COOPER (COupled OPEration Robot): CONTROL DE UNA ESTRUCTURA ARTICULAR RRR ACOPLADA A UN MANIPULADOR

Iván Perea, Gabriel J. García, Carlos A. Jara, Jorge Pomares, Francisco A. Candelas, Fernando Torres
Departamento de Física, Ingeniería de Sistemas y Teoría de la Señal
Carretera San Vicente del Raspeig, s/n, San Vicente del Raspeig, CP 03690
Universidad de Alicante
{ivan.perea, gjgg, carlos.jara, jpomares, francisco.candelas, fernando.torres}@ua.es

Resumen

Este artículo presenta el modelado, construcción y control de un robot articular tipo RRR, especialmente diseñado para ser acoplado al extremo de un robot antropomórfico PA-10 de Mitsubishi. A este robot articular de tres grados de libertad se le ha asignado como nombre el acrónimo COOPER (COupled OPEration Robot), y su principal finalidad es dotar al PA-10 de la capacidad de visualización dentro de entornos dinámicos. La plataforma multi-robot compuesta por los dos manipuladores se pretende utilizar para la supervisión y manipulación de tareas complejas. El artículo se centra en dar una descripción general de cada una de las partes del diseño y desarrollo hardware-software, centrándose principalmente en la arquitectura del controlador. Finalmente, se muestran los resultados obtenidos en la creación de dos esquemas de control, directo e indirecto, que plasman la eficiencia en el desarrollo de controladores gracias a esta nueva arquitectura software.

Palabras Clave: arquitectura software, control visual, diseño electrónico, robótica

1 INTRODUCCIÓN

Una de las principales líneas de investigación llevadas a cabo en la actualidad es hacer que un sistema robótico pueda interaccionar de una manera robusta y fiable dentro de entornos dinámicos [11, 14]. El robot COOPER se ha construido fundamentalmente con el fin de conseguir que un robot industrial pueda manipular, de forma autónoma, objetos sujetos a posibles oclusiones dentro de entornos dinámicos. Para ello, el robot COOPER lleva incorporada una pequeña cámara en el extremo del mismo y para su control se ha implementado un controlador basado en control visual directo. De esta manera, COOPER puede seguir los objetivos cercanos a la herramienta acoplada al extremo del sistema robótico PA-10, salvando las oclusiones que pueda al realizar las tareas de manipulación.

Dejando a un lado la mecánica y la sensorización de las estructuras robóticas, la verdadera inteligencia de un robot reside en su software. Sin embargo la interacción con los elementos hardware supone un problema para el desarrollo de este software dada la gran heterogeneidad existente. Muchas compañías privadas han creado *frameworks* y *middleware* intentando dar solución a este problema, como por ejemplo el Microsoft Robotic Studio [6] que se basa en SOAP una orientación a servicios de alto nivel de las funcionalidades mediante servicios Web. También existen trabajos de universidades y centros de investigación, que han creado sus propios *frameworks* como por ejemplo Player/Stage [12, 13] que se basan también en una interfaz generalizada PADI de acceso a los dispositivos y posteriormente una orientación a servicio con el uso de protocolos como CORBA o Jini, el uso de estas abstracciones hace el sistema pesado para aplicaciones que requieren un alto rendimiento como por ejemplo el caso del control directo. Otras aproximaciones [7] proponen el uso de una arquitectura basada en componentes conectables (*plugins*) que dotan de alta flexibilidad al sistema y evitan recompilar la aplicación a posteriori. Sin embargo esta arquitectura requiere un trabajo previo del diseñador de controladores para aprender a expresar el código en *schemas* [1], entidades que representan el comportamiento deseado del robot y que posteriormente son compiladas creando los *plugins*. Todos los sistemas desarrollados encapsulan funcionalidades de diferentes formas, para proveer abstracciones y mecanismos de reutilización de código para incrementar la productividad en la creación comportamientos de alto nivel. Sin embargo todos estos sistemas acaban proponiendo complejos sistemas de funcionamiento que requieren una especialización en su uso al no haber ninguno ampliamente aceptado. Además, todos estos sistemas se centran en la obtención de comportamientos de alto nivel y no en la implementación de mecanismos

de control a bajo nivel, como el control de posición de una estructura articular o el control visual.

Se requiere por tanto un sistema para el control de COOPER que permita el guiado mediante algoritmos de control visual directo. Puesto que se requiere una alta eficiencia, se decide desarrollar una arquitectura software propia para el control de la estructura. Se diseña por tanto esta arquitectura con el objetivo de poder aplicarse a diferentes estructuras abstrayendo de la heterogeneidad de los sensores y actuadores. La segunda principal aportación de este trabajo consiste en ofrecer un marco para la prueba rápida de nuevos esquemas de control sobre diferentes plataformas robóticas, incluyendo la reutilización de esquemas de control creados previamente en los nuevos. Como última aportación, se ha definido la arquitectura Modelo-Behavior-Controller, que solventa algunas desventajas de las arquitecturas Modelo-Vista-Controlador y sus variantes [4], y constituye un nuevo patrón de arquitectura software para el desarrollo de software dedicado al control de sistemas robóticos.

