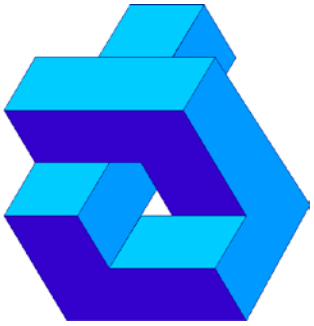
UNIVERSIDAD POLITECNICA DE MADRID



DEPARTAMENTO DE AUTOMATICA
INGENIERIA ELECTRONICA
E INFORMATICA INDUSTRIAL

División de Ingeniería de Sistemas
y Automática (DISAM)

ICE – The Integrated Control Environment

Definición e implantación de un IDE extensible para la construcción de sistemas autónomos dentro del proyecto ASys.

AUTOR: Miquel González Oyaga

TUTOR: Carlos Hernández Corbato
Ricardo Sanz Bravo

Agradecimientos

A mi madre, por estar ahí siempre sin olvidarse de su hijo. Por sus visitas esporádicas, por su apoyo incondicional, por no dejar nunca de creer, por ser un modelo de constancia y sacrificio, por haber dado todo sin esperar nada a cambio . Todo ha sido gracias a ti.

A mi padre, por sus grandes recibimientos. Por dejarme su portátil en un momento tan oportuno. Por sus tomates, sin ellos nada hubiera sido lo mismo.

A mis abuelos por su calidad humana y por la ilusión que transmiten. A mi tía Pili por su buena energía y por su forma de ver la vida. A mis primas, Cristina y Neus por no olvidarse nunca de su primo lejano. A mi primo Borja, por su ayuda en Barcelona y por su buen humor continuo.

A mis compañeros de piso y a la vez amigos por los buenos momentos vividos a lo largo de estos años. A Javier Gómez Miñambres por sus espaguetis con albóndigas y por sus cada vez menos epic meals. A Javier Mata Sanz por ser un gran compañero en el duro camino diario y por su justicia natural.

A mi "Abuela" María Ángeles.

A Zuzana por su apoyo en estos últimos meses y por la luz que pone cada día en mi vida.

A mis compañeros de clase a los largo de estos años en las dos universidades. A Charlio por sus Molinos y el logo del trabajo de robótica. A Marta por ser la dele de los AIAIAI. A Miguel a pesar de emigrar a Industriales. A Mer por el viaje a Irlanda. A Iván por su humor negro. A Fran por su morcilla. A Pablo por ser el delegado, una gran persona y por su ayuda con las mudanzas. A Fernandisko por su Arehucas y por los Carnavales de Las Palmas 2011. A Turley porque está muy loco.

A mis compañeros y amigos del colegio mayor. A SuperJota por dejarme su habitación en Julio de 2012. A Ardillita por hacerme compañía en ese infernal verano y por el interrail 2011. A Jungis por su "uruguasho".

A mis amigos de Pamplona. A Serena por ser mi abogado y un gran amigo. A Xabi por su ojo clínico y su Burguetazo. A Navarrete por enseñarme el primer año Herjomar. A Rafa por su ironía constante y su mee. A Ardaiz porque sin Pon no hay Pin.

A Carlos, por todo el apoyo y dedicación. Sin él no hubiera sido posible este proyecto. A Ricardo, por ayudarme cuando más lo necesitaba.

A todos, muchas gracias.

Jamás te rindas.

Resumen

La oportunidad de desarrollar y probar software en una herramienta personalizada para un grupo de investigación, es un elemento soñado por parte de las compañías. Esto, además de crear una imagen de marca, fomenta un sentimiento de grupo que permite al desarrollador sentirse cómodo en un entorno conocido y amigable. También permite a la compañía establecer una serie de programas determinados con los que trabajar y de esta manera unificar las herramientas que se utilizarán en el desarrollo de un producto.

Es en este marco en el que se desarrolla este proyecto fin de carrera. Continuando con la idea inicial del grupo ASLab de crear un ICe (Integrated Control environment) para el grupo de trabajo, este proyecto sería la primera piedra. Desde el ASLab se deseaba tener un ambiente de trabajo personalizado, creado en y para la herramienta más utilizada por ASLab, Eclipse. La idea es crear un RCP para esta plataforma, en el que manejar sistemas autónomos y donde se puedan integrar, desarrollar y probar las herramientas ya creadas y las que están por crear.

Este proyecto es el encargado de empezar con este trabajo pero no de terminarlo ya que, con cada nueva herramienta desarrollada, este ICe puede cambiar. Está desarrollado sobre Eclipse e incorpora a éste una perspectiva ASLab en la que por defecto se incorporan varias vistas.

Además de esta perspectiva, se ha desarrollado una interfaz amigable, compuesta por varios proyectos plug-ins de Eclipse, con la que comunicarse con las librerías de ROS fuerte sin necesidad de acceder a ellas por medio de la consola como se realiza corrientemente. De esta manera el proyecto mostrará la información de forma más clara de los nodos activos, tópicos que estos publican, así como la posibilidad de lanzar nuevos nodos o eliminar aquellos que no se deseen tener.

ÍNDICE

ÍNDICE DE FIGURAS

Capítulo 1

1. Introducción

1.1. Motivación

Personalmente considero el desarrollo del proyecto fin de carrera como un reto y una oportunidad única para aprender sobre un ámbito de la ingeniería de tu propia elección. Es por esto que he elegido el departamento de automática y electrónica, concretamente el LABoratorio de Sistemas Autónomos (ASLab) de la Universidad Politécnica de Madrid para desarrollar mi proyecto ya que desde siempre me ha atraído mucho la idea de desarrollar, configurar y personalizar el software de un computador.

Además supone un reto personal el enfrentarse a un nuevo sistema y metodología de trabajo y al uso de nuevas herramientas de desarrollo sin tener conocimientos previos. De esta manera, el entorno de trabajo es nuevo para mí ya que nunca he utilizado el sistema operativo Ubuntu, el entorno de desarrollo de Eclipse, el conjunto de librerías de ROS o un repositorio online para el control de versiones.

Es una oportunidad muy interesante poder trabajar con Eclipse ya que se trata de una de las herramientas de programación de software más utilizadas por los desarrolladores. Al tratarse de un programa de software libre existe gran cantidad de información sobre la misma, y da la oportunidad de desarrollar plugins para modificar, personalizar o ampliar su funcionalidad lo que resulta esencial para un grupo de investigación. El trabajar con las librerías para el control de robots de ROS (Robot Operating System) también es una buena oportunidad para iniciarme en una herramienta muy utilizada en la ingeniería.

Considero además muy atractiva la oportunidad de poder desarrollar una herramienta nueva desde cero para poder conocer como es la estructura de una interfaz que se relaciona con el usuario basada en lenguaje de programación Java.

1.2. Marco del proyecto

El proyecto se ha desarrollado en el Laboratorio de Sistemas Autónomos de la Universidad Politécnica de Madrid, **ASLab** (Autonomous Systems Laboratory) se enmarca dentro del proyecto ASys (Autonomous Systems) que se desarrolla en el laboratorio de sistemas autónomos perteneciente a la división de ingeniería de sistemas y automática DISAM.

El laboratorio ASLab posee varias líneas de investigación abiertas, todas ellas orientadas hacia el cumplimiento de objetivos a medio y largo plazo del grupo de investigación, que están recogidos dentro del proyecto ASys.

Figura 1 Logo de la UPM y ASLab.

Proyecto ASys

ASys es el proyecto principal del grupo ASLab. Se trata de un programa de investigación a largo plazo, dirigido al desarrollo de una tecnología para el desarrollo sistemático de sistemas autónomos. Uno de los objetivos principales es el desarrollo de tecnologías universales para la construcción de sistemas de gran autonomía. Éste incluye métodos para el análisis de requisitos, arquitecturas de la autonomía y activos reutilizables, como herramientas, ontologías, etc.

Figura 2 Logo de ASys.

ASys y la ingeniería de sistema autónomos

Los sistemas autónomos durante su operación pueden necesitar soportar perturbaciones externas, cambios en las especificaciones y/o parámetros originales, o dinámicas inesperadas que no siempre son predecibles. Los ingenieros de automática y de producción desean sistemas capaces de trabajar por su cuenta.

Sin embargo, se considera que esta autonomía de un sistema está limitada (por ejemplo, por el ingeniero que se ha encargado del desarrollado del mismo). Esta limitación, en cierto sentido, es una gran diferencia entre la autonomía natural y la artificial. Los sistemas de autonomía natural se consideran más autónomos en el sentido más etimológico de la palabra (siguiendo sus propias leyes de comportamiento). Por otro lado, los sistemas de autonomía artificial se comportan de forma autónoma pero sólo en cierto grado ya que estando limitados por restricciones externas (por ejemplo, relacionadas con la seguridad, la economía o el impacto medioambiental).

El proyecto ASys considera que todo dominio de la autonomía es abstracto e independiente de cualquier sistema autónomo y aplicación particular. Esto obliga a considerar una gran variedad de sistemas, desde aplicaciones basadas en robots a procesos continuos, nodos activos o sistemas software. Ref : The ASys Vision. Ricardo Sanz and Manuel Rodriguez. ASLAB Technical Report R-2007-001. UPM Autonomous Systems Laboratory, 2007.

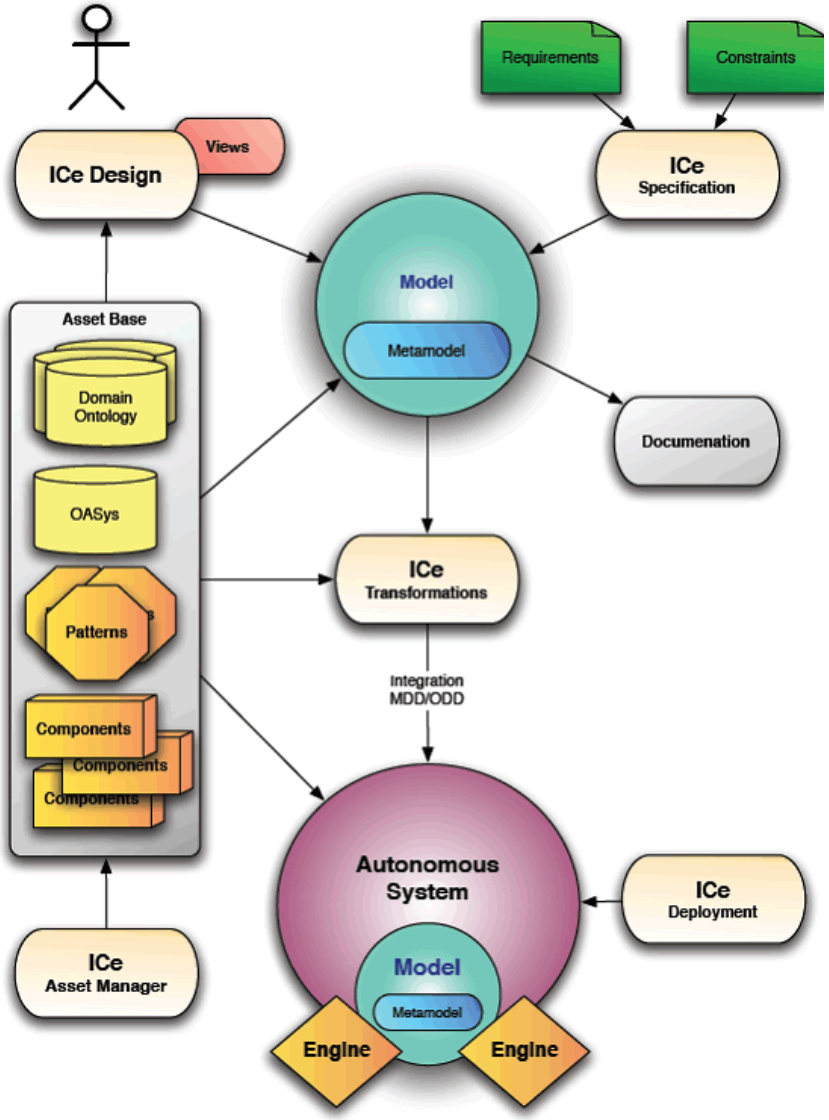


Figura 3 Estructura general del proyecto ASys.

El programa de investigación considera diferentes elementos para materializar las ideas previas (Figura 3): un enfoque de diseño centrado en la arquitectura, una metodología para diseñar sistemas autónomos basados en modelos, una base de activos que aportan elementos modulares para que realicen los roles especificados en las arquitecturas de los patrones.

Los procesos de ingeniería cubren desde las especificaciones y conocimientos iniciales, hasta el producto final, que es el sistema autónomo. La primera etapa del proyecto de investigación se centra en ontologías, como una conceptualización común para describir el área de conocimiento. Son tratados tanto el estudio de ontologías de dominios existentes, como el desarrollo de una ontología para el dominio de los sistemas autónomos OASys. Uno de los objetivos centrales es producir una metodología que explote estas ontologías para generar modelos basados en el conocimiento que contienen. Esto se realiza en la fase de diseño como se refleja en la parte superior de la figura 3.

El enfoque del proyecto ASys es conceptual y centrado en la arquitectura. La conveniencia del control existente y las arquitecturas de control cognitivo se determinarán en términos de cómo encajan con las ideas de investigación de ASys y los productos desarrollados (ontologías, modelos, perspectivas, máquinas). Se considerarán también las posibles adaptaciones y extensiones para las arquitecturas analizadas.

Un entorno importantísimo del proyecto ASys es el interfaz con la que va a trabajar el usuario y como se va a relacionar con los modelos. Aquí es donde encaja el concepto de ICe, Integrated Control Environment. Este es el lugar donde se van a desarrollar y probar los modelos y proyectos de los miembros del proyecto ASys

1.3. Objetivos del proyecto

ASys plantea un marco muy general para el desarrollo de sistemas autónomos. Muchos de los proyectos desarrollados por el grupo ASLab implementan diversas librerías para trabajar con los diferentes sistemas y diversas herramientas dependiendo del ámbito del proyecto. Es por esto que surge la necesidad de crear una herramienta nueva para intentar unificar los sistemas, herramientas y librerías con las que se trabaja y facilitar así la implementación de las mismas por parte del grupo de investigación.

Profundizando un poco más, el grupo de investigación lleva tiempo estudiando la posibilidad de desarrollar un nuevo proyecto llamado **ICe** (Integrated Control Environment). El propósito fundamental de este proyecto es crear un desarrollo y entorno de operación para los sistemas ASys. El objetivo de este proyecto es la construcción de un entorno RCP, Rich Client Platform, basado en eclipse donde manejar los sistemas autónomos desarrollados en proyectos anteriores. Además este entorno permitirá integrar las herramientas de ASys ya disponibles e integrar las que se desarrollarán en el futuro.

Dada esta necesidad y deseo de unificar las herramientas de desarrollo, es en este contexto donde el proyecto marca un objetivo claro.

1.3.1. Objetivo del proyecto

El propósito fundamental del proyecto costa de dos partes bien diferenciadas.

La primera de ellas es el estudio, definición y el desarrollo de un marco de trabajo, **ICe** (Integrated Control environment), para que el grupo de investigación pueda trabajar de manera unificada con una herramienta común. El propósito de esta parte viene dado por el objetivo inicial del proyecto ICe de ASLab de construir proyecto basado en un RCP de Eclipse para personalizar el entorno de desarrollo del grupo.

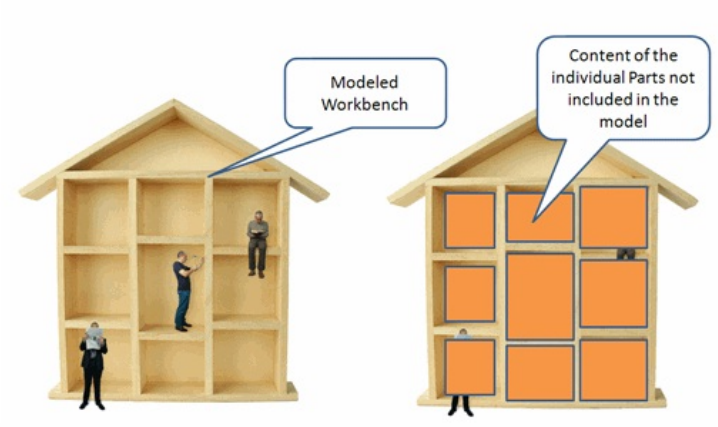


Figura 4 Visión metafórica del objetivo del proyecto.

El segundo objetivo será la implementación de una herramienta que permita trabajar con las librerías ROS de manera más amigable. Dada la exigencia inicial de que la aplicación este construida en eclipse, se aprovecharan las características de este programa para generar tantas vistas como sean necesarias para crear una interfaz amigable y trabajar con las librerías de ROS de manera intuitiva.

La idea es que el usuario no tenga que seguir escribiendo y lanzando comandos por el terminal del sistema para interaccionar con los nodos activos como se hace ahora. La herramienta presentará los elementos necesarios para hacer al usuario una interactuación sencilla. Además deberá mostrar la información de lo que está pasando internamente en el sistema para que el usuario sea capaz de tomar las decisiones correspondientes.

El usuario podrá trabajar con sus proyectos a través de la interfaz ofrecida por su herramienta de desarrollo, y podrá hacer uso de las nuevas funcionalidades ofrecidas de forma sencilla siguiendo las indicaciones del manual de usuario.

Así los principales objetivos son:

- Estudio de las herramientas utilizadas por el grupo ASLab.
- Estudio del entorno Eclipse y las posibilidades que ofrece.
- Desarrollo de una herramienta que de soporte la estructura del proyecto ICe.
- Diseño de una perspectiva ASLab de eclipse.
- Diseño de las vistas necesarias para la interfaz con ROS.
- Desarrollo de un manual de usuario de la herramienta.
- Documentación del trabajo realizado.

1.3.2. Alcance del proyecto

El alcance del proyecto consiste en lograr una serie de hitos durante una etapa de documentación y análisis del problema, el desarrollo de unos productos concretos durante la etapa de investigación y desarrollo, y la realización de la documentación que recoja el trabajo realizado. Paso por paso, el alcance del proyecto incluye las siguientes tareas:

- Estudio de la base teórica de las metodologías sobre las que se fundamenta el proyecto.
- Estudio del estado de investigación en la materia en la que se va a trabajar.
- Familiarización con el entorno y las herramientas desarrollo del grupo ASLab y del proyecto ICe.
- Análisis de las soluciones disponibles para cumplir los requisitos del proyecto.
- Desarrollo de la perspectiva ASLab.
- Desarrollo de las vistas necesarias para la interfaz con ROS.
- Realización de pruebas y verificación.
- Desarrollo de un manual de usuario.
- Desarrollo de la memoria final del proyecto.

1.4. Estructura del documento

Estado del Arte. Se explica que es y cómo se encuentran las plataformas para el desarrollo de software. Además se habla de las librerías ROS y su evolución histórica. También, se estudian los Model Driven Engineering.

Entorno de desarrollo y herramientas utilizadas. Se presenta el entorno usado para la realización del proyecto, así como la explicación del funcionamiento de las herramientas utilizadas para desarrollar el proyecto.

Arquitectura y descripción del sistema. Se analizan los requisitos del proyecto, se desarrolla una arquitectura del sistema y se desarrolla la herramienta.

Ejemplos de uso. Se muestra mediante ejemplos como funcionan todos los componentes de la herramienta.

Resultados y conclusiones. Se muestran los resultados obtenidos y se hace un repaso del trabajo realizado.

Líneas futuras. Se especifica la arquitectura interna y cómo sería posible continuar desarrollando esta herramienta.

Alcance y planificación. Estructura de descomposición del proyecto (EDP) y diagrama de Gantt que representan el trabajo llevado a cabo durante la realización del proyecto.

Anexos. Manual de usuario y documentación relativa al código de la herramienta desarrollada.

Capítulo 2

2. Estado del arte

2.1. Plataformas para el desarrollo de software

A la hora de desarrollar software se deberá elegir cuidadosamente una plataforma adecuada ya que esta decisión será fundamental para el desarrollo del proyecto. Aquí se presenta un breve estudio sobre dos de los entornos de desarrollo más extendidos en el mundo, Microsoft Visual Studio y Eclipse.

Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado, (IDE, por sus siglas en inglés), para sistemas operativos "Windows". Soporta varios lenguajes de programación tales "Visual C++", C Sharp, Visual J# y Visual Basic , al igual que entornos de desarrollo web como aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Figura 5 Logo de Microsoft Visual Studio.

El desarrollador de Microsoft Visual Studio, como no podía ser de otra manera, es Microsoft. El lanzamiento oficial de primera versión se produjo el 30 de Julio de 1998 y el lanzamiento de su última versión, Visual 2012, se ha producido el 15 de Agosto de 2012 en versiones para ordenadores de 32 y 64 bits.

Las versiones de este producto han sido:

Microsoft Visual Studio 97
Microsoft Visual Studio 6
Microsoft Visual Studio .NET (2002)
Microsoft Visual Studio .NET 2003
Microsoft Visual Studio 2005
Microsoft Visual Studio 2008
Microsoft Visual Studio 2010
Microsoft Visual Studio 2012

Figura 6 Versiones de MVS.

Si se estudia brevemente la historia de cada versión se puede crear una visión general del cómo y el porqué de la constante evolución del producto.

Visual Studio 6.0 se lanzó en 1998 y fue la última versión en ejecutarse en la plataforma Windows 9x. Los números de versión de todas las partes constituyentes pasaron a 6.0, incluyendo Visual J++ y Visual InterDev, que se encontraban en las versiones 1.1 y 1.0 respectivamente. Esta versión fue la base para el sistema de desarrollo de Microsoft para los siguientes 4 años, en los que Microsoft migró su estrategia de desarrollo al .NET.

Visual Studio 6.0 fue la última versión en que Visual Basic se incluía de la forma en que se conocía hasta entonces; versiones posteriores incorporarían una versión muy diferente del lenguaje con muchas mejoras, fruto de la plataforma .NET. También supuso la última versión en incluir Visual J++, que proporcionaba extensiones de la plataforma Java, lo que lo hacía incompatible con la versión de Sun Microsystems. Esto acarreó problemas legales a Microsoft, y se llegó a un acuerdo en el que Microsoft dejaba de comercializar herramientas de programación que utilizaran la máquina virtual de Java.

Aunque el objetivo a largo plazo de Microsoft era unificar todas las herramientas en un único entorno, esta versión en realidad añadía un entorno más a Visual Studio 5.0: Visual J++ y Visual Interdev se separaban del entorno de Visual C++, al tiempo que Visual FoxPro y Visual Basic seguían manteniendo su entorno específico.

En la nueva versión, **Microsoft Visual Studio .NET (2002**), se produjo un cambio sustancial, puesto que supuso la introducción de la plataforma .NET de microsoft. .NET es una plataforma de ejecución intermedia multilenguaje, de forma que los programas desarrollados en .NET no se compilan en lenguaje máquina, sino en un lenguaje intermedio denominado Microsoft Intermediate (MSIL).

En una aplicación MSIL, el código no se convierte a lenguaje máquina hasta que ésta se ejecuta, de manera que el código puede ser independiente de plataforma (al menos de las soportadas actualmente por .NET). Las plataformas han de tener una implementación de infraestructuras de lenguaje común (CLI) para poder ejecutar programas MSIL. Actualmente se pueden ejecutar programas MSIL en Linux y Mac OS X usando implementaciones de .NET que no son de Microsoft, tales como Proyecto mono y DotGNU

Visual Studio .NET 2002 supuso también la introducción del lenguaje C#, un lenguaje nuevo diseñado específicamente para la plataforma .NET, basado en C++ y Java. Se presentó también el lenguaje J# (sucesor de J++), el cual, en lugar de ejecutarse en una máquina virtual de Java, se ejecuta únicamente en el framework .NET. El lenguaje Visual Basic fue remodelado completamente y evolucionó para adaptarse a las nuevas características de la plataforma .NET, haciéndolo mucho más versátil y dotándolo con muchas características de las que carecía. Algo similar se llevó a cabo con C++, añadiendo extensiones al lenguaje llamadas Managed Extencions for C++ con el fin de que los programadores pudieran crear programas en .NET. Por otra parte, Visual FoxPro pasa a comercializarse por separado.

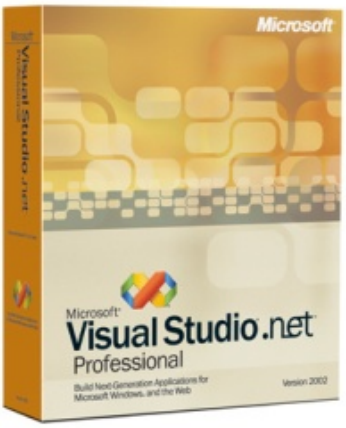


Figura 7 Producto MVS en su versión de 2002.

Todos los lenguajes se unifican en un único entorno. La interfaz se mejora notablemente en esta versión, siendo más limpia y personalizable.

Esta versión requiere un sistema operativo basado en Windows NT. La versión interna de Visual Studio .NET es la 7.0.

Visual Studio .NET 2003 supone una actualización menor de Visual Studio .NET. Se actualiza el .NET Framework a la versión 1.1. También se añade soporte con el fin de escribir aplicaciones para determinados dispositivos móviles, ya sea con ASP.NET o con el .NET Compact Framework. Además, el compilador de Visual C++ se mejora para cumplir con más estándares: el Visual C++ Toolkit 2003.

Visual Studio 2003 se lanza en 4 ediciones: Academic, Professional, Enterprise Developer y Enterprise Architect. La edición Enterprise Architect incluía una implementación de la tecnología de modelado Microsoft Visio, que se centraba en la creación de representaciones visuales de la arquitectura de la aplicación basadas en UML. También se introdujo "Enterprise Templates", para ayudar a grandes equipos de trabajo a estandarizar estilos de programación e impulsar políticas de uso de componentes y asignación de propiedades.

Microsoft lanzó el Service Pack 1 para Visual Studio 2003 el 13 de septiembre de 2006.La versión interna de Visual Studio .NET 2003 es la 7.1 aunque el formato del archivo que emplea es el 8.0.Es compatible solo con Windows XP, Windows Server 2003 o anteriores

Visual Studio 2005 se empezó a comercializar a través de Internet a partir del 4 de octubre de 2005 y llegó a los comercios a finales del mes de octubre en inglés. En castellano no salió hasta el 4 de febrero de 2006. Microsoft eliminó la coletilla .NET de su nombre, pero eso no indica que se alejara de la plataforma .NET, de la cual se incluyó la versión 2.0.

La actualización más importante que recibieron los lenguajes de programación fue la inclusión de tipos genéricos, similares en muchos aspectos a las plantillas de C++. Con esto se consigue encontrar muchos más errores en la compilación en vez de en tiempo de ejecución, incitando a usar comprobaciones estrictas en áreas donde antes no era posible. C++ tiene una actualización similar con la adición de C++/CLI como sustituto de C# manejado.

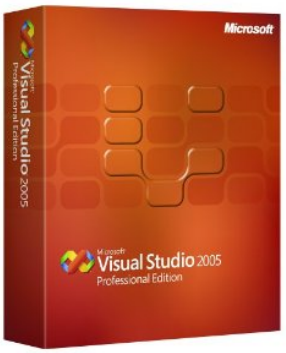


Figura 8 Producto MVS en su versión de 2005.

Se incluye un diseñador de implantación, que permite que el diseño de la aplicación sea validado antes de su implantación. También se incluye un entorno para publicación web y pruebas de carga para comprobar el rendimiento de los programas bajo varias condiciones de carga.

Visual Studio 2005 también añade soporte para arquitecturas de 64 bits. Aunque el entorno de desarrollo sigue siendo una aplicación de 32 bits, Visual C++ 2005 soporta compilación para x86-64 AMD 64, Intel 64 e IA-64. El SDK incluye compiladores de 64 bits así como versiones de 64 bits de las librerías.

Visual Studio 2005 tiene varias ediciones radicalmente distintas entre sí: Express, Standard, Professional, Tools for Office y 5 ediciones Visual Studio Team System. Éstas últimas se proporcionaban conjuntamente con suscripciones a MSDN cubriendo los 4 principales roles de la programación: Architects, Software Developers, Testers y Database Professionals. La funcionalidad combinada de las 4 ediciones Team System se ofrecía como la edición Team Suite. Por otra parte, Tools for the Microsoft Office System está diseñada para extender la funcionalidad a Microsoft Office.

Visual Studio 2008 fue publicado (RTM) el 17 de noviembre de 2007 en inglés, mientras que la versión en castellano no fue publicada hasta el 2 de febrero de 2008.

El nuevo framework (.NET 3.5) está diseñado para aprovechar las ventajas que ofrece el nuevo sistema operativo Windows Vista a través de sus subsistemas Windows Communication Foundation (WCF) y Windows Presentation Foundation (WPF). El primero tiene como objetivo la construcción de aplicaciones orientadas a servicios, mientras que el último apunta a la creación de interfaces de usuario más dinámicas que las conocidas hasta el momento.

A las mejoras de desempeño, escalabilidad y seguridad con respecto a la versión anterior, se agregan, entre otras, las siguientes novedades:

La mejora en las capacidades de pruebas unitarias permiten ejecutarlas más rápido independientemente de si lo hacen en el entorno IDE o desde la línea de comandos. Se incluye además un nuevo soporte para diagnosticar y optimizar el sistema a través de las herramientas de pruebas de Visual Studio. Con ellas se podrán ejecutar perfiles durante las pruebas para que ejecuten cargas, prueben procedimientos contra un sistema y registren su comportamiento, y utilizar herramientas integradas para depuración de programas.

Con Visual Studio Tools for Office (VSTO) integrado con Visual Studio 2008 es posible desarrollar rápidamente aplicaciones de alta calidad basadas en la interfaz de usuario (UI) de Office que personalicen la experiencia del usuario y mejoren su productividad en el uso de Word, Excel, PowerPoint, Outlook, Visio, InfoPath y Project. Una completa compatibilidad para implementación con ClickOnce garantiza el entorno ideal para una fácil instalación y mantenimiento de las soluciones Office.

Visual Studio 2008 permite incorporar características del nuevo Windows Presentation Foundation sin dificultad tanto en los formularios de Windows existentes como en los nuevos. Ahora es posible actualizar el estilo visual de las aplicaciones al de Windows Vista debido a las mejoras en Microsoft Foundation Class Library (MFC) y Visual C++. Visual Studio 2008 permite mejorar la interoperabilidad entre código nativo y código manejado por .NET. Esta integración más profunda simplificará el trabajo de diseño y codificación.

Visual Studio 2008 ahora permite la creación de soluciones multiplataforma adaptadas para funcionar con las diferentes versiones de .NET Framework: 2.0 (incluido con Visual Studio 2005), 3.0 (incluido en Windows Vista) y 3.5 (incluido con Visual Studio 2008).

.NET Framework 3.5 incluye la biblioteca ASP.NET AJAX para desarrollar aplicaciones web más eficientes, interactivas y altamente personalizadas que funcionen para todos los navegadores más populares y utilicen las últimas tecnologías y herramientas Web, incluyendo Silverlight y Popfly.

Visual Studio 2010 es la versión más reciente de esta herramienta, acompañada por .NET Framework 4.0. La fecha del lanzamiento de la versión final fue el 12 de abril de 2010.

Hasta ahora, uno de los mayores logros de la versión 2010 de Visual Studio ha sido el de incluir las herramientas para desarrollo de aplicaciones para Windows 7, tales como herramientas para el desarrollo de las características de Windows 7 y la Ribbon Preview para WPF.

Entre sus más destacables características, se encuentran la capacidad para utilizar múltiples monitores, así como la posibilidad de desacoplar las ventanas de su sitio original y acoplarlas en otros sitios de la interfaz de trabajo.

Además ofrece la posibilidad de crear aplicaciones para muchas plataformas de Microsoft, como Windows, Azure, Windows Phone 7 o Sharepoint. Microsoft ha sido sensible a la nueva tendencia de las pantallas táctiles y con este Visual Studio 2010 también es posible desarrollar aplicativos para pantallas multitáctiles.

Visual Studio 2012 fue dado a conocer mediante la Release Candidate del nuevo entorno de programación de "Windows", cual integra completo soporte para el actual y en fase de desarrollo Windows 8, Microsoft Visual Studio 2012 RC tiene como características el desarrollo completo e integro en el estilo Metro de Windows 8, además soporte para antiguas versiones de Windows al estilo clásico. Visual Studio 2012 ya se encuentra disponible en su última versión que sería la Ultimate puede ser descargado y probado desde la pagina de Microsoft.

ECLIPSE

Eclipse comenzó como un proyecto de IBM Canadá. Fue desarrollado por OTI (Object Technology International) como reemplazo de VisualAge también desarrollado por OTI. En noviembre del 2001, se formó un consorcio para el desarrollo futuro de Eclipse como código abierto. En 2003, fue creada la fundación independiente de IBM.

Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la Licencia Pública Común pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Fundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU.

Figura 9 Logo de la primera versión de Eclipse.

Indagando un poco más en lo que es Eclipse, se puede decir que es un entorno de desarrollo de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entorno de desarrollo integrado, como el IDE de lenguaje de programación Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azure. Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

El resumen de las versiones de Eclipse que han ido existiendo es el siguiente:

Versión	Fecha de lanzamiento	Versión de plataforma
Juno	8 de junio de 2012	4.2
Indigo	22 de junio de 2011	3.7
Helios	23 junio de 2010	3.6
Galileo	24 de junio de 2009	3.5
Ganymede	25 junio de 2008	3.4
Juno	29 de junio de 2007	3.3
Indigo	30 de junio de 2006	3.2

Figura 10 Tabla con las versiones de Eclipse.

Sistemas con componentes distribuidos

La computación desde sus inicios ha sufrido muchos cambios, desde los grandes ordenadores que permitían realizar tareas en forma limitada y de uso un tanto exclusivo de organizaciones muy selectas, hasta los actuales ordenadores ya sean personales o portátiles que tienen las mismas e incluso mayores capacidades que los primeros y que están cada vez más introducidos en el quehacer cotidiano de una persona.

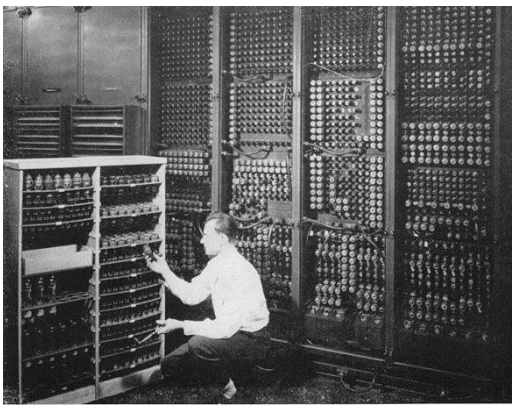


Figura 11 Uno de los primeros ordenadores.

Los mayores cambios se atribuyen principalmente a dos causas, que se dieron desde las décadas de los setenta:

1. El desarrollo de los microprocesadores, que permitieron reducir en tamaño costo a los ordenadores y aumentar en gran medida las capacidades de los mismos y su acceso a más personas.
2. El desarrollo de las redes de área local y de las comunicaciones que permitieron conectar ordenadores con posibilidad de transferencia de datos a alta velocidad.

Es en este contexto que aparece concepto de “Sistemas Distribuidos" que se ha popularizado tanto en la actualidad y que tiene como ámbito de estudio las redes como por ejemplo: Internet, redes de teléfonos móviles, redes corporativas, redes de empresas, etc.

Los sistemas distribuidos suponen un paso más en la evolución de los sistemas informáticos. Deben ser entendidos desde el punto de vista de las necesidades que las aplicaciones plantean y las posibilidades que la tecnología ofrece. Un sistema distribuido se define como un conjunto de computadores interconectados, que comparten un estado y ofrecen una visión general del sistema única.

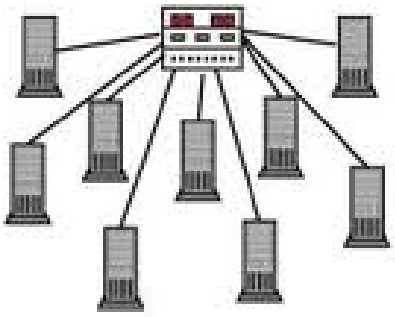


Figura 12 Imagen simbólica de un sistema distribuido.

Los recursos de las diferentes máquinas en red que componen del sistema, se integran de forma que se tiene la sensación de trabajar con sistema completo y disponible en todos los sentidos. El usuario accede a los recursos del sistema distribuido a través de una interfaz gráfica de usuario desde un terminal, despreocupándose de su localización.

Las aplicaciones ejecutan una interfaz de llamadas al sistema como si de un sistema centralizado se tratase. Un servicio de invocación remota resuelve los accesos a los recursos no locales utilizando para ello la interfaz de red. Los sistemas distribuidos proporcionan de forma transparente el proceso de compartir los recursos, facilitando el acceso y la gestión, e incrementando la eficiencia y la disponibilidad del sistema.

Los sistemas distribuidos de componentes muestran los recursos de forma homogénea, ocultando la distribución interna. Es más, el usuario y las aplicaciones propias del sistema no ven una red, sino un sistema que en realidad es totalmente indistinguible de uno centralizado. Cabe destacar que un sistema distribuido no es un sistema en red, ya que estos últimos pueden definirse como un conjunto de sistemas con estados independientes. Un sistema distribuido se define como un sistema con un estado global.

En realidad, la topología y los atributos físicos de la red se encuentran ocultos por los protocolos de red, mientras que la arquitectura de cada máquina está oculta por el sistema operativo que implemente. Además, los componentes de un sistema distribuido pueden ser heterogéneos, es por ello que se requiere de una capa de software, comúnmente denominada middleware, que permite la visión de un sistema único.

El método que utiliza ROS es el clustering. Un cluster puede definirse como un grupo de computadores conectados entre sí a través de un sistema en red y un software que realiza la distribución de la carga de trabajo entre los equipos. Por lo general, este tipo de sistemas cuentan con un centro de almacenamiento de datos único. En el caso de ROS, el nodo máster.

Un sistema distribuido que pretenda ofrecer una visión de sistema único deberá cumplir las propiedades que se presentan a continuación.

Transparencia: El objetivo principal de un sistema distribuido es proporcionar al usuario y a las aplicaciones del sistema una visión de los recursos que parezcan estar gestionados por una sola máquina. La distribución física de los recursos es transparente. Pueden describirse diferentes aspectos de la transparencia:

- Identificación: Los espacios de nombres de los recursos son independientes de la topología de la red y de la propia distribución de los recursos. De esta forma, una aplicación puede referirse a un recurso con un nombre independientemente de en qué nodo se ejecute.
- Ubicación: La ubicación física de los recursos es desconocida tanto para las aplicaciones como para los usuarios. Esto implica también que los recursos pueden migrar entre nodos sin que las aplicaciones se vean afectadas.
- Replicación: No es posible para el usuario ni las aplicaciones conocer cuántas unidades de cada recurso existen.
- Paralelismo: Una aplicación puede ejecutarse en paralelo sin tener que especificarlo, y sin influir como consecuencia sobre la ejecución de la misma. Sobre todo, esta propiedad tiene que ver con la capacidad del sistema de permitir distribuir procesos y memoria.
- Compartición: El que un recurso compartido intente ser accedido simultáneamente desde varias aplicaciones no tiene efectos sobre la ejecución de la aplicación.

- Rendimiento: Es inevitable que al implementar las propiedades de un sistema distribuido se produzca una pérdida del rendimiento del sistema. Es necesario siempre encontrar un compromiso entre las soluciones que este tipo de sistemas aportan, y la capacidad de respuesta unida al rendimiento que se necesite por parte del sistema.

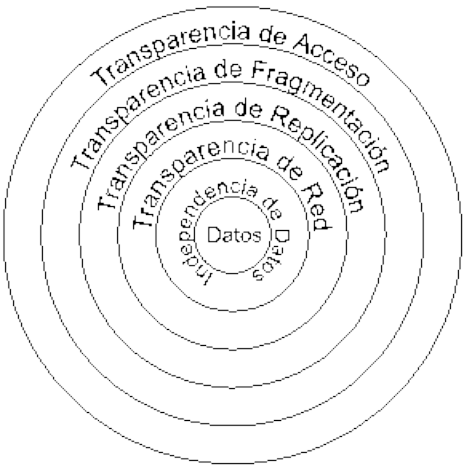


Figura 13 Características de transparencia en un sistema distribuido.

