "stdafx.h"            // This for MFC
#include <OB/CORBA.h>
#include "Robot_impl.h"
#include <direct.h>
#include <math.h>
#include "MotoFile.h"
#include "MotoFileDlg.h"
#include "Robot_Def.h"

#ifndef _MOTOCOM_H_
#include "motocom.h"
#endif
#include <list>using
namespace std;

#define UP          1
#define FRONT       2
```

```
#define DOWN        3
#define LEFT        0
#define RIGHT       1
#define VERTICAL    1
#define AHEAD       2
#define HORIZONTAL  3
#define C_MIN  204.787742827

extern CMotoFileDlg * ptr;      // To get the Robot Object
extern double * dpPosData;
extern double  speedvar;
extern shortsComHandle;
extern bool   bConnected;

void Robot_impl::get_status(CORBA::String_out Estado)
                 throw(CORBA::SystemException)
{
    ptr->Display("Requesting status");
    Estado = CORBA::string_dup(Edo);
}

void  Robot_impl::do_action(const char * action)
                  throw(CORBA::SystemException)
{
   char msg[100]="";

   sprintf(msg,"Requesting <%s> to execute",action);
   ptr->Display(msg);
}
```

```
//////////////////////////////////////////////////////
// Extend Movement
//
//
void Robot_impl::Extend(CORBA::Short direction,
                        CORBA::Long distance,
                        CORBA::Boolean_out result )
{
    char msg[100];
    char dir[20];
    char Data[10];
    // short store the form storage pointer
    unsigned short * spRconf;
    // short stores the return value from BscIsRobotPos
    short sRobotPos2;
    CString csConv;
    spRconf=new unsigned short;
    dpPosData=(double *) new double[11];
    double theta_L, theta_U;        // Angles of the triangle in Degrees
    double dtheta_L, dtheta_U;      // Angles of the triangle in Degrees
    double dtheta_LN, dtheta_UN;    // Angles of the triangle in Degrees
    double dtheta1, dtheta2;        // Angles of the triangle in Degrees
    double theta1, theta2;          // Angles of the triangle in Rad
    double pulsos_l;
    double pulsos_u;
    double djw;
    double theta_x;
```

```
double dtheta_x;
double angC, angB, angH; // Angle of the triangle in
double H;                // Height of the arm
double a,b,d;            // length segments a and b, and distance d.
double m, n;
double c;                // Arc to compute
double pi;
double cn;
double dc;               // Angle C in degrees
double cte1 = 360.0/484000.0;
double cte2 = 360.0/484000.0;
double cte3 = 360.0/283600.0;

///// BscMovj Variables //////////
double Speed = speedvar;
char cFrName[100] = "BASE"; // stores the Coordinate Frame
unsigned short RConf=0;
unsigned short ToolNo = 00; // short stores the tool number
double * dpTargetP;         // target position storage pointer
short sRobotPos3;           // stores the return value from BscMovj

c = 0.0;
pi = 3.1415926535;
cn = 0.0;
// 1 Returns position in Joint//
sRobotPos2 = BscIsLoc(sComHandle, 1, spRconf, dpPosData);
pulsos_l = dpPosData[1];
pulsos_u = dpPosData[2];
dtheta_L = pulsos_l*cte1;
```

```
dtheta_U = pulsos_u*cte2;
theta_L = dtheta_L * pi / 180.0;
theta_U = dtheta_U * pi / 180.0;

a = 570.0;
m = 130.0;
n = 640.0;

// Value b
b = sqrt(m*m+n*n);

//Value angle X
theta_x = atan2(m,n);
dtheta_x = theta_x * 180 / pi;

// Values of theta1 and theta2
dtheta1 = 90.0 - dtheta_L;
dtheta2 = - dtheta_L + dtheta_U + dtheta_x;

// converting to radians
theta1 =dtheta1*pi/180.0;
theta2 =dtheta2*pi/180.0;

// Values to calculate:
// Angle C, side c, height H, distance d. angB, angH

angC = pi/2 + theta_x + theta_U;
c    = sqrt(a*a+b*b-2*a*b*cos(angC));
H    = a*sin(theta1)+b*sin(theta2);
```

```
d    = sqrt(c*c - H*H);
angB = acos((b*b-a*a-c*c)/(-2.0*a*c));
angH = theta1-angB;