Este artículo presenta el diseño, construcción y control del robot COOPER. En primer lugar, en la Sección 2 se ilustra brevemente el proceso de modelado, construcción, selección de accionamientos y una pequeña ilustración de la etapa de potencia. La Sección 3 se centra en la arquitectura software, describiendo de forma detallada cada uno de los módulos que la constituyen. En la Sección 4, se muestran los resultados obtenidos, empleando dicha arquitectura para crear un controlador visual directo y un controlador visual indirecto. Finalmente, se comentan una serie de importantes conclusiones.

2 DISEÑO Y CONSTRUCCIÓN DEL ROBOT COOPER

La necesidad de incorporar la visión a un robot manipulador, con el objetivo de manipular objetos dentro de entornos dinámicos sujetos a posibles oclusiones, ha llevado a la inclusión de un nuevo robot en el entorno de trabajo que pudiese posicionar una cámara en el entorno, evitando la oclusión de las características. Para que este robot pueda realizar adecuadamente dicha tarea, es necesario un esquema de control de rápida respuesta ante estímulos, capaz de anticiparse a eventos que puedan provocar la pérdida de las características. Dicho requisito fuerza a la utilización de un esquema de control visual directo para el guiado de este nuevo robot. Para esto se necesita que la dinámica de este nuevo robot sea perfectamente conocida, motivando el diseño y construcción del robot COOPER. En la Figura 1 puede verse el diseño final de este robot, donde además se muestra la asignación de los sistemas de referencia de D-H.

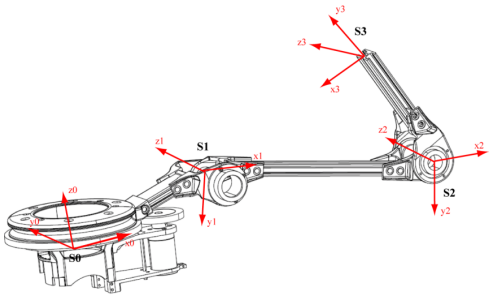

Figura 1: Sistemas DH del modelo del mini-robot.

A partir de las simulaciones se determina el par medio τ que debe proporcionar cada conjunto motor-reductor, τ_1=21.2 Nm, τ_2= 6 Nm y τ_3= 2.2 Nm. En el caso de la primera articulación, se ha incorporado una primera transmisión con una reducción 110:40 y una eficiencia de 72%, por lo tanto el par medio τ resultante para la salida de la reductora planetaria del motor ha de ser τ_1=10.7. La Tabla 1 muestra las características de los motores seleccionados, donde M_b es el valor de par, N_b (rpm) es la velocidad nominal, y η_m la eficiencia.

Tabla 1: Parámetros de los motores

Motor	M_b (mNm)	N_b (rpm)	η_m (%)
Articulación 1	170	6930	91
Articulación 2	93.3	6910	84
Articulación 3	25.8	10100	83

La Tabla 2 muestra las características de las reductoras seleccionadas, donde R es la reducción, M_{cont} es el par continuo soportado, M_{int} es el par intermitente máximo soportado por la reductora y η_r la eficiencia de transmisión.

Tabla 2: Parámetros de las reductoras planetarias

Reductora	R	M_{cont} (Nm)	M_{int} (Nm)	η_r(%)
Articulación 1	113	15	2.5	72
Articulación 2	111	8	12	70
Articulación 3	159	6	7.5	70

El par final de cada grupo motor-reductor se ha calculado considerando $N_f = M_b \cdot R \cdot \eta_r \cdot \eta_m$. Por tanto los resultados obtenidos para los pares son τ_{1f}=12.58 Nm, τ_{2f}= 6.09 Nm, τ_{3f}= 2.38 Nm, excediendo el valor medio obtenido. En el caso de la primera articulación, al aplicar la relación de reducción correspondiente a la última transmisión se obtiene τ_{1f2}=24.908 Nm, que también supera el par medio obtenido en las simulaciones para la misma.