Escalabilidad: Una de las características de los sistemas distribuidos es su modularidad. Esto permite una gran flexibilidad y posibilita la escalabilidad del sistema, definida como la capacidad para crecer sin aumentar su complejidad ni disminuir su rendimiento. Uno de los objetivos del diseño de un sistema distribuido es extender la escalabilidad a la integración de servicios. La escalabilidad presenta dos aspectos.

- El sistema distribuido debe proporcionar espacios de nombres suficientemente amplios, de forma que no supongan una limitación inherente.
- El sistema debe mantener un buen nivel de rendimiento en el acceso a los recursos cuando el sistema crece.

Fiabilidad y tolerancia a fallos: La fiabilidad de un sistema puede definirse como la capacidad de realizar correctamente y en todo momento las funciones para las que se ha diseñado. Se han de tener en cuenta dos aspectos:

- Disponibilidad: Es la fracción de tiempo en la que el sistema está operativo. La disponibilidad se puede incrementar utilizando componentes de mayor calidad, o diseñar el sistema según el criterio de replicación de componentes que permita seguir operando al sistema aún cuando algunos de ellos fallen.
- Tolerancia a fallos: Aún con una alta disponibilidad, un fallo en un momento determinado puede acarrear consecuencias desastrosas. La tolerancia a fallos expresa la capacidad del sistema para seguir operando correctamente ante el fallo de alguno de sus componentes, enmascarando el fallo al usuario o a la aplicación. Por lo tanto, la tolerancia a fallos implica detectar el fallo y continuar con el servicio.

Consistencia: El problema de mayor complejidad es el de la gestión del estado global para evitar situaciones de inconsistencia entre los componentes del sistema. El problema radica en la necesidad de mantener un estado global consistente en un sistema con varios componentes, cada uno de los cuales posee su propio estado local. Los nodos del sistema se hallan físicamente distribuidos, por lo que la gestión del estado global depende fuertemente de los mecanismos de comunicación, a su vez soportados por una red sujeta a fallos.

2.2. ROS

Robot Operating System (ROS) es una estructura software y un conjunto de librerías implementadas para el desarrollo de software para robots, suministrando un sistema semi-operativo para robots. ROS fue desarrollado originalmente en 2007 bajo el nombre switchyard por el laboratorio de inteligencia artificial de Stanford y apoyado por la AI de robots de la mima Universidad.

A partir de 2008, el desarrollo continúa principalmente en Willow Garage, un instituto de investigación de la robótica / incubadora, con más de veinte instituciones que colaboran en el desarrollo de un modelo federado.

Figura 14 Logo de ROS.

ROS ofrece servicios estándar del sistema operativo como la abstracción de hardware de bajo nivel del control del dispositivo, la implementación de la funcionalidad de uso común, de paso de mensajes entre procesos y gestión de paquetes. Se basa en una arquitectura de teoría de grafos donde el procesamiento se lleva a cabo en los nodos que pueden recibir, enviar múltiples sensores, control del estado, la planificación, el actuador y otros mensajes. La biblioteca está dirigida a un sistema operativo tipo Unix, Ubuntu y Linux , mientras que otras variantes como Fedora y Mac OS X son considerados "experimentales" y no aseguran su correcto funcionamiento.

ROS es liberado bajo los términos de la licencia BSD, y es un software de código abierto. Es gratuito para uso comercial y de investigación. Los paquetes ros-pkg están licenciados bajo una variedad de licencias de código abierto.

Algunas aéreas en las que ROS está incluido son:

- Un nodo maestro coordinación
- Publicar o suscribirse a los flujos de datos: imágenes, música, actuadores láser, control, contacto
- Multiplexación de información
- Nodos de creación y destrucción
- Los nodos están perfectamente distribuidos, permitiendo la operación distribuida a través de multi-cores y multi-procesadores, la GPU y grupos
- Iniciar sesiones
- Parámetro servidor
- Sistemas de prueba
- Sistemas de percepción
- Sistemas de identificación de objeto
- Sistemas en movimiento
- Sistemas de planificación
- Sistemas de Agarrando

La tabla con las versiones de ROS es:

Versión	**Fecha de lanzamiento**
Fuerte	23 de Abril de 2012
Electric Emys	30 de Agosto de 2011
Diamondback	2 de Marzo de 2011
C Turtle	3 de Agosto de 2010
Box Turtle	1 de Marzo de 2010
Ros 1.0	22 de Enero de 2010

Figura 15 Tabla con las versiones de ROS.

2.3. Model-Driven Engineering

Model-Driven Engineering, Ingeniería Orientada a Modelos, es una metodología de desarrollo de software que se centra en la creación de modelos, o abstracciones, más cerca de algunos conceptos de dominio particular en lugar de la informática (o algorítmica) conceptos. Tiene el propósito de aumentar la productividad mediante la maximización de la compatibilidad entre sistemas, simplificando el proceso de diseño, y promover la comunicación entre los individuos y equipos que trabajan en el sistema.

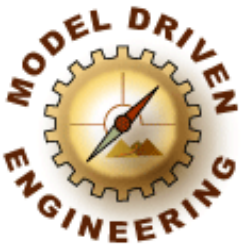


Figura 16 Model-Driven Engineering.

Un paradigma de modelado para el MDE se considera eficaz si los modelos tienen sentido desde el punto de vista del usuario y puede servir como base para la implementación de sistemas. Los modelos son desarrollados a través de una amplia comunicación entre los desarrolladores del producto, diseñadores y miembros del equipo de desarrollo. Como conclusión enfoque de los modelos, que permitan el desarrollo de software y sistemas.

La iniciativa MDE más conocido es el Object Management Group (OMG). Estas siglas que en español significan Grupo de Gestión de Objetos, son un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA. Es una organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas. El grupo está formado por diversas compañías y organizaciones con distintos privilegios dentro de la misma.

Figura 17 Logo de OMG

Una de las plataformas para el desarrollo de estos modelos es Eclipse. Este contiene una serie de plug-ins de modelado (Eclipse Modeling Project) que permite al usuario crear modelos y generar código java de ellos. Además ofrece tecnología de modelos persistentes (XML) y permite consultar, validar y visualizar los modelos transformándolos.

Figura 18 Proyecto MDE con Eclipse.

Model Driven Development Integration

Esta propuesta de proyecto Eclipse se registró para declarar la intención y el alcance de un proyecto de tecnología PMC llamado Model Driven de Integración para el Desarrollo (MDDI). Además, esta propuesta está escrita para solicitar la participación adicional y el aporte de la comunidad Eclipse.

Model Driven Development (MDD) está ganando cada vez más la atención de la industria y las comunidades de investigación. MDD hace hincapié en usar modelos en el ciclo de vida de desarrollo de software y argumenta su automatización a través de la ejecución de un modelo y la transformación de modelos con las técnicas de generación de código. El OMG está promoviendo un enfoque basado en modelos para el desarrollo de software a través de su iniciativa Model Driven Architecture (MDA) y sus normas de apoyo, tales como UML, MOF y QVT. El desarrollo basado en modelos es la aplicación específica de MDA para el desarrollo de software.

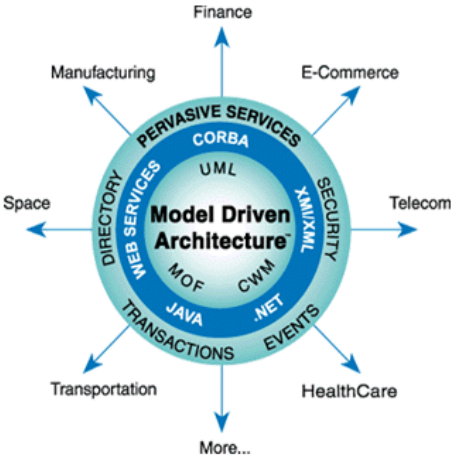


Figura 19 Propósito de Model Driven Architecture de OMG.

El proyecto Eclipse MDDI está dedicado a la realización de una plataforma que ofrece los plug-ins de integración necesarios para la aplicación de un enfoque MDD. Este proyecto producirá marcos extensibles y herramientas de ejemplo, diseñados para soportar varios lenguajes de modelado (UML y lenguajes específicos de dominio) y diferentes metodologías. La plataforma MDDI proporcionará la capacidad de integrar las herramientas de modelado, así como herramientas de apoyo a otros espacios tecnológicos, para crear un entorno MDD totalmente personalizable.

El alcance del proyecto de la plataforma Eclipse, es una base excelente para la creación e integración de herramientas de desarrollo. El proyecto tiene como objetivo ampliar MDDI para permitir un mayor nivel de integración de Model Driven Development. Se pondrá a disposición una plataforma extensible MDD, sobre la base de marcos dedicados a la integración de las tecnologías de modelado y sus herramientas de apoyo. La plataforma proporcionará una base que puede ser aprovechada y ampliada para el desarrollo de entornos de modelado futuros, ya sean de código abierto o comercial.

El proyecto MDDI asegurará la integración de servicios diseñados para la plataforma, así como herramientas externas. Otros espacios tecnológicos como XML o transformación gráfica pueden dar solución a los problemas MDD. Por ejemplo, el espacio de las tecnologías formales ofrece una serie de buenas prácticas y herramientas para validar los modelos. El problema es que estos espacios tecnológicos confían en las herramientas de soporte heterogéneo que en general tienen que ser de forma manual o mediante programación relacionada con puentes específicos. El proyecto MDDI abordará esta cuestión en el contexto de MDD.

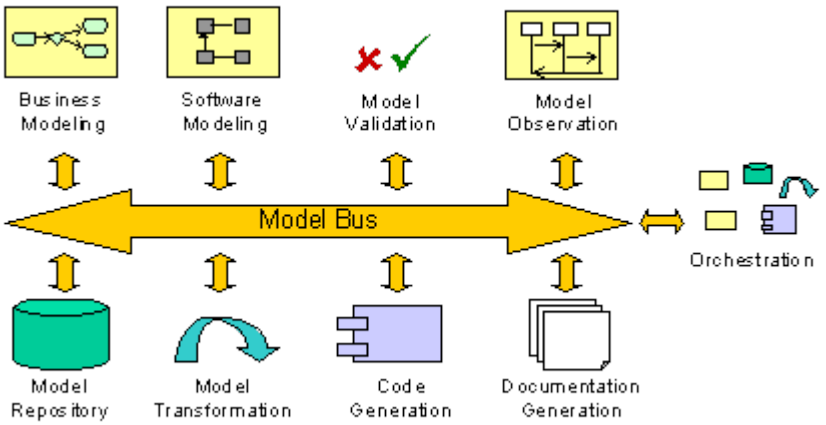


Figura 20 Ejemplo de un entorno basado en TDM.

La integración de las tecnologías de modelado no es trivial, aunque los lenguajes estándar y formatos de intercambio (como XMI) ya existen. Como se ilustra en la **Figura 2.x,** se tiene que lidiar con lenguajes diferentes (estándar o de propiedad, de uso general o de dominio específico), así como con diferentes herramientas y sus limitaciones.

En ese contexto, tres diferentes niveles de integración pueden ser considerados:

Un nivel de **integración tecnológica** asegura la interoperabilidad de las herramientas que proporcionan servicios orientados modelo, también proporcionan capacidades de conectividad a otros terceros herramientas de modelado, a condición de que se puede adaptar para ser utilizado por los meta-modelos. Además proporciona a los instrumentos de apoyo un modelo de ejecución común que sean compatibles.

Un nivel de **integración semántica** asegura la descripción completa de las lenguas, así como los mecanismos de intercambio para facilitar la comparación y traducción de esta semántica entre las herramientas. Explícitamente mapear la semántica utilizada por herramientas es un requisito para apoyar modelos totalmente ejecutables.

Un nivel de **integración metodológica** proporciona capacidades de extensión y personalización para soportar un número sin consolidar de las prácticas de desarrollo basadas en lenguajes de modelado y metodologías basadas en modelos.

El objetivo final es permitir a los expertos en metodología para el diseño de lenguajes de modelado y procesos de desarrollo, y luego a integrar las herramientas existentes (o crear otros nuevos) para proporcionar un entorno dedicado de soporte. Los desarrolladores de aplicaciones pueden entonces utilizar la plataforma a medida de acuerdo a las especificaciones de los expertos para realizar tareas de modelado. La plataforma desarrollada dentro MDDI será por lo tanto un entorno integrado con capacidades de modelado y meta-modelado, donde los principios MDD aplican en ambos niveles.

Capítulo 3

3. Entorno de desarrollo y herramientas utilizadas

En esta sección se van a presentar las herramientas que se han utilizado para la realización de este proyecto y los motivos sobre el porqué de su elección.

3.1 Sistema Operativo

El sistema operativo que se ha utilizado para el desarrollo de este proyecto ha sido **Ubuntu en su versión 10.04.** Ubuntu es un sistema operativo mantenido por Canonical y la comunidad de desarrolladores. Utiliza un núcleo Linux y su origen está basado en Debian GNU/Linux. Ubuntu está orientado al usuario novel y promedio, con un fuerte enfoque en la facilidad de uso y mejorar la experiencia de usuario. Está compuesto de múltiple software normalmente distribuido bajo una licencia de software libre o de código abierto.

Figura 21 Logo de Ubuntu

Algunas de las características técnicas de Ubuntu 10.04, comparado con sus versiones anteriores, son:

- Utiliza un Kernel de Linux en su versión 2.6.32.11.
- Utiliza GNOME 2.30.
- En Kubuntu KDE SC 4.4.2.
- GIMP 2.6.8 sólo disponible desde los repositorios.
- Posee un nuevo artwork.
- Renovado el slideshow del instalador Ubiquity.
- Ha eliminado el HAL para hacer más rápido el inicio del sistema y regresar después de suspendido.
- Drivers open source para Nvida ahora se incorporan por defecto.
- Nuevas características para Ubuntu Enterprise Cloud (UEC).
- Mejorada versión de likewise-open, paquete que provee autenticación para Active Directory y soporte de servidor para Linux.
- Ubuntu One mejora la sincronización.

Explicando un poco más las características anteriormente enumeradas se puede decir que el motor de todo sistema es el kernel. Esta actualización Kernel 2.6.32 incluye mejoras "que no se ven" pero que hacen de Ubuntu 10.04 un sistema más fiable, seguro y robusto.

El Gnome, en su versión Gnome 2.30., es el entorno de escritorio por defecto en Ubuntu. Pequeños añadidos en Nautilus, Empathy, la gestión de usuarios, soporte para iPod, etc.

El nuevo driver libre para tarjetas Nvidia es Nouveau. Al mismo tiempo se ha mejorado la integración de los drivers propietarios de Nvidia.

Mejoras en el Centro de software de Ubuntu. Muy importante, sobre todo para usuarios nuevos, facilitando la instalación de aplicaciones en Ubuntu 10.04

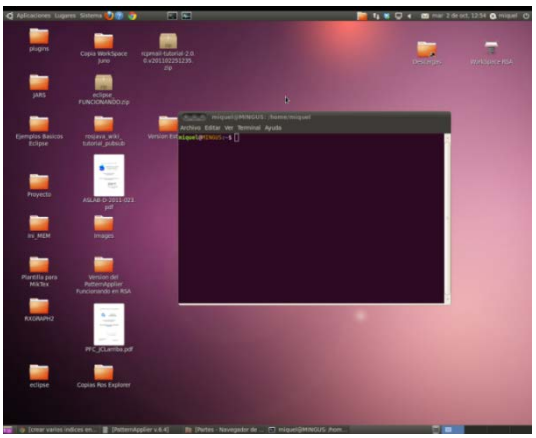


Figura 22 Ejemplo de Escritorio con S.O. Ubuntu.

Una vez enumeradas sus características técnicas se expondrán los motivos del de su elección. Ubuntu como se ha dicho es un sistema operativo de software libre, por lo que no necesita licencia al ser gratuito. Permite una gran versatilidad en el entorno y el sistema si el usuario posee los conocimientos necesarios.

Además el sistema en esta versión es el que mejor trabaja con las versiones de ROS, concretamente en su versión Fuerte. Otros sistemas operativos como Windows y Mac son incompatibles con las versiones de ROS con las que se va a trabajar ya que los responsables del mantenimiento de dichas librerías solo aseguran su correcto funcionamiento con un sistema operativo Ubuntu.

El último motivo de su elección ha sido que la gran mayoría del grupo de investigación trabaja con Ubuntu. Para evitar incompatibilidades, ya que el objetivo del proyecto es crear un nuevo marco ICe que siente las bases para el grupo de trabajo, se ha decidido seguir trabajando con el sistema que trabaja la mayoría.

3.2 ROS y ROS Java

Como ya se ha dicho en el capítulo 2, ROS (Robot Operating System) es un marco de trabajo de software que permite el desarrollo de software para aplicaciones robóticas. La base de ROS consiste en proporcionar unas librerías con la funcionalidad de un clúster informático heterogéneo.

ROS tiene dos tipos básicos que lo componen, uno el Sistema Operativo tal y como se ha descrito hasta ahora, y otro referido a la gestión de paquetes desarrollados por los usuarios. Estos paquetes se organizan en carpetas denominadas stacks. Los paquetes son los encargados de ofrecer la funcionalidad de ROS propiamente dicha, por ejemplo funciones de localización, mapeo, planificación de eventos, escaneo, simulación, etc.

Figura 23 Logo de la página Web de ROS.

Estas librerías que ofrece ROS han sido utilizadas en este proyecto ya que uno de sus objetivos es construir las vistas necesarias para trabajar con ellas de manera más amigable y a más alto nivel, evitando así controlar el sistema ROS mediante comandos por consola. Concretamente se ha trabajado con las librerías de ROS Fuerte y ROS Java.

ROS FUERTE

ROS Fuerte Turtle es la quinta versión de la distribución de ROS. Se lanzó el día 23de Abril de 2012 e incluye numerosas mejoras que permiten y facilitan la integración de ROS con otros software. Esta mejora incluye una nueva versión del sistema de compilación, el cambio al marco de trabajo Qt y a las librerías estándares.

El objetivo de ROS es hacer el código lo más reutilizable posible dentro del ámbito de la robótica, favoreciendo el desarrollo de la comunidad de usuarios y ampliando el abanico de oportunidades de implementación del sistema ROS. Según fuentes de ROS, esta versión es una nueva base sólida para la generación de grandes bibliotecas dentro del campo de la robótica.

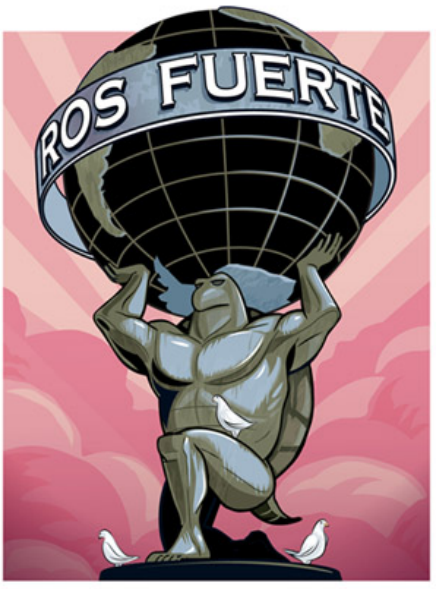


Figura 24 Logo de la ROS Fuerte.

ROS Fuerte Turtle está dirigido principalmente a los sistemas operativos Ubuntu Lucid, Oneiric y Precise, aunque también se puede instalar en varios sistemas Linux como Red Hat, Debian y Gentoo. Además se puede utilizar en otros sistemas operativos como Mac OS X, FreeBSD, Android, y Windows, aunque con una compatibilidad limitada.

Las principales mejores y diferencias con las bibliotecas de las versiones anteriores son:

- Fuerte se aleja del bajo nivel

Las bibliotecas de bajo nivel ROS (ros, ros_comm, actionlib) ahora pueden ser fácilmente compiladas en marcos no basados en ROS. Este nuevo sistema CMake basado en un sistema de construcción, llamado catkin, hace que sea fácil de integrar estas bibliotecas usando CMake find_package como un estándar y las herramientas de pkg-config. El método anterior de compilación del sistema rosbuild sigue estando disponible.

- Nuevo diseño de sistema de archivos

Las bibliotecas de bajo nivel de ROS ahora se pueden instalar en una librería estándar. Esta migración hacia un diseño estándar de jerarquía del sistema de ficheros permite una integración más fácil de ROS con otras herramientas.

- Bibliotecas standard de mensajes ROS

Las estructuras de mensajes de ROS, como los common_msgs, std_msgs y los paquetes rosgraph_msgs ahora pueden ser fácilmente utilizadas por marcos no ROS con mínimas dependencias con el sistema. Esto aumenta la portabilidad de código entre diferentes marcos de software con robots.

- Qt basado en RViz

RViz ha sido reescrito para utilizar el marco de Qt, lo que mejora la integración con plataformas como MAC OS X. La nueva RViz también cuenta con numerosas herramientas y mejoras en la compatibilidad.

- Gazebo 1.0

Gazebo 1.0 es una nueva versión y ahora es la principal en la biblioteca de simulación de software. Se incluye una nueva arquitectura que aumenta significativamente su rendimiento.

- PCL 1.5

PCL 1.5 cuenta con una biblioteca de seguimiento nueva, mejoras de rendimiento y otras mejoras importantes.

- Rosdep 2

Rosdep ha sido reescrito y ahora es una herramienta externa (que hay que instalar por separado). Esto permite a rosdep actualizarse con mayor frecuencia y mejorar la compatibilidad con otras plataformas. El nuevo rosdep también utiliza una base de datos centralizada, lo que hace que sea más fácil presentar las nuevas reglas para la inclusión.

- Nuevo paquete rospkg

La nueva biblioteca independiente rospkg Python que proporciona una API estable para la manipulación del paquete de ROS.

- Separación de la rosemacs

Rosemacs está ahora separada de manera que se puede actualizar más fácilmente.

ROS Java

También ha sido necesario instalar algunas librerías adicionales de ROS, concretamente las de ROS java. Cabe recordar que el desarrollo de nuevos elementos del sistema ROS es posible gracias a que éste utiliza código abierto. Los usuarios y desarrolladores de ROS tienen la oportunidad de crear nuevos componentes, mejorar los ya existentes o simplemente ayudar a que el sistema sea más robusto y flexible. Esto ayuda a que el sistema sea robusto, flexible y heterogéneo cuando se utilice en cualquier plataforma. ROS tiene una ventaja relevante y es que es capaz de implementar componentes desarrollados en códigos de programación distintos.

Esto hace que el sistema sea adaptativo en función del tipo de nodo que se requiera y sobre todo hace que el sistema sea capaz de funcionar correctamente en gran variedad de máquinas. Esta característica ha resultado ser de gran importancia en el desarrollo de este proyecto, ya que Eclipse trabaja fundamentalmente con código Java.

.

Figura 25 Logo de Java.

3.3 Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llamaba RCP (Aplicaciones de Cliente Enriquecido), opuesto al las aplicaciones cliente-liviano basadas en navegadores. Esta plataforma típicamente ha sido usada para desarrollar entornos de desarrollo integrado, en ingles IDE, como el IDE de Java llamado Java Development ToolKit (JDT) y el compilador que se entrega como parte de Eclipse, que también son usados para desarrollar el mismo Eclipse.

Eclipse es un programa que se puede descargar de manera gratuita de la propia página de Eclipse (www.eclipse.org). Posee varias versiones iníciales en función de a lo que se vaya a dedicar el desarrollador, pero la única diferencia entre ellas es los plug-ins iniciales que lo configuran por defecto. Esto no supone problema alguno ya que Eclipse posee una estructura modular, y es por esta modularidad por la que se pueden ir añadiendo módulos (plug-ins) para aumentar su funcionalidad.

Concretamente este proyecto ha sido desarrollado sobre Eclipse Juno en su versión Classic 4.2 para Linux.

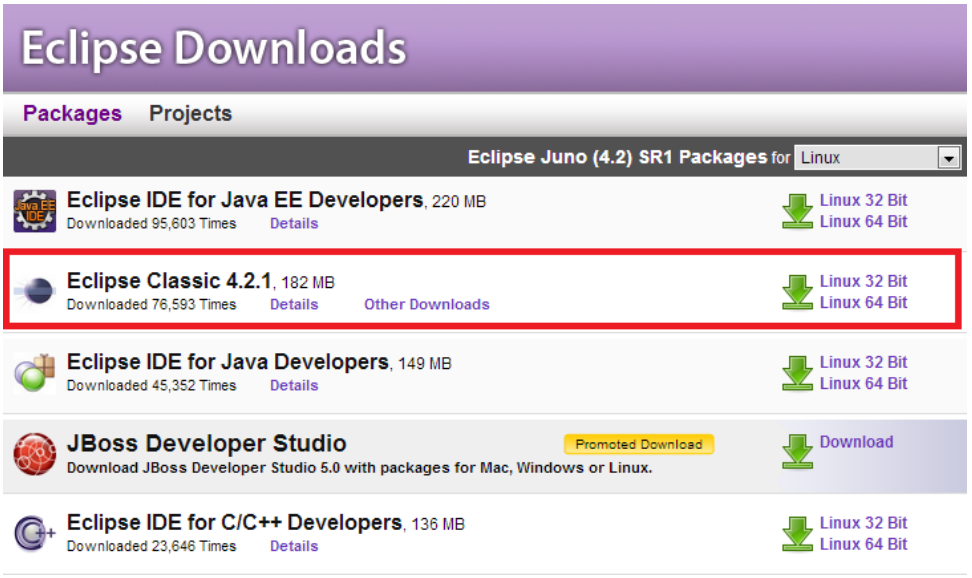


Figura 26 Repositorio de eclipse con Juno 4.2

Se ha decidido trabajar con la última versión de Eclipse estable para ir acorde con las actualizaciones de mercado, ya que en sus nuevas versiones posee más funcionalidades y algunos paquetes solo son válidos para estas.

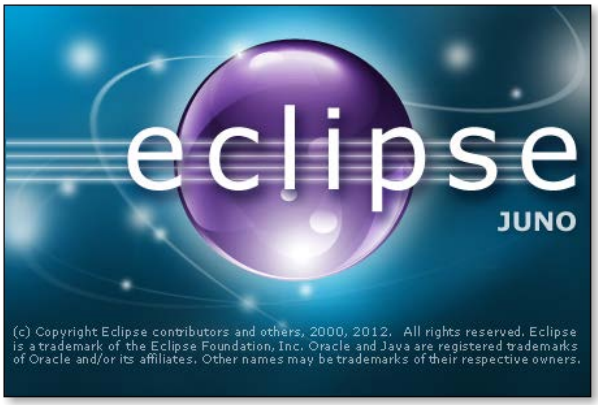


Figura 27 Logo de Eclipse Juno.

3.3.1 Entorno RCP de Eclipse

RCP se corresponde con las siglas de Rich Client Platform. Esto es una herramienta de programación que hace más sencillo la integración de capacidades independientes mediante componentes software. Muchas veces construir una aplicación software requiere de complejas integraciones de muchos componentes. Un RCP es una herramienta que hace mucho más sencillo integrar componentes software independientes.

La plataforma RCP de Eclipse permite la integración sin fisuras de módulos software de diferentes aplicaciones software. Típicamente estas aplicaciones incluyen herramientas graficas avanzadas, herramientas geo-espaciales, tecnología de mapeo...etc.

Usando un RCP, el desarrollador puede integrar componentes independientes con un simple click de ratón.

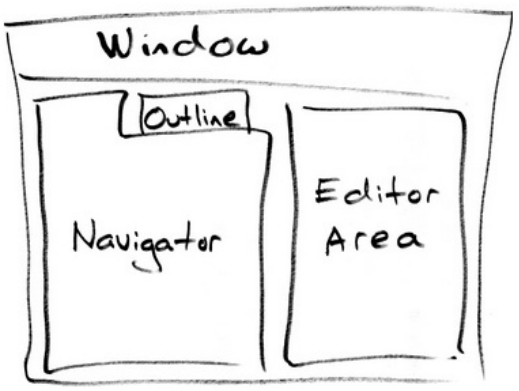


Figura 28 Dibujo esquemático de lo que es un RCP.

La aplicación de programación de eclipse es uno de los mejores ejemplos de integración completa para RCP. Las librerías que incluye Eclipse, permiten al programador controlar cientos de componentes Java, que de otra manera serian incontrolables. Algunos ejemplos de RCP están incluidos en Eclipse IDE y Java Spring.

Para ilustrar con un ejemplo como se crearía fácilmente un RCP sencillo se ha decidido generar un proyecto plugin de Eclipse. En su fase de creación este pregunta al usuario si esta aplicación va a ser pequeña, o si en cambio va a ser un RCP.

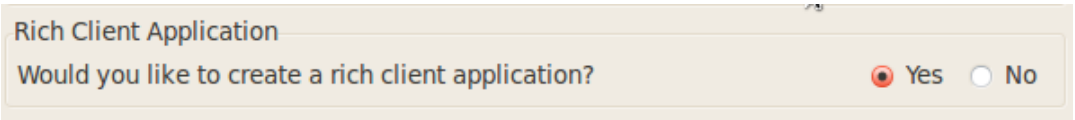


Figura 29 Pregunta sobre crear un RCP de Eclipse.

Si el cliente decidiera que quiere realizar un RCP Eclipse presenta cuatro plantillas por defecto, generando ya una estructura definida y el código correspondiente, ahorrando al programador todo este código.

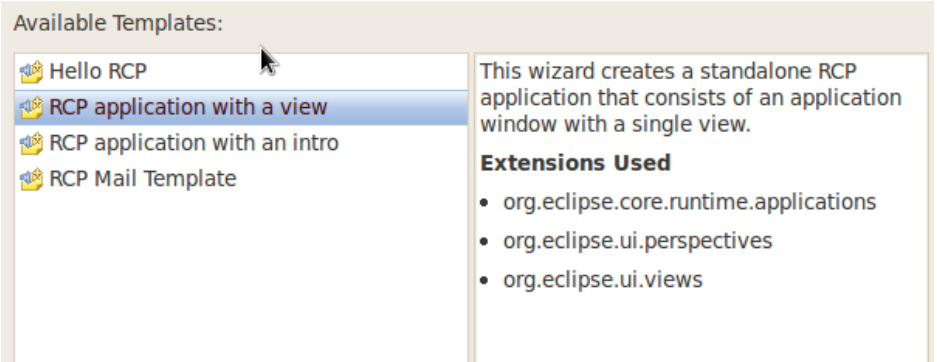


Figura 30 Plantillas RCP de Eclipse.

Internamente el conjunto de librerías que se utilizan para realizar un RCP están definidas en Eclipse. Véase la figura 3.11.

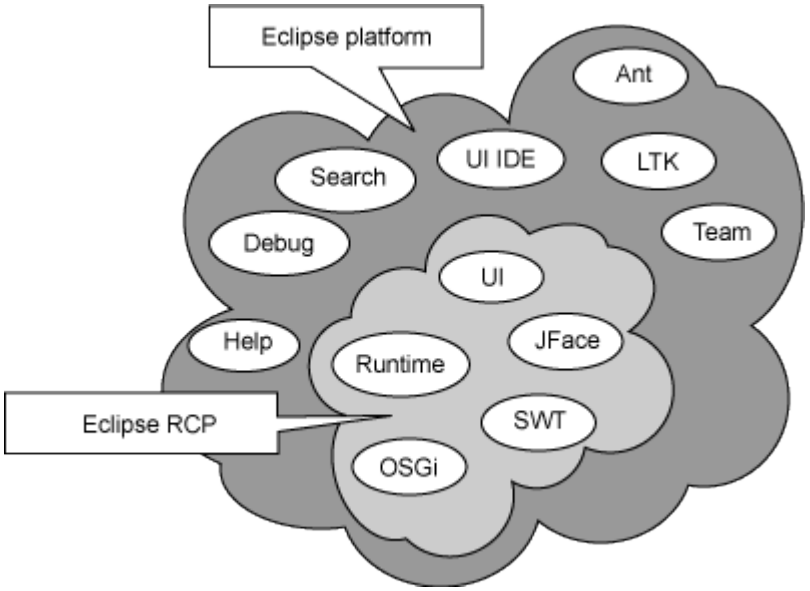


Figura 31 RCP dentro de Eclipse.

3.3.2 Entorno Java de Eclipse

Eclipse está fuertemente basado en lenguaje de programación Java, véase figura 3.12. Contiene lo que ellos llaman **"**The Java Development Environment" (JDE), es decir, el entorno de desarrollo java de la herramienta. Este entorno está basado en herramientas de desarrollo Java (JDT) que componen internamente este entorno y se encuentran en el paquete de Eclipse SDK.

Lenguajes de programación utilizados en Eclipse 3.2.1

Lenguaje	Líneas de código	%
Java	1.911.693	92,66%
ANSI C	133.263	6,46%
C++	10.082	0,49%
JSP	3.613	0,18%
sh	2.066	0,10%
perl	1.468	0,07%
php	896	0,04%
sed	2	0,00%

Figura 32 Lenguajes de programación en Eclipse 3.2.1.

Las herramientas de desarrollo Java (JDT) proporcionan los plugins que implementan un IDE de Java que soporta el desarrollo de cualquier aplicación Java, incluyendo plugins para el propio Eclipse. Añade una naturaleza de proyecto Java y una perspectiva Java al entorno de trabajo de Eclipse, así como varias vistas, editores, asistentes, constructores y herramientas de fusión de código y refactorización.

Además Eclipse contiene una perspectiva Java. Esta está diseñada para trabajar con proyectos Java. Contiene inicialmente un editor, y las vistas Package Explorer, Hierarchy, Outline, Problems, Javadoc y Declaration.

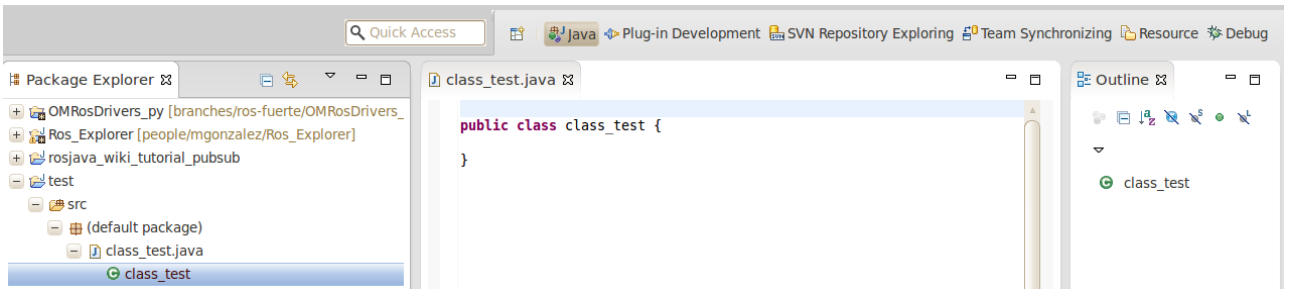


Figura 33 Perspectiva Java en Eclipse.

El proyecto JDT suministra un set de plug-ins que añaden capacidades y características completas Java IDE para la plataforma Eclipse. Los plug-ins que suministra la JDT se podrían categorizar en:

JDT APT

JDT APT añade soporte para el procesamiento de proyectos Eclipse con la notación Java 5 o superior. Ofrece las siguientes características:

Apoyo para los procesadores que funcionan siguiendo la anotación Sun para la línea de comandos.
Contribución de la anotación a base de construir artefactos durante la generación incremental
Contribución de los marcadores de problemas para la anotación de problemas

JDT Core

JDT Core define la estructura de lo que no es la interfaz de usuario. Esto incluye:

Un constructor Java incremental.
Un modelo de Java que proporciona la API para navegar por el árbol de elementos Java. El árbol de elementos Java define una visión centrada en Java del proyecto. Salen a la superficie elementos, fragmentos de paquetes, unidades de compilación, clases binarios, tipos, método etc.
Una infraestructura basada en búsqueda indexada que se utiliza para buscar. Un asistente de código, computación con la jerarquía de tipos, y refactorización.
Soporte de evaluación.

JDT Debug

JDT Debug implementa la compatibilidad de depuración Java y funciona con cualquier compilador JDPA con Java VM. Se lleva a cabo en la parte superior del lenguaje independiente "modelo de depuración" proporcionada por el depurador de la plataforma.

JDT Debug proporciona las características de depuración siguientes:

Puesta en marcha de una máquina virtual de Java, ya sea en modo de ejecución o depuración
Arranque y puesta en una marcha VM Java
Evaluación de expresiones en el contexto de un marco con pila
Páginas para evaluación fragmento de código Java interactivo
Recarga dinámica de clases cuando sea compatible con la máquina virtual Java

JDT Text

JDT Text proporciona las siguientes características:

Búsqueda por palabras claves
Contexto específico (Java, Javadoc) Asistencia para seleccionar el código
Método de nivel de edición
Anotaciones en el margen de los problemas, puntos de descanso, o partidos de búsqueda
Actualización para la edición se lleva a cabo en el Outliner
Ayuda API Javadoc para la especificación y muestra seleccionada del elemento Java en una ventana pop-up
Asistencia para importar automáticamente, crear y organizar las declaraciones de importación
Formato del código

JDT UI

JDT UI implementa la interfaz de usuarios específicos con las siguientes características:

Package Explorer
Type Hierarchy View
Java Outline View
Wizards para la creación de elementos Java

3.3.2.1 Plug-ins en Eclipse

Los complementos en los que se basan muchas herramientas informáticas son los plugins. Un plugin es una aplicación informática que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la interfaz de programación de aplicaciones, en este caso Eclipse.

Esta estructuración de los componentes en otros más sencillos permite que los desarrolladores externos colaboren con la aplicación principal extendiendo sus funciones. Además permite reducir el tamaño de la aplicación al poderse montar por "partes". También permite separar el código fuente de la aplicación para manejar la incompatibilidad de las licencias software.

Figura 34 Logo Eclipse plugins.

En el caso concreto de Eclipse, él mismo está desarrollado a base de plugins lo que le dota de gran versatilidad. Como ya se ha visto antes, los proveedores de Eclipse lo lanzan con diferentes configuraciones y en función de los plugins que contengan inicialmente, recibirá un nombre u otro, pero el usuario puede ampliar sus paquetes instalando nuevos componentes. Pero la característica que lo hace realmente interesante es que sobre este, el desarrollador puede crear sus propias herramientas.

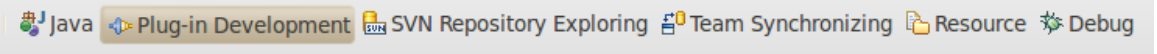


Figura 35 Perspectiva Eclipse de desarrollo de plugins.

Eclipse incorpora una perspectiva concreta para el desarrollo de plugins. En ella vienen por defecto el Package Explorer, la consola, la vista de errores, el outline.

Además Eclipse incorpora la posibilidad de crear un archivo plugin y desarrollarlo. Para ello habrá que hacer File->New-> Plug-in Project.

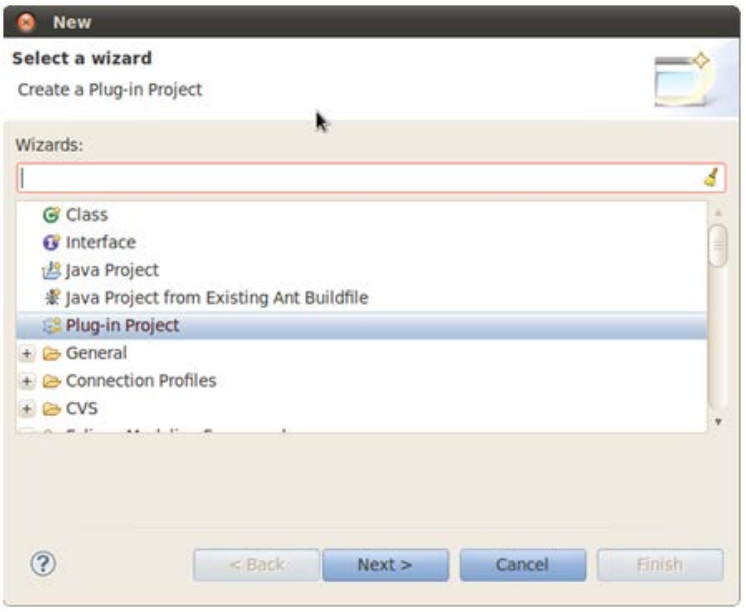


Figura 36 Menú inicial al crear un nuevo proyecto plug-in.

Una vez seleccionado habrá que darle un nombre al proyecto, definir las librerías y carpetas donde se archivarán los paquetes y clases del código y elegir con la versión de java con la que se va a trabajar.