// Validation
if ( abs(c-a-b) < 0.02 )
{
    sprintf(msg,"Arm full extended...no action taken place");
    ptr->Display(msg);
    result = FALSE;
}
else
{
 // Now, we calculate the new distance
 cn = c + distance;
 if ( cn > (a+b) )
 {
  sprintf(msg,"Distance is out of range");
  sprintf(msg,"making full extended move instead");
  ptr->Display (msg);
  cn=(a+b);
 }
    angC  = acos((a*a+b*b-cn*cn)/(2*a*b));
    angB  = acos((a*a+cn*cn-b*b)/(2*a*cn));
    theta1 = angB + angH;

    // Converting to degrees
    dtheta1 = theta1*180/pi;
    dc     = angC*180/pi;
```

```
        //New axis values in degrees
        dtheta_LN = 90 - dtheta1;
        dtheta_UN = dc-dtheta_x-90;

        //To keep End-effector Position
        djw = -(dtheta_LN - dtheta_L) + (dtheta_UN - dtheta_U);

        //Converting to Pulses
        theta_L = dtheta_LN*pi/180.0;
        theta_U = dtheta_UN*pi/180.0;
        pulsos_l = dtheta_LN /cte1;
        pulsos_u = dtheta_UN /cte2;
        djw      = djw/cte3;
        dpPosData[1]=pulsos_l;
        dpPosData[2]=pulsos_u;
        dpPosData[4]=dpPosData[4]-djw;

       sRobotPos3 = BscPMovj (sComHandle, Speed, ToolNo, dpPosData);
    }
    result = TRUE;
}

////////////////////////////////////////////
// Retract

void Robot_impl::Retract(CORBA::Long distance,
                         CORBA::Boolean_out result )
{
```

```
char msg[200];
sprintf(msg,"Requesting to Retract in one amount of %ld",distance);
ptr->Display(msg);
result = TRUE;
char dir[20];
// short store the form storage pointer //
unsigned short * spRconf;
// short stores the return value from BscIsRobotPos //
short sRobotPos2;
CString csConv;
spRconf=new unsigned short;
dpPosData=(double *) new double[11];
char Data[10];
long dist;
double theta_L, theta_U;    // Angles of the triangle in Degrees
double dtheta_L, dtheta_U;  // Angles of the triangle in Degrees
double dtheta_LN, dtheta_UN;// Angles of the triangle in Degrees
double dtheta1, dtheta2;    // Angles of the triangle in Degrees
double theta1, theta2;      // Angles of the triangle in Rad
double pulsos_l, pulsos_L;
double pulsos_u, pulsos_U;
double djw;
double theta_x;
double dtheta_x;
double angC, angB, angH;     // Angle of the triangle in
double H;                    // Height of the arm
double a,b,d;                // Length segments a and b, and d.
double m, n;
double c;                    // Arc to compute
```

```
double pi;
double cn;
double dc;                    // Angle C in degrees
// double distance;
double cte1 = 360.0/484000.0; // Degree vs. Pulses
double cte2 = 360.0/484000.0;
double cte3 = 360.0/283600.0;
///// Variables of BscMovj //////////
double Speed = speedvar;
char cFrName[100] = "BASE";     // Coordinate Frame
unsigned short RConf=0;
unsigned short ToolNo = 00;     // Short stores the tool number
double * dpTargetP;             // Target position storage pointer
short sRobotPos3;               // Return value from BscMovj
c = 0.0;
pi = 3.1415926535;
cn = 0.0;                       // 1 Working with JOINT//
sRobotPos2 = BscIsLoc(sComHandle, 1, spRconf, dpPosData);
pulsos_l = dpPosData[1];
pulsos_u = dpPosData[2];
dtheta_L = pulsos_l*cte1;
dtheta_U = pulsos_u*cte2;
theta_L = dtheta_L * pi / 180.0;
theta_U = dtheta_U * pi / 180.0;
// Links a, b and distance n
a = 570.0;
m = 127.0;
n = 640.0;
// Value of b
```

