UNIVERSIDAD NACIONAL DE LA PATAGONIA
SAN JUAN BOSCO

FACULTAD DE INGENIERIA
2013

Tesis de Licenciatura

UN ENTORNO DE DESARROLLO INTEGRADO
MULTIPLATAFORMA PARA LA INICIACION A LA PROGRAMACION ESTRUCTURADA Y CONCURRENTE

DaVinci Concurrente IDE

Autor:
APU Carlos Germán Tejero

Director:
Lic. Guillermo Feierherd

Propuesta de tesis para alcanzar el título de grado en Licenciatura en Informática

Índice de contenido

•

• Introducción

DaVinci Concurrente (DVC) es un lenguaje de programación, especialmente diseñado para facilitar el aprendizaje de la programación estructurada secuencial y concurrente.

DVC ha sido desarrollado por Daniel Eugenio Aguil Mallea sobre la base del lenguaje de programación Visual DaVinci (desarrollado por el Lic. Raúl Champredonde del Instituto de Investigación en Informática LIDI), brinda un lenguaje simple y en castellano que permite a los alumnos que toman un primer contacto con el mundo de la programación concentrarse en adquirir los aspectos fundamentales.

Las principales características de DVC son las siguientes:

- Lenguaje de programación estructurada simple y claro.
- Todos los elementos del lenguaje se encuentran en castellano.
- Posee todas las estructuras de control de flujo clásicas.
- La ejecución de programas se visualiza a través del movimiento de robots en una ciudad.
- Es posible incluir distintos tipos de elementos (flores, papeles y obstáculos) para plantear distintos tipos de problemas a resolver.

Dado que el trabajo del creador de DaVinci Concurrente incluye únicamente la definición del lenguaje y la implementación de un intérprete para el mismo, se carece hasta ahora de un entorno de desarrollo integrado, que permita la creación de programas de forma sencilla.

Es así que la principal motivación del presente trabajo consiste en crear un entorno que facilite la programación en el lenguaje DaVinci Concurrente y que, brinde todas las características de un entorno moderno de programación.

La mantención y la participación constituyen también motivaciones adicionales del presente. Se pretende que el entorno de desarrollo posea características que faciliten la mantención y participación en su desarrollo, así como la continuación del proyecto por otros interesados.

Asimismo, su utilización en materias que no sólo sean de iniciación a la programación, será estimulada a través de la posibilidad de implementar e intercambiar componentes del entorno fácilmente. Por ejemplo, dado que el planificador de tareas del intérprete es extensible e intercambiable, es sencillo que alumnos de materias como Sistemas Operativos, implementen distintos tipos de estrategias de planificación y experimenten con la herramienta.

En las próximas unidades veremos las decisiones que se tomaron sobre las herramientas a utilizar en el presente trabajo, que características se implementan, que características se planean implementar en futuros trabajos y los distintos módulos que componen la implementación.

1. Elección de las herramientas para el desarrollo

Antes de comenzar a abordar el presente trabajo, resulta conveniente señalar los motivos por los cuales se eligieron las herramientas necesarias para llevarlo adelante. Esta elección se realizó respetando la siguiente secuencia:

- Elección del lenguaje de programación
- Elección del entorno de desarrollo a utilizar
- Elección de la plataforma utilizar

• Elección del lenguaje de programación

Año tras año los alumnos universitarios cambian, así también como sus prácticas y costumbres. Hasta hace unos años, era normal que la mayoría de los alumnos no dispusieran de una computadora de su propiedad, lo que obligaba a las universidades a contar con laboratorios bien equipados. Con el correr de los años esto ha cambiado, de tal manera que hoy el 100% de los alumnos cuentan con una computadora en su domicilio, y una gran mayoría posee una computadora personal o portátil.

Si bien esta situación es natural, el uso de computadoras de propiedad de los alumnos, ha agregado un nuevo problema para los docentes al momento de determinar que herramientas utilizarán para favorecer los procesos de enseñanza y aprendizaje. Este problema es, básicamente, el de la heterogeneidad de los sistemas operativos que utilizan los alumnos. Si bien Windows sigue siendo aún el sistema operativo más utilizado, se observa un incremento en el uso de sistemas operativos como Linux y Mac OSX. Como consecuencia de ello, y con la decisión de evitar condicionamientos sobre el sistema operativo se ha impuesto como condición que debe ser dessarrollado en un lenguaje de programación que permita la ejecución sobre plataformas heterogéneas, tanto de hardware como de software.