Para gobernar los motores del robot se han utilizado tres amplificadores ADS50-5 de Maxon, a su vez las referencias de estos amplificadores están conectadas a una unidad de entradas y salidas CompactDAQ de National Instruments. Esta unidad se conecta mediante interfaz USB al computador que ejecuta los

algoritmos de control. Las principales ventajas del CompactDAQ son su modularidad y su completo interface de programación desde diferentes entornos, incluidos la plataforma .Net y LabView. Por otra parte, se han escogido los amplificadores ADS50-5 que permiten un control directo a bajo nivel de la corriente o par del motor. Además, aunque estos amplificadores trabajan con PWM con las ventajas que esto conlleva, su referencia es una señal analógica lo que facilita el conexionado. La Figura 2 ilustra los componentes básicos del circuito de control de los motores y de realimentación para uno de los tres motores (M1). Se usa un módulo de CompactDAQ con salidas analógicas (AO) para proporcionar los valores de referencia de cada amplificador (SV), mientras que con un modulo de entradas analógicas (AI) se monitoriza el consumo de corriente (In) y la velocidad estimada (Mn) en los amplificadores. Así mismo, con un módulo de entradas digitales rápidas programables (PFI) se leen las señales del encoder diferencial (E1) de cada motor, y con un módulo de entradas digitales estándar (DI) se monitorizan cada final de carrera (FC) que informa de la posición cero del eje correspondiente.

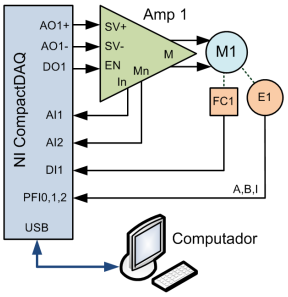

Figura 2: Esquema de control de los motores.

3 ARQUITECTURA SOFTWARE

Los dispositivos hardware comerciales cuyo objetivo es su integración en sistemas automáticos, se distribuyen con los controladores necesarios para su uso en diferentes sistemas operativos. Junto a ellos suelen incluirse librerías, bien estáticas (archivos cuya extensión es **.lib** en windows y **.a** en sistemas Unix) o bien dinámicas (cuya extensión es **.dll** en windows y **.so**), que permiten realizar programas personalizados para la integración de dichos dispositivos. Las librerías estáticas se enlazan directamente con el código del usuario, mientras que las librerías dinámicas se cargan en tiempo de ejecución. En el caso de las librerías estáticas, se obtiene la ventaja de que el sistema se puede distribuir sin incluir las librerías, y además, suele tener una ejecución más rápida y ligera. No obstante, una posterior revisión de la librería implica tener que recompilar el código nuevamente. En el caso de las librerías estáticas ofrecen múltiples ventajas en otros sentidos. En primer lugar un cambio interno en la librería no requiere recompilar el código, siempre que se mantenga estable la interfaz de uso de la librería. Y en segundo lugar, se pueden realizar diferentes librerías dinámicas con la misma interfaz de acceso, por lo que es posible cambiar el funcionamiento del sistema, cambiando la librería empleada sin realizar modificaciones en el código del software original [7]. Esto es particularmente útil en el caso de componentes que comparten un interfaz muy estable, pero cuya funcionalidad puede diferir en gran medida.

A menudo el software desarrollado para el uso de los dispositivos de entrada/salida, depende directamente de las librerías del fabricante. Esto ocurre en menor grado en el caso de las librerías dinámicas, ya que sólo dependen de la interfaz de acceso a la funcionalidad de la misma. Una de las aportaciones del presente artículo es el diseño y desarrollo de una arquitectura software, que independiza la programación de controladores genéricos para estructuras robóticas ó articuladas, de las implementaciones o tecnologías concretas de los distintos accionamientos. Este sistema además de poder aplicarse a diferentes estructuras, tiene la ventaja de ofrecer un marco para la prueba rápida de nuevos esquemas de control sobre diferentes arquitecturas. Además, permite combinar varios controladores de manera cooperativa ó jerárquica obteniendo una gran cantidad de combinaciones posible. Así, se constituye la segunda principal aportación de este trabajo, al presentar un marco para el prototipado rápido de controladores aplicados a la robótica, que permite su desarrollo de forma sencilla y directa puesta en marcha, sin requerir tener en cuenta la implementación de la comunicación con los dispositivos de entrada/salida. Para la implementación de este software se ha definido la arquitectura Model-Behavior-Controller, que solventa algunas desventajas de las arquitecturas Modelo-Vista-Controlador y sus variantes y constituye un nuevo patrón de arquitectura software para el desarrollo de software dedicado al control de sistemas robóticos.

4.1 Descripción general de la arquitectura

A continuación se describe como se ha realizado el diseño de la arquitectura software que permite controlar una estructura articulada mediante diversos algoritmos de control de forma general.

En la Figura 3 se muestra un diagrama de componentes que ilustra la arquitectura software del sistema. Las principales características del sistema son: adaptabilidad para poder probar de manera sencilla diferentes tipos de controlador, poder

reutilizar otros controladores previos como parte de los nuevos, e independencia de la tecnología de control empleada, siendo fácilmente extensible al uso con nuevos dispositivos de entrada/salida. Además, es muy importante que la arquitectura proporcione un buen rendimiento para el controlador que permita unos tiempos de ciclo de cómo máximo 20ms para lógicas de control muy complejas. Para satisfacer estos requisitos se ha definida una arquitectura inspirada en la arquitectura MVC (Modelo–Vista-Controlador). Esta nueva arquitectura se ha denominado Modelo-Comportamiento-Controlador ó MBC (Model-Behavior-Controller).