```
b = sqrt(m*m+n*n);
//Value of angle X
theta_x = atan2(m,n);
dtheta_x = theta_x * 180 / pi;
// Values of angles theta1 y theta2
dtheta1 = 90.0 - dtheta_L;
dtheta2 = - dtheta_L + dtheta_U + dtheta_x;
// converting to radians
theta1 =dtheta1*pi/180.0;
theta2 =dtheta2*pi/180.0;
// Values to calculate:
// Angle C, side c, height H, distance d. angB, angH
angC = pi/2 + theta_x + theta_U;
c    = sqrt(a*a+b*b-2*a*b*cos(angC));
H    = a*sin(theta1)+b*sin(theta2);
d    = sqrt(c*c - H*H);
angB = acos((b*b-a*a-c*c)/(-2.0*a*c));
angH = theta1-angB;
// Validation
if ( abs(c-a-b) < 0.02 )
{
    sprintf(msg,"Arm full extended...no action taken place");
    AfxMessageBox (msg);
    //result = FALSE;
}
else
{
    // Now, we calculate the new distance
    cn = c - dist;
```

```
 if ( cn > (a+b) )
 {
     sprintf(msg,"Distance is out of range making a full extended");
     AfxMessageBox (msg);
     cn=(a+b);
 }
 sprintf(msg,"Extending AHEAD");
 AfxMessageBox (msg);
 angC  = acos((a*a+b*b-cn*cn)/(2*a*b));
 angB  = acos((a*a+cn*cn-b*b)/(2*a*cn));
 theta1 = angB + angH;
 // Converting to degrees
 dtheta1 = theta1*180/pi;
 dc     = angC*180/pi;
 //New values of axis in grades
 dtheta_LN = 90 - dtheta1;
 dtheta_UN = dc-dtheta_x-90;
 //To keep End-Effector Position
 djw = -(dtheta_LN - dtheta_L) + (dtheta_UN - dtheta_U);
 //conversion to pulses
 theta_L = dtheta_LN*pi/180.0;
 theta_U = dtheta_UN*pi/180.0;
 pulsos_l = dtheta_LN /cte1;
 pulsos_u = dtheta_UN /cte2;
 djw      = djw/cte3;
dpPosData[1]=pulsos_l;
dpPosData[2]=pulsos_u;
dpPosData[4]=dpPosData[4]-djw;
// Retracting to new position //
```

```
        sRobotPos3 = BscPMovj (sComHandle, Speed, ToolNo, dpPosData);
    }
    result = TRUE;
}

// Turn
//
//
void Robot_impl::Turn(CORBA::Short direction, CORBA::Short degrees,
CORBA::Boolean_out result ) {
    char msg[100];

    // New variables to BscIsLoc //
    // short store the form storage pointer
    unsigned short * spRconf;
    // short stores the return value from BscIsRobotPos
    short sRobotPos2;
    spRconf=new unsigned short;

    ///// Variables of BscPMovj /////
    double Speed = speedvar;
    unsigned short ToolNo = 00; // short stores the tool number
    double * pTargetP;          // target position storage pointer
    short sRobotPos3;
    pTargetP = new double[11];

    sprintf(msg,"Requesting to Turn (%d), direction);
    ptr->Display(msg);
    sprintf(msg,"in one amount of %ld", degrees );
```

```
ptr->Display(msg);

if ( direction == RIGHT )
degrees=-degrees;

// Check actual robot position //
sRobotPos2 = BscIsLoc(sComHandle, 1, spRconf, dpPosData);
// 0 Return value in CARTESIAN//  // 1 Return value in PULSES //
if (sRobotPos2 == 0)
{
    sprintf(msg,"Got the current robot position successfully");
    ptr->Display(msg);

    // Value of Pos is in degrees, so multiplies to convert to pulses //
    // Factor of 1° - 1555.55 pulses
    (*dpPosData) = (*dpPosData)+(degrees*1555.55);
    // Moving axis S.
    pTargetP = dpPosData;
    sRobotPos3 = BscPMovj (sComHandle, Speed, ToolNo, pTargetP);
    if (sRobotPos3 == 0)
    {
            sprintf(msg,"Move to new robot position sucessfully");
            ptr->Display(msg);
    }
    else
    {
        sprintf(msg,"Falla en mov. joint S ");
        ptr->Display(msg);
    }
```

```
    }
    else
    {
            sprintf(msg,"Not able to get new position");
            ptr->Display(msg);
    }
    result = TRUE;
}

// Turn_EF
//
//
void Robot_impl::Turn_EF(CORBA::Short direction, CORBA::Short
degrees, CORBA::Boolean_out result ) {
    char msg[100];

    // New variables for BscIsLoc //
    // short store the form storage pointer
    unsigned short * spRconf;
    // short stores the return value from BscIsRobotPos
    short sRobotPos2;
    spRconf=new unsigned short;