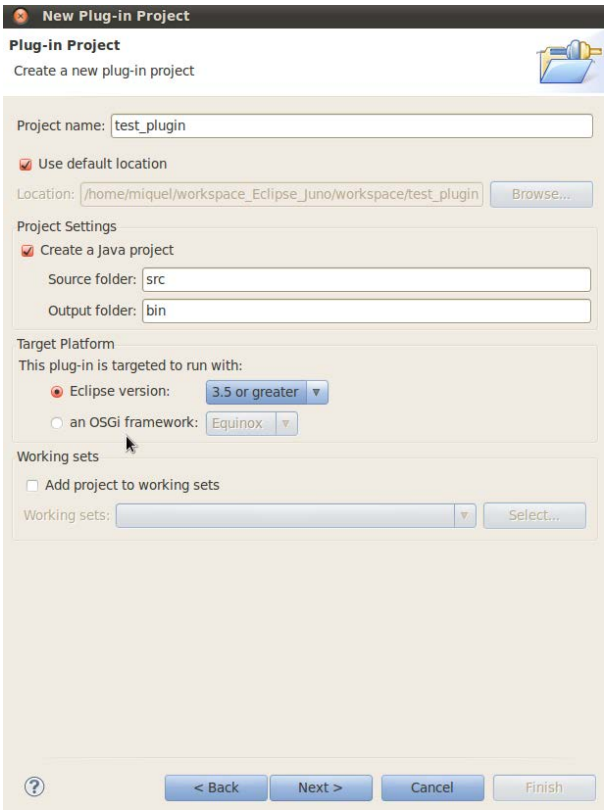


Figura 37 Segundo paso al crear un nuevo proyecto plug-in.

A continuación se le dará el nombre a la clase que activará el plug-in, se establecerá la identificación (ID) para que Eclipse pueda identificarlo y se establecerá su número de versión en el caso de que no sea la inicial.

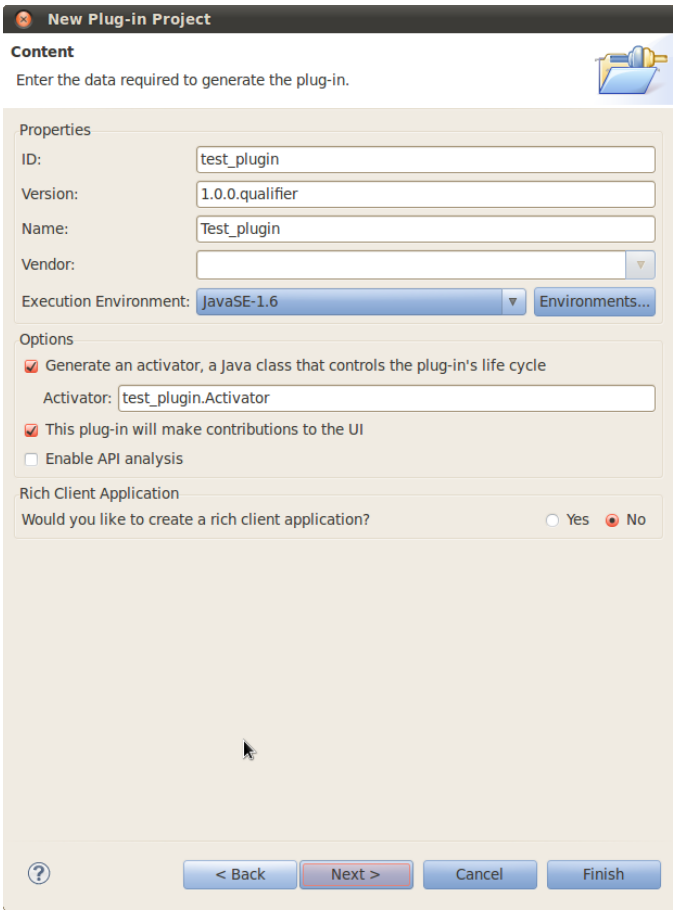


Figura 38 Paso 3 al crear un nuevo proyecto plug-in.

Eclipse por defecto incluye plantillas para crear proyectos plug-in que ya incorporan unos componentes predeterminados y generan el código necesario para ensamblar todo.

Figura 39 Proyectos Plug-in Standard ofrecidos por Eclipse.

En el caso de que se quiera construir un plug-in diferente, Eclipse ofrece la opción de crear un plug-in personalizado en la que el usuario puede definir con un solo “click” de ratón los componentes de Eclipse que desea incluir. El mismo programa sugiere muchas posiblidades en funcion de lo que el desarrollador quiera hacer e incluye un menú en el que comenta lo que contiene cada libreria.

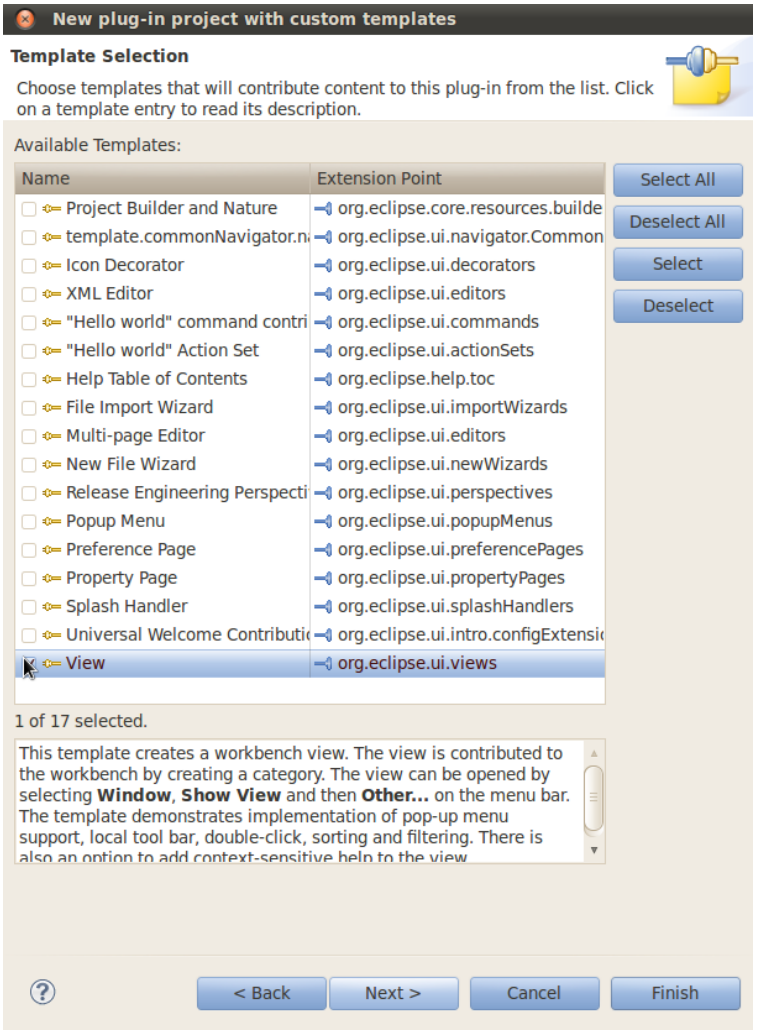


Figura 40 Librerías ofrecidas por Eclipse en un plug-in personalizado.

Por último habría que destacar que cada proyecto plug-in incluye un fichero llamado “plugin.xml”, escrito en XML, donde se establecen las librerías que incluye este, las extensiones que tiene. Este archivo es esencial para crear un proyecto plug-in ya que recoge un resumen de todos los componentes que se utilizan.

```
<plugin>
   <extension>

   </extension>

</plugin>
```

Figura 41 Código de un plug-in en XML.

3.3.3 Entorno gráfico de Eclipse

Eclipse, además de ser una herramienta para el desarrollo de código Java y proyectos plug-in como se ha visto en los puntos anteriores, también permite el desarrollo de aplicaciones gráficas. Esta funcionalidad se incorpora por medio de una herramienta para el desarrollo gráfico llamada GEF (Graphical Editing Framework).

GEF incorpora librerías estándar para la incorporación de tablas, gráficos y diagramas interactivos en un conjunto de herramientas standard (SWT). Esto complementa el marco de trabajo SWT, dando a los desarrolladores mucha más libertad sobre cómo construir su aplicación y como mostrar la información en ellas.

Figura 42 Logo de GEF.

El proyecto GEF de eclipse está compuesto por tres partes principales: Draw2D, Zest y GEF. La parte interna llamada GEF recibe el mismo nombre que el proyecto.

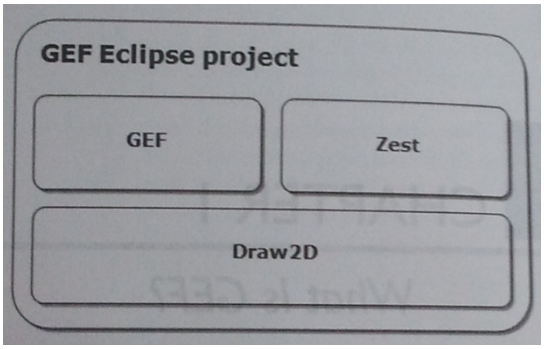


Figura 43 Overview del proyecto GEF de Eclipse.

Draw2D es una herramienta para dibujar y mostrar información grafica de manera sencilla, pero no permite un comportamiento interactivo. Zest está construido sobre Draw2D y proporciona una interfaz JFace-Like para que sea fácil unir a este modelo de Draw2D, un modelo de Java. GEF también está construido sobre Draw2D, proporcionando aplicaciones ricas y pesadas para producir diagramas avanzados con interacciones complejas.

3.3.3.1 Herramienta Zest de Eclipse

Zest es una capa construida sobre Draw2D para la adaptación de tu modelo al diagrama. No es una aplicación muy pesada y permite la interacción directa con los objetos construidos en el grafo. Además permite definir diversos parámetros como son la dirección de los conectores, nuevas formas para los objetos, diferentes colores y anchos…etc. Es por esto que se ha decidido utilizar esta herramienta para la representación de los nodos y tópicos de ROS.

Figura 44 Logo de ZEST.

La herramienta Zest posee los siguientes componentes fundamentales:

- GraphNode – Nodos en el graph con sus propiedades asociadas
- GraphConnections – Flechas o líneas del graph con las que se conectan dos nodos.
- GraphContainer – Uso del grafico dentro del graph
- Graph – Vista donde se pintan GraphNodes, GraphConnections y GraphContainer.

Zest Eclipse proporciona a los administradores de diseño gráfico un gestor de diseño gráfico y determina cómo los nodos (y las flechas) de un gráfico están dispuestos en la pantalla. Entre los administradores de diseño (layout managers) se encuentran las siguientes características:

Layout Manager	**Descripción**
TreeLayoutAlgorithm	El grafico es mostrado con la forma de un árbol vertical
HorizontalTreeLayoutAlgorithm	Igual que el TreeLayoutAlgorithm pero horizontal
RadialLayoutAlgorithm	La raíz es el centro, los otros nodos son situados alrededor de la raíz
GridLayoutAlgorithm	Presenta una estructura cuadrada
SpringLayourAlgorithm	Algoritmo que hace que todas las conexiones tengan aprox. el mismo tamaño
HorizonalShift	Mueve los nodos superpuestos a la derecha
CompositeLayoutAlgorithm	Combina varios algoritmos

3.3.4 Paquetes utilizados en Eclipse

Para aumentar la funcionalidad de Eclipse Classic se ha necesitado una serie de plug-ins para desarrollar entornos gráficos, vistas, clientes ricos y nuevas funcionalidades.

Para ello se va a realizar de la siguiente manera. Las demás versiones de Eclipse, excepto el classic, incluyen por defecto el “Eclipse Marketplace”, que es mercado en el que se recogen los plugins que se los usuarios van desarrollando y son aprobados por la pagina de Eclipse.

Para instalar la pestaña de Market se realizarán los siguientes pasos. Una vez arrancado Eclipse se pulsará la pestaña de Help->Install New Software.

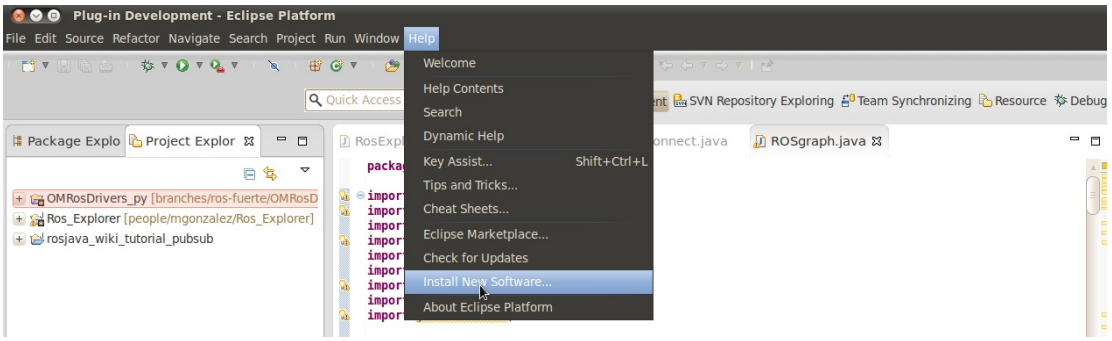


Figura 45 Paso 1 para Instalar el MarketPlace en Juno Classic.

Una vez arrancada la interfaz de Install se buscará el link que viene por defecto internamente para la gestión de actualizaciones de Juno (http://download.eclipse.org/releases/juno) y se seleccionará la Feature de General Purpose Tools.

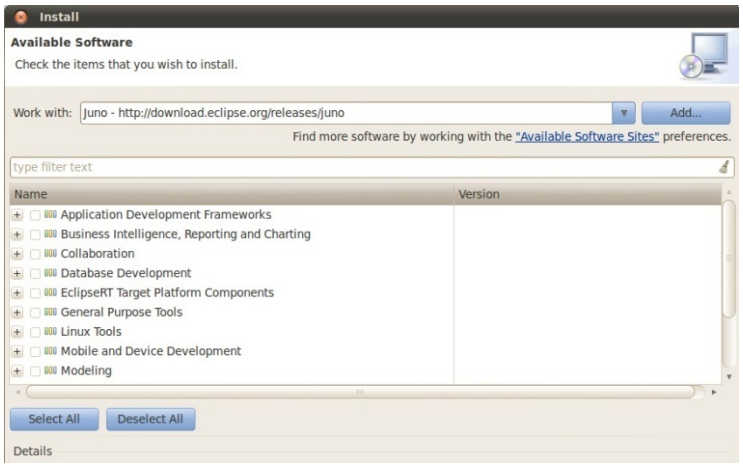


Figura 46 Paso 2 para Instalar el MarketPlace en Juno Classic.

Una vez desplegado el General Prupose Tools se mostrarán todos los contenidos del paquete, y se clickará sobre el MarketPlace Client.

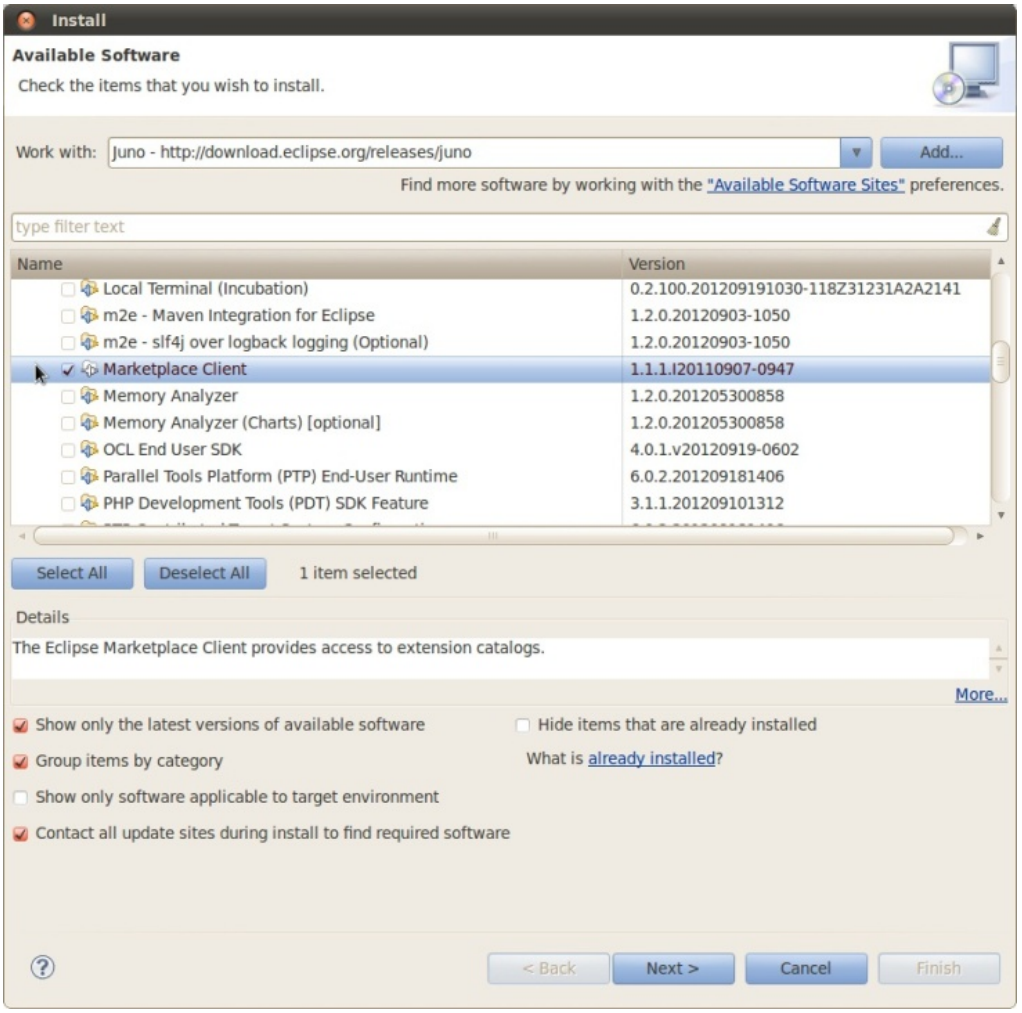


Figura 47 Paso 3 para Instalar el MarketPlace en Juno Classic.

Una vez seleccionado se presionará next, se aceptarán los términos y cuando esté instalado será necesario reiniciar Eclipse para que los cambios surjan efecto. Ya con el MarketPlace instalado se han utilizado una serie de paquetes para ampliar la funcionalidad de Eclipse. Se va a explicar con un ejemplo de cómo se podría instalar un plug-in, ya que con la herramienta MarketPlace es realmente intuitivo y sencillo.

El primer paquete que se ha de instalar es EMF, Eclipse Modeling Framework. Inicialmente se adaptó con esta versión de Eclipse un pluglet a su versión moderna de plugin. Para realizar esta transformación era necesaria esta librería ya que el pluglet contenía modelos UML. El proyecto EMF es un marco de modelado y herramienta para la generación de código para herramientas de construcción y otras aplicaciones basadas en modelos de datos.

Para instalar este paquete, una vez el eclipse esté arrancado, se pulsará Help->Eclipse MarketPlace.

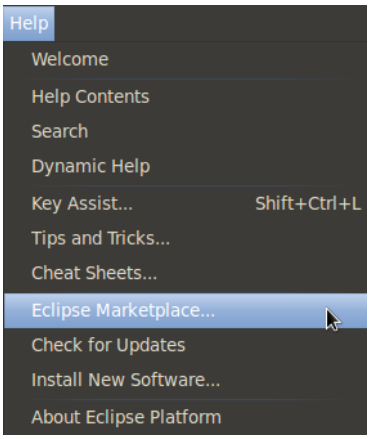


Figura 48 Paso 1 para la instalación de un plug-in del MarketPlace.

Una vez abierta la ventana del MarketPlace introducir en nombre del plugin a instalar, en este caso EMF, se buscará el elemento concreto y se pulsará install aceptando la licencia y los pasos necesarios. Una vez instalada, como siempre, se necesitará reiniciar Eclipse.

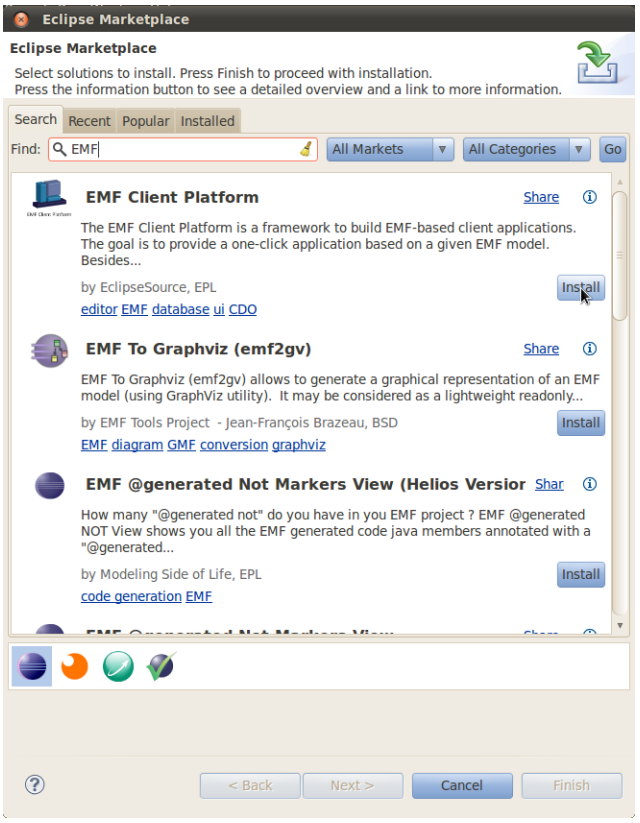


Figura 49 Paso 2 para la instalación de un plug-in del MarketPlace.

Todos los pasos para la instalación de nuevo software se realizarán así, con lo que no se explicará da uno en detalle. Los demás paquetes instalados para el desarrollo del proyecto son:

- Window Builder.
- EMF Transaction Update.
- RCP
- GEF
- Zest

Por último también se ha instalado un plugin adicional que se ha decidido explicar ya que es diferente al resto. Durante todo el proyecto se ha trabajado con un repositorio, montado en los servidores del grupo ASLab, para el almacenamiento de los proyectos. Este repositorio guarda una característica fundamental ya que permite un control de versiones de cada proyecto. No destruye las versiones ni las "pisa" sino que las va almacenando todas y el usuario puede recuperar versiones anteriores.

El grupo de investigación ASLab trabaja con un repositorio SVN, que se sincroniza desde el propio Eclipse. Para obtener esta herramienta existen dos posibles plugins. El Subversive y el Subclipse. Por dar menos problemas, por lo menos en mi caso, he decidido trabajar con el Subclipse. Esta herramienta está en el Market con lo que se instalará fácilmente siguiendo los pasos anteriores. Una vez reiniciado el Eclipse salta una ventana para configuración de JavaHL. Si no se tuviera las librerías instaladas habrá que incluirlas, por medio de esta ventana, para el correcto funcionamiento del SVN.

Capítulo 4

4. Arquitectura y descripción del sistema

4.1. Overview de la herramienta

Esta herramienta ha surgido de una antigua idea de ASLab de crear un **ICe** para el grupo de investigación. Su propósito fundamental es crear un entorno de desarrollo y operación para los sistemas ASys. Esto se haría mediante la construcción de un entorno RCP, Rich Client Platform, basado en eclipse donde manejar los sistemas autónomos desarrollados en proyectos anteriores. Además este entorno permitirá integrar las herramientas de ASys ya disponibles e integrar las que se desarrollarán en el futuro.

Figura 50 Logo de ICe

Siguiendo las especificaciones, la plataforma se ha construido sobre Eclipse. La herramienta creada es un **proyecto plug-in** que incorpora varios plug-ins a su vez como pueden ser vistas, nuevas acciones…etc. En este programa se ha incorporado una nueva perspectiva, perspectiva ASLab en la que por defecto se lanzan una serie de vistas previamente configuradas. Con ellas y siguiendo el manual de usuario, se puede interactuar con las librerías de ROS de manera muy intuitiva ya que en ningún momento hace falta escribir ningún comando por consola. La figura 4.2 muestra un Overview de la perspectiva creada.

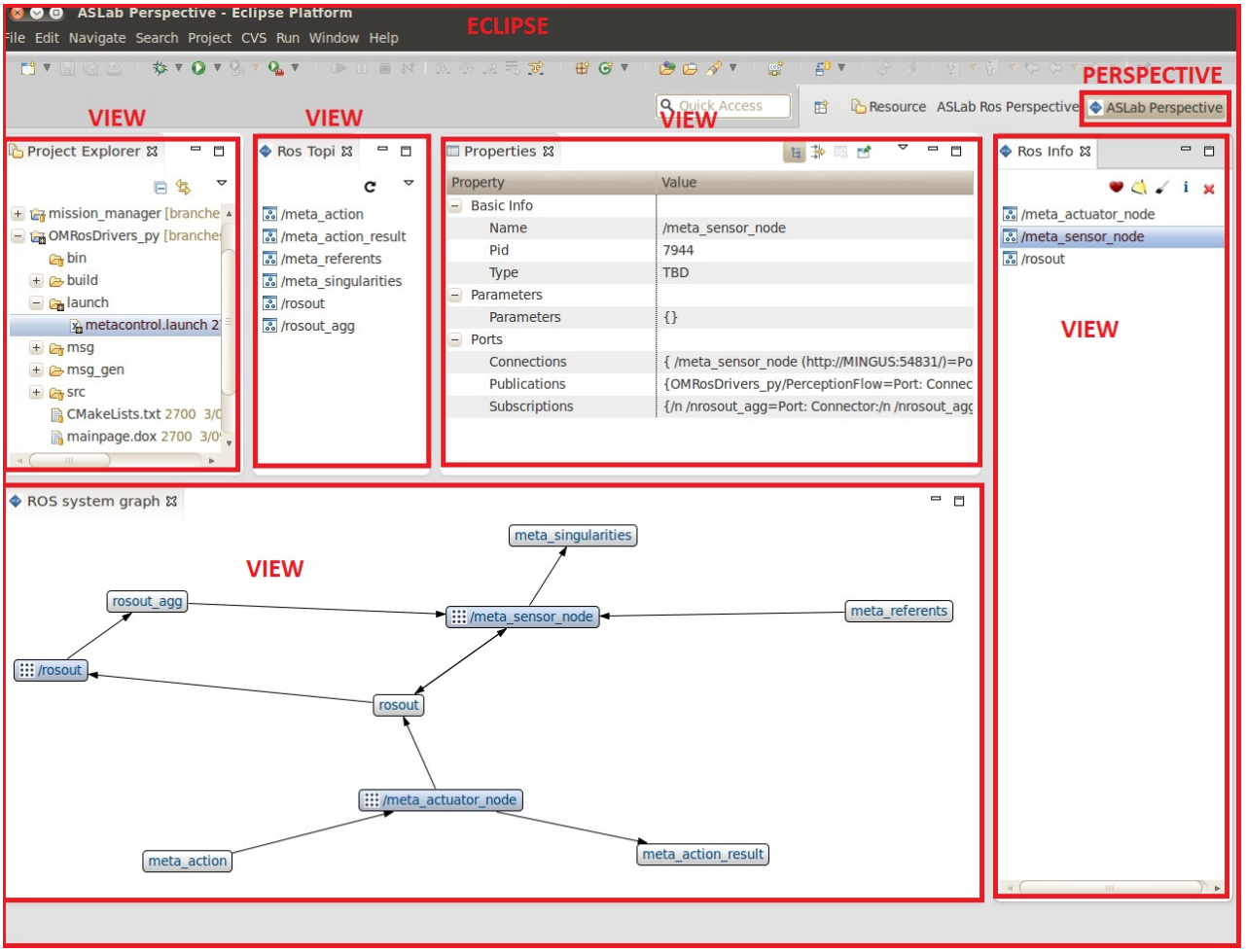


Figura 51 Overview del ICe de ASLab.

Si se observa la figura 51, a simple vista se aprecia que el ICe está compuesto por 5 vistas iniciales y una perspectiva que engloba todo. Dos de las vistas incluidas son dos de las vistas del propio Eclipse y sobre las cuales se ha extendido la funcionalidad para que se comportaran de la manera deseada. Además incluye dos plug-ins adicionales para el lanzamiento de nodos que se explicarán en los puntos que vienen a continuación.

Además cabe destacar que todas las clases de proyecto y archivos de código están escritos en Java, exceptuando los .xml que genera Eclipse. Estas clases Java están agrupadas en paquetes de manera que estas quedan organizadas. Todas cuelgan de la carpeta inicial para el proyecto de Eclipse, scr, y también de un nuevo paquete llamado “org.aslab.asys.ice”, que es la rama concreta donde se encontraría el proyecto dentro del software ASLab. Esto aparece visualmente en la figura 52.

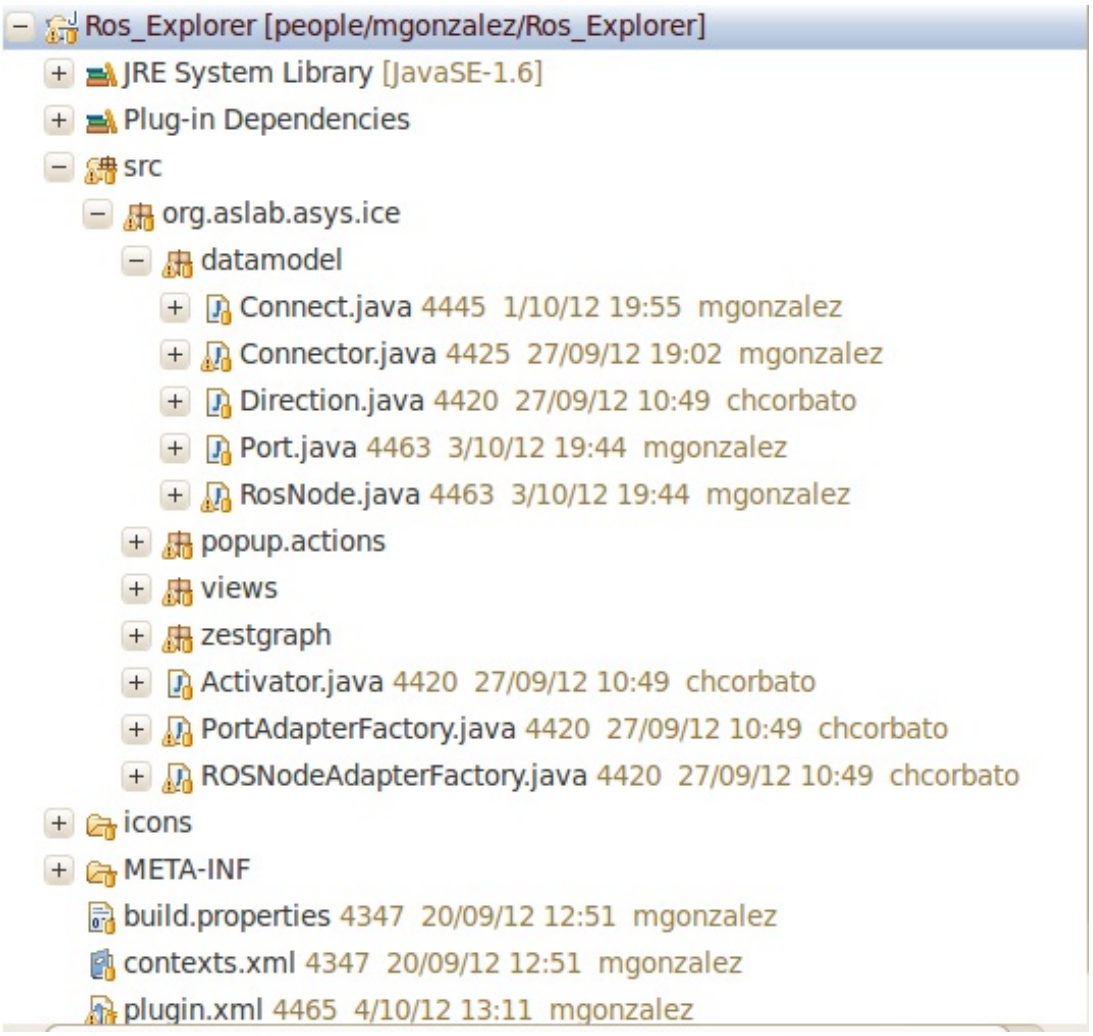


Figura 52 Organización de los packages del proyecto.

De la carpeta org.aslab.asys.ice cuelgan cuatro paquetes a su vez y tres clases Java. Los paquetes han sido creados para tener en ella las clases java agrupadas por tematica, y existen los packages de modelo de datos, “datamodel”, el que contiene las clases Java de los plug-ins de los menús PopUp, “popup.actions”, el que contiene las clases de las vistas del ICe, “views” y el que contiene los archivos que necesita Zest para pintar el grafo, “zestgraph”.

El paquete de views se anida a su vez con un nuevo paquete que contiene las clases que necesita la vista de propiedades de Eclipse para mostrar en la tabla las mismas.

Figura 53 Package Properties.

Por último destacar que el paquete zestgraph contendrá los archivos Java que necesita el plug-in Zest para pintar el diagrama de grafos correspondiente.

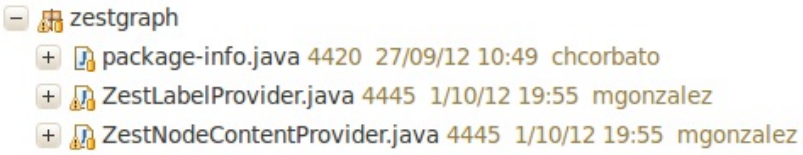


Figura 54 Package zestgraph.

4.2. Perspectiva ASLab

Una perspectiva en Eclipse es un marco que engloba varias vistas por defecto. Se trata de un proyecto plug-in de Eclipse si se realiza por separado, pero como está dentro del proyecto ICe habrá que especificar esta nueva extensión utilizada dentro del fichero "plugin.xml". En este fichero Xml se le indicará cual es el punto de Eclipse que se desea extender, en este caso es el paquete org.eclipse.ui.perspectives.

```
</extension>
<extension
      point="org.eclipse.ui.perspectives">
   <perspective
         class="org.aslab.asys.ice.views.Perspective"
         icon="icons/aslab.png"
         id="ros_explorer.views.Perspective"
         name="ASLab Perspective">
   </perspective>
</extension>
```

Figura 55 Extensión en el XML para perspectivas.

Además se le indicará cual es la clase que debe consultar cuando se activa esta perspectiva y ésta se encuentra anidada en los paquetes anteriores, concretamente en org.aslab.asys.ice.view, e iría controlada por la clase Perspective.java.

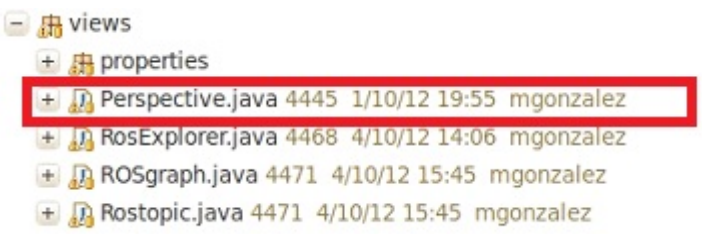


Figura 56 Clase Perspectiva.java.

Clase Perspective

La clase perspective hereda las propiedades de la clase de Eclipse *IPerspectiveFactory*. Es por ello que debe implementar como fijo al menos el método *createInitialLayout()*. Una vez creado este layout inicial, por código se indicará como organizar y que vistas utilizar en la misma, y esto se hará dentro del método *addViews()*. Estas se organizarán según aparecen en el código, es decir, una vez situada una vista, la siguiente lo hará tomando el espacio restante como total ya que las vistas se sitúan indicando el porcentaje del espacio restante que deben ocupar.

Estas vistas se pueden situar tomando como referencia cuatro puntos del *IPageLayout*. Estos son la parte de arriba (*TOP*), la parte de abajo (*BOTTOM*), el lado izquierdo (*LEFT*), y el lado derecho (*RIGHT*). En este método están situadas las cinco vistas iniciales. El Project Explorer se encontraría en la parte izquierda, a su derecha estaría la vista de Ros Topic, a la derecha del Ros Topic se encontraría la vista de Properties y a la derecha del todo se encontraría la vista de Ros Info. En la parte de abajo está la vista de Ros Graph.

```
IFolderLayout bottom =
        factory.createFolder(
            "bottomRight", //NON-NLS-1
            IPageLayout.RIGHT,
            0.8f,
            factory.getEditorArea());
    bottom.addView(ROS_EXPLORER_VIEW_ID);
```

Figura 57 Código para situar el Ros Info dentro de la Perspective.

Tomando como ejemplo como situar la vista Ros Info, veasé figura 57, se va a explicar cómo se haría. Primero se le pone una etiqueta de donde se va a situar, en el siguiente campo se indica donde va situado tomando como referencia los cuatro puntos disponibles del IPageLayout, en el siguiente campo se le indica el porcentaje de espacio que debe ocupar y en el último se le indica que actualice el area restante de la perspectiva. Una vez hecho esto se añade al espacio creado para la vista, la vista que se desee.

Además la clase perspective.java tiene otros métodos para añadir más partes de eclipse. El método *addPerspectiveShortcuts()* permite añadir tres botones de acceso rápido a Eclipse. Se han añadido el de la perspective de Resource, el CVS y la sincronización con el equipo donde se trabaja.

```
private void addPerspectiveShortcuts() {
    factory.addPerspectiveShortcut("org.eclipse.team.ui.TeamSynchronizingPerspective"); //NON-NLS-1
    factory.addPerspectiveShortcut("org.eclipse.team.cvs.ui.cvsPerspective"); //NON-NLS-1
    factory.addPerspectiveShortcut("org.eclipse.ui.resourcePerspective"); //NON-NLS-1
}
```

Figura 58 Código de la función addPerspectiveShortcuts.

Para añadir nuevos Wizard se tiene la función *addNewWizardShortcuts()*. Mediante ella se han añadido tres botones de acceso rápido para crear una nueva carpeta, un nuevo archivo y para hacer un checkout de un proyecto.

```
private void addNewWizardShortcuts() {
    factory.addNewWizardShortcut("org.eclipse.team.cvs.ui.newProjectCheckout");//NON-NLS-1
    factory.addNewWizardShortcut("org.eclipse.ui.wizards.new.folder");//NON-NLS-1
    factory.addNewWizardShortcut("org.eclipse.ui.wizards.new.file");//NON-NLS-1
}
```

Figura 59 Código de la función addNewWizardShortcuts.

Por último para añadir a la vista botones de acceso directo se tiene la función *addViewShortcuts()*.Con este método se añaden 10 botones de acceso rápido.

```
private void addViewShortcuts() {
    factory.addShowViewShortcut("org.eclipse.ant.ui.views.AntView"); //NON-NLS-1
    factory.addShowViewShortcut("org.eclipse.team.ccvs.ui.AnnotateView"); //NON-NLS-1
    factory.addShowViewShortcut("org.eclipse.pde.ui.DependenciesView"); //NON-NLS-1
    factory.addShowViewShortcut("org.eclipse.jdt.junit.ResultView"); //NON-NLS-1
    factory.addShowViewShortcut("org.eclipse.team.ui.GenericHistoryView"); //NON-NLS-1
    factory.addShowViewShortcut(IConsoleConstants.ID_CONSOLE_VIEW);
    factory.addShowViewShortcut(JavaUI.ID_PACKAGES);
    factory.addShowViewShortcut(IPageLayout.ID_RES_NAV);
    factory.addShowViewShortcut(IPageLayout.ID_PROBLEM_VIEW);
    factory.addShowViewShortcut(IPageLayout.ID_OUTLINE);
}
```

Figura 60 Código de la función addViewShortcuts().

Con el código de la perspectiva funcionando, si esta se incorporara mediante un plug-in a Eclipse saldría una nueva perspectiva ASLab.