Otro aspecto a tener en cuenta al momento de la elección del lenguaje de programación es la accesibilidad del mismo. Un objetivo del presente trabajo es, que dé lugar a la participación de interesados en la modificación, mantención o extensión del mismo. Por lo tanto, el lenguaje seleccionado, debe ser ampliamente conocido y utilizado en el mundo académico, y las herramientas de desarrollo deben ser fácilmente accesibles a cualquiera, preferentemente de forma libre y gratuita.

En la actualidad, los cinco lenguajes de programación más utilizados en el mundo son: C, C++, C#, Objetive-C y JAVA. Analizando cada uno de ellos, vemos:

- C: Si bien es un lenguaje multiplataforma ampliamente utilizado, esta orientado a la programación de sistemas de bajo nivel. Si bien fue utilizado para el desarrollo de aplicaciones de usuario final, actualmente es utilizado sólo para desarrollo de sistemas operativos, compiladores, motores de bases de datos, etc.
- C++: Creado para agregar orientación a objetos a C y reemplazarlo, es cada vez menos utilizado para el desarrollo de aplicaciones de usuario final y cada vez más utilizado para desarrollo de sistemas de bajo nivel. Existen herramientas de desarrollo en la mayoría de la plataformas existentes, sin embargo, no cuenta con una librería estandarizada multiplataforma de componentes reutilizables que lo acompañe.
- C#: Es un lenguaje especialmente diseñado para la plataforma .Net de Microsoft. Si bien existen intentos de implementaciones multiplataforma de .Net como MONO, la realidad es que .Net sólo puede ejecutarse sobre sistemas operativos Windows.
- Objetive-C: Es un lenguaje con orientación a objetos, para el cual existen herramientas de desarrollo en la mayoría de la plataformas existentes. Sin embargo, no cuenta con una librería

estandarizada multiplataforma de componentes reutilizables que lo acompañe, si bien la mayoría de los existentes se basan en NEXTSTEP.

- JAVA: Es un lenguaje orientado a objetos, compilado e interpretado. Creado con la premisa de ser multiplataforma, viene acompañado de un conjunto de componentes reutilizables, que facilitan el desarrollo.

Por los motivos antes mencionados, el presente trabajo será realizado utilizando el lenguaje de programación JAVA, dado que cumple con las impuestas.

- Elección del entorno de desarrollo a utilizar

Dentro del mundo de la programación en JAVA, existen numerosos entornos de desarrollo. Los más utilizados por los desarrolladores de todo el mundo son:

- Eclipse
- IntelliJ IDEA
- NetBeans

La Tabla1 resume las principales características de cada uno:

	NetBeans	**Eclipse**	**IntelliJ IDEA**
Primera versión	3.1 (1999)	3.0 (21/06/2004)	1.0 (01/2001)
Versión actual	7.3 (21/02/2013)	4.2 (01/03/2013)	12.0.4 (14/02/2013)
Ediciones	JAVA SE, JAVA EE, C/C++, PHP y ALL	Numerosas	Community y Ultimate
Licencia	CDDL - GPL2	EPL	APL 2 para Community y Propietaria para Ultimate
Lenguaje	JAVA	JAVA	JAVA
Plataforma	JAVA SE	JAVA SE–OSGI-SWT	JAVA SE
Sistema operativo	Multiplataforma	Multiplataforma	Multiplataforma

Tabla1

Analizando cada uno de estos IDEs resulta que:

IntelliJ IDEA: Es un gran entorno, del cual existen dos ediciones:

- Community: es libre y gratuita, pero posee numerosas limitaciones.
- Ultimate: es propietaria y requiere el pago de una licencia para poder utilizarla.

Dadas sus limitaciones la edición Community no puede utilizarse, y la edición Ultimate, al ser paga, limitaría la participación de personas que no pudieran afrontar los costos de su licencia. Por lo tanto, IntelliJ IDEA no puede ser utilizado para el presente trabajo.

Eclipse: Es el entorno de desarrollo más utilizado en el mundo por los programadores JAVA. Esto se debe en gran parte a que su licencia libre, es poco restrictiva y permite a empresas (como RedHat, IBM, Alfresco, Liferay, etc) empaquetarlo y distribuirlo junto con extensiones desarrolladas por ellas mismas, como entornos de desarrollos para sus plataformas (Jboss Developer Studio, Liferay Developer Studio, etc).

NetBeans: Es el entorno de desarrollo el más antiguo de los tres considerados. Si bien no es el más utilizado, dado que su licencia libre es muy restrictiva, es uno de los más amigables con el usuario.