    ///// Variables of BscPMovj /////////
    double Speed = speedvar;
    unsigned short ToolNo = 00; // short stores the tool number
    double * pTargetP;          // target position storage pointer
    short sRobotPos3;
    pTargetP = new double[11];
```

```
///// variables of BscIsLoc in Turn Effector /////
short sRobotPosTE;
short sRobotPos5;

if ( direction == RIGHT )
degrees=-degrees;
sRobotPosTE = BscIsLoc(sComHandle, 1, spRconf, dpPosData);
        if (sRobotPosTE == 0)
        {
            sprintf(msg,"Got the current robot position sucessfully");
            ptr->Display(msg);
            // 1 degree - 415.583 pulses...
            dpPosData[5] = dpPosData[5] + (degrees*415.583);
            pTargetP = dpPosData;
            sRobotPos5 = BscPMovj (sComHandle, Speed, ToolNo, pTargetP);
            if (sRobotPos5 == 0)
            {
                sprintf(msg,"Move to the new position");
                ptr->Display(msg);
            }
            else
            {
                sprintf(msg,"Failure in mov. joint T (Turn Effector)");
                ptr->Display(msg);
            }
        }
        else
        {
```

```
                sprintf(msg,"It's not connected the XRC)");
                ptr->Display(msg);
            }
    sprintf(msg,"Comand TURN_EF = %d",degrees);
    ptr->Display(msg);
    result = TRUE;
}

// Turn_Wrist
//
//
void Robot_impl::Turn_Wrist(CORBA::Short direction, CORBA::Short
degrees, CORBA::Boolean_out result ) {
    char msg[100];

    // New variables for BscIsLoc //
    // short store the form storage pointer
    unsigned short * spRconf;
    // short stores the return value from BscIsRobotPos
    short sRobotPos2;
    spRconf=new unsigned short;

    ///// Variables of BscPMovj /////////
    double Speed = speedvar;
    unsigned short ToolNo = 00; // short stores the tool number
    double * pTargetP;          // target position storage pointer
    short sRobotPos3;
    pTargetP = new double[11];
```

```
///// variables of BscIsLoc Wrist ////
short sRobotPosWrist;
// short stores the return value from BscIsRobotPos
short sRobotPos4;
sprintf(msg,"Requesting to Turn Wrist (%d)",direction);
ptr->Display(msg);
sprintf(msg,"in one amount of %ld",degrees);
ptr->Display(msg);

if ( direction == RIGHT )
degrees=-degrees;

// Actual Position of Robot //
// 1 return value in Pulses //
sRobotPosWrist = BscIsLoc(sComHandle, 1, spRconf, dpPosData);

if (sRobotPosWrist == 0)
  {
   sprintf(msg,"Got the current robot position sucessfully");
   ptr->Display(msg);
   // Pos in grades, pulses/grade...
   *(dpPosData+4) = *(dpPosData+4)+(degrees*808.081);
   pTargetP = dpPosData;
   sRobotPos4 = BscPMovj (sComHandle, Speed, ToolNo, pTargetP);
   if (sRobotPos4 == 0)
    {
      sprintf(msg,"Move to the new position");
      ptr->Display(msg);
    }
```

```
        else
           {
            sprintf(msg,"Failure mov. joint B ");
            ptr->Display(msg);
           }
        }
        else
            {
             sprintf(msg,"It's not connected the XRC)");
             ptr->Display(msg);
            }
    result = TRUE;
}

// MoveV ////////////////////
/////
//
void Robot_impl::MoveV(CORBA::Short direction, CORBA::Long distance,
CORBA::Boolean_out result ) {
    char msg[100];
    sprintf(msg,"Requesting to MoveV (%d) in one amount
    of %ld",direction,distance);
    ptr->Display(msg);
    char Data[20];
    //double pos;
    double dSpeed = speedvar;
    // temp char array that stores the Coordinate Frame
    char cFrName[100] = "BASE";
    unsigned short rconf= 0;
```

```
// short stores the tool number
unsigned short uToolNo = 00;
// target position storage pointer
double * dpTargetP;
// short stores the return value from BscIMov
short sRobotPos;
//CString csConv;