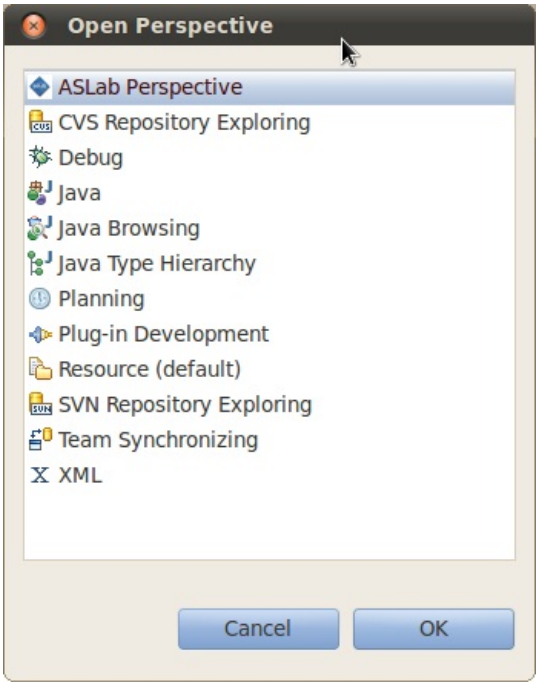


Figura 61 Abrir la perspectiva ASLab.

4.3. Plug-in con PopUps Menus

Dentro del plug-in total que es el marco ICe, existen funcionalidades adicionales que se han añadido a Eclipse. Estos complementos corresponden con un requisito de la interfaz de alejar lo máximo posible al usuario del código de bajo nivel. Antes, el usuario tenía que lanzar los nodos de su proyecto por el terminal indicando manualmente el nombre del proyecto y el nombre del archivo que quería lanzar. Gracias a estas dos nuevas funcionalidades el usuario solo deberá seleccionar el archivo que desee lanzar y pulsar el nuevo botón que surge para este propósito.

Existen dos modos de lanzamiento de archivo. El usuario podrá lanzar archivos con terminación .launch, siempre que estos estén correctamente construidos. Serán lanzados con la configuración que indique su código. El otro modo de lanzamiento permitirá ejecutar archivos con terminación .py, que serán nodos ROS individuales escritos en Python.

Estos dos comportamientos vendrán definidos en dos ficheros Java que se encuentran dentro del Package “org.aslab.asys.ice.popup.actions”.

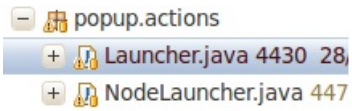


Figura 62 Carpetas donde están los PopupMenus.

CLASE Launcher

Primero se explicará la funcionalidad del lanzador de ficheros .launch que está recogida en el fichero Launcher.java. Añadiendo código XML dentro del fichero .xml del proyecto se indicará la clase que se desea extender, en este caso “org.eclipse.core.resources.IResource”, y el filtro de extensión del archivo con el que se quiere que se active, en este caso .launch.

```
<objectContribution
      objectClass="org.eclipse.core.resources.IResource"
      adaptable="true"
      nameFilter="*.launch"
      id="PopUp_Launcher.contribution1">
   <menu
         label="ASLab Launcher"
         path="additions"
         id="PopUp_Launcher.menu1">
      <separator
            name="group1">
      </separator>
   </menu>
   <action
         label="Launch File"
         class="org.aslab.asys.ice.popup.actions.Launcher
         menubarPath="PopUp_Launcher.menu1/group1"
         enablesFor="1"
         id="PopUp_Launcher.newAction">
   </action>
</objectContribution>
```

Figura 63 Extensión XML para añadir el Launcher.

Este fichero Launch.java contiene varios métodos para comunicarse con la consola e indicarle cual es el fichero que se quiere lanzar. Si la acción se quisiera ejecutar por consola, se haría invocando al comando de ROS "roslaunch" seguido del nombre del proyecto y del nombre del fichero. Esto viene dentro de la función *run()* que está escuchando la acción.

```
public void run(IAction action) {

    String cmd = "roslaunch" + " " + project.getName()+ " "+ resource.getName();

    try{
        Runtime rt = Runtime.getRuntime();
        Process p = rt.exec(cmd);}

    catch (Exception e)
    {
        e.printStackTrace();
    }
```

Figura 64 Código para el lanzamiento de un .launch.

Para recoger los nombres del recurso que se ha utilizado se tiene el método *selectionChanged()*. Este recogerá la acción mediante un *IAction* y la selección con un *ISelection* y realizará el comportamiento que se desea. En este caso es recoger el nombre del proyecto y el fichero para transmitírselos al método *run()*.

```
    public void selectionChanged(IAction action, ISelection selection) {

        if (selection instanceof IStructuredSelection) {
               IStructuredSelection ssel = (IStructuredSelection) selection;
               Object obj = ssel.getFirstElement();
               IFile file = (IFile) Platform.getAdapterManager().getAdapter(obj,
                       IFile.class);
               if (file == null) {
                   if (obj instanceof IAdaptable) {
                       file = (IFile) ((IAdaptable) obj).getAdapter(IFile.class);
                   }

               }
               if (file != null) {

                   resource = (IResource) file;
                   project = file.getProject();
               }
          }
    }
}
```

Figura 65 Código que recoge la acción de un .launch.

```
miquel@MINGUS:~$ roslaunch OMRosDrivers_py metacontrol.launch
```

Figura 66 Código Roslaunch por terminal.

Resumiendo, el comportamiento de este plug-in evita que el usuario tenga que introducir una secuencia como la de la figura 66 y pueda lanzar directamente el fichero que desee lanzar, con el menú PopUp que surge en la figura 67.

Figura 67 Menú adicional que surge al pulsar sobre un .launch.

CLASE NodeLauncher

La clase NodeLauncher.java es muy similar a la anterior, pero presenta una funcionalidad diferente. Esta lanza archivos .py correspondientes a nodos de ROS programados en Python. Se ha deseado, aprovechando la oportunidad que produce lanzar los nodos uno por uno, guardad donde se encuentra el nodo (Path) para poder acceder a él de manera sencilla.

También se ha escrito en XML la extensión del plug-in para aplicar la nueva funcionalidad, y es muy similar al de la clase Launch.java

```
<objectContribution
      objectClass="org.eclipse.core.resources.IResource"
      adaptable="true"
      nameFilter="*.py"
      id="PopUp_NodeLauncher.contribution1">
   <menu
         label="ASLab Node Launcher"
         path="additions"
         id="PopUp_NodeLauncher.menu1">
      <separator
            name="group1">
      </separator>
   </menu>
   <action
         label="Launch Node"
         class="org.aslab.asys.ice.popup.actions.NodeLauncher"
         menubarPath="PopUp_NodeLauncher.menu1/group1"
         enablesFor="1"
         id="PopUp_NodeLauncher.newAction">
   </action>
</objectContribution>
```

Figura 68 Extensión XML para añadir el NodeLauncher.

Para compartir la información del path del nodo lanzado ha sido necesario conectarlo con los demás plug-ins. Para ello se ha implementado a la clase la extensión ISelectionListener y IPropertyChange. Lo que se desea es informar a las demás vistas de que se ha producido un cambio de este tipo y ellas actuarán en consecuencia. Se ha creado una lista de clases, que escuchan el cambio de esta, llamada *myListeners* en la que existen dos métodos para actuar con ella. El método *addPropertyChangeListener()* que añade plug-ins externos a la lista para luego ser notificados y *removePropertyChangeListener()* que borrará la lista en caso de que se desee.

```
static private List<IPropertyChangeListener> myListeners = new ArrayList<IPropertyChangeListener>();

// A public method that allows listener registration
static public void addPropertyChangeListener(IPropertyChangeListener listener) {
    if(!myListeners.contains(listener)) // TODO runtime exception here
        myListeners.add(listener);
}
// A public method that allows listener registration
static public void removePropertyChangeListener(IPropertyChangeListener listener) {
    myListeners.remove(listener);
}
```

Figura 69 Código del NodeLauncher para manejar la lista de Listeners.

Una vez establecido cuales son los suscriptores al cambio se define una acción que realiza el método *updateListeners()*. Se deseaba enviar el path, y esto se hace enviando a las demás vistas un *IResource* y que ellas interpreten la información como deseen.

```
private void updateListeners(){
    for (Iterator iter = myListeners.iterator(); iter.hasNext();) {
        IPropertyChangeListener element = (IPropertyChangeListener) iter.next();
        IResource myFile = resource;
        element.propertyChange(new PropertyChangeEvent(this,
                IDNodeLauncherChange , null , resource ));

    }
}
```

Figura 70 Acción al actualizar los Listeners.

La acción que realiza el PopUpMenu es muy similar al de la clase Launcher.java. En este caso se realiza una llamada a la consola con los comandos de ROS “rosrun” indicando en nombre del proyecto y el nombre del nodo que se desea lanzar. Esta función *run()* es diferente a las demás ya que después de ejecutarse tiene que notificar a los demás plug-ins que ha cambiado algo. Esto se realiza llamando a la función *updateListeners().*Véase figura 71.

```
public void run(IAction action) {

    String cmd = "rosrun" + " " + project.getName()+ " "+ resource.getName();

    String NodeName = resource.getName();

    try{
        Runtime rt = Runtime.getRuntime();
        Process p = rt.exec(cmd);}

    catch (Exception e)
    {
        e.printStackTrace();
    }

    System.out.println("Action executed ==> "+cmd);
    System.out.println("");

    updateListeners();
```

Figura 71 Código para el lanzamiento de un .py.

El menú que aparecerá para lanzar el nodo será el siguiente (Figura 72)

Figura 72 Menú adicional que surge al pulsar sobre un .py.

4.4. Vistas incluidas en la herramienta

La perspectiva tiene cinco vistas incluidas en ella. Estas son el Project Explorer, la vista de propiedades de Eclipse, y tres vistas creadas que son la Ros Info, la Ros Topic View y la ROSGraph view. Cabe recordar que todas las vistas trabajan con los modelos de datos de la carpeta datamodel. En ella se encuentran las clases con las que trabaja todo el proyecto que contienen todos los modelos de datos que se van recogiendo. La clase más importante dentro de los modelos de datos será la clase RosNode.java.

El paquete que contiene los ficheros .java de las clases es la de views. En ella se encuentra otro paquete a su vez que contiene los ficheros que extienden la vista de propiedades para que por ella se muestren las propiedades de los nodos en los que se pincha.

Figura 73 Package correspondiente a las vistas.

Se ha decidido incluir el Project Explorer porque es una vista muy utilizada por los desarrolladores en la que ven sus proyectos y las carpetas que contienen los archivos fuentes que están desarrollando. Además será en esta vista donde se puedan lanzar los dos plu-ings con PopUpMenu explicados en los puntos anteriores.

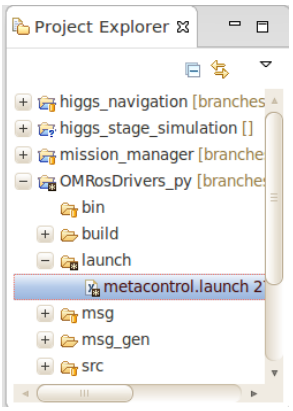


Figura 74 Vista del Project Explorer.

Un requisito de la herramienta con ROS era poder ver las propiedades de los nodos de manera fácil, sin tener que acceder a través de la consola a su información individual. Para ello se ha extendido la funcionalidad de la vista de Propiedades que viene con Eclipse dentro de la clase ROSNodePropertySource.java que se encuentra en la carpeta de properties.

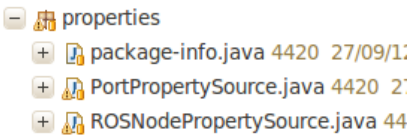


Figura 75 Carpeta de Properties.

CLASE ROSNodePropertySource

El archivo ROSNodePropertySource.java debe heredar las características de las vistas properties que llevarán la extensión IPropertySource. Deben implementar unos métodos que indiquen los campos que se desean mostrar en la vista.

```
public class ROSNodePropertySource implements IPropertySource {
```

Figura 76 Cabecera de la ROSNodePropertySource.java.

El tipo de variable más importante con el que trabaja esta clase es el *IPropertyDescriptor*, que será el descriptor de propiedades de la vista Eclipse. Esta clase implementa tres métodos fundamentales, obligados por heredar propiedades de la clase *IPropertySource*, para la identificación de la información a mostrar. Toda la información está recogida en los modelos de la carpeta datamodel, principalmente dentro de la clase *RosNode.java*.

El método *getPropertyDescriptors()* es el encargado de crear los campos a mostrar dentro de la vista, y trabajará con un Array de *IPropertyDescriptors.* Existen tres categorías fundamentales de los datos a mostrar. La información básica que incluirá el nombre del nodo (*Name*), el tipo de nodo (*type*), el path donde se encuentra el nodo (*Path*) y el numero del proceso interno dentro de la CPU(*Pid*). La segunda categoría fundamental recoge, en caso de que existan, los parámetros fijados para el nodo (*Parameters*). Por último la tercera categoría contiene la información de las conexiones del nodo que son las conexiones (*Connections*), publicaciones (*Publications*) y subscripciones (*Subscription*s).

```
public IPropertyDescriptor[] getPropertyDescriptors() {

    TextPropertyDescriptor nameProp=new TextPropertyDescriptor("name", "Name");
    nameProp.setCategory("Basic Info");
```

Figura 77 Método getPropertyDescriptors().

Mediante *TextPropertyDescriptor* se crean uno por uno los campos descritos anteriormente y se les asigna la categoría a la que corresponden. Una vez creados todos los descriptores, estos se pasan a formato Array y se devuelven a la vista para que los utilice.

```
IPropertyDescriptor[] propertyDescriptors = new IPropertyDescriptor[]{
        nameProp, typeProp,pathProp,pidProp,pubProp,subProp,conProp,paraProp
};
```

Figura 78 Array del método getPropertyDescriptors().

El segundo método fundamental de la clase es el *getPropertyValue()*. Este recibirá el objeto del cual se desea mostrar la información, analizará sus campos internos y cuando encuentre un campo conocido mostrará su información en el caso de que este objeto tenga el método correspondiente para devolverla. Por defecto este método recibe un *Object* genérico, pero en este caso recibirá un objeto de la clase RosNode que es la que contiene toda la información del nodo.

```
public Object getPropertyValue(Object id) {
    if (id.equals("name")) {
        return node.getName();
    }
```

Figura 79 Método getPropertyValue().

El tercer método, *setPropertyValue()*, es el que permite modificar la información de los objetos clicando sobre el campo que se desee cambiar. Este recibe dos objetos, el primero es el campo que se desea modificar (*id*) y el segundo es el nuevo contenido (*value*). Dentro del método se analiza el contenido del identificador, y en caso de corresponder con algún campo del objeto principal, y de que este presente un método para actualizar sus campos, estos se modificarán.

```
public void setPropertyValue(Object id, Object value) {
    String s = (String) value;
    Integer i = new Integer((String)value);

    if (id.equals("name")) {
      node.setName(s);
    }
```

Figura 80 Método setPropertyValue().

Una vez implementados los siguientes métodos, el resultado final es el esperado. Si el usuario pincha encima de un nodo de la lista, y si este se ha lanzado correctamente, la vista de propiedades publicará la información del nodo seleccionado.

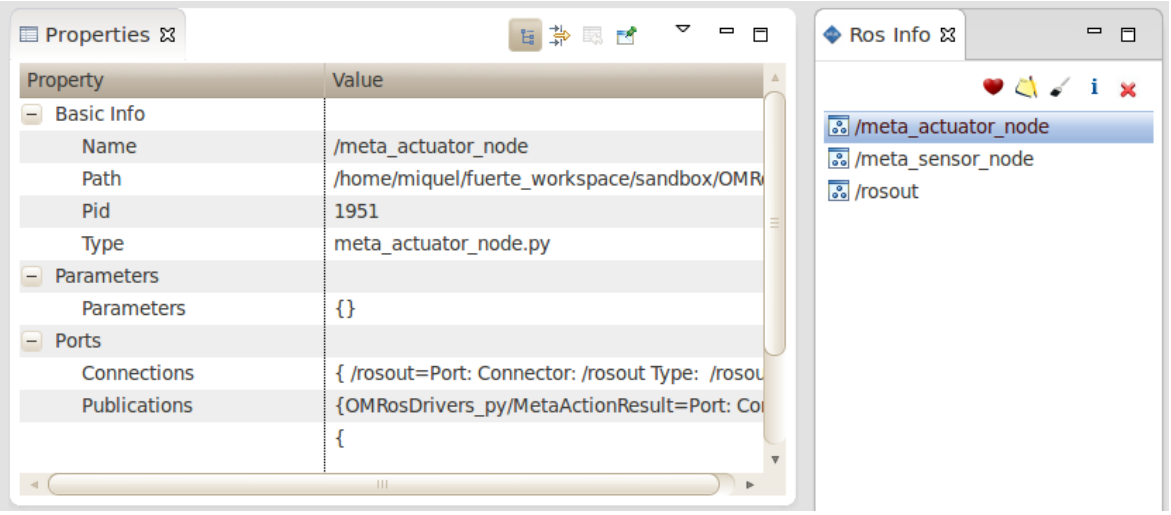


Figura 81 Resultado final de la extensión de la Properties View.

4.4.1. Ros Info View

La Ros Info View, es la clase que contiene toda la información recogida del sistema ROS. Por ello deberá ir conectada a todas las demás clases para escuchar sus eventos y recoger de esta manera la información que estas generan para su correcto almacenamiento. Asimismo esta clase contiene las herramientas necesarias para generar una vista en la que se van a mostrar los nodos activos del sistema ROS.

CLASE RosExplorer

La clase *RosExplorer* hereda las características de una vista estándar de Eclipse (*ViewPart*) e implementa los métodos necesarios para conectarse con otras vistas (*ISelectionListener* y *IPropertyChangeListener*). Toda la información de los nodos se va a registrar en un *Map,* con el nombre de ***nodemap***, en el que la *Key* es una variable de tipo *String* que será el nombre del nodo y el segundo campo será un *Object* de tipo *RosNode*. Se ha recogido la información en un *Map* porque a la hora de encontrar la información de un nodo, esta se puede obtener simplemente con su *Key* que es el nombre del nodo.

Además, esta vista va conectada con el PopUp plug-in de lanzamiento de nodos individuales, con lo que es necesario crear un nuevo mapa para recoger la información que envía esta clase. El mapa recibe el nombre de ***pathmap***, con el primer campo una variable de tipo *String* y en el segundo un objeto de tipo *IResource* que será el que contenga la información del nodo lanzado. El análisis de esta *IResource* se analizará en el método correspondiente para ser recuperado cuando sea necesario.

```
public class RosExplorer extends ViewPart implements  ISelectionListener, IPropertyChangeListener {

    /**
     * HashMap of the RosNodes running in the RosMaster in which you are connected.
     */
    private Map<String, RosNode> nodemap = new HashMap<String, RosNode>();

    private Map< String, IResource> pathmap = new HashMap<String,IResource>();

    public static final String ID = "org.aslab.asys.ice.views.RosExplorer";

    public static final String IDRosExplorerChange = "RosExplorerChange";

    public static final String IDROSGraphChange = "ROSGraphChange";

    public static final String IDNodeLauncherChange = "NodeLauncherChange";
```

Figura 82 Cabecera de la clase RosExplorer.

Una vez declarados los contenedores de información, se crean las acciones que se van a incluir en la vista. Una acción para lanzar el *RosCore* que será necesaria para lanzar los nodos uno por uno, ya que con el *RosRun* no se lanza el nodo maestro de Ros. Una acción para actualizar la lista de nodos de manera manual. La acción para lanzar el *Rxgraph* también se incluirá además de una acción para "matar" el nodo seleccionado. Si se desea ver la información del nodo en un mensaje existe la acción "nodeinfo" y por ultimo también se ha creado una acción para ver el código de los nodos lanzados, pero este se desarrollará en futuras extensiones de la herramienta.

```
private Action Launch_RosCore;
private Action Node_list;
private Action Rxgraph;
private Action killnode;
private Action nodeinfo;
private Action nodecode;
private Action doubleClickAction;
```

Figura 83 Lista de acciones del RosExplorer.

Será necesario crear una lista de *IPropertyChangeListeners* que almacenará en un *ArrayList* las clases que serán informadas cuando esta vista genere un evento de tipo change. Esta lista se actualiza con dos métodos. El método addPropertyChangeListener() que añadirá las clases que escuchan los cambios, y el método *removePropertyChangeListener()* que eliminará los elementos de la lista.

```
static private List<IPropertyChangeListener> myListeners = new ArrayList<IPropertyChangeListener>

// A public method that allows listener registration
static public void addPropertyChangeListener(IPropertyChangeListener listener) {
    if(!myListeners.contains(listener)) // TODO runtime exception here
        myListeners.add(listener);
}
// A public method that allows listener registration
static public void removePropertyChangeListener(IPropertyChangeListener listener) {
    myListeners.remove(listener);
}
```

Figura 84 Lista de Listeners del RosExplorer.

Internamente también existe una clase auxiliar generada al heredar las propiedades de la vista de Eclipse. Esta presenta un método a sobrescribir para añadir elementos a la vista. El método es *getElements()* recibirá un *Object*, que es el mapa de nodos y devolverá los elementos en formato *Array*. Siempre que se refresque la vista internamente el programa llamará a este método para actualizar los elementos existentes.

El método para definir como se controlará esta vista es *createPartControl()*. En este se añade a las clases RosGraph.java y el Plug-in NodeLauncher.java esta vista como *Listener* para que esta sea notificada cuando las otras dos clases cambien. Una vez añadida a la lista de las otras dos clases, se define como actua el viewer y qué campos va a tener.

```
public void createPartControl(Composite parent) {


    ROSgraph.addPropertyChangeListener(this);

    // add this view as a selection listener to the workbench page
    getSite().getPage().addSelectionListener(ID, (ISelectionListener) this);

    NodeLauncher.addPropertyChangeListener(this);

    // add this view as a selection listener to the workbench page
    getSite().getPage().addSelectionListener(ID, (ISelectionListener) this);


    viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);
    ViewContentProvider provider = new ViewContentProvider(nodemap);
    viewer.setContentProvider(provider);
    viewer.setLabelProvider(new ViewLabelProvider());
    viewer.setSorter(new NameSorter());
    getSite().setSelectionProvider(viewer);
    viewer.setInput(nodemap.values().toArray());
```

Figura 85 Método createPartControl() del RosExplorer.

A continuación se irán añadiendo la lista de los botones y acciones a la vista. El método *hookContextMenu()* será el encargado de escuchar las acciones que suceden en la vista y de recuperar el objeto seleccionado con sus propiedades. El método *fillContextMenu()* añadirá nuevas acciones cuando el usuario presione con el ratón derecho encima de un objeto. A este ContextMenu se han añadido las acciones de RosNodeInfo, KillNode y ShowNode Code.

También hay que definir qué botones va a tener la vista en la toolbar mediante el método *fillLocalToolBar()*. En esta barra se han añadido las cinco acciones existentes, es decir, *Launch_RosCore, Node_list, Rxgraph, nodeinfo, killnode* con sus correspondientes iconos. Una vez establecidas las acciones se ha dado una funcionalidad específica a cada una de ellas. Esto se hace con un método llamado *makeActions()* en el que se define el comportamiento deseado al presionar en los botones. Dentro de la clase existe un método, showMessage() que recibe una cadena de caracteres y lanza un mensaje en una ventana externa con el contenido de la cadena.

El método propertyChange() es el encargado de identificar los eventos que llegan a la clase. Este recibe un evento de tipo PropertyChangeEvent, interpreta de donde viene y define un comportamiento específico para su tratamiento.

```
public void propertyChange(PropertyChangeEvent event) {
    if( event.getProperty().equals(IDROSGraphChange)) {

        String aux = (String) event.getNewValue();

        Object obj = FindNode(aux);

        if(obj!=null)
        showMessage("Node Info: \n"+obj.toString());
        else
            showMessage("It's not in NodeList");

    }
if( event.getProperty().equals(IDNodeLauncherChange)) {
```

Figura 86 Método propertyChange del RosExplorer.

Asimismo existe un método *FindNode()* que recibe una cadena de caracteres con el nombre del nodo a recuperar y devuelve el *RosNode* que recoge del *nodemap*. El método rosNodeList() lanzará un comando a la consola y devolverá un *Array* de *Strings* con los nombres de los nodos activos. Si se busca un método para rellenar el *nodemap* este es *CreateNodeList()* y si se necesita un método para rellenar los campos de un *RosNode* individualmente este es *createNodeInfo()*.

Por último debe existir un método que avise a los *Listeners* de la lista de que la clase ha cambiado, y este es *updateListeners()*. Cuando se llame a este método se lanzará un evento de tipo *IDRosExplorerChange* y se enviará los valores del *nodemap* en una *ArrayList* para que la información sea utilizada por los subscriptores.

```
private void updateListeners(){
    for (Iterator iter = myListeners.iterator(); iter.hasNext();) {
        IPropertyChangeListener element = (IPropertyChangeListener) iter.next();
        List<RosNode> nodelist = new ArrayList<RosNode>(nodemap.values());
        element.propertyChange(new PropertyChangeEvent(this,
                IDRosExplorerChange , null , nodelist ));

    }
```

Figura 87 Método updateListeners del RosExplorer.

4.4.2. Ros Topic

La Ros Topic View contiene la lista de ROS topics. En ella simplemente se muestra la lista de los topics que publican los nodos ROS activos en el sistema. La vista está controlada por Rostopic.java.

CLASE Rostopic

Esta clase está subscrita a los cambios del RosExplorer, de manera que cuando está clase publique un evento, la vista de topics se debe actualizar por lo que debe implementar *ISelectionListener* y *IPropertyChange*. Además hereda de *ViewPart* por lo que debe sobreescribir algunos métodos para su correcto funcionamiento.

```
public class Rostopic extends ViewPart implements  ISelectionListener, IPropertyChangeListener
```

Figura 88 Cabecera de la clase Rostopic.

Sus atributos fundamentales son dos Action, una para refrescar la lista de topics manualmente y otra para detectar cuando el usuario hace dobleclick en ella. También tiene un TableViewer, que contendrá la vista con los objetos en ella y un Map que contiene el nombre de los topics en ambos campos. Este mapa solo se usa para pintar el nombre de los mismos.

```
private TableViewer viewer;
private Action refresh;
private Action doubleClickAction;

public static final String IDRosExplorerChange = "RosExplorerChange";

private Map<String, String> topicmap = new HashMap<String, String>();
```

Figura 89 Atributos de la clase Rostopic.

Al heredar de la clase *ViewPart* presenta una clase interna ViewContentProvider. Dentro de la misma existe un getElements() que recibe un Object y devuelve una lista de objects para que sean pintados. Concretamente este método recibe el mapa de topics, topicmap, y devuelve sus nombres. Siempre que la vista se refresque se llamará a este método.

```
public Object[] getElements(Object parent) {
     return topicmap.values().toArray();
}
```

Figura 90 Método getElements() de la clase Rostopic.

El método createPartControl() se llamará al inicio y es el que creará el control de la vista. Al principio subscribirá a la vista al cambio de acciones de la clase RosExplorer y después añadirá las características del viewer añadiéndole sus correspondientes barras de herramientas y botones.

```
public void createPartControl(Composite parent) {


    RosExplorer.addPropertyChangeListener(this);

    // add this view as a selection listener to the workbench page
    getSite().getPage().addSelectionListener(ID, (ISelectionListener) this);
```

Figura 91 Método createPartControl() de la clase Rostopic.

El método *hookContextMenu()* será el encargado de escuchar las acciones que suceden en la vista y de recuperar el objeto seleccionado con sus propiedades. El método *fillContextMenu()* añadirá nuevas acciones cuando el usuario presione con el ratón derecho encima de un objeto. El método *fillLocalToolBar()* será el encargado de añadir botones a la barra de herramientas local de la vista. En esta barra se ha añadido la acción de *refresh*.

Una vez establecidas las acciones se ha dado una funcionalidad específica a cada una de ellas. Esto se hace con un método llamado *makeActions()* en la que se define el comportamiento deseado al presionar en los botones. El método *createTopicList()* será el encargado de crear el *topicmap*, vaciándolo siempre que se llame a este método. Por último el método *propertyChange()* recibirá un evento de que alguna de las vistas a las que esta está subscrita ha cambiado. En este caso solo puede ser de tipo *RosExplorer* ya que es a la única vista a la que está subscrito. Cuando reciba este tipo de evento llamará a la acción *refresh* y la vista quedará actualizada.

```
@Override
public void propertyChange(PropertyChangeEvent event) {

    if( event.getProperty().equals(IDRosExplorerChange)) {
        refresh.run();
        }
    }
```

Figura 92 Método propertyChange() de la clase Rostopic.

4.4.3. Ros Graph View

La vista Ros Graph View es la que mostrará el diagrama de grafo de los nodos activos en el sistema. Se podrá interactuar con los elementos de la misma de manera que estos escuchen eventos. La clase que rige esta vista es ROSgraph.java

CLASE ROSgraph

La clase *ROSgraph.java* incluye la herramienta ZEST para pintar gráficamente los nodos. Para ello se deben incluir las librerías de esta herramienta.

```
import org.eclipse.zest.core.viewers.GraphViewer;
import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.core.widgets.ZestStyles;
import org.eclipse.zest.layouts.LayoutAlgorithm;
import org.eclipse.zest.layouts.LayoutStyles;
import org.eclipse.zest.layouts.algorithms.SpringLayoutAlgorithm;
import org.eclipse.zest.layouts.algorithms.TreeLayoutAlgorithm;
```

Figura 93 Librerías importadas en la clase ROSgraph.

Como además se desea que esta vista reciba y envié eventos a la clase *RosExplorer* también habrá que conectarlas mediante *ISelectionListener* y *IPropertyChangeListener*. Asimismo hereda de la clase ViewPart por lo que también deberá sobrescribir algunos de sus métodos.

Sus atributos principales son dos identificadores, uno para detectar los cambios en el *RosExplorer* y otro para identificar los cambios propios. También tiene un atributo de tipo *String* que se utilizará para registrar el nombre del nodo sobre el que se haya pinchado con el ratón. Además el atributo tipo *Graph* es necesario para utilizar correctamente las librerías de *ZEST*.

```
public class ROSgraph extends ViewPart implements  ISelectionListener, IPropertyChangeListener{

        public static final String ID = "org.aslab.asys.ice.views.ROSgraph";

        public static final String IDRosExplorerChange = "RosExplorerChange";

        public static final String IDROSGraphChange = "ROSGraphChange";

        private Graph graph;

        private String NodeName = new String();
```

Figura 94 Cabeceras de la clase ROSgraph.

Como en las dos vistas anteriores, esta también tiene una lista de *IPropertyChangeListener* sobre la que incluirá mediante el método *addPropertyChangeListener()* las clases que escucharan sus cambios y mediante el método *removePropertyChangeListener()* eliminará los componentes de la lista.

Mediante el método *createPartControl()* se establecerá como se inicia la vista y sus características. Es aquí donde se inscribe esta vista en la lista de *Listeners* de la clase *RosExplorer* para pintar los nodos cada vez que la clase publique un evento.

```
public void createPartControl(Composite parent) {

    RosExplorer.addPropertyChangeListener(this);

    // add this view as a selection listener to the workbench page
    getSite().getPage().addSelectionListener(ID, (ISelectionListener) this);
```

Figura 95 Método createPartControl() de la clase ROSgraph.

El método *propertyChange()* es el que recibirá los *PropertyChangeEvent* e interpretará de qué clase son los cambios. Internamente, esta clase tendrá un *HashMap* de nodos en el que recogerá en en el primer campo el nombre del mismo y en el segundo guardará su *GraphNode*. El *GraphNode* es el elemento que usa *ZEST* para pintar los nodos. Dentro de este método también se establecerá el sentido de las conexiones entre nodos y su dirección mediante *GraphConnection.*

```
public void propertyChange(PropertyChangeEvent event) {

    if( event.getProperty().equals(IDRosExplorerChange)) {
        nodes = (List<RosNode>) event.getNewValue(); //Lista del Rosnode

        graph.clear();

        Map <String,GraphNode> mapa = new HashMap<String,GraphNode>();
```

Figura 96 Método propertyChange() de la clase ROSgraph.

Por último con el método updateListeners() se informará a las clases subscritas a los cambios de la clase ROSgraph de que esta ha cambiado. Se les enviará el nombre del nodo sobre el que se ha pinchado.

```
private void updateListeners(){
    for (Iterator iter = myListeners.iterator(); iter.hasNext();) {
        IPropertyChangeListener element = (IPropertyChangeListener) iter.next();
        String nodename = NodeName;
        element.propertyChange(new PropertyChangeEvent(this,
                IDROSGraphChange , null , nodename ));

    }
```

Figura 97 Método updateListeners() de la clase ROSgraph.

Capítulo 5

5. Ejemplos de Uso

En el siguiente apartado, se mostrará cómo funciona la aplicación individualmente con cada una de sus vistas. Para ello se trabajará siempre con el mismo ejemplo en todas ellas y será lanzando del proyecto OMRosDrivers_py, de su carpeta launch el fichero metacontrol.launch, exceptuando el primer apartado. Este proyecto contiene un fichero .launch y el código de dos nodos.

5.1. Lanzamiento de nodos

Para el lanzamiento de los nodos habrá que encontrarse en el Project Explorer sobre un proyecto ya desarrollado que permita en algún archivo lanzar nodos de manera conjunta o de manera individual.

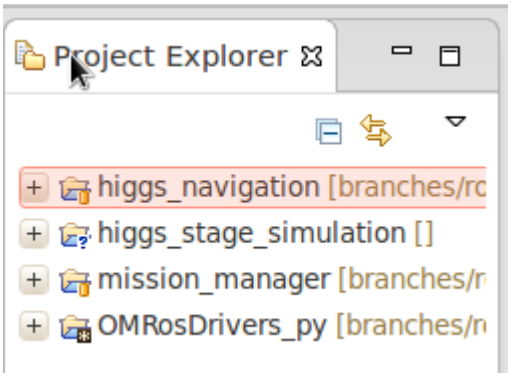


Figura 98 Vista del Project Explorer.

Existen dos modos de lanzamiento:

El primero de ellos es el lanzamiento de ficheros de código con extensión .launch. Dentro de los mismos irá el código relativo para lanzar los nodos que se quieran de manera conjunta. Ordenando que se ejecute la acción sobre el fichero este se lanzará evitando escribir por pantalla el RosLaunch con su nombre del proyecto y fichero correspondientes.

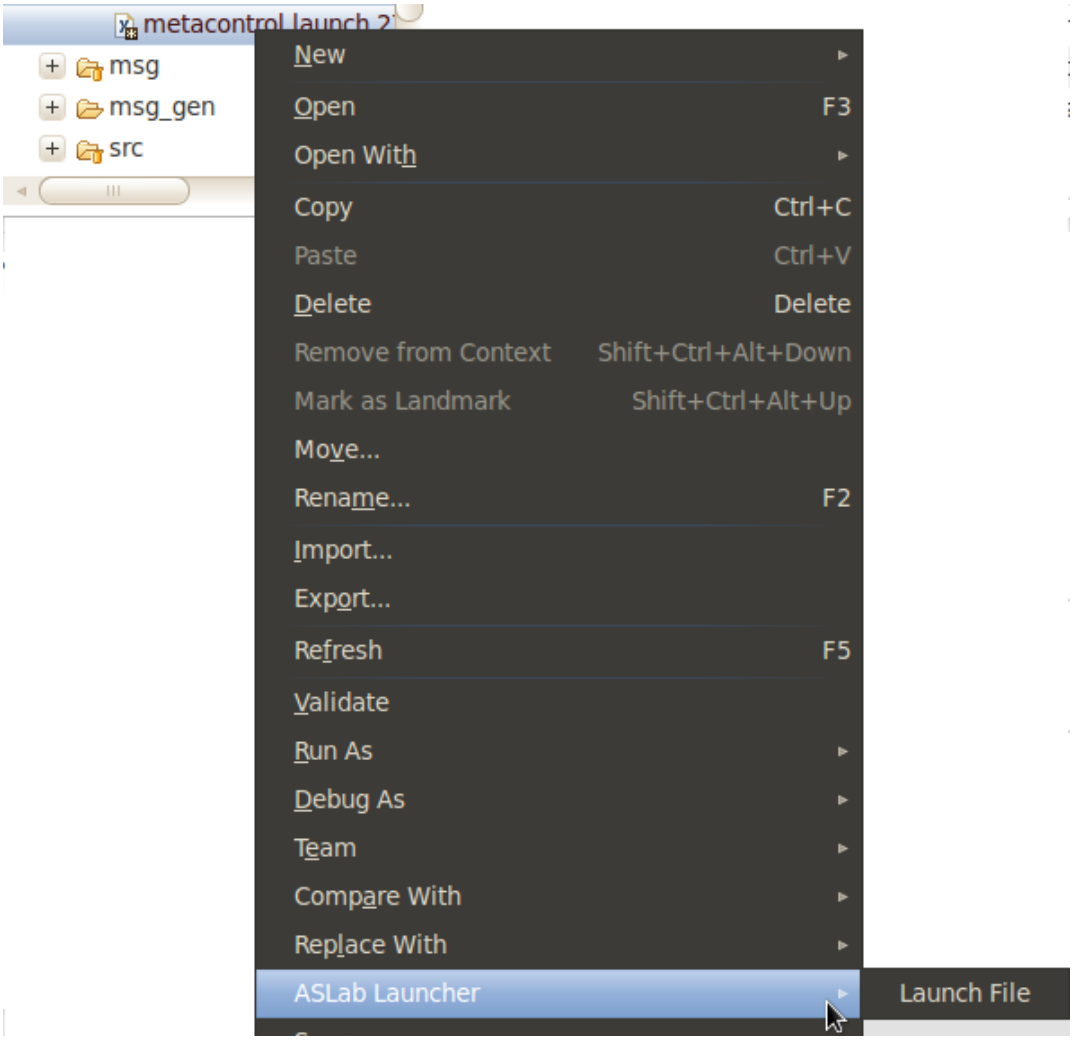


Figura 99 Lanzamiento de un *.launch.

Tras el lanzamiento del archivo se mostrará por consola el comando ejecutado sobre qué proyecto y sobre qué archivo. Se mostrará un mensaje de error si no se hubiera lanzado correctamente.

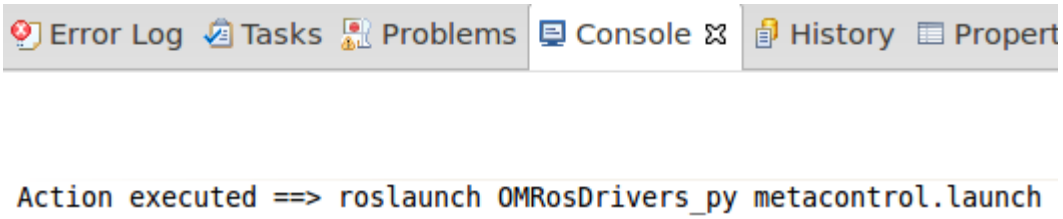


Figura 100 Consola tras el lanzamiento de un *.launch.

El segundo de ellos es el lanzamiento de ficheros de código con extensión .py Estos deberán corresponder a nodos ROS escritos en python. Dentro de los mismos irá el código del nodo. Activando la opción de lanzamiento se ejecutará el código RosRun de lanzamiento correspondiente. Para que esto se lance correctamente deberá estar "corriendo" un RosCore, ya que la acción RosRun no lo lanza por defecto, como sí lo hace la acción RosLaunch.