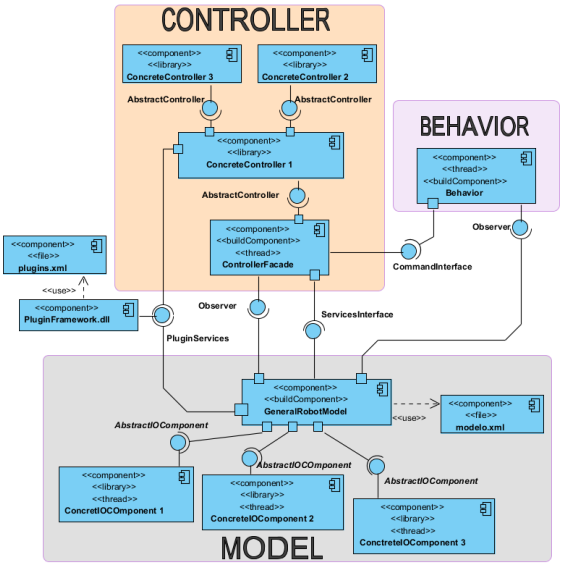

Figura 3: Arquitectura del sistema.

El componente modelo, tiene como principales finalidades las siguientes: en primer lugar, proporcionar una capa de datos fácilmente adaptable a diferentes estructuras robóticas y a los dispositivos de interacción de las mismas y en segundo lugar proporcionar una interacción rápida y homogénea sobre a los elementos de entrada/salida cuya configuración obtiene de archivo XML llamado “modelo.xml”.

A diferencia con la arquitectura MVC, en esta nueva arquitectura, el controlador no constituye parte de la interfaz de usuario [4], sino que representa los controladores encargados de tomar las decisiones de actuación sobre el modelo en función de las referencias de control establecidas. Para establecer las referencias de control, genera una serie de comandos que son ofrecidos al componente de comportamiento.

El componente de comportamiento es el encargado de tomar las decisiones de alto nivel y puede constituirse como un componente que proporciona un comportamiento autónomo del sistema ó bien un componente que proporciona un acceso para el control del sistema bajo supervisión de un operador. En el primer caso se pueden considerar esquemas de IA como máquinas de estado o algoritmos de planificación, mientras que en el segundo caso están las interfaces gráficas en terminales de control, la operación remota y la teleoperación a través de dispositivos de interfaz avanzados para operadores humanos, e incluso, se pueden considerar sistemas de comportamiento hibrido que incluyan varias de estas aproximaciones.

Finalmente, se encuentra el componente *PluginFramework* que da soporte a las funcionalidades básicas de un sistema de *plugins*, como la creación de instancias de forma dinámica y la localización de las mismas a partir de los servicios que las caracterizan. Los servicios proporcionados se identifican por el tipo de interfaz *software* que implementan, concretamente *AbstractIOComponent* y *AbstractController*. Así, es posible añadir nuevos elementos de entrada salida, y nuevos controladores a posteriori con el sistema ya en producción. Esta característica permite probar rápidamente los controladores desarrollados en diferentes estructuras robóticas, ya que un sistema de *plugins* dota de gran flexibilidad a la arquitectura [7]. Toda esta información sobre los elementos a crear se encuentra de nuevo en un archivo de configuración XML llamado "*plugins.xml*".

4.2 El modelo

El modelo es el elemento del sistema encargado de mantener actualizados los datos relativos a las entradas y salidas del sistema. También se encarga de mantener actualizados los parámetros físicos del robot, como posición articular, posición 3D y orientación del extremo, velocidad articular, velocidad del extremo y pares articulares. Para realizar estas tareas en primer lugar, requiere ser configurado. Para ello, carga la configuración de un archivo XML llamado “modelo.xml” en el que se ha realizado una definición de los parámetros del robot, nombre, tipo, dirección y cantidad (identifica el número de elementos en los parámetros que se corresponde con un vector). Posteriormente estos parámetros se asocian a Componentes de entrada/salida, que se encargan de mantener actualizado los datos en caso de ser parámetros de entrada, o bien, de actualizar los valores de salida en caso de ser parámetros de actuación. Estos componentes están representados en la Figura 3 como *ConcreteComponent*.