// New variables for BscIsLoc //
//dpPosData =(double *) new double[11];
// short store the form storage pointer
unsigned short * spRconf;
// short stores the return value from BscIsRobotPos
short sRobotPos2;
spRconf=new unsigned short;

if ( direction == DOWN )
distance=-distance;
if (bConnected)
{
 // Actual Robot Position //
 sRobotPos2 = BscIsLoc(sComHandle, 0, spRconf, dpPosData);
 if (sRobotPos2 == 0)
  {
    // Moves in Z axis the value of pos in mm.
    *(dpPosData+2)=*(dpPosData+2)+distance;
    dpTargetP = dpPosData;
    // Moves to new position //
    sRobotPos = BscMovj(sComHandle, dSpeed, cFrName,
```

```
        rconf, uToolNo, dpTargetP);
        if (sRobotPos == 0)
          {
            sprintf(msg,"Moving the Arm...");
            ptr->Display(msg);
          }
        else
          {
           sprintf(msg,"Not able to move..no action taken place");
           ptr->Display(msg);
          }
      }
      else
         {
          sprintf(msg,"Not able to get new robot position");
          ptr->Display(msg);
         }
    }
    else
    sprintf(msg,"Robot Not Connected");
    ptr->Display(msg);
    result = TRUE;
}

// MoveH
//
//
void Robot_impl::MoveH(CORBA::Short direction, CORBA::Long distance,
CORBA::Boolean_out result ) {
```

```
char msg[100];

// New variables for BscIsLoc //
// short store the form storage pointer
unsigned short * spRconf;
// short stores the return value from BscIsRobotPos
short sRobotPos2;
spRconf=new unsigned short;

///// Variables de BscMovj //////////
double dSpeed = speedvar;
// temp char array that stores the Coordinate Frame
char cFrName[100] = "BASE";
unsigned short RConf=0;
// short stores the tool number
unsigned short uToolNo = 00;
// target position storage pointer
double * dpTargetP;
// short stores the return value from BscMovj
short sRobotPos;
CString csConv;
dpTargetP = new double[11];

sprintf(msg,"Requesting to MoveH (%d) in one amount of
%ld",direction,distance);
ptr->Display(msg);

if ( direction == RIGHT )
distance=-distance;
```

```
if (bConnected)
{
    // Actual Robot Position //
    sRobotPos2 = BscIsLoc(sComHandle, 0, spRconf, dpPosData);
    if (sRobotPos2 == 0)
    {
        sprintf(msg,"Got the current robot position successfully");
        ptr->Display(msg);
        // Moving in axis X, a value for POS in mm.
        (*dpPosData) = (*dpPosData)+distance;
        // This move two axis S and L.
        dpTargetP = dpPosData;

        // Moves to new position //
        sRobotPos = BscMovj(sComHandle, dSpeed, cFrName,
        RConf, uToolNo, dpTargetP);
        if (sRobotPos == 0)
            {
                sprintf(msg,"Move to the new position");
                ptr->Display(msg);
            }
        else
            {
                sprintf(msg,"Failure in lineal movement");
                ptr->Display(msg);
            }
    }
    else
```

```
                {
                        sprintf(msg,"Not able to get new robot position");
                        ptr->Display(msg);
                }
    }
    result = TRUE;
}

// Home
//
//
void Robot_impl::Home(CORBA::Boolean_out result ) {
    char msg[200];
    sprintf(msg,"Requesting to HOME");
    ptr->Display(msg);
    // short store the form storage pointer
    unsigned short * spRconf;
    // short stores the return value from BscIsRobotPos
    short sRobotPos;
    spRconf=new unsigned short;

    // Variables of BscMovJ
    double dSpeed = speedvar;
    // temp char array that stores the Coordinate Frame
    char cFrName[100] = "BASE";
    unsigned short RConf=0;
    // short stores the tool number
    unsigned short uToolNo = 00;
    // target position storage pointer
```

```
double * dpTargetP1;
// short stores the return value from BscMovj
short sRobotPos2;

// Array for X,Y,X,Tx,Ty,Tz
double array[12]={399.747, -371.324, 568.530,
178.01, -6.74, -50.26, 0, 0, 0, 0, 0};

//bConnected = TRUE;
if (bConnected)
{
    sRobotPos = BscIsLoc(sComHandle, 0, spRconf, dpPosData);
    sRobotPos=0;
    if (sRobotPos==0)
    {
     dpTargetP1 = &array[0];
     // Moves robot with joint motion to a target
     // position in a specified frame system
     sRobotPos2 = BscMovj(sComHandle, dSpeed, cFrName,
     RConf, uToolNo, dpTargetP1);
     if (sRobotPos2 == 0)
     {
      sprintf(msg,"Moving to HOME");
      ptr->Display(msg);
     }
     else
        {
         sprintf(msg,"Failure in lineal mov. rc=%d",sRobotPos);
         ptr->Display(msg);
```