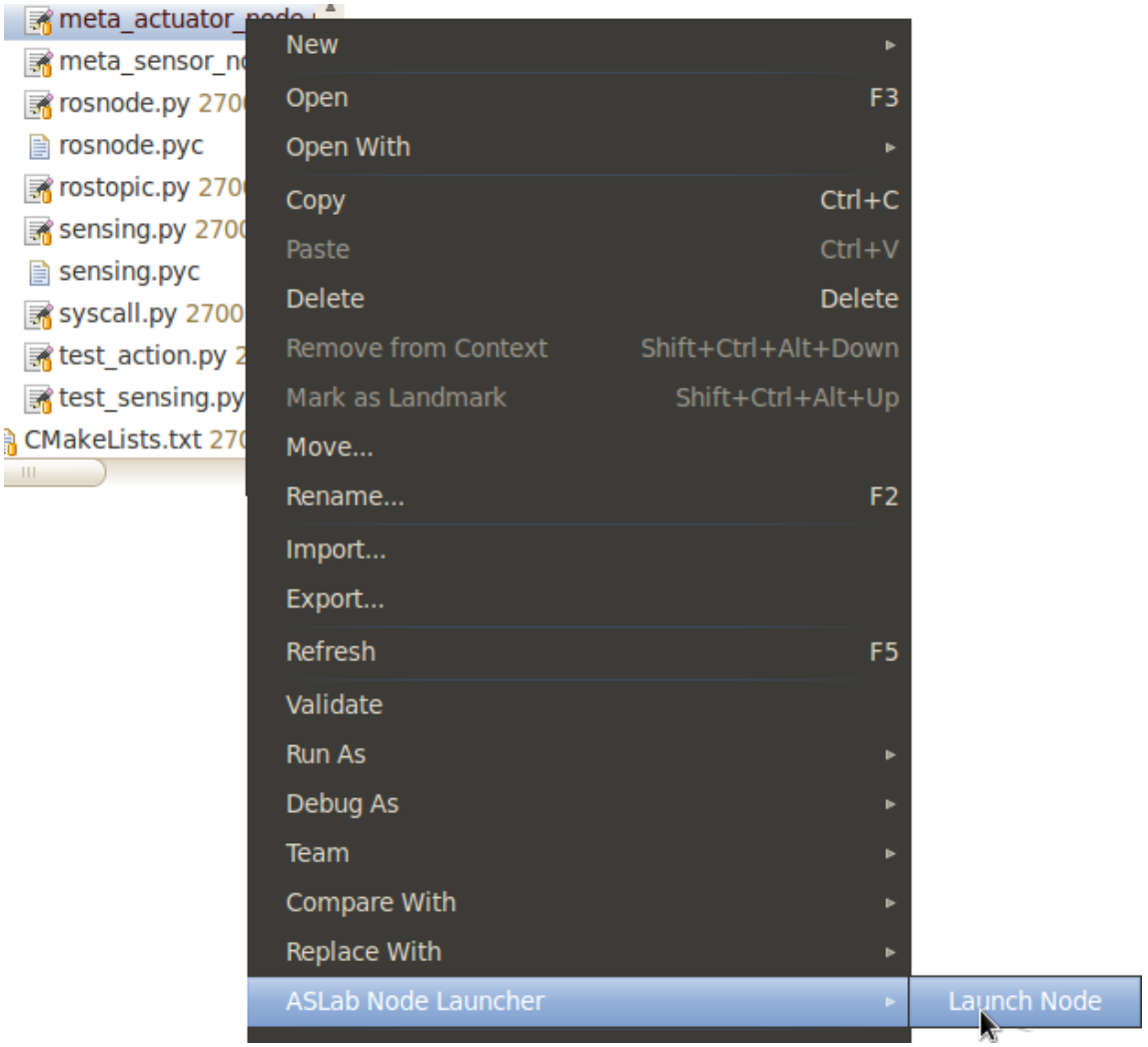


Figura 101 Lanzamiento de un *.py.

Tras el lanzamiento del nodo se mostrará por consola el comando ejecutado sobre qué proyecto y sobre qué archivo. Se mostrará un mensaje de error si no se hubiera lanzado correctamente.

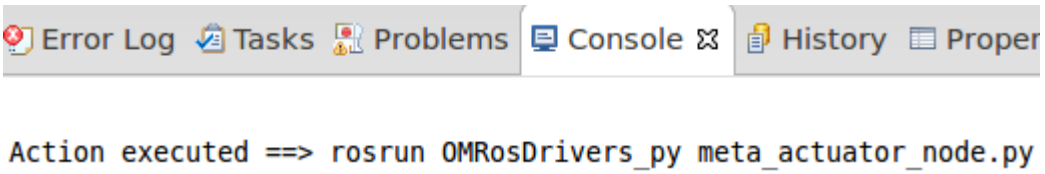


Figura 102 Consola tras el lanzamiento de un *.py.

5.2. Listado de tópicos

Una vez lanzados los nodos en un sistema ROS, estos publican y/o reciben una serie de topics. En la vista Ros Topic se muestran los topics activos en el sistema para que el usuario los vea de forma directa. Inicialmente, si no ha sido lanzado el RosMaster la vista se encontrará vacía.

La vista posee un botón para la actualización de sus tópicos manualmente. La vista se actualiza de dos maneras posibles. La primera es mediante el botón, que generará un refresh interno de la vista. La segunda forma se producirá por estar conectada con la vista Ros Info. Cuando esta vista genere un evento de su tipo, la vista RosTopic la leerá y la lista se actualizará.

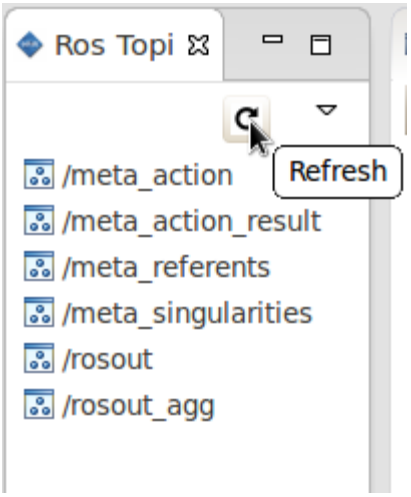


Figura 103 Refresh en la vista Ros Topic.

Cada vez que se refresque la lista de tópicos, ya sea de una manera o de otra, se mostrará por consola un mensaje para que el usuario aprecie la actividad dentro de la vista.

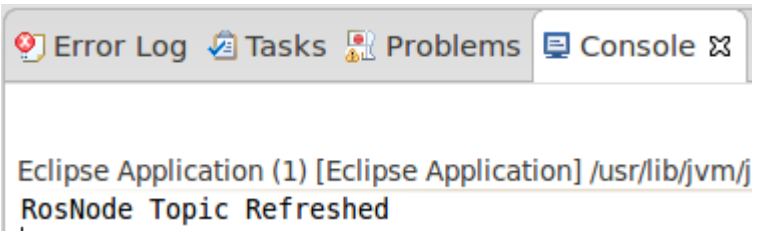


Figura 104 Mensaje al refrescar la vista Ros Topic.

5.3. Funcionamiento de la vista Ros Info

La vista donde se muestran en una lista los nodos activos del sistema es la Ros Info. Inicialmente estará vacía ya que para actualizarla será necesario pulsar el botón de RosNode List. Ella tiene dos maneras de acceder y interactuar con la lista de nodos ROS. La primera de ellas es mediante uno de los cinco botones y la segunda es pinchando con el ratón derecho sobre los nodos y eligiendo una de las opciones deseadas.

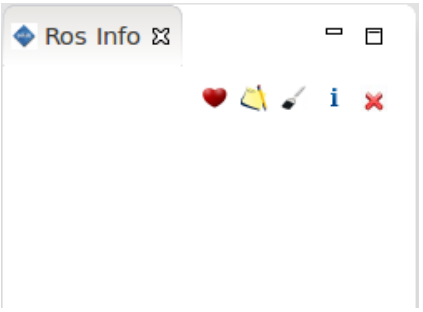


Figura 105 Lista de nodos inicial en la vista Ros Info.

Para generar la lista de nodos ROS y que estos se muestren por pantalla, habrá que pulsar el botón de RosNode List. Una vez creada internamente en la clase correspondiente, aparecerá en la vista una lista con todos los nodos activos.

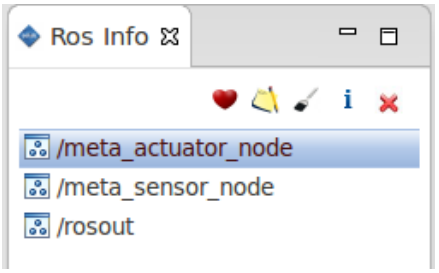


Figura 106 Lista de nodos en la vista Ros Info.

Por consola el usuario recibirá un mensaje que indicará que la lista se ha actualizado correctamente.

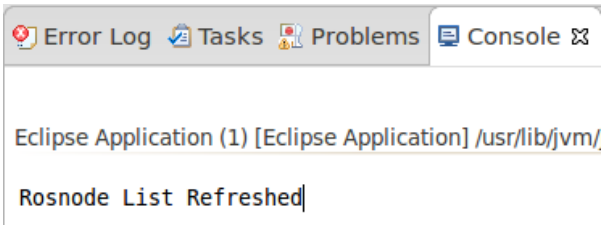


Figura 107 Consola tras generar la lista de nodos en la vista Ros Info.

Si se deseara ver las acciones ocultas de la vista, se deberá pulsar ratón derecho sobre uno de los objetos de la lista. Al pulsar Node Info saldrá una ventana que mostrará la información del nodo ROS.

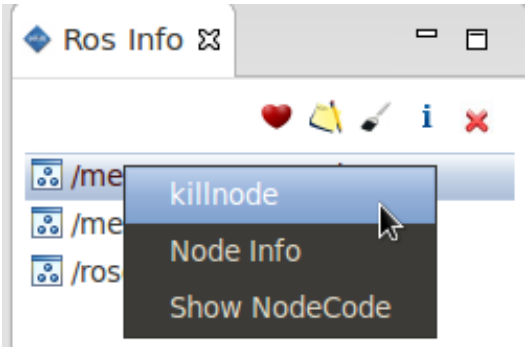


Figura 108 Acciones al pulsar botón derecho en la vista Ros Info.

Si se desea eliminar un nodo, se pulsará el botón kill node o click derecho y se seleccionará la acción correspondiente sobre el nodo seleccionado. Esto provocará que el sistema elimine el nodo de la lista y a su vez del sistema ROS activo. Cuando se realice esta tarea, se mostrará por consola un mensaje para que el usuario vea que nodo ha eliminado. Siempre después de esta acción se producirá una llamada a la acción RosNode List para que la lista se actualice automáticamente sin que el usuario tenga que pulsar ningún botón.

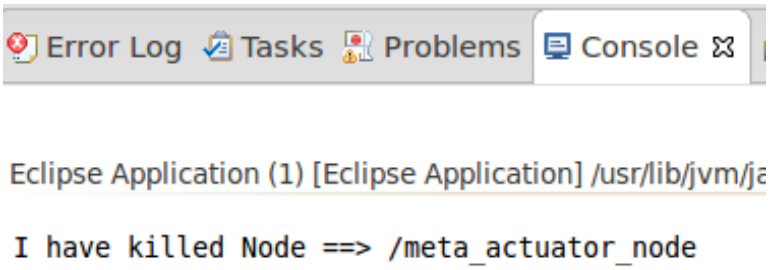


Figura 109 Acciones al pulsar botón derecho en la vista Ros Info.

5.4. Representación gráfica de los nodos

Se ha deseado pintar los nodos de ROS corriendo en un sistema en un diagrama de grafos, en el que se identifiquen de manera sencilla los nodos, los tópicos y sus relaciones. Todos estos grafos y conexiones se imprimen sobre la vista ROSgraph. Inicialmente, la vista generará cuatro nodos por defecto en los que se puede leer el mensaje en ingles, Generate the Node List.

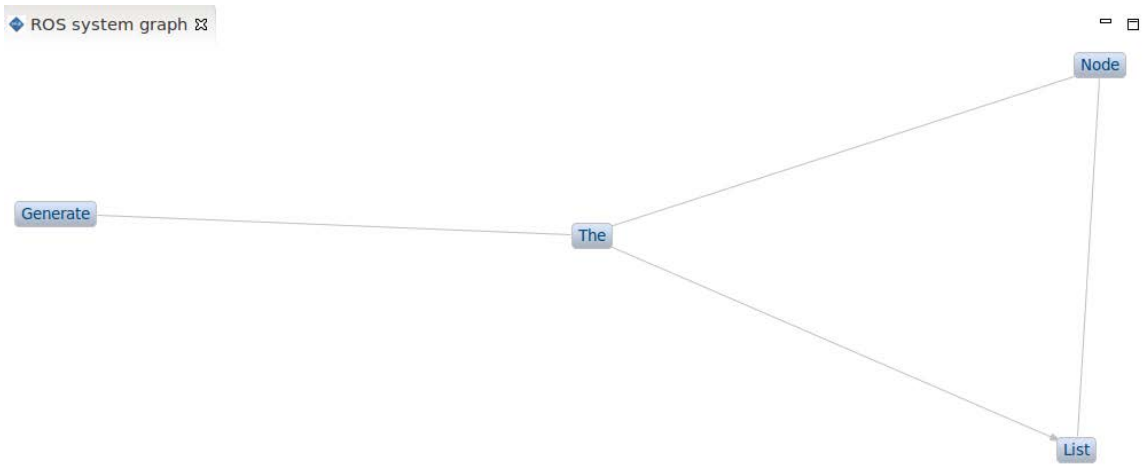


Figura 110 Nodos iniciales de la vista ROSgraph.

Solo existe una manera posible para que la vista se actualize y es que el mapa de nodos de la vista Ros Info se actualice. Cada vez que esta presenta un cambio, se genera un evento que recoge esta vista que a su vez se actualiza.

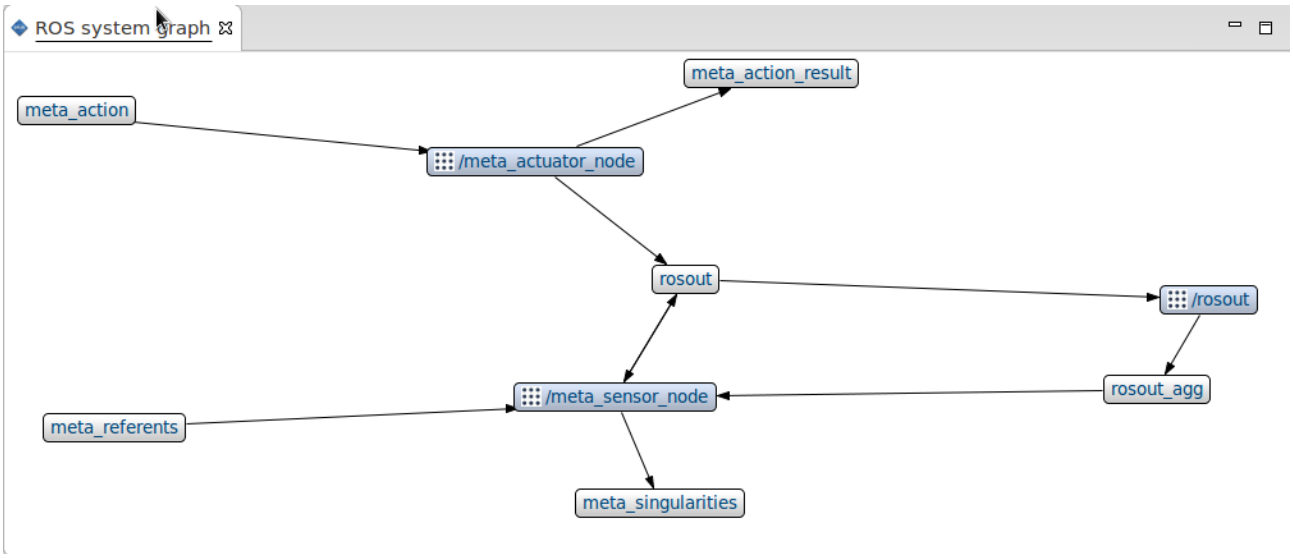


Figura 111 Nodos activos en la vista ROSgraph.

Si se deseara interactuar con los nodos del grafo, estos son sensibles al click del ratón. Pulsando con el botón izquierdo sobre un nodo del sistema, la vista sacará una ventana con la información del nodo en el caso de que este lo sea y el nodo se iluminará.

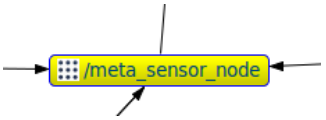


Figura 112 Nodos seleccionado en la vista ROSgraph.

La ventana con la información será como se ve en la figura 113

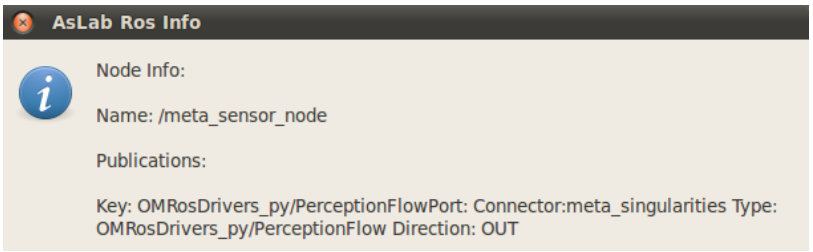


Figura 113 Información mostrada al pulsar un nodo de la vista ROSgraph.

En el caso de que al pulsar sobre un nodo del grafo, este no sea un nodo de ROS, saltará un mensaje que indicará al usuario que este *GraphNode* no está en la lista de nodos.

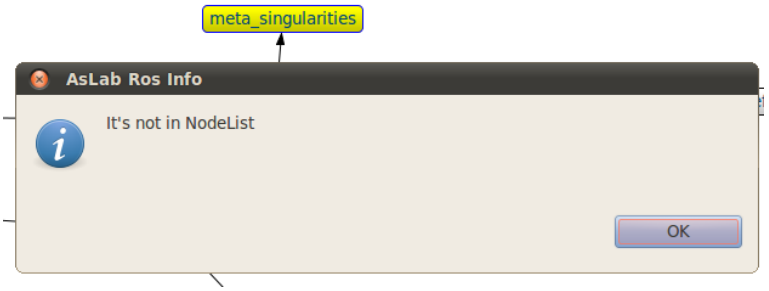


Figura 114 Información mostrada al pulsar un topic de la vista ROSgraph.

Por último, se mostrará por consola el nombre del nodo seleccionado.

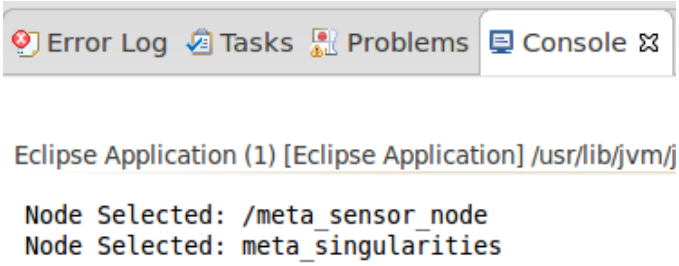


Figura 115 Información mostrada por consola en la vista ROSgraph.

Capítulo 6

6. Conclusiones

Como resultado del proyecto se ha obtenido una herramienta que permite al usuario interactuar con los nodos de ROS sin necesidad de “bajar” al nivel de consola de comandos e integrar toda esta nueva funcionalidad en un plug-in para el sistema de desarrollo Eclipse.

Interactuación con un sistema ROS

El uso de sistemas ROS está muy extendido a la hora de desarrollar robots y sistemas móviles, pero lamentablemente su interfaz no es nada atractiva, por no decir inexistente. Actualmente se han de escribir comandos en el terminal para apreciar lo que está pasando dentro del sistema tanto en sus nodos, como sus tópicos, parámetros etc.

Tener toda la información disponible de forma visual, en una sola ventana de lo que está pasando es esencial para la localización y depuración de errores en el sistema que se está diseñando. De esta manera se puede apreciar si el sistema se está comportando de la manera esperada rápidamente, si está bien conectada con el nodo deseado, si el tópico es correcto etc.

Herramienta ICe

Con la implementación de la herramienta como un plug-in y el uso de la plataforma Eclipse para el desarrollo de extensiones se ha conseguido que el proceso de lanzamiento de nodos, visualización de los mismos se realice de manera sencilla.

Desde el punto de vista de funcionalidad, se han cumplido las necesidades que se exigían para la interactuación con sistemas ROS. El trabajo directo de la herramienta sobre los dichos sistemas, gracias a las librerías de Eclipse como ZEST o las de extensión de sus vistas, ha permitido una buena relación de la misma con los sistemas de nodos, permitiendo su modificación hasta cierto punto de manera dinámica.

Conclusiones generales

Se concluye que los objetivos del proyecto se han logrado satisfactoriamente. La herramienta desarrollada además de cumplir los requisitos, resulta útil y cómoda para su manejo. Su uso puede aportar

Además el plug-in está totalmente abierto a su extensión, ya que prácticamente se podría extender en el sentido que se quisiera y/o necesitara. Se podrían incorporar nuevas vistas, nuevos menus, nuevas funciones para interactuar con los nodos ROS desde el entorno de desarrollo y de esta manera comprobar que el nuevo proyecto funciona de la manera deseada.

Gracias al desarrollo de este proyecto, la utilización de estas herramientas y el análisis realizado para el mismo, han aportado un conocimiento muy valioso sobre el papel que los plug-ins, ya sean perspectivas, vistas, PopUpMenus, tienen el Eclipse, como se compone este internamente y de su enorme potencial de extensión.

Capítulo 7

7. Líneas futuras

La herramienta posee un potencial inmenso para seguir acercando al usuario las librerías de ROS e incorporarlas a Eclipse ya que se podrían incorporar tantas nuevas herramientas como fueran necesarias. Analizando la perspectiva ya existente, se podrían incorporar tres funcionalidades a corto plazo y otra línea futura sobre la que seguir trabajando.

La primera funcionalidad a desarrollar a corto podría ser, que la herramienta mostrara el código del nodo lanzado manualmente. La información ya está disponible en cada nueva entidad RosNode que se ha creado, en el campo path, simplemente habría que desarrollar el código de la acción ya creada en el RosExplorer llamada ShowNode Code.

Dentro de *run()* habría que recoger el nombre del objeto sobre el que se ha pulsado y pedir al Map de nodos que devuelva el path del Objeto que al corresponda al nombre. Si se da este path absoluto a un editor de texto adecuado se debería mostrar el código del nodo.

La segunda funcionalidad a desarrollar seria sobre la vista RosTopic. En ella, siguiendo el ejemplo de lo que ya se hace en vista Ros Info, se podría también mostrar la información de cada topic en la Propierties view, con el nodo que lo publica, a que nodo está conectado etc. La tercera de ellas seria incorporar las variables de entorno necesarias para que Eclipse no tuviera que ser lanzado por consola para cargar correctamente las librerías binarias de ROS.

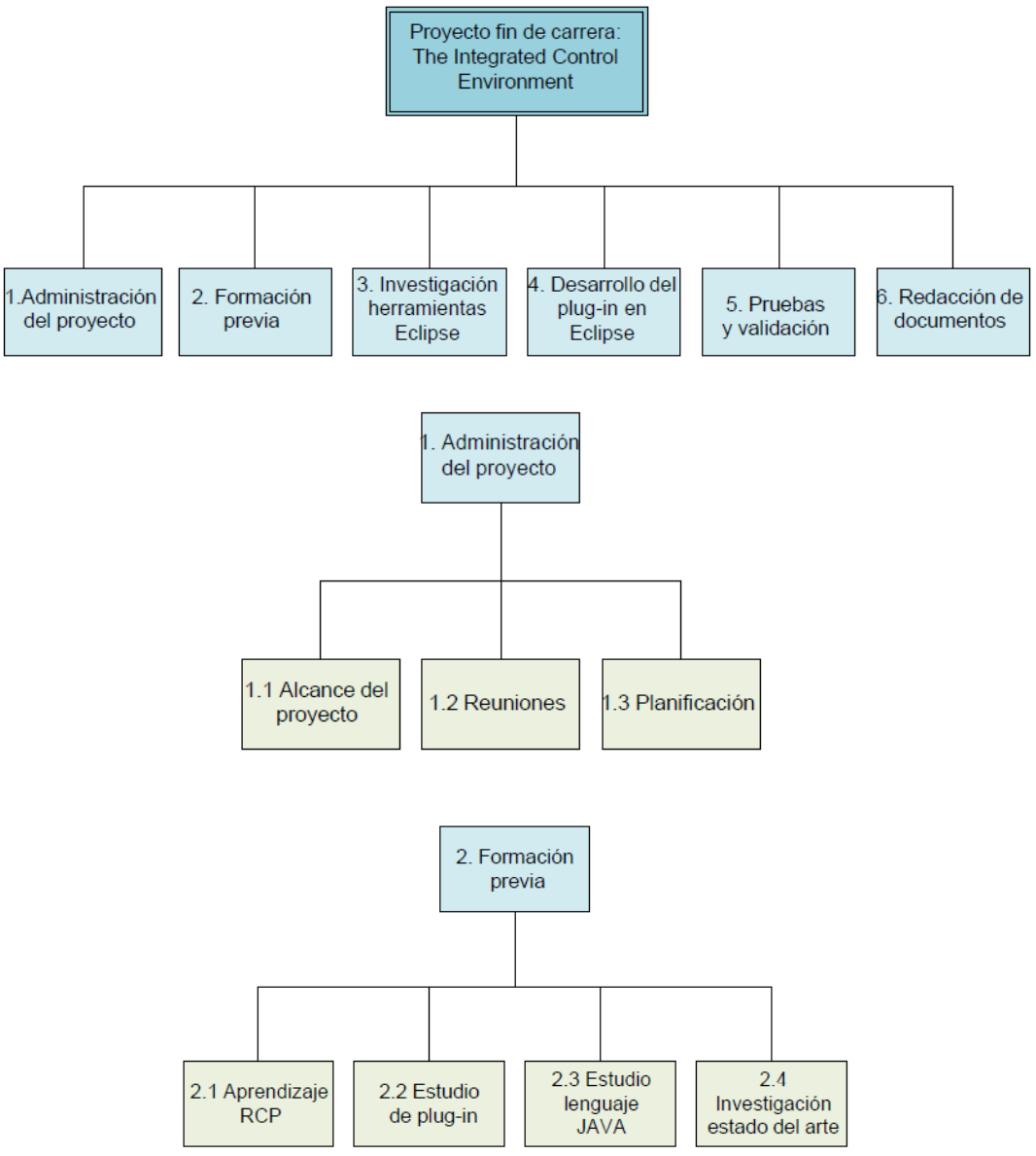
La principal línea futura sobre la que seguir trabajando sería la ampliación de la interacción con la vista ROSgraph. Añadiendo nuevos eventos y acciones a ejecutar cuando el usuario pulsara botones con el ratón.

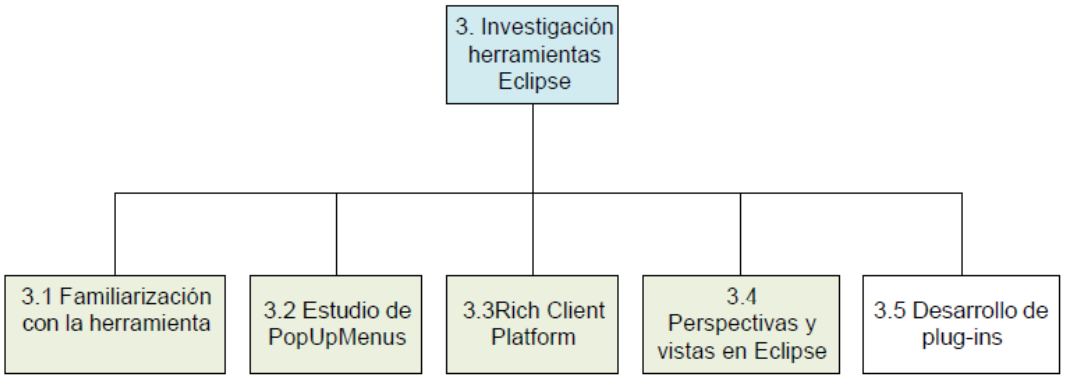
El objetivo real de esta herramienta es acercar al usuario a lo que sucede dentro de un sistema. Es por ello que también sería interesante poder modificar los parámetros del sistema de manera dinámica. Esto podría llegar a realizarse a través de la Properties View. Si el método de la clase ROSNodePropertySource.java estuviera más desarrollado y permitiera modificar los parámetros de los nodos y guardar los cambios, estos nodos deberían a su vez modificar sus parámetros y variar su funcionalidad.

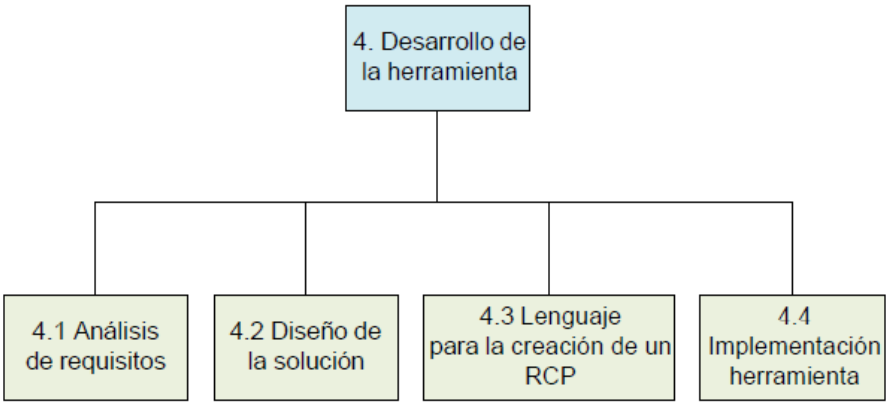
Capítulo 8

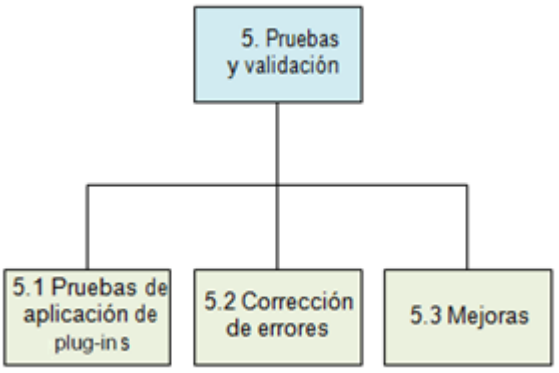
8. Alcance y planificación del proyecto

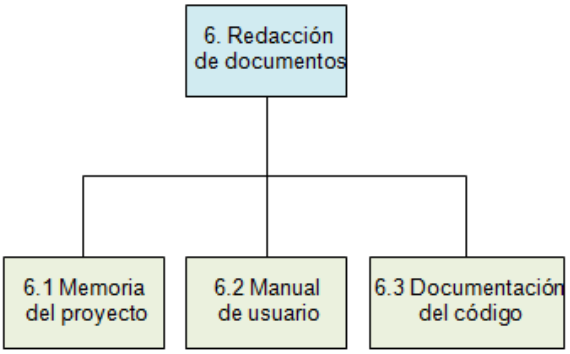
8.1. Estructura de descomposición del trabajo




3. Investigación herramientas Eclipse
3.1 Familiarización con la herramienta
3.2 Estudio de PopUpMenus
3.3Rich Client Platform
3.4 Perspectivas y vistas en Eclipse
3.5 Desarrollo de plug-ins


4. Desarrollo de la herramienta
4.1 Análisis de requisitos
4.2 Diseño de la solución
4.3 Lenguaje para la creación de un RCP
4.4 Implementación herramienta


5. Pruebas y validación
5.1 Pruebas de aplicación de plug-ins
5.2 Corrección de errores
5.3 Mejoras


6. Redacción de documentos
6.1 Memoria del proyecto
6.2 Manual de usuario
6.3 Documentación del código

8.2. Diagrama de Gantt

	Task Name	Duration	Start	Finish	Predecessors
1	Inicio del proyecto	1 day	Thu 22/03/12	Thu 22/03/12	
2	Estudios de la tematica del proyecto	30 days	Fri 23/03/12	Thu 03/05/12	1
3	Aprendizaje RCP	5 days	Fri 23/03/12	Thu 29/03/12	1
4	Estudio de plug-ins	5 days	Fri 30/03/12	Thu 05/04/12	3
5	Estudio lenguaje JAVA	14 days	Fri 06/04/12	Wed 25/04/12	4
6	Investigación estado del arte	6 days	Thu 26/04/12	Thu 03/05/12	5
7	Investigacion herramientas Eclipse	40 days	Fri 04/05/12	Thu 28/06/12	6;2
8	Familiarización con la herramienta	10 days	Fri 04/05/12	Thu 17/05/12	6
9	Estudio PopUp Menu	10 days	Fri 18/05/12	Thu 31/05/12	8
10	Rich Client Platform	5 days	Fri 01/06/12	Thu 07/06/12	9
11	Perspecticas y vistas en eclipse	5 days	Fri 08/06/12	Thu 14/06/12	10
12	Desarrollo de plug-ins	10 days	Fri 15/06/12	Thu 28/06/12	11
13	Desarrollo de la herramienta	60 days	Fri 29/06/12	Thu 20/09/12	12;7
14	Analisis de requisitos	10 days	Fri 29/06/12	Thu 12/07/12	12
15	Diseño de la solucion	10 days	Fri 13/07/12	Thu 26/07/12	14
16	Lenguaje para la creación de un RCP	10 days	Fri 27/07/12	Thu 09/08/12	15
17	Implementación de la herramienta	30 days	Fri 10/08/12	Thu 20/09/12	16
18	Pruebas y validación	5 days	Fri 21/09/12	Thu 27/09/12	17;13
19	Pruebas de aplicación de plug-ins	2 days	Fri 21/09/12	Mon 24/09/12	17
20	Corrección de errores	2 days	Tue 25/09/12	Wed 26/09/12	19
21	Mejoras	1 day	Thu 27/09/12	Thu 27/09/12	20
22	Redacción de documentos	11 days	Fri 28/09/12	Fri 12/10/12	21;18
23	Memoria del proyecto	8 days	Fri 28/09/12	Tue 09/10/12	21
24	Manual de usuario	2 days	Wed 10/10/12	Thu 11/10/12	23
25	Documentación del código	1 day	Mon 15/10/12	Mon 15/10/12	24;22

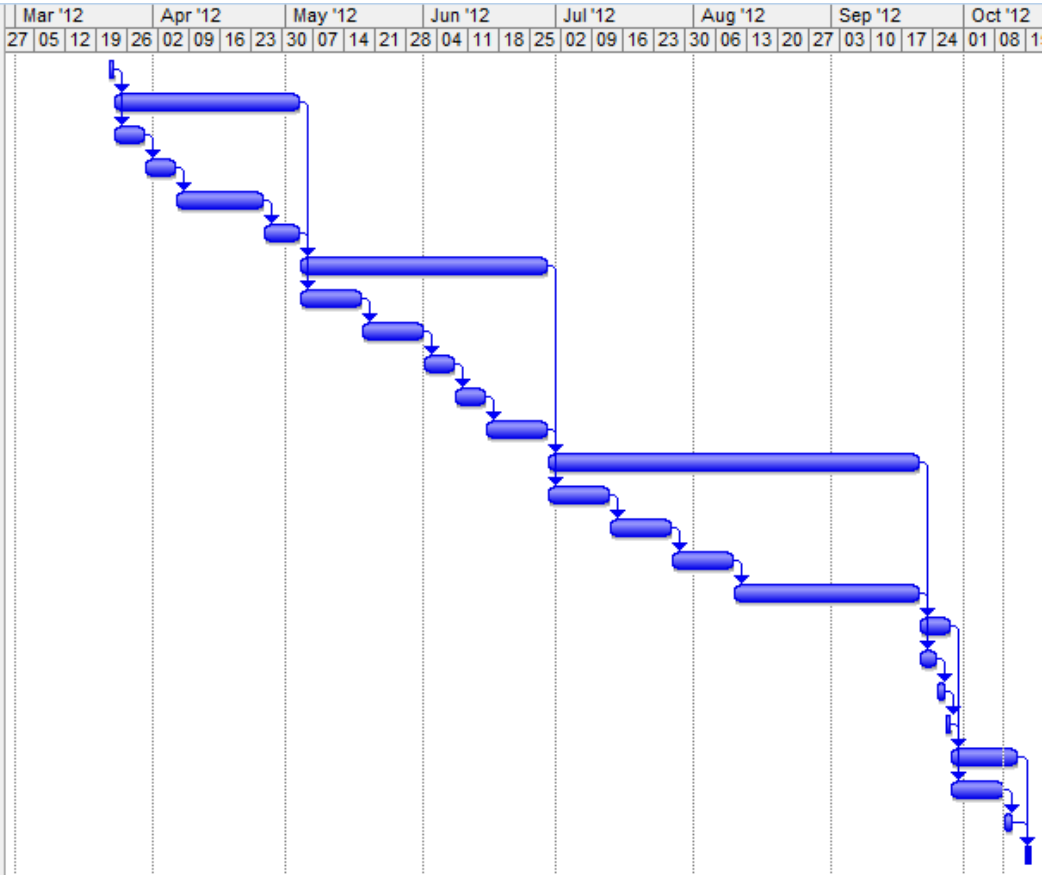


Figura 116 Diagrama de Gantt

BIBLIOGRAFÍA

[1] Julita Bermejo-Alonso, Ricardo Sanz, Manuel Rodríguez y Carlos Hernández. An ontological framework for autonomous systems modeling. Autonomous System Laboratory (ASLab). Universidad Politécnica de Madrid, España.

[2] J. Bermejo-Alonso. "OASys: ontology for autonomous systems". Ph.D. dissertation. E.T.S.I.I.M., Universidad Politécnica de Madrid, 2010.

[3] Jeff McAffer, Erich Gamma, John Wiegand. *The Eclipse Graphical Editing Framework (GEF).* Editorial: Addison-Wesley. Agosto 2011

[4] Jeff McAffer, Paul VanderLey, Simon Archer. *OSGi and Equinox* Editorial: Addison-Wesley. Mayo 2010

[5] Jeff McAffer, Jean-Michel Lemieux, Chris Aniszczyk. *Eclipse Rich Client* Platform. Editorial: Addison-Wesley. Mayo 2010

[6] Eric Clayberg, Dan Rubel. *Eclipse Plug-ins*. Editorial: Addison-Wesley. 2009

[7] Ángel García Beltrán y José María Arranz Santamaría. *Programación orientada a objetos con Java.* E.T.S.I.I.M., Universidad Politécnica de Madrid, 2007.

[8] Bruno R.Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java.* World Wide Series in Computer Science. 2000

[9] Joshua Bloch. *Effective Java*. Editorial: Addison-Wesley. 2008

[10] Steve John Metsker, William C. Wake. *Design Patterns in Java.* Editorial: Addison-Wesley. 2006

[11] Martin Fowler. *Refactoring. Improving the Design of Existing Code*. Editorial: Addison-Wesley. 1999

[12] Getting started with Eclipse. IBM. http://www.ibm.com/developerworks/library/os-ecov/

[13] Tutoriales Vogella. http://www.vogella.com/tutorials.html

http://www.omg.org/

[14] Página de ROS. http://www.ros.org/wiki/

[15] Tutoriales ROS. http://www.ros.org/wiki/ROS/Tutorials

[16] Respuestas en stackoverflow. http://stackoverflow.com/

[17] Wikipedia. http://es.wikipedia.org/wiki/Wikipedia:Portada

[18] Guía de ayuda de Eclipse.

ANEXO 1: Manual de usuario

The Integrated Control Environment

Manual de Usuario

ÍNDICE

Índice de ilustraciones

Sección 1 Introducción

The Integrated Control Environment es un proyecto planteado por el grupo ASLab. Dicho grupo de investigación lleva tiempo estudiando la posibilidad de desarrollar una nueva herramienta cuyo propósito sea crear un entorno de desarrollo y operación para los sistemas ASys.

Esto surge de la necesidad y el deseo de unificar las herramientas de desarrollo y evitar así problemas de incompatibilidad. El objetivo fundamental de este proyecto es el desarrollo de un entorno rico, RCP, basado en Eclipse donde construir nuevos sistemas y manejar los sistemas autónomos desarrollados en proyectos anteriores.

El presente manual muestra al usuario como utilizar esta primera parte del desarrollo de este proyecto ICe. En este primera parte se ha desarrollado una herramienta capaz de interactuar con nodos ROS por medio de varias vistas y una perspectiva Eclipse desarrollada para ASLab. Se supone que el usuario está familiarizado con el sistema operativo Ubuntu, que tiene instalado en su equipo las librerías de ROS en su versión Fuerte y las librerías de ROS Java, que además trabaja con una versión de Eclipse igual o superior a Juno y que tiene instalado el plug-in para eclipse ZEST. También se supone que el usuario está familiarizado con Eclipse y la interacción con sistemas basados en ROS.

Sección 2 Instalación del plug-in

A continuación se explica paso por paso como importar el plug-in e incorporarlo a Eclipse. Para ello deberemos tener un usuario en el repositorio SVN de ASLab. El URL del repositorio es: svn+ssh://software.aslab.upm.es/home/svnroot.

El primer paso es lanzar la herramienta Eclipse desde la consola de comandos. En el caso de estar trabajando con Ubuntu se indicará dentro de que carpeta está el ejecutable de Eclipse y se pulsará enter. Hecho esto arrancará Eclipse.