Para conseguir la creación de un modelo adaptable, el subsistema posee una arquitectura de repositorio de datos basada en el patrón *Reflection* [4], de manera que los distintos elementos del modelo se crean en

base a los metadatos almacenados en un archivo XML y cuyo contenido sigue las reglas de la gramática mostrada en la Figura 4. En esta gramática se describe cada elemento que compone el modelo, con su nombre (que corresponde con un identificador), su tipo de dato básico, su dirección (entrada o salida), y su cantidad (este último es útil, por ejemplo para un puerto de datos, donde los datos se reciben en forma de vector y por lo tanto este valor será mayor de 1). Nótese que las reglas derivadas de la regla A_1, sólo se emplean para obtener independencia del orden en que aparezcan los elementos de configuración antes mencionados.

$$S \rightarrow \{S\} < data > A_1 < /data >$$
$$A_1 \rightarrow (A_{11}A_{12}|A_{12}A_{11}|A_{21}A_{22}|A_{22}A_{21}|A_{31}A_{32}|A_{32}A_{31})$$
$$A_{11} \rightarrow (N_1D_1|D_1N)_1$$
$$A_{12} \rightarrow (C_1T_1|T_1C_1)$$
$$A_{21} \rightarrow (N_1T_1|T_1N_1)$$
$$A_{22} \rightarrow (C_1D_1|D_1C_1)$$
$$A_{31} \rightarrow (T_1D_1|D_1T_1)$$
$$A_{32} \rightarrow (C_1N_1|N_1C_1)$$
$$N_1 \rightarrow <name>L_1</name>$$
$$L_1 \rightarrow \{L_1\}L_2$$
$$L_2 \rightarrow [a\text{-}zA\text{-}Z._\ ]$$
$$D_1 \rightarrow < direction > D_2 < /direction >$$
$$D_2 \rightarrow input \mid output$$
$$T_1 \rightarrow < type > T_2 < /type >$$
$$T_2 \rightarrow (bit \mid byte \mid char \mid double \mid long \mid float \mid long)$$
$$C_1 \rightarrow < count > C_2 < /count >$$
$$C_2 \rightarrow (1|2|3|4|5|6|7|8|9)C_3$$
$$C_3 \rightarrow (1|2|3|4|5|6|7|8|9|0)*$$

Figura 4: Gramática para el archivo de meta-datos para la configuración del modelo

En segundo lugar los elementos que implementan la interfaz *AbstractIOComponent* actúan como puertos de conexión a los dispositivos, independizando al modelo de las tecnologías y particularidades de los mismos. Los elementos que implementan la interfaz *AbstractIOComponent*, y por tanto los servicios que los caracteriza, permiten interaccionar con el dispositivo concreto, y dependen del tipo de puerto con que se corresponden. Los componentes de entrada/salida, o puertos, pueden ser o de salida, o de entrada, estos últimos a su vez, pueden activarse por evento y por tanto desencadenan una tarea (como por ejemplo una señal de paro o un final de carrera) o por lectura activa cuando se necesita el dato [2] (éstos corresponden a componentes que proporcionan un servicio que no está ligado a una tarea). Esta interacción con los sensores y actuadores depende directamente de los drivers software utilizados para conectar las tarjetas de entradas/salidas empleadas. En el caso, del robot COOPER se emplea una tarjeta de la empresa National Instrument, que proporciona una serie de archivos *DLL* (*Dynamic-Link Libraries*) para la comunicación con el CompactDAQ por lo que hay que enlazar estas *DLL* para poder interaccionar con las entradas salidas del sistema. Si posteriormente se desea utilizar otro tipo de sistema de entradas/salidas hay que recompilar el código, con la *DLL* del nuevo fabricante creando un nuevo *plugin*, encapsulando así en estos elementos la dependencia de la tecnología e independizando al modelo.

4.3 El controlador

Este elemento no se corresponde exactamente con el controlador del patrón Modelo-Vista-Controlador. Según la el patrón MVC, el controlador y la vista componen la interfaz de usuario y el controlador solo se encarga de manejar las entradas del usuario [4]. En el caso propuesto, los comandos recibidos en el controlador no se traducen en llamadas directas a servicios del Modelo. En la arquitectura MBC se extiende el comportamiento del controlador, comportándose como un componente con entidad propia que ejecuta esquemas de control que intentan minimizar el error entre un valor observado en el modelo, y unas referencias que pueden modificarse a través de los comandos recibidos. Estas modificaciones de las referencias provocan que el controlador tome nuevas decisiones de actuación. En este sistema la lógica de la aplicación recae en gran medida en el controlador al contrario que en el caso de la arquitectura MVC donde toda la lógica recae en el modelo provocando lógica altamente acoplada. El esquema MBC, ofrece un modelo independiente de la lógica de control, obteniendo un bajo acoplamiento entre estos elemenos, y pudiendo así, sustituir el controlador incluso en tiempo de ejecución. De esta forma se obteniene una evidente mejora en el reparto de responsabilidades.