```
            }
        }
        else
        {
            sprintf(msg,"Not able to Read Current Position ! ");
            ptr->Display(msg);
        }
    }
    else
    {
        sprintf(msg," It's not Connected the XRC !! ");
        ptr->Display(msg);
    }
    result = TRUE;
}

// Ready
//
//
void Robot_impl::Ready(CORBA::Boolean_out result ) {
    char msg[100];
    sprintf(msg,"Requesting to Move READY position");
    ptr->Display(msg);
    result = TRUE;
}

// Speed
//
//
```

```
void Robot_impl::Speed(CORBA::Short velocity, CORBA::Boolean_out
result )
{
 char msg[100];
 sprintf(msg,"Requesting to SetUP velocity at %d percent",velocity);
 ptr->Display(msg);
 speedvar = velocity;
 sprintf(msg,"Velocity set to %d percent",velocity);
 ptr->Display(msg);
 result = TRUE;
}

// Learn
//
//
void Robot_impl::Learn(CORBA::Long  var, CORBA::Boolean_out result )
{
    char msg[100];
    sprintf(msg,"Requesting to learn current
    location with number %ld",var);
    ptr->Display(msg);
    result = TRUE;
}

// Goto
//
//
void Robot_impl::Goto(CORBA::Long  var, CORBA::Boolean_out result )
{
```

```
    char msg[100];
    sprintf(msg,"Requesting to Go at location [%ld]",var);
    ptr->Display(msg);
    result = TRUE;
}

// Gripper
//
//
void Robot_impl::Gripper(CORBA::Long  dist, CORBA::Boolean_out
result )
{
    char msg[100];
    // short stores the return value from BscMovj
    short sComplete;
    short JobOK;
    // temp char array that stores the Coordinate Frame
    char cName[100] = "RBHERROF.JBI";
    char cName2[100] = "RBHERRON.JBI";
    // temp char array that stores the Coordinate Frame
    sprintf(msg,"Requesting to Move the Gripper");
    ptr->Display(msg);

    if (dist>=50)
    {
        sComplete = BscSelectJob(sComHandle, cName);
        if (sComplete==0)
        {
                        JobOK = BscStartJob(sComHandle);
```

```
                    if (JobOK==0)
                    {
                        sprintf(msg,"Gripper ON");
                        ptr->Display(msg);
                    }
                    else
                    {
                     sprintf(msg,"Current Job not specified");
                     ptr->Display(msg);
                    }
    }
    else
    {
        sprintf(msg,"Not able to get the job selected");
        ptr->Display(msg);
    }
}
else if (dist <49)
    {
        sComplete = BscSelectJob(sComHandle, cName2);
        if (sComplete==0)
        {
            JobOK = BscStartJob(sComHandle);
            if (JobOK==0)
                    {
                        sprintf(msg,"Gripper OFF");
                        ptr->Display(msg);
                    }
                    else
```