Tanto Eclipse como NetBeans son entornos muy utilizados, con una comunidad muy activa de desarrolladores que los usan y los mantienen. Cualquiera de los dos entornos podrían ser utilizados para llevar adelante el proyecto.

Como la elección del entorno de desarrollo se ve influenciada por la elección de la plataforma a utilizar, posponemos la decisión de cual utilizar, hasta haber decidido la plataforma.

• Elección de la plataforma a utilizar

Para escribir un nuevo entorno de desarrollo tenemos principalmente dos caminos:

1. Desarrollo del entorno desde cero
2. Desarrollo del entorno sobre una plataforma preexistente

Describimos a continuación los puntos a favor y en contra de cada camino.

• Desarrollo del entorno desde cero

En este apartado veremos los puntos a favor y en contra de desarrollar el entorno desde cero (sin utilizar una plataforma para el desarrollo de entornos).

Puntos a favor:

- Curva de aprendizaje más corta: al no estar basado sobre una gran plataforma, el resultado del desarrollo es más pequeño, lo que facilita el aprendizaje y conocimiento del mismo.
- Sin limitaciones: al no estar apoyado sobre una plataforma existente, no existen las limitaciones o restricciones que impondría la misma.

Puntos en contra:

- Resultado poco profesional: los resultados profesionales, se alcanzan a través de años de maduración y evolución. Al ser un desarrollo nuevo desde cero, difícilmente se obtenga un resultado muy profesional en sus primeras versiones.
- Evolución depende de una comunidad más pequeña: al ser un desarrollo iniciado por una sola persona en una universidad pequeña, difícilmente pueda generarse una gran comunidad de desarrollo alrededor del mismo.
- Carencia de documentación: como todo producto nuevo, carece de documentación existente, la cual debería generarse desde cero.

• Desarrollo del entorno sobre una plataforma preexistente

En este apartado veremos los puntos a favor y en contra de desarrollar el entorno sobre una

plataforma preexistente.

Puntos a favor:

- Resultado muy profesional: el resultado de desarrollar sobre una plataforma existente, dara por resultado un producto muy similar al resto de los productos desarrollados sobre la misma plataforma.
- En continua evolución apoyada sobre la comunidad: dado que la sobre plataforma se apoyan otros proyectos, la continua evolución de los mismos, produce indefectiblemente la continua evolución de la plataforma.
- Re-utilización de código preexistente: en vez de tener que escribir nuevamente código existente, apoyarse sobre la existencia del mismo.
- Abundante documentación: las plataformas existentes, no son nuevas, y contienen mucha documentación acerca de ellas, además de libros, tutorias, etc.

Puntos en contra:

- Curva de aprendizaje más larga: no sólo es necesario conocer el lenguaje de programación sobre el que esta construido el producto, sino que también es necesario conocer la plataforma para poder participar en el desarrollo.
- Limitaciones impuestas por la plataforma: al desarrollar sobre una plataforma existente, esta impone restricciones y limitaciones, a través sus abstracciones.

Luego de analizar los puntos a favor y en contra de cada uno de los caminos, se ha optado por el camino de desarrollar el entorno sobre un plataforma existente. Esta decisión se basa principalmente en la idea de que el camino elegido favorecerá y potenciará la participación de personas interesadas en el proyecto.

Ya hemos elegido el camino a seguir, corresponde retomar la discusión sobre la plataforma a utilizar. Dado que nos hemos quedado solo con dos entornos de desarrollo posibles de utilizar (Elicpse y NetBeans) y que cada uno de ellos cuenta con una plataforma para la construcción de entornos de desarrollo y de aplicaciones en general, vamos a analizarlas brevemente para, decidir cuál es la más adecuada para el presente trabajo. Las plataformas son:

- Eclipse RCP
- NetBeans RCP

A continuación veremos los puntos a favor y en contra de cada una:

• Eclipse RCP

Puntos a favor:

- La más utilizada en el mundo de la programación JAVA
- Posee una plataforma madura, pero en constante desarrollo y mejora
- Su desarrollo se apoya en una extensa comunidad, compuesta por numerosas empresas

Puntos en contra:

- Si bien su interface es extremadamente flexible, es poco amigable con el usuario
- No se encuentra construida totalmente sobre la plataforma JAVA SE, y requiere de conocimiento no estándares como SWT (Standard Widget Toolkit, alternativa a SWING para desarrollo de interfaces de usuario en JAVA) y OSGI (Open Services Gateway Initiative, plataforma dinámica de módulos y servicios).