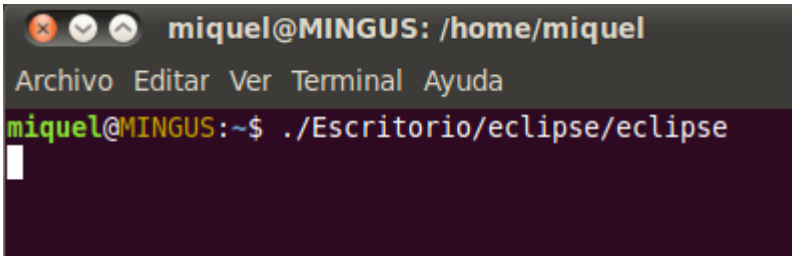


Ilustración 1 Lanzamiento de Eclipse desde el terminal.

Una vez lanzado Eclipse se accederá al repositorio de ASLab. En la vista SVN Repository Exploring se deberá tener configurado el repositorio con el URL que se indica anter. Habrá que acceder a la carpeta donde se encuentra el directorio, que es: /root/people/mgonzalez/Ros_Explorer/Ros_Explorer_jar. Se pincha sobre ella con el botón derecho y se selecciona Export.

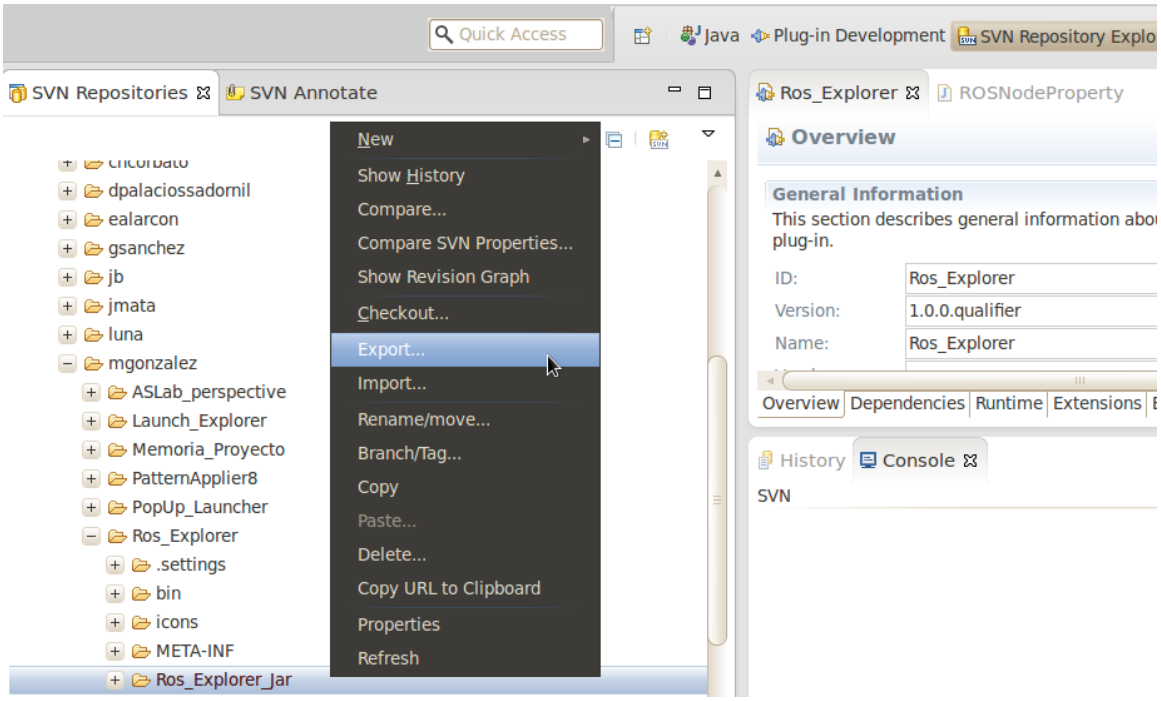


Ilustración 2 Descarga del plug-in del repositorio.

A continuación habrá que seleccionar en que directorio se desea guardar el .jar que contiene al plug-in. En el caso de este tutorial se ha seleccionado el escritorio como se aprecia en las siguientes ilustraciones.

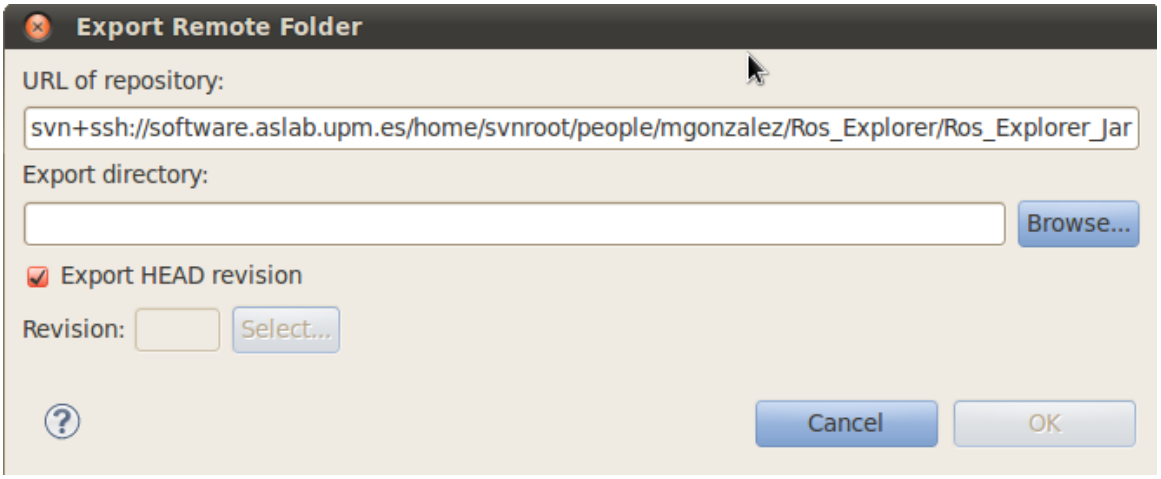


Ilustración 3 Export paso 1.

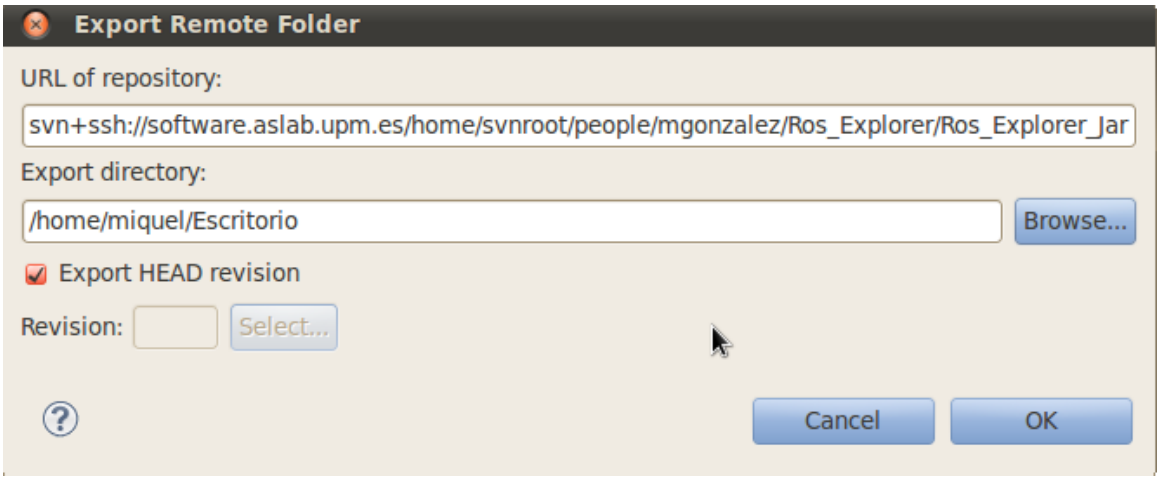


Ilustración 4 Export paso 2.

Al no tratarse de un proyecto de Eclipse no hará falta incorporarlo al espacio de trabajo, sino que habrá que incorporarlo a sus plug-ins. Se recomienda añadirlo a la carpeta de dropins para mantener ordenados que plug-ins son de Eclipse y que plug-ins son propios. Dicho esto, se copia el fichero .jar recientemente descargado y se incluye en la carpeta de dropins donde se encuentra el ejecutable de Eclipse.

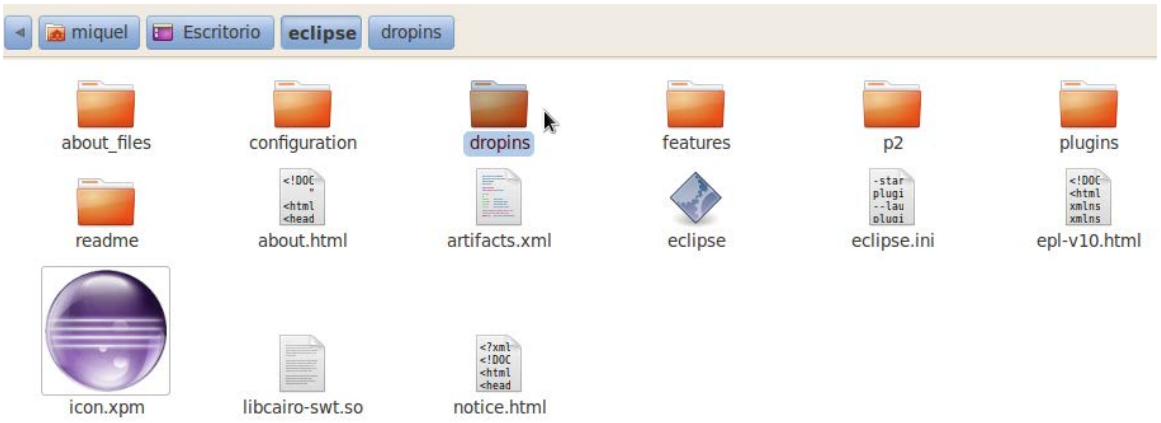


Ilustración 5 Incorporación del plug-in a Eclipse paso 1.

Ilustración 6 Incorporación del plug-in a Eclipse paso 2.

Para actualizar las dependencias entre los plug-ins que incorpora Eclipse al iniciarse será necesario reiniciarlo. Pulsando en File->Restart este se reiniciará incluyendo el contenido recientemente puesto en la carpeta de dropins.

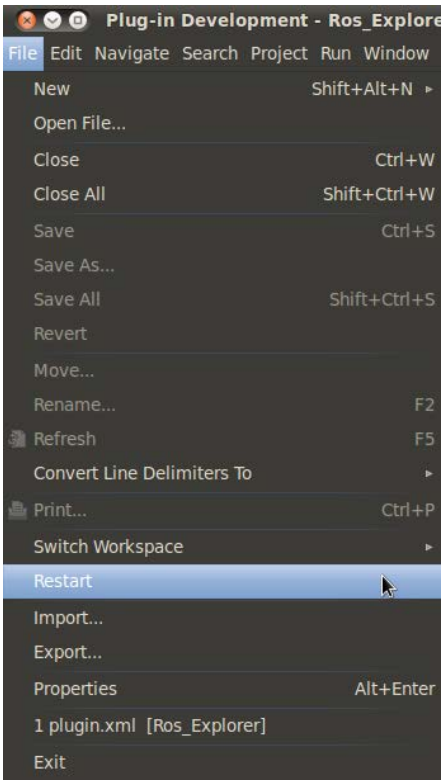


Ilustración 7 Reinicio de Eclipse.

Hecho todo esto el plug-in quedará incorporado a Eclipse.

Sección 3 Herramientas que componen el plug-in

Una vez instalado la herramienta en Eclipse se explicará de qué está compuesta. El plug-in está compuesto principalmente por tres vistas, dos lanzadores de archivos y una perspectiva.

Los lanzadores de archivos no se encuentran directamente visibles entre las herramientas de Eclipse ya que solo aportan nuevas funcionalidades. Estas dos herramientas permitirán al usuario lanzar ficheros con las extensiones *.launch y *.py, pero su lanzamiento será explicado más adelante.

Para acceder a las vistas hay que ir a la pestaña superior de Window->Show View ->Other… Una vez clicado sobre este menú, aparece una nueva carpeta ASLab que contiene las tres vistas incorporadas, Ros Info, Ros system graph y Ros Topic. El comportamiento de las vistas se explica

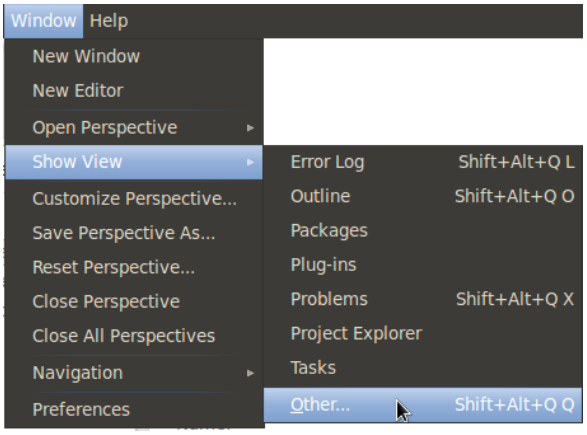


Ilustración 8 Abrir nuevas vistas.

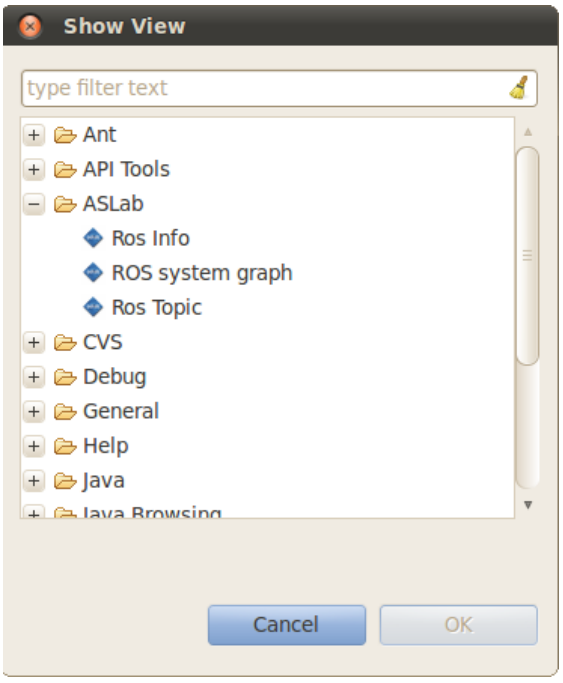


Ilustración 9 Vistas incorporadas.

También está definida una perspectiva ASLab compuesta por estas tres vistas indicadas en la Ilustración 9, y dos vistas más que son el Project Explorer y la vista de Propiedades de Eclipse. Para abrir esta perspectiva hay que clicar en Window->Open Perspective->Other. Entonces se selecciona sobre la ventana de abrir perspectivas la perspectiva ASLab.

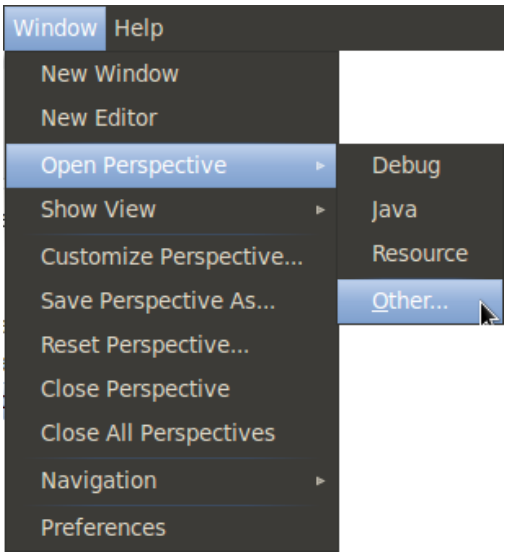


Ilustración 10 Abrir nueva perspectiva ASLab.

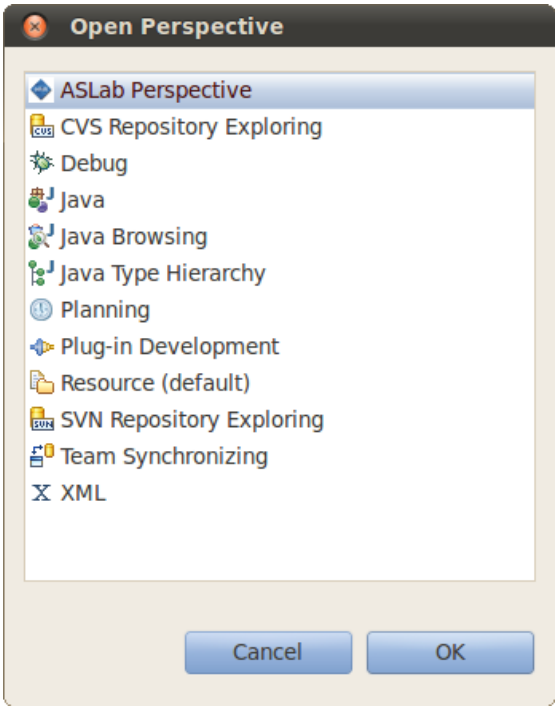


Ilustración 11 Perspectiva incorporada.

La perspectiva ASLab inicialmente estará compuesta por: En su parte superior izquierda se encontrará el Project Explorer, a su izquierda la vista Ros Topic, a la izquierda de esta view la vista de propiedades. En la parte derecha se encontrará la vista de Ros Info, y en la parte inferior izquierda estará la vista Ros system graph para pintar los nodos con sus relaciones.

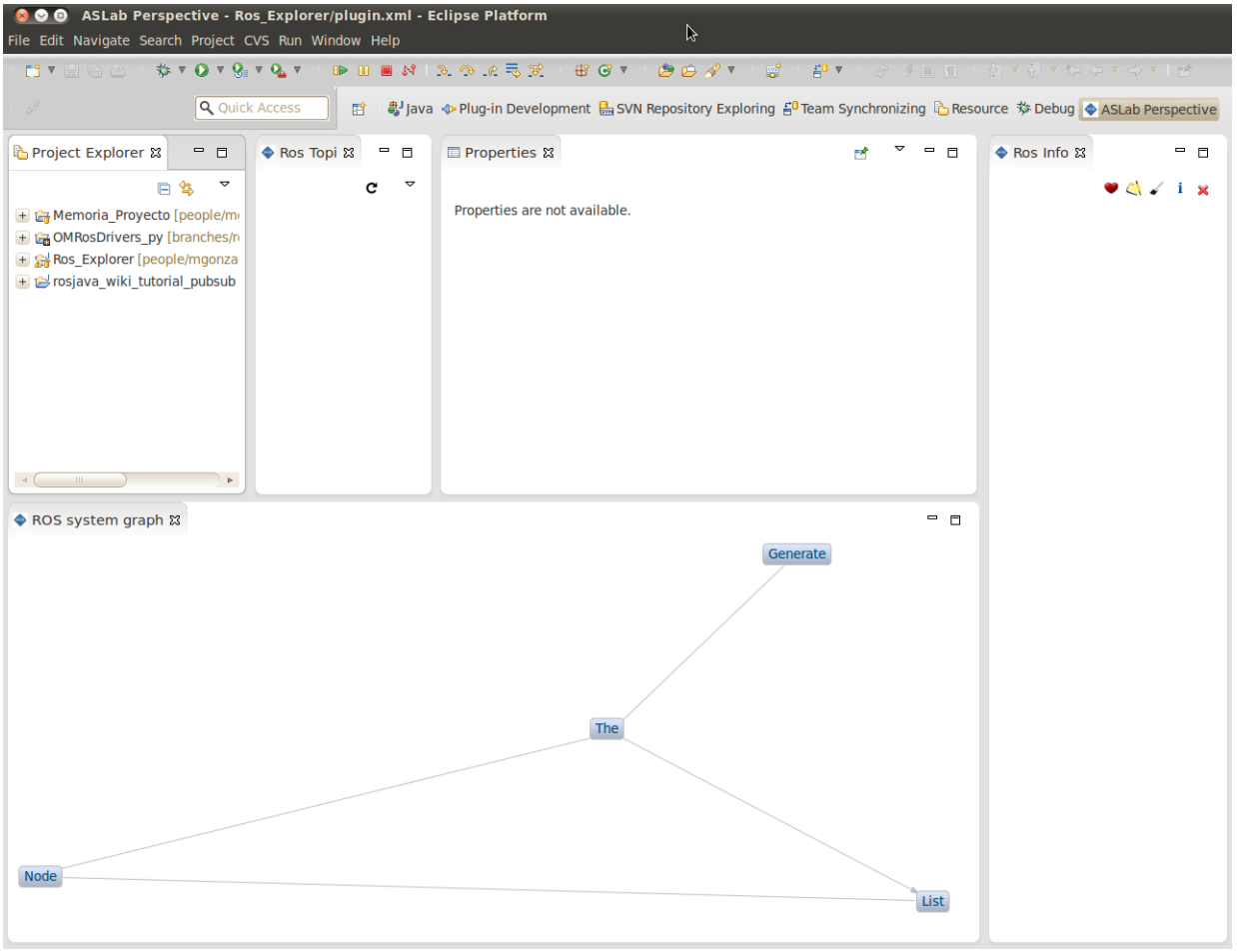


Ilustración 12 Perspectiva inicial ASLab.

Sección 4 Funcionamiento de la herramienta

Una vez explicado de qué está compuesto este plug-in se explicará su funcionamiento. Lo primero que se va a explicar es como lanzar los nodos de un sistema ROS desde Eclipse. Se han estudiado tres posibles casos para el lanzamiento de nodos. El nodo puede estar escrito en diferentes lenguajes de programación, y para esta herramienta se han estudiado tres casos, que el nodo estuviera escrito en Java, en un fichero *.launch en el que podrían ser lanzados varios nodos simultáneamente o en python. Es por esto que en esta herramienta existen **tres maneras de lanzar nodos desde Eclipse** dependiendo del lenguaje en que estén escritos.

La primera forma de lanzar un nodo ROS escrito en Java local es configurando adecuadamente las herramientas externas de Eclipse. Para ello se pulsará en la barra de herramientas superior la pestaña de Run->Run Configurations…

Ilustración 13 Run Configuration en Eclipse.

Pulsando este botón se abre una ventana sobre las configuraciones disponibles en Eclipse. Se selecciona la configuración de aplicaciones Java, Java Application y se crea una nueva clicando botón derecho sobre ella.

Hecho esto se le da un nombre a la nueva configuración y se pone el nombre del proyecto en el que se encuentra la clase Java que contiene al nodo. El proyecto debe estar en el espacio de trabajo de Eclipse para que esta herramienta externa lo pueda localizar correctamente. La clase principal que va a abrir este fichero es org.ros.RosRun, con lo que esto mismo se pone en Main class:

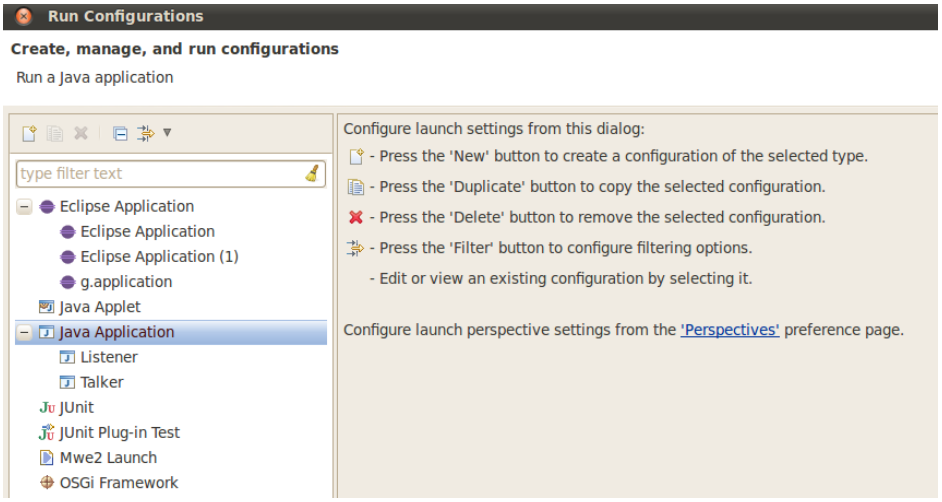


Ilustración 14 Run Configuration paso 1.

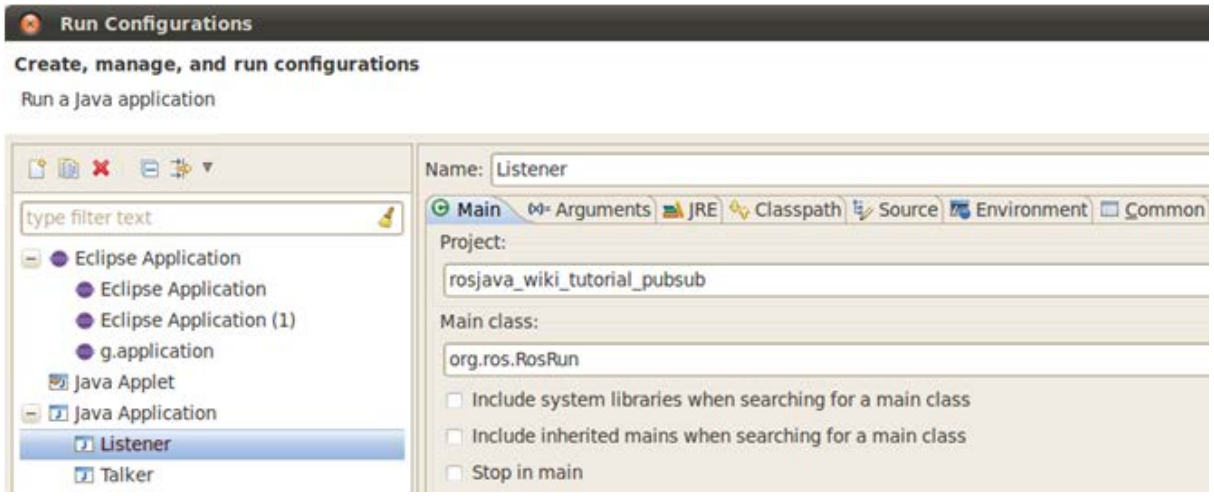


Ilustración 15 Run Configuration paso 2.

Por último, en la pestaña de Arguments, hay que seleccionar el path a partir de la carpeta src del proyecto en donde se encuentra la clase .java si especificar su extensión.

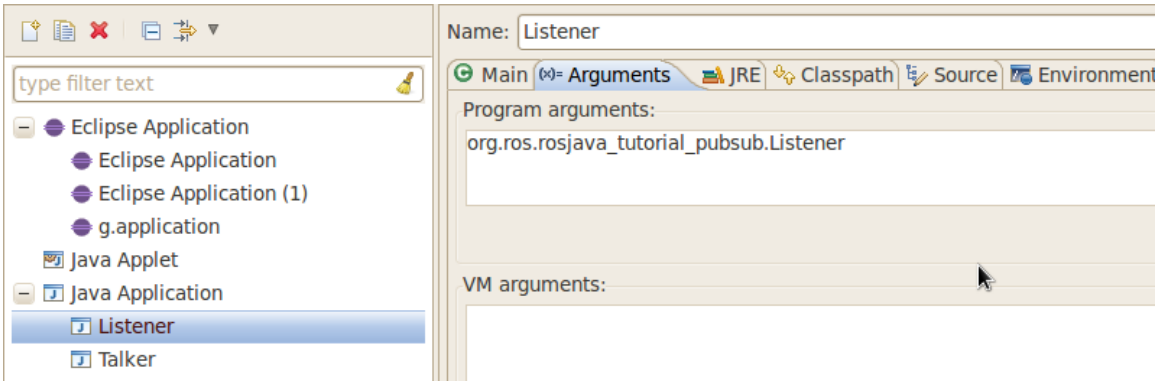


Ilustración 16 Run Configuration paso 3.

Hecho esto cuando se pulsa el botón de acceso rápido Run As, sale esta nueva configuración creada. Si que quiere lanzar este nodo, simplemente se tiene que clicar sobre el nombre de la configuración que se le ha dado y este se lanzará.

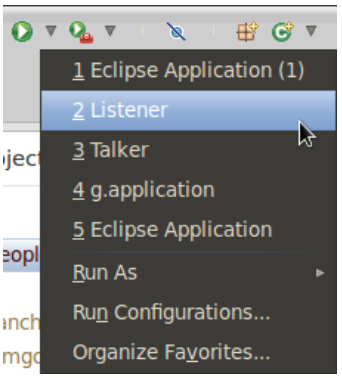


Ilustración 17 Boton Run As....

La segunda opción para lanzar nodos es que el nodo esté escrito en un fichero de lanzamiento con la extensión .launch y que en él se especifique que nodos se van a lanzar. Para este lanzamiento hay que clicar sobre la vista del Project Explorer con el botón derecho sobre el fichero que se desee lanzar y a continuación seleccionar ASLab Launcher->Launch File.

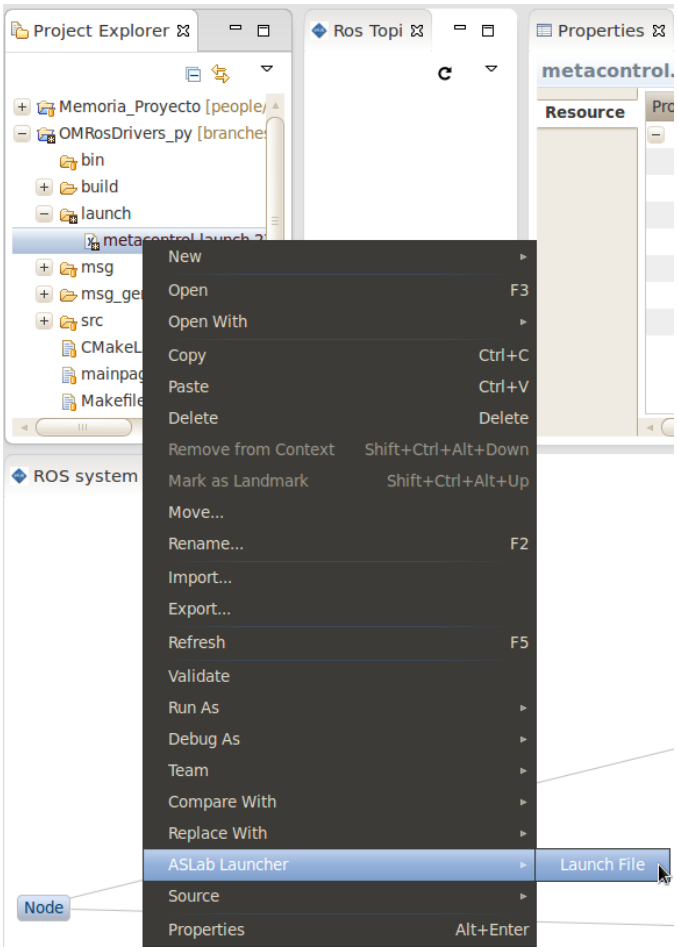


Ilustración 18 Lanzamiento de ficheros *.launch

La tercera opción para lanzar nodos es que este esté escrito en código python en un fichero con la extensión *.py. La acción que se realiza internamente es llamar a la instrucción RosRun de las librerías de ROS, y estas lanzan el nodo del proyecto especificado. Esta instrucción no lanza un RosCore, por lo que deberá que ser lanzado. Para esto se clica en la vista Ros Info sobre el icono correspondiente al RosCore y este se lanzará. Además aparece un mensaje que indica que el RosCore ha sido lanzado.

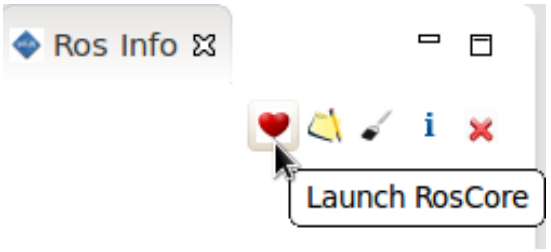


Ilustración 19 Lanzamiento del RosCore

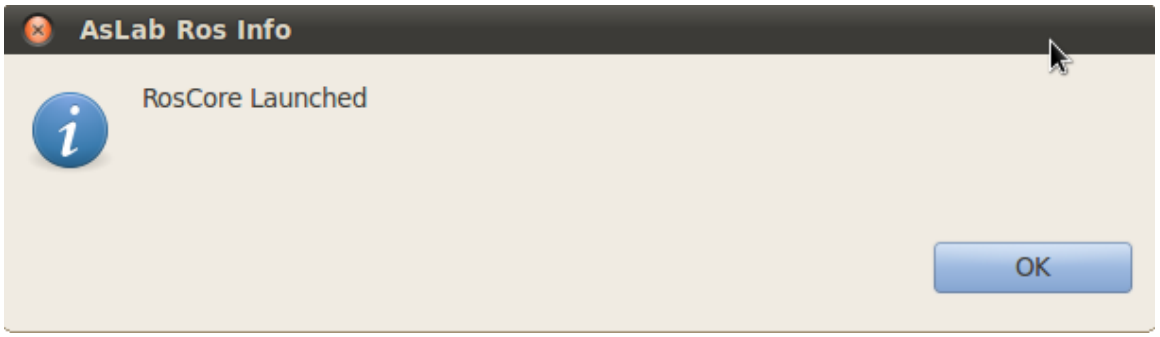


Ilustración 20 Mensaje RosCore lanzado

Una vez lanzado el nodo Master, para continuar con el lanzamiento hay que clicar sobre la vista del Project Explorer con el botón derecho sobre el fichero que se desee lanzar y a continuación seleccionar ASLab Launcher->Launch File.

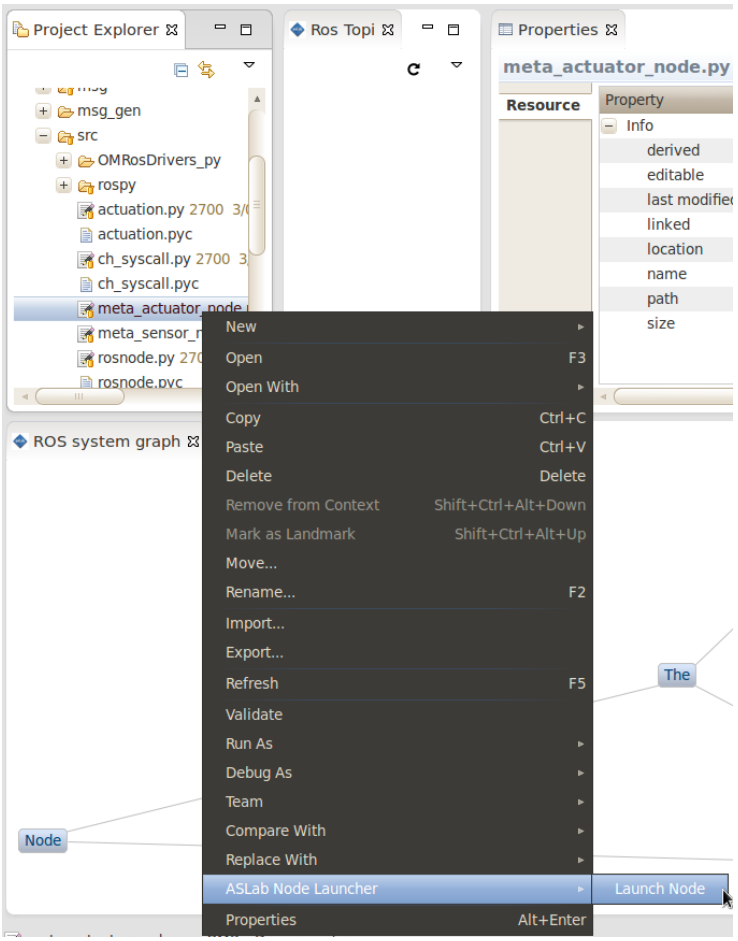


Ilustración 21 Lanzamiento de un fichero *.py

A continuación se va a explicar cómo generar la lista de nodos con su información correspondiente y como actualizar el grafo inicial de nodos. En las siguientes ilustraciones se supodrá que solo está lanzado el nodo Master.

Para generar la lista de nodos se debe clicar sobre la vista Ros Info sobre el icono Node List. Esto generará la lista de nodos dentro de la vista y a su vez actualizará tanto la lista de tópicos de la vista Ros Topic como el grafo con los nodos inicial.

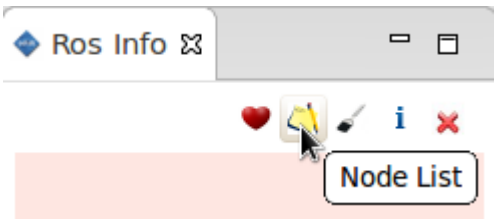


Ilustración 22 Generación de la lista de nodos ROS

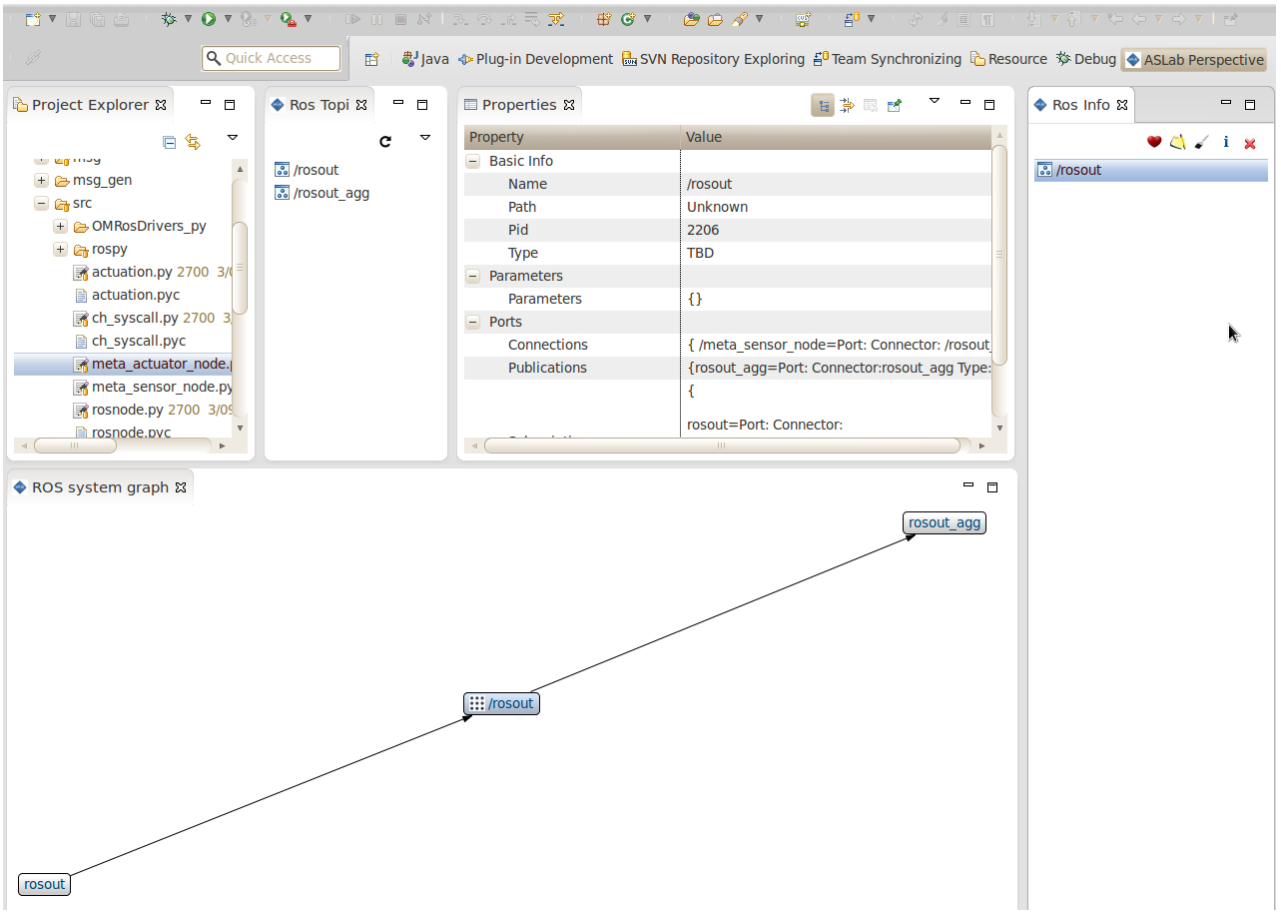


Ilustración 23 Vista con RosCore lanzado

De esta parte en adelante se ha lanzado el fichero metacontrol.launch que se encuentra dentro del proyecto OMRosDrivers_py/launch. Este proyecto ha sido descargado del repositorio de ASLab para Higgs. La dirección de este repositorio es: svn+ssh://software.aslab.upm.es/home/svn_repositories/Higgs, y el proyecto esta concretamente en: branches/ros-fuerte/OMRosDrivers_py. Este fichero contiene dos nodos llamados meta_actuator_node y meta_sensor_node.