```
                        {
                         sprintf(msg,"Current Job not specified");
                         ptr->Display(msg);
                        }
            }
            else
            {
                sprintf(msg,"Not able to get the job selected");
                ptr->Display(msg);
            }
        }
    else
    {
        sprintf(msg," It's not Connected the XRC !! ");
        ptr->Display(msg);
    }
    result = TRUE;
}
```

Apéndice C

CONFIGURACIÓN BÁSICA ETHERNET

Esta información está disponible en el manual del robot [34], sin embargo se anexa cómo realizar una conexión Ethernet, debido a los frecuentes problemas encontrados al realizar una comunicación exitosa en la práctica. Algunas de estas recomendaciones, no son englobadas adecuadamente en dicho manual.

El primer paso es instalar la tarjeta para interfase Ethernet JANCD-XIF02, referenciada como la tarjeta XIF02. Esta permite transmitir datos por medio de Ethernet como medio de transmisión. Dicha tarjeta debe ser instalada en la tarjeta base opcional JANCD-XCP02 para el controlador XRC.

La tarjeta XIF02 usa TCP/IP como protocolo de comunicación. Algunas características de este tipo de comunicación son:

Comunicación entre varias estaciones

Se permite que múltiples estaciones se conecten a la línea de transmisión. Sin cambiar ningún parámetro en la conexión, la comunicación puede llevarse a cabo especificando la estación destino. Las estaciones de comunicación incluyen XRC conectados a la tarjeta XIF02 y computadoras personales conectadas mediante tarjetas de red.

Procesamiento paralelo con el robot

Ya que las comunicaciones de red son procesadas por el CPU de la tarjeta XIF02, el desempeño en las tareas del robot no disminuye.

C.1. Configuración de la tarjeta Ethernet

Insertar la tarjeta XCP02, con la tarjeta XIF02 montadas dentro del CPU rack del controlador XRC, y ajustar los parámetros usando el programador portátil (*teach pendant*).

Se requiere asignar una dirección IP Para identificar a la red local y a la estación. Donde se debe de utilizar un número diferente para cada nodo. La dirección consiste de 4 bytes, y cada byte terminando con un punto. Ejemplo 192.168.10.1

Asignar una Sub Net Mask Una dirección IP está compuesta de una dirección de red y una dirección host. De cualquier manera la notación normal no diferencia entre la dirección de red y la dirección del host. Por lo que la Sub Net Mask divide las secciones distintas de una dirección IP.

Para una dirección de red, el mismo número es especificado para cada red. Se muestra un ejemplo a continuación de la configuración de la misma.

Cuando la dirección IP de la estación actual es digamos 142.168.10.1 y la sub net mask es 255.255.255.0

Sub net mask: 255.255.255.0 - 11111111. 11111111. 11111111.00000000 - Dirección de la red - Dirección Host -

Por lo tanto, la dirección de la red y la dirección host son como sigue:

Dirección de la red: 192.168.10.0 Dirección Host: 1

Gateway

Para comunicaciones entre redes con direcciones de red diferenciadas por la sub net mask, los paquetes de datos son enviados y recibidos a través de una estación "gateway". Por lo que habrá que fijar la dirección IP del gateway para la comunicación entre diferentes redes.

Server Address

En principio, cuando la función DCI o la función independiente se usan en el controlador XRC, este es fijado como un cliente mientras que la PC se fija como un

servidor. Para transmitir con una tarjeta XIF02 cuando el XRC es el cliente y la PC es el servidor, la dirección IP debe ser fijada. Dividir la dirección IP de 32 bits del servidor (PC) entre 8 bits de 4 secciones, y escribir cada sección en notación decimal.

Ejemplo:

11000000.10101000.00001010.01100101 - 192.168.10.101

C.2. Procedimiento para establecer los parámetros de comunicación en el controlador

Los parámetros de comunicación deben ser fijados en modo mantenimiento. Esto es, al encender el controlador XRC, se mantiene presionada el botón *TOP MENU*, luego seleccionar *SYSTEM* del menú principal, seleccionar *SETUP*, seleccionar *OPTION BOARD*, seleccionar *ETHERNET*, seleccionar *USE*, seleccionar el parámetro de comunicación a cambiar, ingresar el parámetro de comunicación y presionar *ENTER*, nuevamente presionar *ENTER*, seleccionar *YES*.

Utilizar el botón *SELECT* para seleccionar opciones en los sub menús.

Como recomendación, utilizar el comando *ping*, para verificar que la dirección IP asignada al controlador sea alcanzable antes de intentar cualquier conexión previa del robot, mediante cualquier tipo de *software* de comunicación y/o control.

En el entorno Windows, verificar en las propiedades de la red este habilitado el protocolo TCP/IP (Protocolo Internet) y dentro de las propiedades del protocolo se asigne una dirección IP dentro del mismo dominio asignado en la *address direction* del controlador, así como la misma Subnet Mask y el Default Gateway. No es recomendable utilizar la dirección IP automática establecida por Windows, ya que esta puede asignar una dirección fuera del dominio fijado en el controlador.

Apéndice D

CONFIGURACIÓN ORBACUS

D.1. Instalación de ORBacus

Es necesario tener debidamente instalado primero MS Visual Studio 6.0 ó una versión posterior. Escoger un procedimiento de instalación completa. Y seguir las opciones por *default* recomendadas. Esto ayuda a especificar automáticamente las rutas de las librerías y de ambientes.

ORBacus®viene en un archivo ejecutable para descomprimir. Bajar en un directorio temporal. Checar el archivo *INSTALL.MICROSOFT* para seguir las instrucciones de instalación.

Ejecutar el `runconfig.bat`. No se requiere ningún tipo de cambio aquí. Usar las opciones por default. Posteriormente para iniciar el proceso de compilación bajo windows, simplemente ejecutar.