- NetBeans RCP

Puntos a favor:

- Construido totalmente sobre la plataforma JAVA SE
- Posee una plataforma madura, pero en constante desarrollo y mejora
- La interface de usuario es muy simple y amigable con el usuario

Puntos en contra:

- Su desarrollo no se apoya en una extensa comunidad, sino en una única empresa (Oracle)
- No es la más utilizada en el mundo de la programación JAVA

Luego de analizar los puntos a favor y en contra de cada una de las dos plataformas, se ha decidido utilizar NetBeans RCP. Esta decisión se funda en dos aspectos principales:

- La plataforma NetBeans RCP sólo requiere conocimientos de JAVA SE, lo que favorecería la participación de personas interesadas.
- El entorno de desarrollo producto de utilizar la plataforma, sera muy similar al entorno de desarrollo del propio fabricante. El entorno de desarrollo NetBeans es mucho más amigable con el usuario que Eclipse, por lo que es de esperar que el resultado de utilizar NetBeans RCP resulte, más amigable que el resultado de utilizar Eclipse RCP.

2. Elección de las características a implementar

Las características a implementar para el entorno de desarrollo, han sido escogidas en base a la experiencia de más de 10 años de programación.

Para una fácil lectura, vamos a agruparlas en cuatro grandes grupos:

- Características básicas del entorno
- Características de la edición de código
- Características de la ejecución de programas
- Características de la depuración de programas

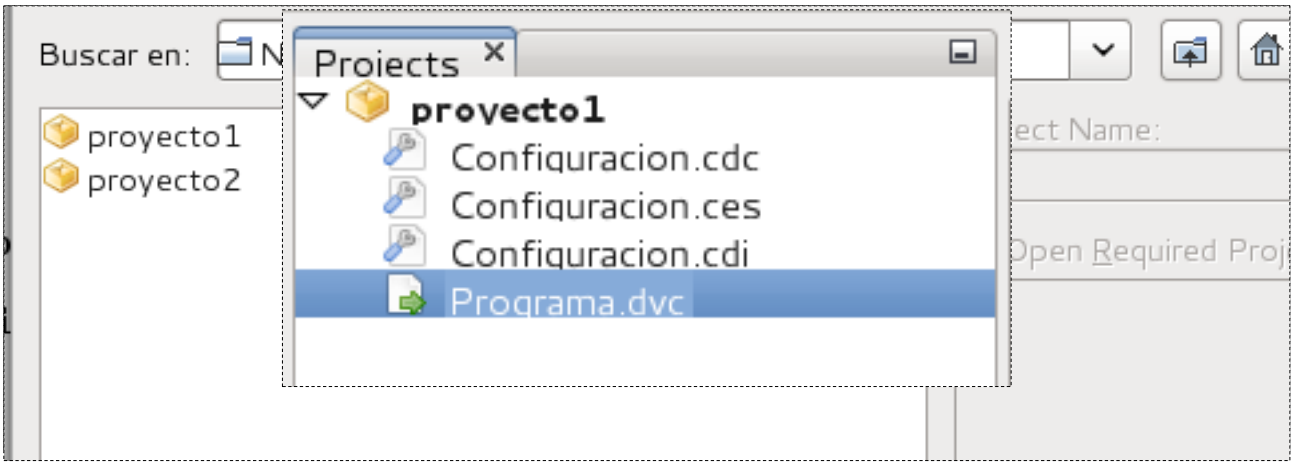
- Características básicas del entorno

En este grupo de características, encontramos las mínimas e indispensables para la manipulación de proyectos de DaVinciConcurrente. Las características a implementar son:

- Formato de proyectos
- Creación de proyectos
- Apertura de proyectos

- Formato de proyectos

Un proyecto de DaVinci Concurrente, es un conjunto de archivos en formato de texto cada uno de



los cuales contiene, información sobre distintas características del proyecto. Debe existir un archivo con el código fuente del proyecto, uno con la configuración de la ciudad, uno con la configuración del intérprete y uno con la configuración de la entrada-salida.

- ## Creación de proyectos

El entorno debe permitir que el usuaria cree nuevos proyectos de DaVinci Concurrente, les asigne un nombre y determine la ubicación en el disco donde guardarlo.

- ## Apertura de proyectos

El entorno debe permitir la apertura de proyectos previamente guardados en el disco, para poder trabajar en ellos.