Una vez dicho esto, si se deseara lanzar la herramienta externa incorporada en las librerías de ROS Rxgraph habría que pulsar el botón correspondiente. Esta es utilizada para el visionado gráfico de los nodos.

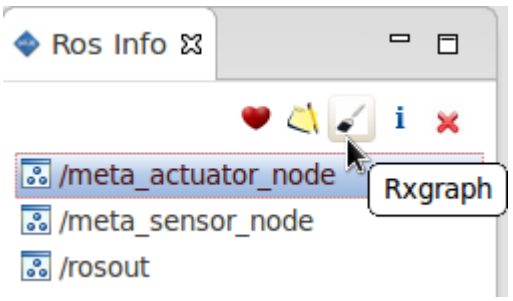


Ilustración 24 Lanzamiento de Rxgraph

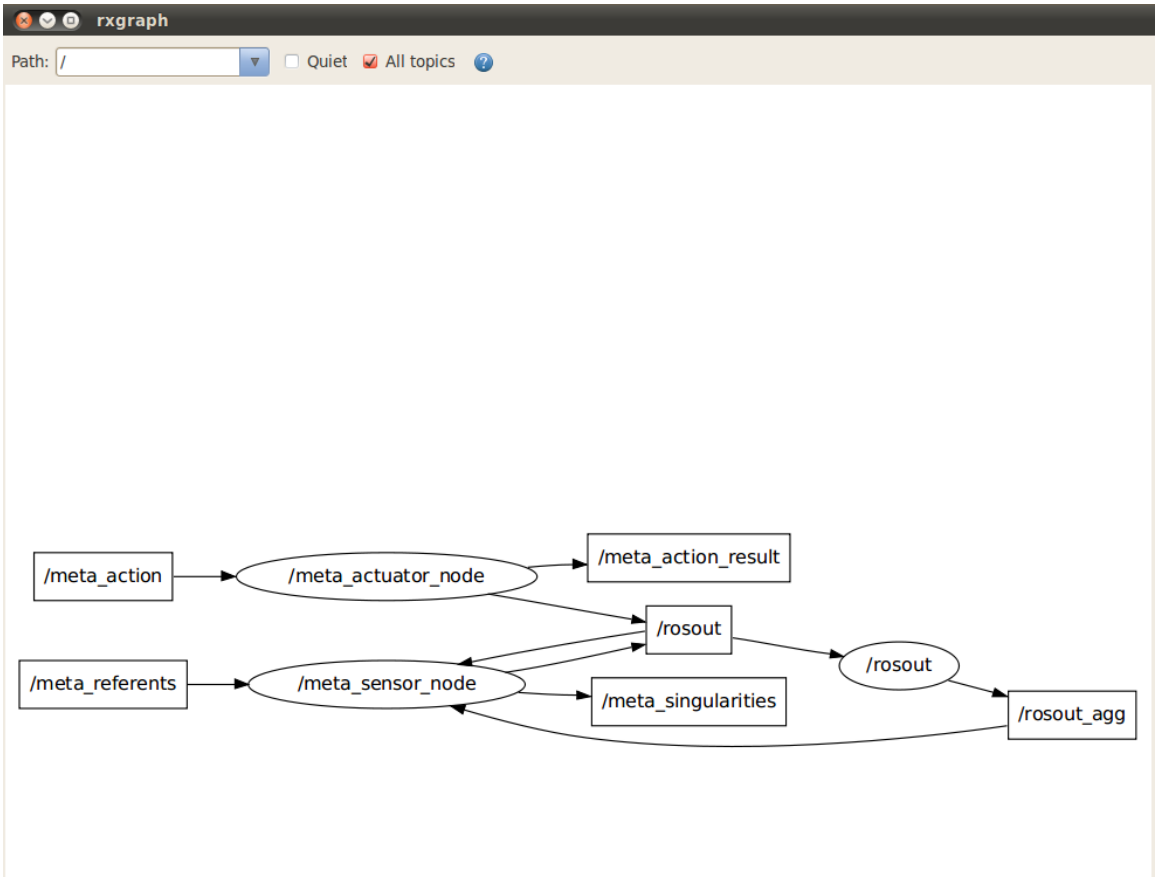


Ilustración 25 Ventana externa Rxgraph

Si se desea ver la información interna de un nodo existen dos posibilidades. La primera de ellas es seleccionar el nodo en la vista Ros Info y clicar sobre el botón Node Info. Esto provocará que salga una ventana externa con la información interna del nodo.

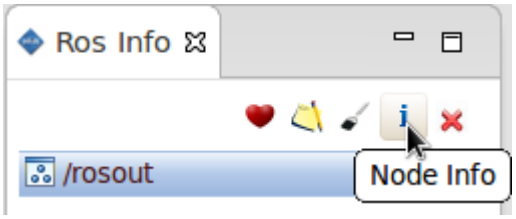


Ilustración 26 Boton Node Info

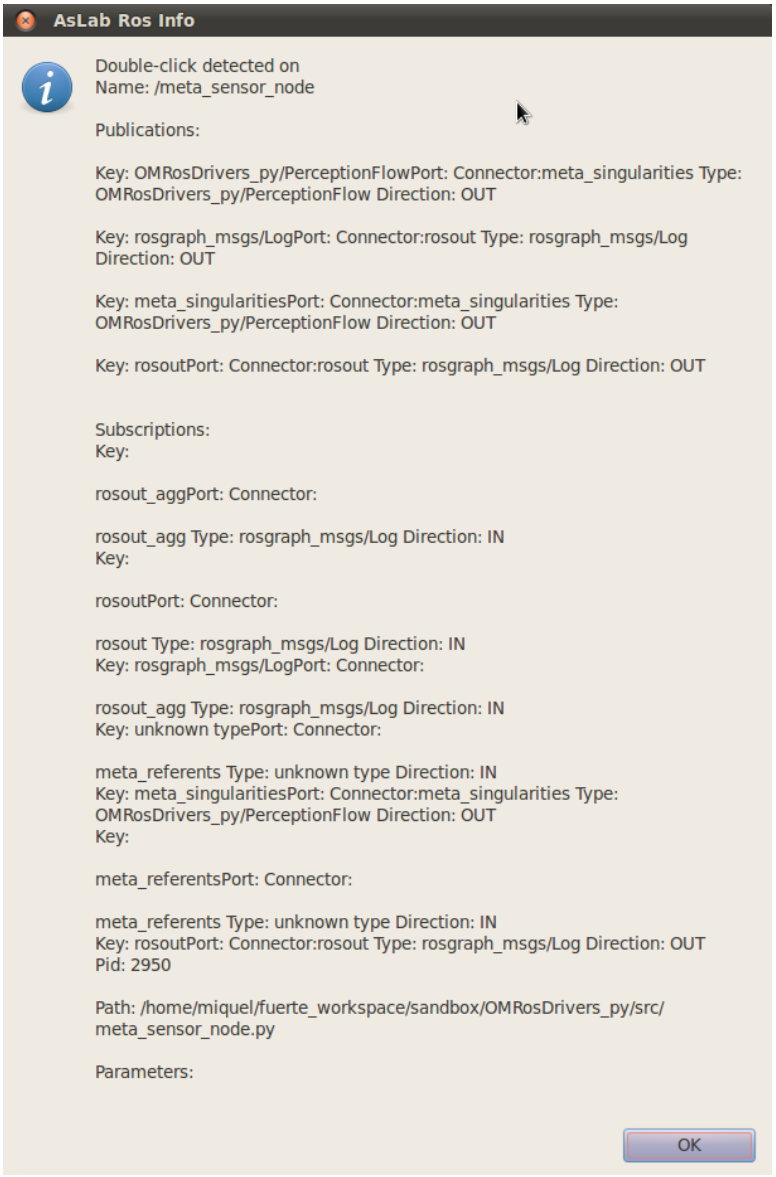


Ilustración 27 Mensaje ASLab con la información del nodo

La segunda opción es clicar con el botón derecho sobre el objeto de la lista Ros Info y elegir la opción de Node Info. La ventana de información será similar a la de la ilustración 27.

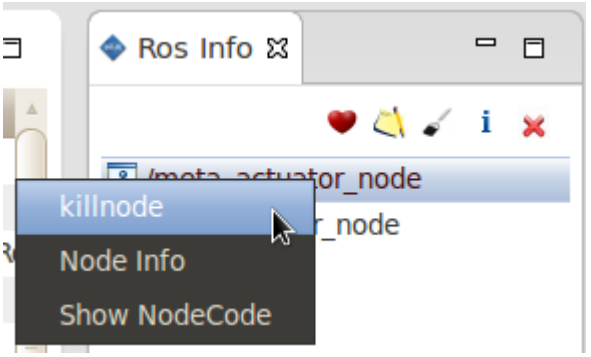


Ilustración 28 PopUp Menu sobre un nodo

Para eliminar un nodo ROS del sistema existen a su vez dos opciones. La primera es clicar el botón killnode una vez seleccionado el objeto. La segunda es clicar con el botón derecho sobre el objeto y seleccionar la opción killnode.

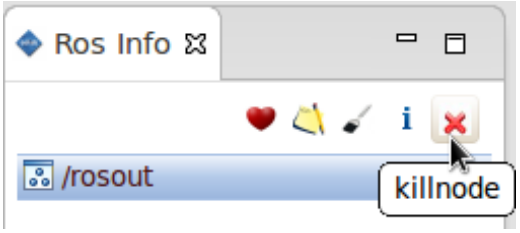


Ilustración 29 Botón killnode

Existe una tercera forma de ver la información del nodo y es por medio de la vista de propiedades. Si se clica con el botón izquierdo sobre la vista Ros Info, en la vista de properties aparece la información del nodo. Aparecen tres campos básicos: La información básica con su nombre, el path de donde esta, el Pid y el tipo. El segundo sería los parámetros del nodo y el tercero sus conexiones.

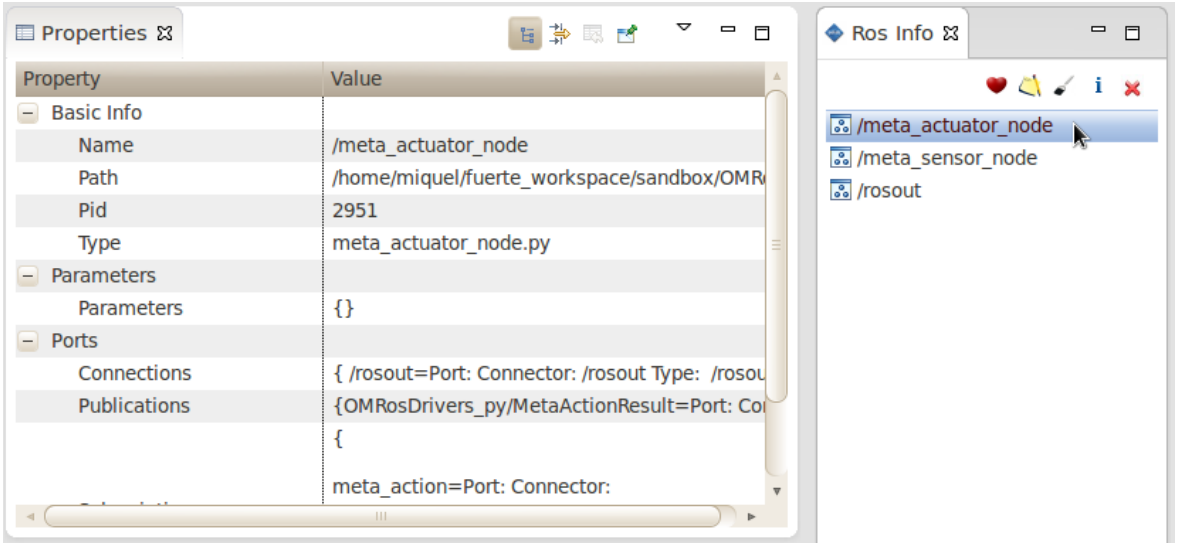


Ilustración 30 Vista Propierties

Por último habría que destacar que la vista en la que se pintan los nodos se actualiza siempre que se presiona el botón de Node List de la vista Ros Info. Una vez pulsado se genera un evento que es leído por la vista y actúa en consecuencia.

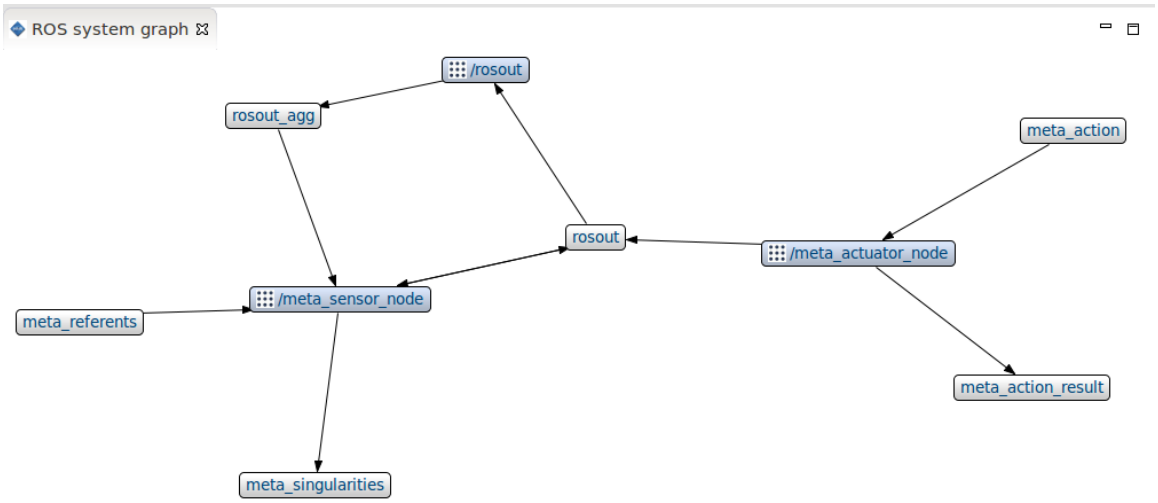


Ilustración 31 Propiedades de un nodo y diagrama de grafos del sistema.

Para finalizar existe una tabla resumen del comportamiento de los botones de la vista Ros Info y de las características de la ventana de propiedades.

Botón	Acción
Launch RosCore	Lanza el RosCore
Node List	Crea la lista de nodos ROS y manda actualizar la lista de tópicos y el grafo de nodos
Rxgraph	Lanza el Rxgraph
Node Info	Lanza una ventana con la información que contiene el nodo
killnode	Elimina del sistema el nodo seleccionado

Ilustración 32 Tabla resumen de Ros Info

Property	Sub-Property	Value
Basic info		
	Name	Nombre del nodo
	Path	Path absoluto del nodo
	Pid	Numero de proceso dentro de la CPU
	Type	Tipo de nodo
Parameters		
	Parameters	Parámetros del nodo
Ports		
	Connections	Conexiones del nodo
	Publications	Publicaciones del nodo
	Subscriptions	Subscripciones del nodo

Ilustración 33 Tabla resumen de la vista de propiedades

Sección 5 Interacción con los modelos del sistema

También existe la posibilidad de interaccionar con los nodos del grafo. Esta posibilidad deja la puerta abierta a nuevas implementaciones ya que por ahora se han desarrollado dos muy simples para comprobar que se podía.

A la hora de interactuar con los nodos existen dos tipos de mensajes. El primero de ellos corresponderá a la información del nodo sobre el que se ha clicado. En el caso de que este sea un nodo surgirá una ventana con su información. En el caso de que el usuario haya clicado sobre un elemento que no fuera un nodo saltaría un mensaje de aviso diciendo que el objeto seleccionado no está en la lista de nodos.

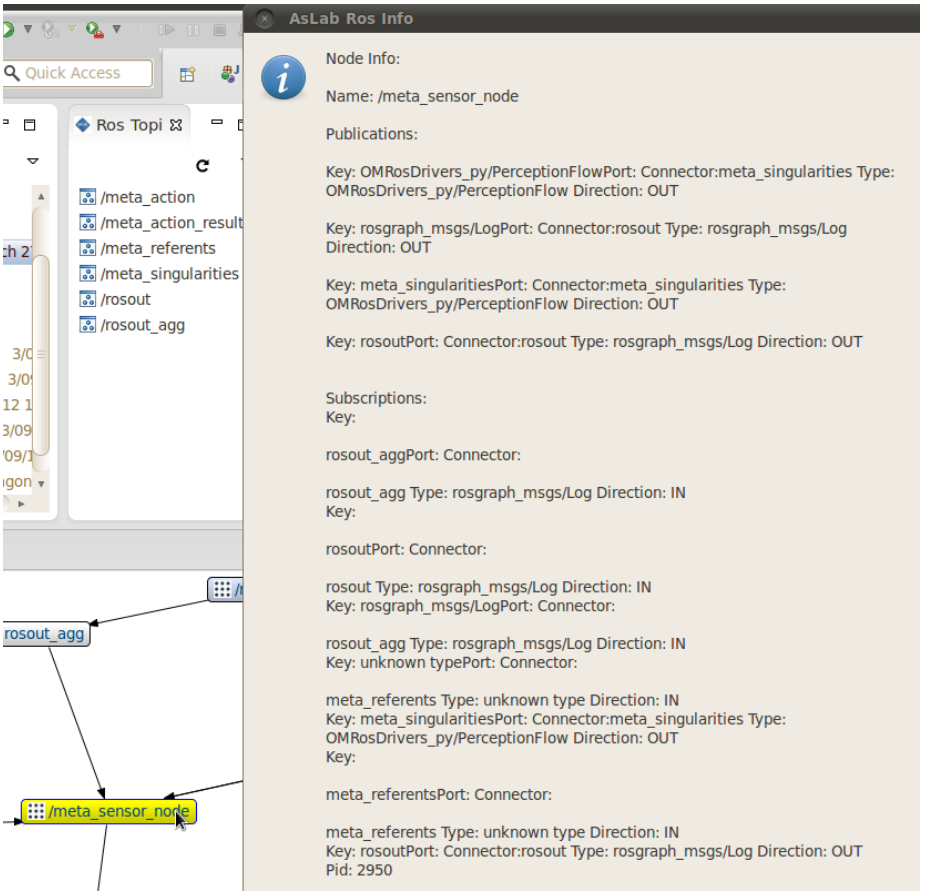


Ilustración 34 Interacción con la herramienta grafica 1.

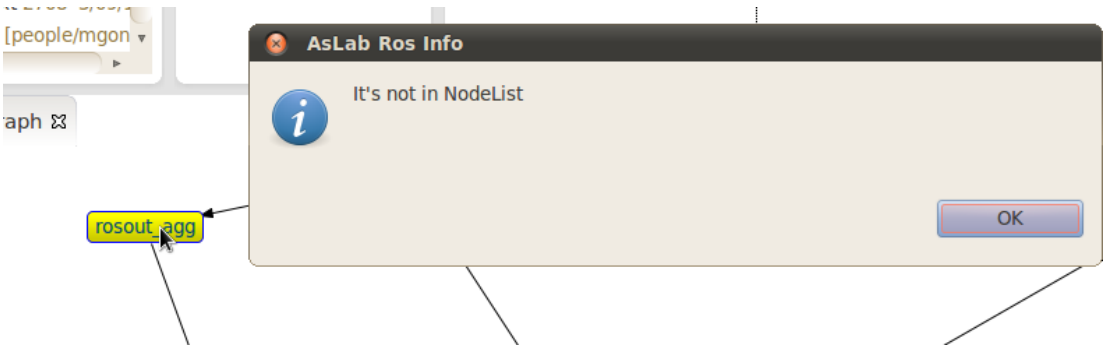


Ilustración 35 Interacción con la herramienta grafica 2.

ANEXO 2: Código de la herramienta

En este segundo anexo se incluirá el código de las clases principales de la herramienta.

Clase Activator.java

```
package org.aslab.asys.ice;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IViewReference;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;
/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

	// The plug-in ID
	public static final String PLUGIN_ID = "Ros_Explorer"; //$NON-NLS-1$

	// The shared instance
	private static Activator plugin;

	/**
	 * The constructor
	 */
	public Activator() {
	}

	/*
	 * (non-Javadoc)
	 * @see
org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.BundleContext)
	 */
	public void start(BundleContext context) throws Exception {
		super.start(context);
		plugin = this;
	}

	/*
	 * (non-Javadoc)
	 * @see
org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext)
	 */
	public void stop(BundleContext context) throws Exception {
		plugin = null;
		super.stop(context);
	}

	/**
	 * Returns the shared instance
	 *
	 * @return the shared instance
	 */
	public static Activator getDefault() {
		return plugin;
	}
```

```
	/**
	 * Returns an image descriptor for the image file at the given
	 * plug-in relative path
	 *
	 * @param path the path
	 * @return the image descriptor
	 */
	public static ImageDescriptor getImageDescriptor(String path) {
		return imageDescriptorFromPlugin(PLUGIN_ID, path);
	}

	// added by Carlos to get one view from another
	public static IViewPart getView(IWorkbenchWindow window, String viewId)
{
	    IViewReference[] refs = window.getActivePage().getViewReferences();
	    for (IViewReference viewReference : refs) {
	        if (viewReference.getId().equals(viewId)) {
	            return viewReference.getView(true);
	        }
	    }
	    return null;
	}
}
```

Carpeta datamodel:

Clase Connect.java

```
package org.aslab.asys.ice.datamodel;

/**
 * @author miquel
 *
 * Elemento que indica una unica conexion del nodo ROS con algun otro elemento
 *
 */
public class Connect {

	/**
	 * Tipo de conexion
	 */
	private String type;

	/**
	 * Nombre del nodo al que estÃ¡ conectado
	 */
	private String node;

	/**
	 * DirecciÃ³n de la conexion
	 */
	private Direction direction;

	/**
	 * Constructor del elemento con su tipo y su conexiÃ³n
	 * @param type tipo de conexion
	 * @param direction Direccion de la conexiÃ³n
	 */
	public Connect(String type, Direction direction) {
		this.setType(type);
		this.setDirection(direction);
	}

	/**
	 Constructor del elemento con su tipo y su conexiÃ³n
	 * @param type tipo de conexion
	 * @param direction Direccion de la conexiÃ³n
	 * @param node Nombre del nodo al que estÃ¡ conectado
	 */
	public Connect(String type,String node, Direction direction) {
		this.setType(type);
		this.setNode(node);
		this.setDirection(direction);
	}

	private void setNode(String node) {
		this.node=node;

	}

	public String getNode(){
		return node;
	}
```

```
	public void setType(String n){
		this.type=n;
	}

	public String getType(){
		return type;
	}

	public void setDirection(Direction direction){
		this.direction=direction;
	}

	public Direction getDirection(){
		return direction;
	}

}
```

Clase Connector.java

```
package org.aslab.asys.ice.datamodel;

import java.util.HashSet;
import java.util.Set;

public class Connector {
	private String name;

	private String type;

	/**
	 * Lista de publicadores
	 */
	private Set<RosNode> publishers = new HashSet<RosNode>();

	/**
	 * Lista de subscriptores
	 */
	private Set<RosNode> subscribers = new HashSet<RosNode>();


	public Connector(String name) {
		super();
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}
```

```
	public Set<RosNode> getPublishers() {
		return publishers;
	}
	public void setPublishers(Set<RosNode> publishers) {
		this.publishers = publishers;
	}
	public Set<RosNode> getSubscribers() {
		return subscribers;
	}
	public void setSubscribers(Set<RosNode> subscribers) {
		this.subscribers = subscribers;
	}


	/**
	 * Añade un subscriptor a la lista
	 * @param node Nodo subscrito
	 */
	public void addSubscriber(RosNode node){
		subscribers.add(node);
		node.setSubscribed(this);
	}
	/**
	 * AÃ±ade un publicador a la lista
	 * @param node Nodo que publica
	 */

	public void addPublisher(RosNode node){
		publishers.add(node);
		node.setPublished(this);
	}
}
```

Clase Direction.java

```
package org.aslab.asys.ice.datamodel;

/**
 * @author miquel
 *     DirecciÃ³nes de una conexion (IN, OUT, INOUT)
 */
public enum Direction {
	IN,
	OUT,
	INOUT
}
```

Clase Port.java

```
package org.aslab.asys.ice.datamodel;

public class Port {

	/**
	 * Conector del puerto
	 */
	private Connector connector;
```

```
	/**
	 * Tipo de la conexiÃ³n del puerto
	 */

	private String type;

	/**
	 * DirecciÃ³n de la conexiÃ³n
	 */

	private Direction direction;

	public Port(){


	}

	public void setConnector(Connector n){
		this.connector=n;
	}

	public Connector getConnector(){
		return connector;
	}

	public void setType(String n){
		this.type=n;
	}

	public String getType(){
		return type;
	}

	public void setDirection(Direction direction){
		this.direction=direction;
	}

	public Direction getDirection(){
		return direction;
	}

	public String toString(){
		return ("\nPort: \n	Connector:"+connector.getName()+"\n
	Type: "+type + "\n	Direction: "+direction) ;
	}
}
```

Clase RosNode.java

```
package org.aslab.asys.ice.datamodel;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
```

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

import org.eclipse.core.resources.IResource;

/**
 * @author miquel
 *     Modelo en java de un nodo ROS con los atributos necesarios para
 *     almacenar su informaciÃ³n.
 */
public class RosNode  {

	/**
	 *  Name of the RosNode
	 */
	private String name;

	/**
	 * SerÃ¡ indefinido hasta que no se lance el nodo correspondiente
	 *  Tipo de nodo ROS
	 */
	private String type="TBD";// TO BE DEFINED

	/**
	 *  SerÃ¡ desconocido hasta que no se lance el nodo correspondiente
	 *  Path del nodo ROS
	 */
	private String path="Unknown";

	/**
	 * Conectores del nodo ROS
	 */

	private Set <Connect> connectors = new HashSet<Connect>();
	/**
	 * Publicaciones del nodo ROS
	 */
	private Map<String, Port> publications= new HashMap<String, Port>();

	/**
	 * Subscripciones del nodo ROS
	 */
	private Map<String, Port> subscriptions= new HashMap<String, Port>();

	/**
	 * Parametros del nodo ROS
	 */
	private Map<String, Object> parameters= new HashMap<String, Object>();

	/**
	 * Process Identifier del nodo ROS dentro del computador
	 */
	private int pid;

	/**
	 * HashMap de Conexiones del nodo ROS
	 */
```

```
	private Map<String, Port> connections= new HashMap<String, Port>();

	public RosNode(){

	}

	/**
	 * Example constructor to build a ROSNode only with a name and a type
	 * @param name
	 * @param type
	 */
	public RosNode(String name, String type) {
		super();
		this.name = name;
		this.type = type;
	}

	/**
	 * @param is InputStream que devuelve el RosNode Info.
	 * @param nam Nombre del nodo
	 * @param aux IResource que define el path del nodo
	 * @throws IOException Excepcion.
	 * Constructor de la clase RosNode. Filtra la informaciÃ³n recibida en
el is y construye
	 * un nodo ROS que aÃ±adir a la lista correspondiente.
	 */
	public RosNode(InputStream is,String nam,IResource aux) throws
IOException{

		String name = aux.getName();

		String path = aux.getLocation().toString();

		this.setType(name);

		this.setPath(path);

		 createNode(is,nam);


	}

	/**
	 * @param is InputStream que devuelve el RosNode Info.
	 * @param nam Nombre del nodo
	 * @throws IOException Posible excepcion
	 * Constructor de la clase RosNode. Filtra la informaciÃ³n recibida en
el is y construye
	 * un nodo ROS que aÃ±adir a la lista correspondiente.
	 */
	public RosNode(InputStream is,String nam) throws IOException{

	  createNode(is,nam);

	}

	/**
	 * @param is InputStream a filtrar
	 * @param nam Nombre del nodo
	 * @throws IOException Posible excepcion
```

```
        * Metodo que filtra la informaciÃ³n y construye el nodo ROS
        */
       public void createNode(InputStream is,String nam) throws IOException{

                 BufferedReader br = new BufferedReader (new InputStreamReader
(is));

               String cad = br.readLine(); //Cadena que leerÃ¡ linea a linea

               this.setName(nam);

               int k=0;

               int i=0;

                String topic=" ";
                     String to=" ";
                     Direction dir = Direction.INOUT;


                while (cad!=null)
                {

                     String c=cad; //Copia de la cadena a filtrar

                     if(c.equals("")){ // No interesa la informaciÃ³n
                           k=0;
                     }


                     if(k==1){ //Es un publicador
                           int  intIni = c.indexOf("/");
                           int  intFin = c.indexOf("[");

                              Port aux = new Port();

                          Connector connector = new
Connector(cad.substring(intIni+1,intFin-1));
                          //Relleno una clase conector con el nombre de arriba
                          aux.setConnector(connector);
                          //Le doy al puerto el connector del connector.

                          Connect aux1= new Connect(cad.substring(intIni+1,intFin-
1),nam,Direction.OUT);

                          connectors.add(aux1);

                          intIni = c.indexOf("[");
                              intFin = c.indexOf("]");
                              aux.setType(cad.substring(intIni+1,intFin));
                              //Establezco el Typo de puerto
                              connector.setType(cad.substring(intIni+1,intFin));
                              //Establezco el Typo de conector



                              connector.addPublisher(this);

                          aux.setDirection(Direction.OUT);
```

```
                        publications.put(aux.getType(), aux);
                        //AÃ±ado el puerto a lista de publicaciones



                    }

                    if(k==2){ //Es un subscriptor

                        int  intIni = c.indexOf("/");
                        int  intFin = c.indexOf("[");

                        Port aux = new Port ();

                        Connector connector = new
Connector(cad.substring(intIni+1,intFin-1));
                       aux.setConnector(connector);

                       Connect aux1= new Connect(cad.substring(intIni+1,intFin-
1),nam,Direction.IN);
                        connectors.add(aux1);

                       intIni = c.indexOf("[");
                        intFin = c.indexOf("]");
                        aux.setType(cad.substring(intIni+1,intFin));

                        connector.setType(cad.substring(intIni+1,intFin));
                        connector.addSubscriber(this);

                       aux.setDirection(Direction.IN);
                        this.setSubscriptions(aux.getType(), aux);
                    }



                    if(k==4){ // Es un puerto

                        int  intIndex2 = cad.indexOf("topic:");
                       if(intIndex2 == - 1){
                       }else{
                        int  intIni1 = c.indexOf("/");
                        topic=cad.substring(intIni1-1);

                       }
                       intIndex2 = cad.indexOf("to:");
                       if(intIndex2 == - 1){
                       }else{
                        int  intIni1 = c.indexOf("/");
                        to=cad.substring(intIni1-1);
                       }

                       intIndex2 = cad.indexOf("inbound");
                       if(intIndex2 == - 1){
                       }else{

                        dir=Direction.IN;


                        i=1;
                       }
```

```
                intIndex2 = cad.indexOf("outbound");
                if(intIndex2 == - 1){
                }else{

                    dir=Direction.OUT;

                    i=1;
                }

             }


             int  intIndex = cad.indexOf("Publications");

            if(intIndex == - 1){

            }else{
              k=1;
            }

            intIndex = cad.indexOf("Subscriptions");

            if(intIndex == - 1){

            }else{
              k=2;
            }


            intIndex = cad.indexOf("Pid");

            if(intIndex == - 1){

            }else{
               k=3;
            }


            intIndex = cad.indexOf("Connections");

            if(intIndex == - 1){

            }else{
               k=4;
            }

            if(k==3){ //Es el numero del proceso interno

                    int  intIni1 = c.indexOf(" ");


this.setPid(Integer.parseInt((cad.substring(intIni1+1))));

                    k=0;
             }

            if(i==1){
             Port aux=new Port();
             Connector connector = new Connector(topic);
             aux.setConnector(connector);
```

```
                connector.setType(to);
                aux.setType(to);
                aux.setDirection(dir);
                this.setConnections(aux.getType(), aux);
                i=0;
               }

               cad = br.readLine();

               }
            this.setParameters(nam);
                                      }



public void setName(String n){
      name=n;
}

public String getName(){
      return name;
}

public void setConnectors(String type, Direction direction){
      Connect aux = new Connect (type,direction);
      connectors.add(aux);
}

public Set<Connect> getConnectors(){
      return connectors;
}

public void setPublications(String name,Port port){
      publications.put(name, port);

}

public Map<String, Port>  getPublications(){
      return publications;
}

public void setSubscriptions(String name,Port port){
      this.subscriptions.put(name, port);

}


public Map<String, Port>  getSubscriptions(){
      return subscriptions;
}

/**
 * @param name Nombre del nodo
 * Funcion que aÃ±ade los parametros del nodo ROS
 */
public void setParameters(String name){

      String cmd = "rosparam list";
```

```
            try{

                  Process p = Runtime.getRuntime().exec(cmd);

              InputStream is = p.getInputStream();

              BufferedReader br = new BufferedReader (new InputStreamReader
(is));

              String aux = br.readLine();

              while (aux!=null)
              {
                   String cad=aux;
                   int  intIndex2 = cad.indexOf(name);
                if(intIndex2 == - 1){
                }else{
                   if(name!=null){
                   parameters.put(cad,name);
                   }
                }
                   aux = br.readLine();


              }
              }

            catch (Exception e)
          {
              //Excepciones si hay algÃºn problema al arrancar el ejecutable o
al leer su salida.
              e.printStackTrace();
          }
      }

      public Map<String, Object>  getParameters(){
            return parameters;
      }

      public void setPid(int pid){
            this.pid=pid;
      }

      public int getPid(){
            return pid;
      }

      public void setPath(String path){
            this.path=path;
      }

      public String getPath(){
            return path;
      }

      public String pathToString(){
            return "\nPath: " +path;
      }
```

```
public void setConnections(String name,Port port){
	this.connections.put(name, port);

}


public Map<String, Port>  getConnections(){
	return connections;
}

public String publicationsToString(){

	Iterator<String> iterator = publications.keySet().iterator();
	String pub = new String();
	pub +="\nPublications: ";

	while(iterator.hasNext()){

		String key = iterator.next().toString();
		String value = publications.get(key).toString();
		pub += "\n\nKey: "+key + value;

	}
	return pub;
}

public String subscriptionsToString(){
	Iterator<String> iterator = subscriptions.keySet().iterator();

	String pub = new String();
	pub +="\nSubscriptions:";

	while(iterator.hasNext()){

		String key = iterator.next().toString();
		String value = subscriptions.get(key).toString();
		pub += "\n\nKey: "+key + value;

	}
	return pub;
}

public String connectorsToString(){

	Iterator<Connect> iterator = connectors.iterator();

	String pub = new String();
	pub +="\nConnectors";

	while(iterator.hasNext()){

		String key = iterator.next().getType();

		pub += "\nConnect: "+key ;

	}
	return pub;
}

public String parametersToString(){
```

```
		Iterator<String> iterator = parameters.keySet().iterator();

		String pub = new String();
		pub +="\nParameters: ";

			if(iterator!=null){

				while(iterator.hasNext()){

					if(iterator!=null){
					String key = iterator.next().toString();
					String value = parameters.get(key).toString();
					pub += "\n\n Key: "+key + "\nValue "+value;
					}

				}

			}
			return pub;
	}

	public String connectionsToString(){
		Iterator<String> iterator = connections.keySet().iterator();

		String pub = new String();
		pub +="\nConnections:";

		while(iterator.hasNext()){

			String key = iterator.next().toString();
			String value = connections.get(key).toString();
			pub += "\nKey: "+key + value;

		}
		return pub;
	}


	public String toString() {


		return "\nName: "+getName() +"\n" + publicationsToString() +"\n"+
subscriptionsToString()
				+ "\nPid: "+ getPid() +" \n "+pathToString() +"\n"+
parametersToString();

	}

	public String getType() {
		return type;
	}

	public void setType(String s) {
		this.type = s;
	}

	public void setSubscribed(Connector connector) {
		Port p = new Port();
		p.setType(connector.getType());
		p.setDirection(Direction.IN);
```

```
            p.setConnector(connector);
            subscriptions.put(connector.getName(), p);
            setSubscriptions(connector.getName(), p);
      }

      public void setPublished(Connector connector) {
            Port p = new Port();
            p.setType(connector.getType());
            p.setDirection(Direction.OUT);
            p.setConnector(connector);
            publications.put(connector.getName(), p);
            setSubscriptions(connector.getName(), p);
      }

}
```

Carpeta popup.actions

Clase Launcher.java

```
package org.aslab.asys.ice.popup.actions;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.core.runtime.Platform;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.IObjectActionDelegate;
import org.eclipse.ui.IWorkbenchPart;


/**
 *
 * @author miquel
 * Clase que indica el comportamiendo que debe realizar el PopUpMenu.
 * Cuando se pulsa ratÃ³n derecho sobre un fichero .launch se despliega este
nuevo menÃº
 */
public class Launcher implements IObjectActionDelegate {

	private Shell shell;

	/**
	 * IResource que contiene el path del proyecto del dichero pulsado
	 */

	private IResource resource = null;

	private IProject project = null;
	/**
	 * Constructor for Action1.
	 */
	public Launcher() {
		super();
	}

	/**
	 * @see IObjectActionDelegate#setActivePart(IAction, IWorkbenchPart)
	 */
	public void setActivePart(IAction action, IWorkbenchPart targetPart) {
		shell = targetPart.getSite().getShell();
	}

	/**
	 * @see IActionDelegate#run(IAction)
	 */
	public void run(IAction action) {

		String cmd = "roslaunch" + " " + project.getName()+ " "+
resource.getName();
```

```
            try{
                  Runtime rt = Runtime.getRuntime();
                  Process p = rt.exec(cmd);}

            catch (Exception e)
        {
            e.printStackTrace();
        }
            System.out.println("Action executed ==> "+cmd+"\n");
      }

      /**
       * FunciÃ³n que realiza una tarea cuando se selecciona un File tipo
*launch.
       * @author miquel
       * @see IActionDelegate#selectionChanged(IAction, ISelection)
       */
      public void selectionChanged(IAction action, ISelection selection) {

            if (selection instanceof IStructuredSelection) {
                  IStructuredSelection ssel = (IStructuredSelection)
selection;
                  Object obj = ssel.getFirstElement();
                  IFile file = (IFile)
Platform.getAdapterManager().getAdapter(obj,
                        IFile.class);
                  if (file == null) {
                      if (obj instanceof IAdaptable) {
                          file = (IFile) ((IAdaptable)
obj).getAdapter(IFile.class);
                      }
                  }
                  if (file != null) {
                       resource = (IResource) file;
                       project = file.getProject();
                  }
               }
      }
                                       }
```

Clase NodeLauncher.java

```
package org.aslab.asys.ice.popup.actions;



import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.aslab.asys.ice.datamodel.RosNode;
import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.core.runtime.Platform;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.util.PropertyChangeEvent;
```

```
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.IObjectActionDelegate;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.ui.IWorkbenchPart;


/**
 * @author miquel
 * Clase que se activa cuando se pulsa con el ratÃ³n derecho sobre un FILE
 * y este tiene extensiÃ³n .py
 */
public class NodeLauncher implements IObjectActionDelegate,
ISelectionListener, IPropertyChangeListener {

	private Shell shell;

	/**
	 * IResource que se envia cuando se produce un evento de tipo
NodeLauncherChange
	 */
	private IResource resource = null;

	/**
	 * Identificador que se lanza a las demas clases cuando se produce un
evento
	 * del tipo NodeLauncherChange
	 */
	public static final String IDNodeLauncherChange = "NodeLauncherChange";

	private IProject project = null;

	/**
	 * Lista de Listeneres a los cambios del NodeLauncher
	 */
	static private List<IPropertyChangeListener> myListeners = new
ArrayList<IPropertyChangeListener>();  // Carlos: para mantener una lista de
listeners de los cambios en el nodemap

	// A public method that allows listener registration
	static public void addPropertyChangeListener(IPropertyChangeListener
listener) {
		if(!myListeners.contains(listener))
			myListeners.add(listener);
	}
	// A public method that allows listener registration
	static public void removePropertyChangeListener(IPropertyChangeListener
listener) {
		myListeners.remove(listener);
	}

	public NodeLauncher() {
		super();
	}

	/**
	 * @see IObjectActionDelegate#setActivePart(IAction, IWorkbenchPart)
	 */
	public void setActivePart(IAction action, IWorkbenchPart targetPart) {
```

```
			shell = targetPart.getSite().getShell();
		}

		/**
		 * @see IActionDelegate#run(IAction)
		 */
		public void run(IAction action) {

			String cmd = "rosrun" + " " + project.getName()+ " "+
resource.getName();

			try{
				Runtime rt = Runtime.getRuntime();
				Process p = rt.exec(cmd);}

			catch (Exception e)
		  {
		      e.printStackTrace();
		  }

			System.out.println("Action executed ==> "+cmd+"\n");

			updateListeners();

		}

		/**
		 * @see IActionDelegate#selectionChanged(IAction, ISelection)
		 */
		public void selectionChanged(IAction action, ISelection selection) {

			 if (selection instanceof IStructuredSelection) {
				IStructuredSelection ssel = (IStructuredSelection)
selection;
				Object obj = ssel.getFirstElement();
				IFile file = (IFile)
Platform.getAdapterManager().getAdapter(obj,
						IFile.class);
				if (file == null) {
				    if (obj instanceof IAdaptable) {
					file = (IFile) ((IAdaptable)
obj).getAdapter(IFile.class);
				    }
				}
				if (file != null) {

					resource = (IResource) file;
					project = file.getProject();
				}
			}

		}
		@Override
		public void propertyChange(PropertyChangeEvent event) {
		}

		@Override
		public void selectionChanged(IWorkbenchPart part, ISelection selection)
{
		}
```

```
	/**
	 * Avisa a las clases subscritas de que se produce un evento
	 */
	private void updateListeners(){
		for (Iterator iter = myListeners.iterator(); iter.hasNext();) {
			IPropertyChangeListener element = (IPropertyChangeListener)
iter.next();
			IResource myFile = resource;
			element.propertyChange(new PropertyChangeEvent(this,
					IDNodeLauncherChange , null , resource ));

		}
	}

}
```

Carpeta views

Clase Perspectiva.java

```
package org.aslab.asys.ice.views;

import org.eclipse.jdt.ui.JavaUI;
import org.eclipse.ui.IFolderLayout;
import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;
import org.eclipse.ui.console.IConsoleConstants;

public class Perspective implements IPerspectiveFactory {

	private IPageLayout factory;

	//IDENTIFICADORES QUE PRODUCEN LOS EVENTOS
	//DE CAMBIO EN LAS CLASES

	private static final String ROS_EXPLORER_VIEW_ID =
			"ros_explorer.views.RosExplorer";

	private static final String ROS_GRAPH_VIEW_ID =
			"ros_explorer.views.ROSgraph";

	private static final String ROS_TOPIC_VIEW_ID =
			"Ros_explorer.views.Rostopic";


	@Override
	public void createInitialLayout(IPageLayout factory) {
		this.factory = factory;
		addViews();
		addActionSets();
		addNewWizardShortcuts();
		addPerspectiveShortcuts();
		addViewShortcuts();
	}

	/**
	 * Situa las vistas dentro de la perspectiva
	 */
	private void addViews() {
		// Creates the overall folder layout.