```
nmake /f Makefile.mak
```

Para instalar los paquetes en algún directorio específico:

```
nmake /f Makefile.mak install
```

En caso de que aparezca el mensaje de "The LIB/INCLUDE environment variable needs to be set", o sí se tienen problemas para encontrar los archivos *include*

del sistema. Probablemente debe ajustarse correctamente el ambiente de desarrollo de Visual C++. Para arreglar esto, buscar en el directorio bin de C++, el archivo tipo *batch* llamado vcvars32. Si está utilizando Visual C++ ejecute:

```
vcvars32
```

En el *command prompt* de DOS, teclee `nmake /f Makefile.mak install`

D.2. Troubleshooting

Se recomienda que después de algún error en la compilación del ORBacus®, ejecutar `nmake /f Makefile.mak clean`. Esta instrucción borrar cualquier archivo previo instalado que pueda causar cualquier problema con el nuevo procedimiento de compilación.

Si se tiene algún problema relacionado con mensajes del tipo *Perl not found*. Es necesario instalar un interprete de Perl. Puede instalar por ejemplo *ActivePerl for Windows*. Y siga las instrucciones por default para este compilador.

Apéndice E

CARACTERÍSTICAS TÉCNICAS MOTOMAN UP6

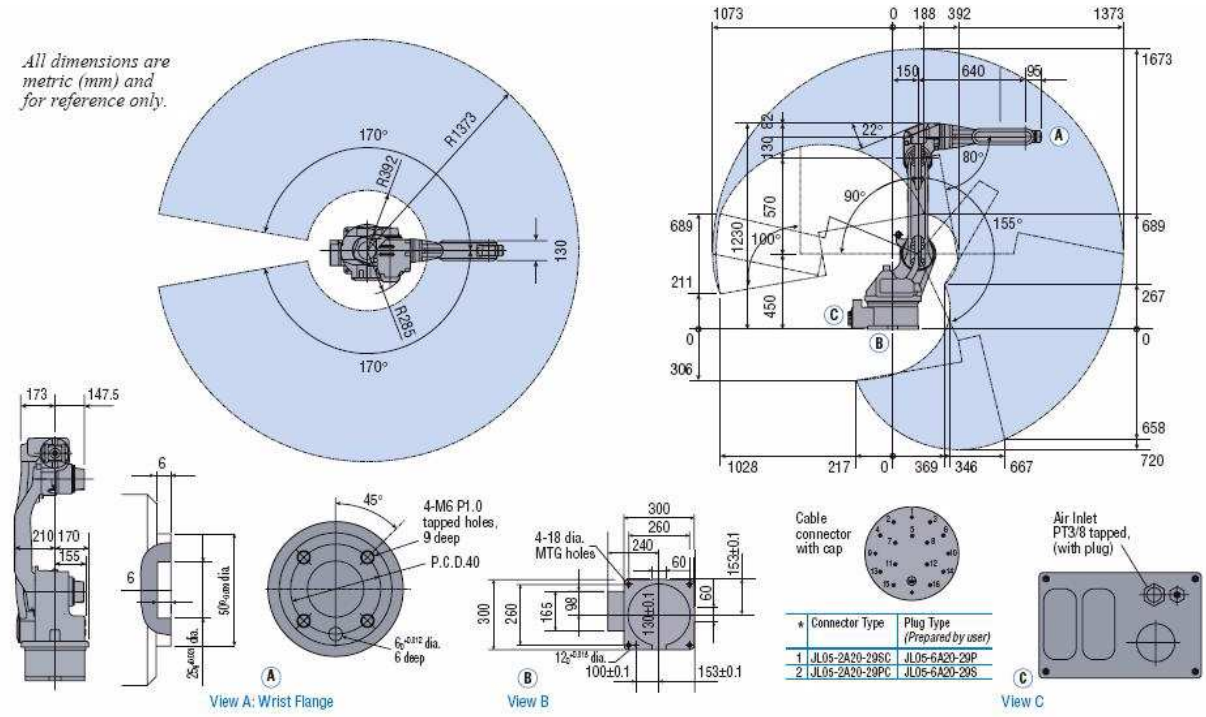


Figura E.1: Dimensiones y Rangos de Trabajo del UP6 [13].