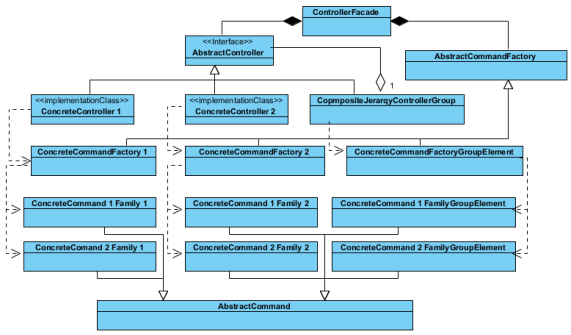

Figura 5: Diseño software del controlador

En la Figura 5, se muestra, el diseño software del componente controlador. En esta figura se muestra como los controladores se han implementado con un patrón *Composite* [10], que permite tener controladores complejos a partir de agrupación ó composición de varios controladores, al mismo nivel o de forma jerárquica. Una vez instanciado el

controlador, las interacciones con el mismo, se realizan mediante comandos concretos que implementan la interfaz *AbstractCommand* y que creados por una factoría. Para esto se implementa un patrón *AbstractFactory* [10] que se encarga de construir los comandos adecuadamente dependiendo del controlador utilizado. Para la interacción con el resto de componentes se ha definido una Fachada (patrón *Facade*) [10] de manera que mantiene el bajo acoplamiento de los elementos del componente publicando únicamente la interfaz abstracta *AbstractCommand*. Las implementaciones concretas de esta interfaz, corresponden con acciones que se pueden realizar sobre el controlador y son ofrecidas al componente comportamiento.

4.4 El plug-in framework

Los sistemas de *plugins* proporcionan una buena flexibilidad al software ya que permiten una gran reutilización de los componentes (*plugins*) desarrollados, permiten instanciar dinámicamente elementos de forma que se pueden crear soluciones adaptadas al problema, y además no requiere recompilar todo el software en caso de añadir nuevos elementos a posteriori [7]. Por estos motivos se ha diseñado una arquitectura software que incluye un pequeño *framework* que da soporte a las funcionalidades básicas de un sistema de *plugins*, para la creación de instancias de forma dinámica y localización de las mismas a partir de los metadatos que las caracterizan. Para esto, se ha creado una clase especializada a la que se denominará creador (*CreatorConcreteComponets*, que implementa el patrón [10] GRASP C*reator*). Este elemento se encarga de la instanciación de objetos abstractos a partir de *DLL* específicas. De esta forma, mediante la configuración en un archivo de texto, *XML* u otros medios, se puede establecer dinámicamente el paquete o DLL que hay que utilizar y las clases a instanciar. En la Figura 7, se muestra la gramática para el archivo de meta-datos empleado en la instanciación de los componentes desde el creador.

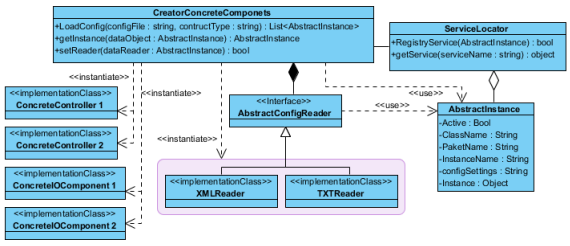

Figura 6: Diseño software del *framework* de *plugins.*

En la Figura 6, se muestra el diseño software del componente. Para poder representar a las instancias de los objetos, existe un tipo abstracto de dato (TAD) llamado *AbstractInstance*, que contiene toda la información de meta-datos para poder crear un objeto a partir de una *DLL* ó la instancia del propio objeto una vez instanciado. El elemento empleado para leer los metadatos es el lector que implementa la interfaz *AbstractConfigReader*. Como se puede ver en la Figura 6 se pueden emplear diferentes tipos de lector, y por tanto, este se debe crear de una forma particular. Su instanciación se realiza mediante un método llamado *getInstance()* del creador que permite crear la instancia asociada a un *AbstractInstance*, de forma que si no había sido creado previamente (su parámetro *instance* es igual a *null*) lo crea y devuelve el *AbstractInstance*. De esta manera, los meta-datos para la creación del mismo se aportan desde fuera del *framework* de *plugins*, por ejemplo desde la vista ó desde el programa principal. A continuación, utilizando el método del creador *setReader(reader : AbstractInstance):bool* se asigna la instancia del elemento lector, enviada por parámetro, como lector actual. A partir de que el creador tiene un lector asociado, puede comenzar a instanciar los componentes, utilizando los metadatos contenidos en el archivo pasado como parámetro del método *LoadConfig(configFile : string, constructType : string) : List<AbstractInstance>*.