		IFolderLayout bottom =
				factory.createFolder(
					"bottomRight",
					IPageLayout.RIGHT,
					0.8f,
					factory.getEditorArea());
			bottom.addView(ROS_EXPLORER_VIEW_ID);

		IFolderLayout botLeft =
				factory.createFolder(
					"botLeft",
					IPageLayout.BOTTOM,
					0.45f,
					factory.getEditorArea());
```

```
                botLeft.addView(ROS_GRAPH_VIEW_ID);

        IFolderLayout topLeft =
                        factory.createFolder(
                                "topLeft",
                                IPageLayout.LEFT,
                                0.25f,
                                factory.getEditorArea());
                topLeft.addView(IPageLayout.ID_PROJECT_EXPLORER);


                IFolderLayout midLeft =
                                factory.createFolder(
                                        "midLeft",
                                        IPageLayout.LEFT,
                                        0.25f,
                                        factory.getEditorArea());
                        midLeft.addView(ROS_TOPIC_VIEW_ID);

        IFolderLayout topRight =
                        factory.createFolder(
                                "topRight",
                                IPageLayout.LEFT,
                                1f,
                                factory.getEditorArea());
                topRight.addView(IPageLayout.ID_PROP_SHEET);
        }

        private void addActionSets() {
                factory.addActionSet("org.eclipse.debug.ui.launchActionSet");
                factory.addActionSet("org.eclipse.debug.ui.debugActionSet");
                factory.addActionSet("org.eclipse.debug.ui.profileActionSet");
                factory.addActionSet("org.eclipse.jdt.junit.JUnitActionSet");
                factory.addActionSet("org.eclipse.team.ui.actionSet");
                factory.addActionSet("org.eclipse.team.cvs.ui.CVSActionSet");

        factory.addActionSet("org.eclipse.ant.ui.actionSet.presentation");
                factory.addActionSet(JavaUI.ID_ACTION_SET);
                factory.addActionSet(JavaUI.ID_ELEMENT_CREATION_ACTION_SET);
                factory.addActionSet(IPageLayout.ID_NAVIGATE_ACTION_SET);
        }

        private void addPerspectiveShortcuts() {

        factory.addPerspectiveShortcut("org.eclipse.team.ui.TeamSynchronizingPer
spective");

        factory.addPerspectiveShortcut("org.eclipse.team.cvs.ui.cvsPerspective")
;

        factory.addPerspectiveShortcut("org.eclipse.ui.resourcePerspective");
        }

        private void addNewWizardShortcuts() {

        factory.addNewWizardShortcut("org.eclipse.team.cvs.ui.newProjectCheckout
");

        factory.addNewWizardShortcut("org.eclipse.ui.wizards.new.folder");
                factory.addNewWizardShortcut("org.eclipse.ui.wizards.new.file");
```

```
	}

	private void addViewShortcuts() {
		factory.addShowViewShortcut("org.eclipse.ant.ui.views.AntView");

	factory.addShowViewShortcut("org.eclipse.team.ccvs.ui.AnnotateView");

	factory.addShowViewShortcut("org.eclipse.pde.ui.DependenciesView");
		factory.addShowViewShortcut("org.eclipse.jdt.junit.ResultView");

	factory.addShowViewShortcut("org.eclipse.team.ui.GenericHistoryView");
		factory.addShowViewShortcut(IConsoleConstants.ID_CONSOLE_VIEW);
		factory.addShowViewShortcut(JavaUI.ID_PACKAGES);
		factory.addShowViewShortcut(IPageLayout.ID_RES_NAV);
		factory.addShowViewShortcut(IPageLayout.ID_PROBLEM_VIEW);
		factory.addShowViewShortcut(IPageLayout.ID_OUTLINE);
	}

}
```

Clase RosExplorer.java

```
package org.aslab.asys.ice.views;



import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.aslab.asys.ice.datamodel.*;
import org.aslab.asys.ice.popup.actions.NodeLauncher;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.*;

import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.FileLocator;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Platform;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.util.PropertyChangeEvent;
import org.eclipse.jface.viewers.*;
import org.eclipse.swt.graphics.Image;
import org.eclipse.jface.action.*;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.ui.*;

import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.SWT;
import org.osgi.framework.Bundle;


/**
 * @author miquel
```

```
 *
 * Clase que hereda de ViewPart y que contiene el mapa de nodos ROS con toda
su informaciÃ³n
 * Toda la informaciÃ³n se almacena en esta clase. Esta conectada mediante
listeners con otras clases
 *
 */
public class RosExplorer extends ViewPart implements  ISelectionListener,
IPropertyChangeListener {


	/**
	 * Mapa que almacena los nodos ROS con su nombre en el campo String y
	 * el objeto en su cambo RosNode
	 */
	private Map<String, RosNode> nodemap = new HashMap<String, RosNode>();

	/**
	 * Mapa que almacena los path de los nodos
	 */

	private Map< String, IResource> pathmap = new
HashMap<String,IResource>();


	public static final String ID = "org.aslab.asys.ice.views.RosExplorer";

	//IDENTIFICADORES QUE PRODUCEN LOS EVENTOS
	//DE CAMBIO EN LAS CLASES

	public static final String IDRosExplorerChange = "RosExplorerChange";

	public static final String IDROSGraphChange = "ROSGraphChange";

	public static final String IDNodeLauncherChange = "NodeLauncherChange";


	private TableViewer viewer;

	//Acciones que producen los botones y popupmenus

	private Action Launch_RosCore;
	private Action Node_list;
	private Action Rxgraph;
	private Action killnode;
	private Action nodeinfo;
	private Action nodecode;
	private Action doubleClickAction;

	/**
	 * Lista de Listeneres a los cambios del RosExplorer
	 */
	static private List<IPropertyChangeListener> myListeners = new
ArrayList<IPropertyChangeListener>();  // Carlos: para mantener una lista de
listeners de los cambios en el nodemap

	// A public method that allows listener registration
	static public void addPropertyChangeListener(IPropertyChangeListener
listener) {
		if(!myListeners.contains(listener))
```

```
                myListeners.add(listener);
        }
        // A public method that allows listener registration
        static public void removePropertyChangeListener(IPropertyChangeListener
listener) {
                myListeners.remove(listener);
        }


        class ViewContentProvider implements IStructuredContentProvider {


                public ViewContentProvider(Map<String,RosNode> data){
                        nodemap = data;
                }

                public void inputChanged(Viewer v, Object oldInput, Object
newInput) {
                }

                public void dispose() {
                }

                @Override
                public Object[] getElements(Object parent) {
                        return nodemap.values().toArray();
                }
        }


        class ViewLabelProvider extends LabelProvider implements
ITableLabelProvider {

                public String getColumnText(Object obj, int index) {
                        RosNode node = (RosNode) obj;
                        return node.getName();
                }
                public Image getColumnImage(Object obj, int index) {
                        return getImage(obj);
                }
                public Image getImage(Object obj) {
                        return PlatformUI.getWorkbench().

        getSharedImages().getImage(ISharedImages.IMG_OBJ_ELEMENT);
                }
        }


        class NameSorter extends ViewerSorter {
        }

        public RosExplorer() {
        }


        public void createPartControl(Composite parent) {

                ROSgraph.addPropertyChangeListener(this);
```

```
		// add this view as a selection listener to the workbench page

		getSite().getPage().addSelectionListener(ID, (ISelectionListener)
this);

		NodeLauncher.addPropertyChangeListener(this);

		// add this view as a selection listener to the workbench page

		getSite().getPage().addSelectionListener(ID, (ISelectionListener)
this);


		viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL |
SWT.V_SCROLL);
		ViewContentProvider provider = new ViewContentProvider(nodemap);
		viewer.setContentProvider(provider);
		viewer.setLabelProvider(new ViewLabelProvider());
		viewer.setSorter(new NameSorter());
		getSite().setSelectionProvider(viewer);
		viewer.setInput(nodemap.values().toArray());


		// Create the help context id for the viewer's control

	PlatformUI.getWorkbench().getHelpSystem().setHelp(viewer.getControl(),
"Ros_Explorer.viewer");
		makeActions();
		hookContextMenu();
		hookDoubleClickAction();
		contributeToActionBars();
	}

	private void hookContextMenu() {
		MenuManager menuMgr = new MenuManager("#PopupMenu");
		menuMgr.setRemoveAllWhenShown(true);
		menuMgr.addMenuListener(new IMenuListener() {
			public void menuAboutToShow(IMenuManager manager) {
				RosExplorer.this.fillContextMenu(manager);
			}
		});
		Menu menu = menuMgr.createContextMenu(viewer.getControl());
		viewer.getControl().setMenu(menu);
		getSite().registerContextMenu(menuMgr, viewer);
	}

	private void contributeToActionBars() {
		IActionBars bars = getViewSite().getActionBars();
		fillLocalPullDown(bars.getMenuManager());
		fillLocalToolBar(bars.getToolBarManager());
	}

	private void fillLocalPullDown(IMenuManager manager) {

	}

	private void fillContextMenu(IMenuManager manager) {
		manager.add(killnode);
		manager.add(nodeinfo);
		manager.add(nodecode);
```

```
		manager.add(new
Separator(IWorkbenchActionConstants.MB_ADDITIONS));
	}

	private void fillLocalToolBar(IToolBarManager manager) {
		manager.add(Launch_RosCore);
		manager.add(Node_list);
		manager.add(Rxgraph);
		manager.add(nodeinfo);
		manager.add(killnode);
	}

	private void makeActions() {

		Launch_RosCore = new Action() {
			public void run() {
				try{
					String[] cmd = {"roscore"};

					Runtime rt = Runtime.getRuntime();
					Process p = rt.exec(cmd);}

				catch (Exception e)
			  {
			      e.printStackTrace();
			  }
				showMessage("RosCore Launched");
			}
		};
		Launch_RosCore.setText("Launch RosCore");
		Launch_RosCore.setToolTipText("Launch RosCore");

		Bundle bundle3 = Platform.getBundle("Ros_Explorer");
		ImageDescriptor myImage3 = ImageDescriptor.createFromURL(
			FileLocator.find(bundle3,new Path("icons/roscore.png"),
					null));

		Launch_RosCore.setImageDescriptor(myImage3);



		Node_list = new Action() {
			public void run() {


				CreateNodeList();
				viewer.setInput(null);
				viewer.setInput(rosNodeList());


				System.out.println("");
				System.out.println("Rosnode List Refreshed");
				System.out.println("");

				updateListeners();

			}
		};
		Node_list.setText("Node List");
		Node_list.setToolTipText("Node List");
```

```
		Bundle bundle = Platform.getBundle("Ros_Explorer");
		ImageDescriptor myImage = ImageDescriptor.createFromURL(
			FileLocator.find(bundle,new Path("icons/list.png"),
						null));

		Node_list.setImageDescriptor(myImage);


		Rxgraph = new Action() {
		public void run() {

				try{
					String cmd = "rxgraph";

					Process p3 =
Runtime.getRuntime().exec(cmd);
					}

				 catch (Exception e)
			{
			    //Excepciones si hay algÃºn problema al
arrancar el ejecutable o al leer su salida.*/
				e.printStackTrace();
			}
		       }
		};
		Rxgraph.setText("Rxgraph");
		Rxgraph.setToolTipText("Rxgraph");

		Bundle bundle1 = Platform.getBundle("Ros_Explorer");
		ImageDescriptor myImage1 = ImageDescriptor.createFromURL(
			FileLocator.find(bundle1,new Path("icons/rxgraph.png"),
						null));

		Rxgraph.setImageDescriptor(myImage1);

		nodecode = new Action() {
			public void run() {


			}
		};

		nodecode.setText("Show NodeCode");
		nodecode.setToolTipText("Show NodeCode");


		killnode = new Action() {
			public void run() {

				ISelection selection = viewer.getSelection();
				Object obj =
((IStructuredSelection)selection).getFirstElement();

				int  intIni = (obj.toString()).indexOf("/");
			int  intFin =(obj.toString()).indexOf("P");
				killNode(obj.toString().substring(intIni,intFin-2));

				System.out.println("");
```

```
                        System.out.println("I have killed Node ==>
"+(obj.toString().substring(intIni,intFin-2)));
                        System.out.println("");

                        System.out.println("");
                        System.out.println("Rosnode List Refreshed");
                        System.out.println("");


                }
            };
            killnode.setText("killnode");
            killnode.setToolTipText("killnode");

            Bundle bundlek = Platform.getBundle("Ros_Explorer");
            ImageDescriptor myImagek = ImageDescriptor.createFromURL(
                    FileLocator.find(bundlek,new Path("icons/delete.png"),
                                        null));

            killnode.setImageDescriptor(myImagek);

            nodeinfo = new Action() {
                public void run() {

                        ISelection selection = viewer.getSelection();
                        Object obj =
((IStructuredSelection)selection).getFirstElement();

                        showMessage("Node Info: \n"+obj.toString());



                }

            };
            nodeinfo.setText("Node Info");
            nodeinfo.setToolTipText("Node Info");

      nodeinfo.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().

      getImageDescriptor(ISharedImages.IMG_OBJS_INFO_TSK));

            doubleClickAction = new Action() {
                public void run() {
                        ISelection selection = viewer.getSelection();
                        Object obj =
((IStructuredSelection)selection).getFirstElement();
                        showMessage("Double-click detected on
"+obj.toString());
                }
            };
      }



      private void hookDoubleClickAction() {
            viewer.addDoubleClickListener(new IDoubleClickListener() {
                public void doubleClick(DoubleClickEvent event) {
                        doubleClickAction.run();
```

```
                }
            });
        }

private void showMessage(String message) {
            MessageDialog.openInformation(
                  viewer.getControl().getShell(),
                  "AsLab Ros Info",
                  message);
      }

/* Escucha los eventos que vienen de otras vistas
 * @see
org.eclipse.jface.util.IPropertyChangeListener#propertyChange(org.eclipse.jfac
e.util.PropertyChangeEvent)
 */

public void propertyChange(PropertyChangeEvent event) {
      if( event.getProperty().equals(IDROSGraphChange)) {

            String aux = (String) event.getNewValue();

            Object obj = FindNode(aux);

            if(obj!=null)
            showMessage("Node Info: \n"+obj.toString());
            else
                  showMessage("It's not in NodeList");

      }
if( event.getProperty().equals(IDNodeLauncherChange)) {


      IResource aux = (IResource) event.getNewValue();

      String nameTot = aux.getName();

      int  intFin =(nameTot.toString()).indexOf(".");

      String name = (String) nameTot.subSequence(0, intFin);

      pathmap.put(name, aux);


      }
}

private RosNode FindNode(String name){

      return nodemap.get(name);

}

/**
 *
 * @return rosnode lista de nombres
 * Crea una lista de Strings con los nombres de los nodos ROS activos
 */
private String[] rosNodeList(){
```

```
      int n=0;
      try{
            String cmd = "rosnode list";

            Process p = Runtime.getRuntime().exec(cmd);

        InputStream is = p.getInputStream();

        BufferedReader br = new BufferedReader (new InputStreamReader (is));

        // Se lee la primera linea
        String aux = br.readLine();

        while (aux!=null)
        {
            // y se lee la siguiente.
            aux = br.readLine();
            n++;

        }

      }

      catch (Exception e)
    {
        //Excepciones si hay algÃºn problema al arrancar el ejecutable o al
leer su salida.*/
        e.printStackTrace();
    }

      String [] cad = new String[n];


      try{


            String cmd = "rosnode list";

            Process p = Runtime.getRuntime().exec(cmd);

        InputStream is = p.getInputStream();

        BufferedReader br = new BufferedReader (new InputStreamReader (is));

        String aux = br.readLine();

        int i=0;



        while (aux!=null)
        {

             cad[i]=aux;

            aux = br.readLine();
            i++;
         }
        }
```

```
        catch (Exception e)
    {
       e.printStackTrace();
    }

        return cad;

}



/**
 * Crea el mapa de nodos ROS
 */
public void CreateNodeList() {

        int i;

        String aux[] = rosNodeList();

        for(i=0;i<aux.length;i++){

                nodemap.put(aux[i],createNodeInfo(aux[i]));

        }

}

/**
 * Elimina el nodo ROS con el nombre entregado
 * @param node Nombre del nodo ROS que se desea eliminar
 */
private void killNode(String node){

        String cmd = "rosnode kill " + node;

        try{
                Runtime rt = Runtime.getRuntime();
                Process pk = rt.exec(cmd);}

        catch (Exception e)
    {
      e.printStackTrace();
    }


        CreateNodeList();

        this.nodemap.clear();
        Node_list.run();

}

/**
 *
 * @param node Nombre del nombre a rellenar
 * @return RosNode Nodo ROS rellenado
 * Rellena el nodo ROS y lo devuelve
 */
private RosNode createNodeInfo(String node) {
```

```
        RosNode aux=new RosNode();

        String cmd = "rosnode info " + node;

        try{

              Process p = Runtime.getRuntime().exec(cmd);

          InputStream is = p.getInputStream();

          String nombre = node.substring(1, node.length());

          if(pathmap.get(nombre)!=null){

          aux = new RosNode(is,node,pathmap.get(nombre));

          }
          else{

          aux = new RosNode(is,node);
          }


        }
        catch (Exception e)
    {
          //Excepciones si hay algÃºn problema al arrancar el ejecutable o al
leer su salida.*/
          e.printStackTrace();
    }

        return aux;

}


        public void setFocus() {
              viewer.getControl().setFocus();
        }

        // method to notify the listeners that the map has changed
        private void updateListeners(){
              for (Iterator iter = myListeners.iterator(); iter.hasNext();) {
                    IPropertyChangeListener element = (IPropertyChangeListener)
iter.next();
                    List<RosNode> nodelist = new
ArrayList<RosNode>(nodemap.values());
                    element.propertyChange(new PropertyChangeEvent(this,
                                IDRosExplorerChange , null , nodelist ));

              }
        }
        @Override
        public void selectionChanged(IWorkbenchPart part, ISelection selection)
{

        }

                                        }
```

Clase ROSgraph.java

```
package org.aslab.asys.ice.views;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;


import org.aslab.asys.ice.datamodel.Connect;
import org.aslab.asys.ice.datamodel.Direction;
import org.aslab.asys.ice.datamodel.RosNode;


import org.eclipse.core.runtime.FileLocator;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Platform;
import org.eclipse.draw2d.ColorConstants;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.util.PropertyChangeEvent;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;

import org.eclipse.swt.widgets.Composite;

import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.core.widgets.ZestStyles;
import org.eclipse.zest.layouts.LayoutAlgorithm;
import org.eclipse.zest.layouts.LayoutStyles;
import org.eclipse.zest.layouts.algorithms.SpringLayoutAlgorithm;
import org.eclipse.zest.layouts.algorithms.TreeLayoutAlgorithm;

import org.eclipse.ui.ISelectionListener;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.part.ViewPart;
import org.osgi.framework.Bundle;


/**
 *
 * @author Miquel
 * Clase que hereda de ViewPart y que pinta los nodos ROS en un grafo.
 * Los nodos ROS son de color azul oscuro y llevan el simbolo de ROS
 * y los topicos no llevan simbolo y son color azul claro
 *
 */
public class ROSgraph extends ViewPart implements  ISelectionListener,
IPropertyChangeListener{

            public static final String ID =
"org.aslab.asys.ice.views.ROSgraph";
```

```
	public static final String IDRosExplorerChange =
"RosExplorerChange";

	public static final String IDROSGraphChange = "ROSGraphChange";

	private Graph graph;

	private String NodeName = new String();

	/**
	 * Lista de Listeneres a los cambios del ROSgraph
	 */

	static private List<IPropertyChangeListener> myListeners = new
ArrayList<IPropertyChangeListener>();  // Carlos: para mantener una lista de
listeners de los cambios en el nodemap

	// A public method that allows listener registration
	static public void
addPropertyChangeListener(IPropertyChangeListener listener) {
		if(!myListeners.contains(listener))
			myListeners.add(listener);
	}
	// A public method that allows listener registration
	static public void
removePropertyChangeListener(IPropertyChangeListener listener) {
		myListeners.remove(listener);
	}

	 private int layout = 1;

	List<RosNode> nodes;


	@Override
	public void createPartControl(Composite parent) {

		RosExplorer.addPropertyChangeListener(this);

		// add this view as a selection listener to the workbench
page
		getSite().getPage().addSelectionListener(ID,
(ISelectionListener) this);

		//Create a initial graph
		 graph = new Graph(parent, SWT.NONE);

	    GraphNode node1 = new GraphNode(graph, SWT.NONE, "The");
	    GraphNode node2 = new GraphNode(graph, SWT.NONE, "List");
	    GraphNode node3 = new GraphNode(graph, SWT.NONE, "Node");
	    GraphNode node4 = new GraphNode(graph, SWT.NONE, "Generate");
	    // Lets have a directed connection
	    new GraphConnection(graph, ZestStyles.CONNECTIONS_DIRECTED,
node1,
	        node2);
	    // Lets have a dotted graph connection
	    new GraphConnection(graph, ZestStyles.CONNECTIONS_DOT, node2,
node3);
	    // Standard connection
	    new GraphConnection(graph, SWT.NONE, node3, node1);
```

```
                // Change line color and line width
                GraphConnection graphConnection = new GraphConnection(graph,
SWT.NONE,
                    node1, node4);

                graph.setLayoutAlgorithm(new
SpringLayoutAlgorithm(LayoutStyles.NO_LAYOUT_NODE_RESIZING), true);
                graph.addSelectionListener(new SelectionAdapter() {
                  @Override
                  public void widgetSelected(SelectionEvent e) {

                   String aux = e.toString();

                    int  intIni = aux.indexOf(":");
                   int  intFin = aux.indexOf("detail");

                   NodeName=(aux.substring(intIni+2,intFin-1));

                System.out.println("Node Selected: "+NodeName);

                updateListeners();
                   }
                 });


            }
            @Override
            public void setFocus() {

            }

            public void setLayoutManager() {
                switch (layout) {
                case 1:
                  graph.setLayoutAlgorithm(new
TreeLayoutAlgorithm(LayoutStyles.NO_LAYOUT_NODE_RESIZING), true);
                  layout++;
                  break;
                case 2:
                  graph.setLayoutAlgorithm(new
SpringLayoutAlgorithm(LayoutStyles.NO_LAYOUT_NODE_RESIZING), true);
                  layout = 1;
                  break;

                }
            }


            private LayoutAlgorithm setLayout() {
                      LayoutAlgorithm layout;
                      layout = new
TreeLayoutAlgorithm(LayoutStyles.NO_LAYOUT_NODE_RESIZING);

                      return layout;

            }
```

```
		@Override
		public void selectionChanged(IWorkbenchPart part, ISelection
selection) {

		}
		/* (non-Javadoc)
		 * @see
org.eclipse.jface.util.IPropertyChangeListener#propertyChange(org.eclipse.jfac
e.util.PropertyChangeEvent)
		 */
		@Override
		public void propertyChange(PropertyChangeEvent event) {

			if( event.getProperty().equals(IDRosExplorerChange)) {
				nodes = (List<RosNode>) event.getNewValue(); //Lista
del Rosnode recogida de RosInfo

				graph.clear();

				Map <String,GraphNode> mapa = new
HashMap<String,GraphNode>();

				//Refresh the Graph Node List
				for( Iterator<RosNode> i = nodes.iterator();
i.hasNext();){

					RosNode a = i.next();

					GraphNode aux = new GraphNode(graph,
SWT.NONE,a.getName());

					Bundle bundle =
Platform.getBundle("Ros_Explorer");
					ImageDescriptor myImage =
ImageDescriptor.createFromURL(
							FileLocator.find(bundle,new
Path("icons/ros.png"),
									null));

					aux.setImage(myImage.createImage());

					aux.setBorderColor(ColorConstants.black);

					mapa.put(a.getName(),aux);


					for( Iterator<Connect> j =
a.getConnectors().iterator(); j.hasNext();){

						Connect b = j.next();

						String name= b.getType();

				if(mapa.get(name)==null){

					GraphNode aux2 = new GraphNode(graph,
SWT.NONE,name);
```

```
                                aux2.setBackgroundColor(ColorConstants.white);

                                aux2.setBorderColor(ColorConstants.black);

                                        mapa.put(name,aux2);


      if(b.getDirection()==Direction.IN){
                                                GraphConnection auxgraph;
                                    auxgraph =new GraphConnection(graph,
ZestStyles.CONNECTIONS_DIRECTED, aux2,
                                                 aux);

      auxgraph.setLineColor(ColorConstants.black);

                                }
                                else{
                                      GraphConnection auxgraph;
                                      auxgraph=new GraphConnection(graph,
ZestStyles.CONNECTIONS_DIRECTED,aux ,
                                                aux2);

      auxgraph.setLineColor(ColorConstants.black);

                                }

                               }
                               else{
                                if(b.getDirection()==Direction.IN){
                                      GraphConnection auxgraph;
                                      auxgraph=new GraphConnection(graph,
ZestStyles.CONNECTIONS_DIRECTED,mapa.get(name),
                                                 aux);

      auxgraph.setLineColor(ColorConstants.black);
                                }
                                else{
                                      GraphConnection auxgraph;
                                      auxgraph=new GraphConnection(graph,
ZestStyles.CONNECTIONS_DIRECTED,aux ,
                                                mapa.get(name));

      auxgraph.setLineColor(ColorConstants.black);
                                }

                               }

                                }

                    }

                        graph.setLayoutAlgorithm(new
SpringLayoutAlgorithm(LayoutStyles.NO_LAYOUT_NODE_RESIZING), true);

                    }
                    else {
                          graph.clear();
                          }
                    }
```

```
		/**
		 * Avisa a las clases subscritas de que se produce un evento
		 */

		private void updateListeners(){
			for (Iterator iter = myListeners.iterator();
iter.hasNext();) {
				IPropertyChangeListener element =
(IPropertyChangeListener) iter.next();
				String nodename = NodeName;
				element.propertyChange(new PropertyChangeEvent(this,
						IDROSGraphChange , null , nodename ));

			}
		}

}
```

Clase Rostopic.java

```
package org.aslab.asys.ice.views;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Map;


import org.eclipse.core.runtime.FileLocator;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Platform;
import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.util.IPropertyChangeListener;
import org.eclipse.jface.util.PropertyChangeEvent;
import org.eclipse.jface.viewers.DoubleClickEvent;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.jface.viewers.ViewerSorter;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.ui.IActionBars;
import org.eclipse.ui.ISelectionListener;
```

```
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;
import org.osgi.framework.Bundle;



/**
 * @author miquel
 *     Clase que hereda de ViewPart y que pinta en una lista los topicos de
 *     los nodos ROS activos en tu sistema.
 */
public class Rostopic extends ViewPart implements  ISelectionListener,
IPropertyChangeListener {

	public static final String ID = "org.aslab.asys.ice.views.Rostopic";

	private TableViewer viewer;
	private Action refresh;
	private Action doubleClickAction;

	public static final String IDRosExplorerChange = "RosExplorerChange";

	private Map<String, String> topicmap = new HashMap<String, String>();

	class ViewContentProvider implements IStructuredContentProvider {

		public void inputChanged(Viewer v, Object oldInput, Object
newInput) {
		}
		public void dispose() {
		}

		public Object[] getElements(Object parent) {
			return topicmap.values().toArray();
		}
	}
	class ViewLabelProvider extends LabelProvider implements
ITableLabelProvider {
		public String getColumnText(Object obj, int index) {
			return getText(obj);
		}
		public Image getColumnImage(Object obj, int index) {
			return getImage(obj);
		}
		public Image getImage(Object obj) {
			return PlatformUI.getWorkbench().

	getSharedImages().getImage(ISharedImages.IMG_OBJ_ELEMENT);
		}
	}
	class NameSorter extends ViewerSorter {
	}

	/**
	 * The constructor.
	 */
	public Rostopic() {
```

```
	}

	/**
	 * This is a callback that will allow us
	 * to create the viewer and initialize it.
	 */
	public void createPartControl(Composite parent) {

		//Se subscribe a los cambios de RosExplorer
		RosExplorer.addPropertyChangeListener(this);

		// add this view as a selection listener to the workbench page

		getSite().getPage().addSelectionListener(ID, (ISelectionListener)
this);


		viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL |
SWT.V_SCROLL);
		viewer.setContentProvider(new ViewContentProvider());
		viewer.setLabelProvider(new ViewLabelProvider());
		viewer.setSorter(new NameSorter());
		viewer.setInput(getViewSite());

		// Create the help context id for the viewer's control

	PlatformUI.getWorkbench().getHelpSystem().setHelp(viewer.getControl(),
"Rostopic.viewer");
		makeActions();
		hookContextMenu();
		hookDoubleClickAction();
		contributeToActionBars();
	}

	private void hookContextMenu() {
		MenuManager menuMgr = new MenuManager("#PopupMenu");
		menuMgr.setRemoveAllWhenShown(true);
		menuMgr.addMenuListener(new IMenuListener() {
			public void menuAboutToShow(IMenuManager manager) {
				Rostopic.this.fillContextMenu(manager);
			}
		});
		Menu menu = menuMgr.createContextMenu(viewer.getControl());
		viewer.getControl().setMenu(menu);
		getSite().registerContextMenu(menuMgr, viewer);
	}

	private void contributeToActionBars() {
		IActionBars bars = getViewSite().getActionBars();
		fillLocalPullDown(bars.getMenuManager());
		fillLocalToolBar(bars.getToolBarManager());
	}

	private void fillLocalPullDown(IMenuManager manager) {
		//manager.add(action1);
		manager.add(new Separator());

	}

	private void fillContextMenu(IMenuManager manager) {
```

```
			//manager.add(action1);
		//	manager.add(action2);
			// Other plug-ins can contribute there actions here
			manager.add(new
Separator(IWorkbenchActionConstants.MB_ADDITIONS));
		}

		private void fillLocalToolBar(IToolBarManager manager) {
			manager.add(refresh);
			//manager.add(action2);
		}

		private void makeActions() {
			refresh = new Action() {
				public void run() {
					//viewer.refresh();
					createTopicList();
					viewer.refresh();
					System.out.println("RosNode Topic Refreshed");
				}
			};
			refresh.setText("Refresh");
			refresh.setToolTipText("Refresh");


			Bundle bundle = Platform.getBundle("Ros_Explorer");
			ImageDescriptor myImage = ImageDescriptor.createFromURL(
				FileLocator.find(bundle,new Path("icons/refresh.png"),
					null));

			refresh.setImageDescriptor(myImage);



			doubleClickAction = new Action() {
				public void run() {
					ISelection selection = viewer.getSelection();
					Object obj =
((IStructuredSelection)selection).getFirstElement();
					showMessage("Double-click detected on
topic=>"+obj.toString());
				}
			};
		}

		private void hookDoubleClickAction() {
			viewer.addDoubleClickListener(new IDoubleClickListener() {
				public void doubleClick(DoubleClickEvent event) {
					doubleClickAction.run();
				}
			});
		}
		private void showMessage(String message) {
			MessageDialog.openInformation(
				viewer.getControl().getShell(),
				"Sample View",
				message);
		}

		/**
```

```
         * Passing the focus request to the viewer's control.
         */
        public void setFocus() {
              viewer.getControl().setFocus();
        }

        public void createTopicList(){

              try{

                    topicmap.clear();

                    String cmd = "rostopic list";

                    Process p = Runtime.getRuntime().exec(cmd);

              InputStream is = p.getInputStream();

              BufferedReader br = new BufferedReader (new InputStreamReader
(is));

              String aux = br.readLine();

              while (aux!=null)
              {
               String cad=aux;

               topicmap.put(cad,cad);

                  aux = br.readLine();
               }

               }

               catch (Exception e)
          {
              e.printStackTrace();
          }
        }

        @Override
        public void propertyChange(PropertyChangeEvent event) {

              if( event.getProperty().equals(IDRosExplorerChange)) {
                    refresh.run();
                    }
              }

        @Override
        public void selectionChanged(IWorkbenchPart part, ISelection selection)
{
              // TODO Auto-generated method stub

        }
                                        }
```

Carpeta views.properties

Clase PortPropertySource.java

```
package org.aslab.asys.ice.views.properties;

import org.aslab.asys.ice.datamodel.Port;
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.ui.views.properties.TextPropertyDescriptor;




/**
 * @author miquel
 *     Clase que hereda IPropertySource para que se muestre
 *en la vista de propiedades las caracteristicas de un puerto de un nodo ROS
 */
public class PortPropertySource implements IPropertySource {

	private final Port node;

	public PortPropertySource(Port node) {
		this.node = node;
	}

	@Override
	public Object getEditableValue() {
		return this;
	}

	@Override
	public IPropertyDescriptor[] getPropertyDescriptors() {

		TextPropertyDescriptor conProp=new
TextPropertyDescriptor("connector", "Connector");
		TextPropertyDescriptor typeProp =new
TextPropertyDescriptor("type", "Type");
		TextPropertyDescriptor dirProp =new
TextPropertyDescriptor("direction", "Direction");

		IPropertyDescriptor[] propertyDescriptors = new
IPropertyDescriptor[]{
				conProp, typeProp,dirProp
		};

		return propertyDescriptors;

	}

	@Override
	public Object getPropertyValue(Object id) {
	    if (id.equals("connector")) {
	        return node.getConnector();
	    }
	    if (id.equals("type")) {
	       return node.getType();
	    }
```

```
            if (id.equals("direction")) {
               return node.getDirection();
            }

            return null;
        }


        @Override
        public boolean isPropertySet(Object id) {
              return false;
        }

        @Override
        public void resetPropertyValue(Object id) {
              // TODO Auto-generated method stub
        }

        @Override
        public void setPropertyValue(Object id, Object value) {
              String s = (String) value;

            if (id.equals("direction")) {
            }

            if (id.equals("type")) {
              node.setType(s);
            }

            if (id.equals("connector")) {
               }
        }

}
```

Clase ROSNodePropertySource.java

```
package org.aslab.asys.ice.views.properties;

import java.util.ArrayList;
import java.util.List;

import org.aslab.asys.ice.datamodel.RosNode;
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.ui.views.properties.PropertyDescriptor;
import org.eclipse.ui.views.properties.TextPropertyDescriptor;


public class ROSNodePropertySource implements IPropertySource {

        private final RosNode node;

        private transient List < IPropertyDescriptor > descriptorList = new
ArrayList < IPropertyDescriptor > ();

        public ROSNodePropertySource(RosNode node) {
```

```
		this.node = node;
	}

	@Override
	public Object getEditableValue() {
		return this;
	}

	@Override
	public IPropertyDescriptor[] getPropertyDescriptors() {

		TextPropertyDescriptor nameProp=new
TextPropertyDescriptor("name", "Name");
		nameProp.setCategory("Basic Info");

		TextPropertyDescriptor typeProp =new
TextPropertyDescriptor("type", "Type");
		typeProp.setCategory("Basic Info");

		TextPropertyDescriptor pathProp =new
TextPropertyDescriptor("path", "Path");
		pathProp.setCategory("Basic Info");

		TextPropertyDescriptor pidProp =new TextPropertyDescriptor("pid",
"Pid");
		pidProp.setCategory("Basic Info");

		TextPropertyDescriptor pubProp=new
TextPropertyDescriptor("publications", "Publications");
		pubProp.setCategory("Ports");

		TextPropertyDescriptor subProp =new
TextPropertyDescriptor("subscriptions", "Subscriptions");
		subProp.setCategory("Ports");

		TextPropertyDescriptor paraProp =new
TextPropertyDescriptor("parameters", "Parameters");
		paraProp.setCategory("Parameters");

		TextPropertyDescriptor conProp =new
TextPropertyDescriptor("connections", "Connections");
		conProp.setCategory("Ports");


		IPropertyDescriptor[] propertyDescriptors = new
IPropertyDescriptor[]{
				nameProp,
typeProp,pathProp,pidProp,pubProp,subProp,conProp,paraProp
		};

		return propertyDescriptors;

	}

	@Override
	public Object getPropertyValue(Object id) {
	    if (id.equals("name")) {
	        return node.getName();
	    }
	    if (id.equals("type")) {
```

```
        return node.getType();
    }

    if (id.equals("path")) {
        return node.getPath();
    }

    if (id.equals("pid")) {
        return node.getPid();
    }
    if (id.equals("publications")) {
        return node.publicationsToString();
    }

    if (id.equals("subscriptions")) {
        return node.subscriptionsToString();
    }
    if (id.equals("connections")) {
        return node.connectionsToString();
    }
    if (id.equals("parameters")) {
        return node.parametersToString();
    }
    return null;

}


@Override
public boolean isPropertySet(Object id) {
       // TODO probably return true for ports and parmeters, etc.
       return false;
}


@Override
public void resetPropertyValue(Object id) {
       // TODO Auto-generated method stub
}

@Override
public void setPropertyValue(Object id, Object value) {
       String s = (String) value;
       Integer i = new Integer((String)value);

    if (id.equals("name")) {
      node.setName(s);
    }
    if (id.equals("type")) {
      node.setType(s);
    }
    if (id.equals("path")) {
            node.setType(s);
        }
    if (id.equals("pid")) {
            node.setPid(i.intValue());
        }
    if (id.equals("publications")) {

       //node.setPublications(s);
```

```
                }
                if (id.equals("subscriptions")) {

                        //node.setType(s);
                }
                if (id.equals("connections")) {

                        //node.setName(s);
                }
                if (id.equals("parameters")) {

                }
        }

}
```