$$S \rightarrow < componets > S_1 < /componets >$$
$$S_1 \rightarrow \{S_1\}(S_2 \mid S_3)$$
$$S_2 \rightarrow \text{<iocomponent>}A_1\text{</iocomponent>}$$
$$S_3 \rightarrow \text{<controler>}A_1\text{</controler>}$$
$$A_1 \rightarrow (A_{11}A_{12}|A_{12}A_{11}|A_{21}A_{22}|A_{22}A_{21}|A_{31}A_{32}|A_{32}A_{31})$$
$$A_{11} \rightarrow (N_1D_1|D_1N)_1$$
$$A_{12} \rightarrow (C_1P_1|T_1P_1)$$
$$A_{21} \rightarrow (N_1P_1|P_1N_1)$$
$$A_{22} \rightarrow (C_1D_1|D_1C_1)$$
$$A_{31} \rightarrow (P_1D_1|D_1P_1)$$
$$A_{32} \rightarrow (C_1N_1|N_1C_1)$$
$$N_1 \rightarrow \text{<name>}L_1\text{<name>}$$
$$P_1 \rightarrow \text{<paket>}L_1\text{</paket>}$$
$$C_1 \rightarrow \text{<class>}L_1\text{</class>}$$
$$D_1 \rightarrow \text{<config>}L_4\text{</config>}$$
$$L_1 \rightarrow \{L_1\}L_2$$
$$L_2 \rightarrow [\text{a-zA-Z._ }]$$
$$L_3 \rightarrow \{L_3\}(L_1|T_1)$$
$$L_4 \rightarrow \{L_4;\}L_1 = L_3$$
$$T_1 \rightarrow (1|2|3|4|5|6|7|8|9)T_2$$
$$T_2 \rightarrow (1|2|3|4|5|6|7|8|9|0)*$$

Figura 7: Gramática para el archivo de meta-datos para la instanciación de *plugins*

4 CONTROLADOR VISUAL

Con el objetivo de controlar el robot COOPER mediante visión, se han desarrollado dos tipos de controladores visuales distintos. La arquitectura software descrita en este artículo permite trabajar

sobre el diseño del controlador utilizando como herramientas distintos controladores diseñados previamente. En concreto, se han desarrollado dos controladores visuales: un controlador visual indirecto y un controlador visual directo.

El control visual es una técnica ampliamente desarrollada en la literatura en las últimas décadas [5]. Estos sistemas controlan el movimiento de la cámara (normalmente situada en el extremo de un robot manipulador) en función de la variación en la imagen de unas determinadas características visuales. En función de la salida del controlador se pueden encontrar sistemas de control visual indirecto cuando la salida del controlador es la velocidad (lineal y angular) de la cámara [3],y sistemas de control visual directo cuando la salida del controlador es el par que se le debe aplicar a cada articulación del robot [8].

Para un sistema de control visual indirecto, la ley de control proporciona la velocidad de la cámara, $\mathbf{v}^C$, que minimice exponencialmente el error en imagen:

$$\mathbf{v}^C = -\lambda \hat{\mathbf{L}}_s^+ \mathbf{e}_s \qquad (1)$$

donde λ es una ganancia proporcional, $\hat{\mathbf{L}}_s^+$ es una estimación de la pseudoinversa de la matriz de interacción [3] y $\mathbf{e}_s = (\mathbf{s} - \mathbf{s}_d)$ es el error en el espacio imagen, donde $\mathbf{s}$ representa las características visuales extraídas en la posición actual del robot y $\mathbf{s}_d$ indica el conjunto de características visuales que se obtendrían en la posición final deseada del robot.

Esta velocidad, $\mathbf{v}^C$, puede transformarse fácilmente a una velocidad en el extremo del robot. Utilizando la arquitectura software desarrollada para el control de COOPER resulta sencillo implementar el controlador visual indirecto propuesto. Para ello, se emplea un controlador básico desarrollado para la arquitectura software consistente en posicionar a COOPER a partir de la velocidad en el extremo. La estructura software descrita en este artículo permite desarrollar fácilmente distintos esquemas de control que aprovechan otros controladores previamente desarrollados, de forma que se obtenga una estructura de control anidada que mejore exponencialmente los tiempos de desarrollo de nuevos esquemas de control.

El otro controlador visual desarrollado sobre la plataforma software descrita en el Apartado 3, se ha implementado gracias al conocimiento preciso de los parámetros dinámicos de COOPER. Este sistema de control visual directo se encarga de aplicar al robot los pares articulares adecuados para alcanzar la posición deseada. En control basado en Jacobiana traspuesta se emplea la siguiente ley de control:

$$\tau = \mathbf{J}(\mathbf{q},\mathbf{s},\mathbf{Z})^T K_p \mathbf{e} - K_v \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \qquad (2)$$

donde $K_p \in \Re^{2Mx2M}$ y $K_v \in \Re^{nxn}$ son las ganancias proporcionales y derivativas respectivamente. Estas constantes son simétricas y definidas positivas. G es la matriz de gravedad y $\tau \in \Re^{nx1}$ son los pares articulares. En el primer término de (2) aparece la misma función de error en el espacio imagen que en el controlador indirecto, $\mathbf{e} = (\mathbf{s} - \mathbf{s}_d)$. Un estudio acerca de la estabilidad de este regulador puede verse en [9] $J(q,s,Z)^T$ se define como la traspuesta de la matriz Jacobiana y se calcula de acuerdo con la siguiente expresión:

$$\mathbf{J}(\mathbf{q},\mathbf{s},\mathbf{Z}) = \mathbf{L}_s(\mathbf{s},\mathbf{Z}) \cdot \mathbf{J}_g(\mathbf{q}) \in \mathbb{R}^{2M \times n} \qquad (3)$$

donde $\mathbf{L}_s(\mathbf{s},\mathbf{Z})$ representa la matriz de interacción y $\mathbf{J}_g(\mathbf{q})$, la matriz Jacobiana del robot.

En la Figura 8 se puede ver la comparativa de ambos esquemas. A la izquierda, se muestra el esquema de un controlador compuesto por un controlador visual indirecto CVI, que utiliza servicios de un controlador de velocidad en el extremo CVE que a su vez utiliza los servicios de un controlador de velocidades articulares CVA. A la derecha en contraposición se muestra un controlador visual directo, que actúa directamente sobre el modelo, obteniendo así una arquitectura más acoplada pero al mismo tiempo más eficiente.

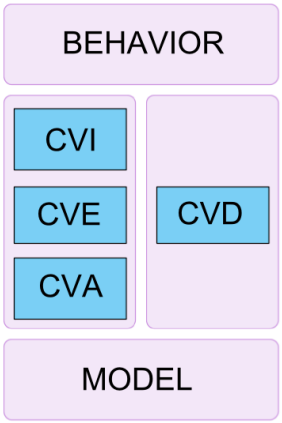

Figura 8: Comparativa entre un controlador visual directo y un controlador visual indirecto.

5 CONCLUSIONES

La plataforma software desarrollada, permite la creación de software para el control de robots, de una manera sencilla y rápida. Los esquemas obtenidos gracias a la disposición de la arquitectura definida

MBC, son fácilmente trazables y altamente adaptables a situaciones concretas para el control de estructuras robóticas o articuladas. Se han desarrollador dos tipos de controlador visual que han sido aplicados al robot COOPER. Los resultados muestran como el controlador visual indirecto requiere de la intervención de más elementos en la consecución de las acciones que un controlador visual directo, obteniendo así con éste último una mejor respuesta ante los estímulos detectados por la cámara.

Agradecimientos

Este trabajo está financiado por el Ministerio de educación y ciencia (Proyecto DPI2008-02 647).

Referencias

[1] Arkin, Ronald C., (1998) Behavior-Based Robotics, vol., The MIT Press.

[2] Charles Lesire, David Doose, Hugues Cassé, (2011), "Validation of real-time properties of a robotic software architecture", in *6th National Conference on Control Architectures of Robots*, Grenoble , France.

[3] Chaumette, F. y Hutchinson, S.A., (2006) "Visual servo control. I. Basic approaches", *Robotics & Automation Magazine, IEEE*, vol. 13, pp. 82-90.

[4] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerland, Michael Stal, (1996) Pattern - Oriented Software Architecture, vol. 1, Wiley.

[5] Garcia, Gabriel, Corrales, Juan, Pomares, Jorge y Torres, Fernando, (2009) "Survey of Visual and Force/Tactile Control of Robots for Physical Interaction in Spain", *Sensors*, vol. 9, pp. 9689-9733.

[6] Jackson, J., (2007) "Microsoft Robotics Studio: A Technical Introduction", *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, vol. 14, pp. 82-87.

[7] José M. Cañas, Jesús Ruíz-Ayúcar, Carlos Agüero, Francisco Matín, (2007) "Jde-neoc: Componente Oriented Software Architecture for Robotics", *Journal of Physical Agents*, vol. 1, pp. 1-6.

[8] Kelly, R. y Marquez, A., (1995) "Fixed-eye direct visual feedback control of planar robots", *Journal of Systems Engineering*, vol. 5, pp. 239-248.

[9] Kelly, Rafael, Cervantes, Ilse, Alvarez-Ramirez, Jose, Bugarin, Eusebio y Monroy, Carmen, (2008), "On Transpose Jacobian Control for Monocular Fixed-Camera 3D Direct Visual Servoing", in *Robot Manipulators*, pp. 243-258.

[10] Larman, Craig, (2004) An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third ed. vol. 1, Addison Wesley Professional.

[11] Patrick M. McDowell, Brian S. Bourgeois and Frederick E. Petry., (2009) "Robot Control in Dynamic Environments Using Memory-Based Learning", *DESIGN AND CONTROL OF INTELLIGENT ROBOTIC SYSTEMS*, vol. 177, pp. 153-170.

[12] Richard T. Vaughan, Brian P. Gerkey, Andrew Howard, (2003), "On device abstractions for portable, reusable robot code", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Las Vegas, (USA), pp. 2421-2427.

[13] T. Collett, B. MacDonald, B. Gerkey, (2005), "Player 2.0: Toward a Practical Robot Programming Framework", in *Australasian Conf. on Robotics and Automation (ACRA 2005)*, Sydney (Australia).

[14] Vukobratović, Miomir, (1997) "How to Control Robots Interacting with Dynamic Environment", *Journal of Intelligent and Robotic Systems*, vol. 19, pp. 119-152.