- ## Características de la edición de código

En este conjunto de características, encontramos las mínimas e indispensables para la asistencia en la edición de código en proyectos de DaVinciConcurrente. Las características a implementar son:

- Ayuda
- Realce de sintaxis

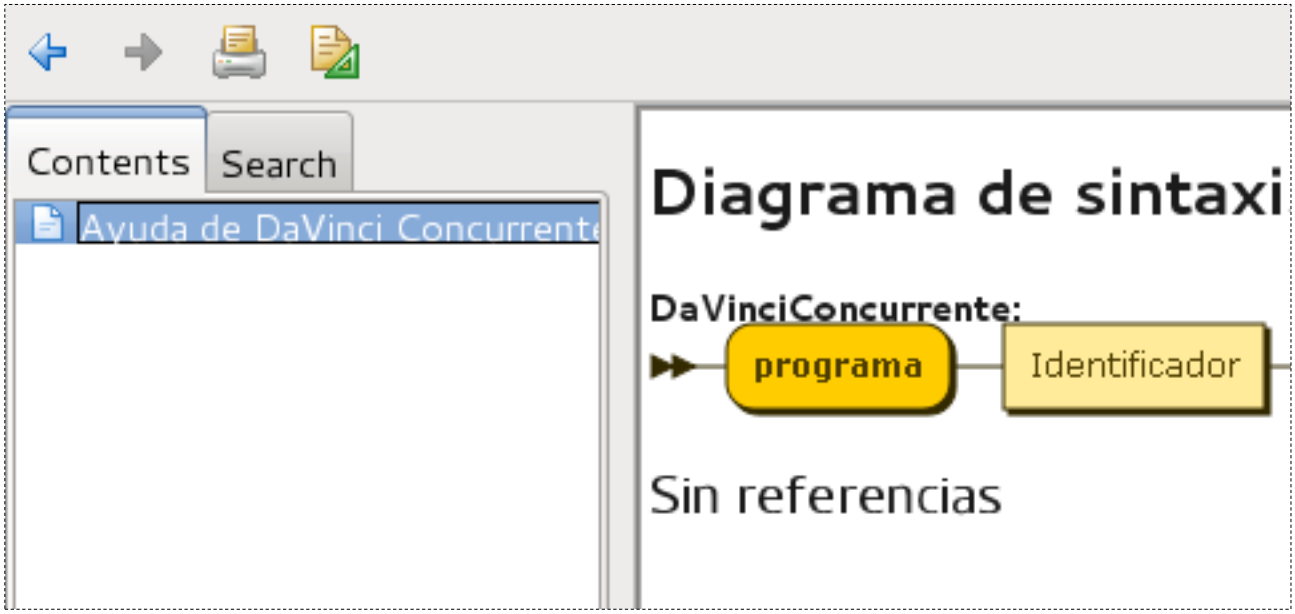
- Auto-completado
- Demarcación de errores

Se han definido también un conjunto de características a implementar en futuros trabajos. Estas son:

- Editor visual de programas
- Navegación de código por hiperenlaces
- Demarcación de usos de identificadores
- Plantillas de auto-completado de código
- Re-factoreo de código
- Advertencias y sugerencias acerca del código

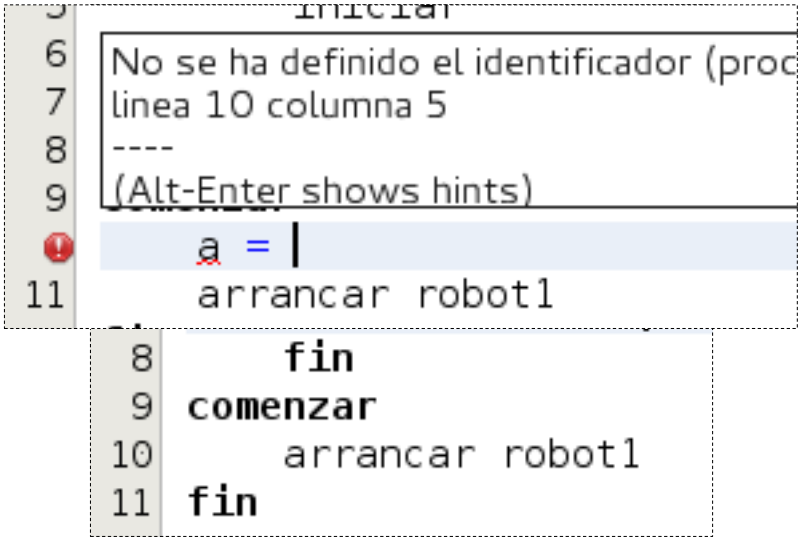
- ## Ayuda

El entorno debe permitir, acceder a la ayuda en cualquier momento, obteniendo tanto contenidos de referencia del lenguaje DaVinci Concurrente como del mismo entorno.



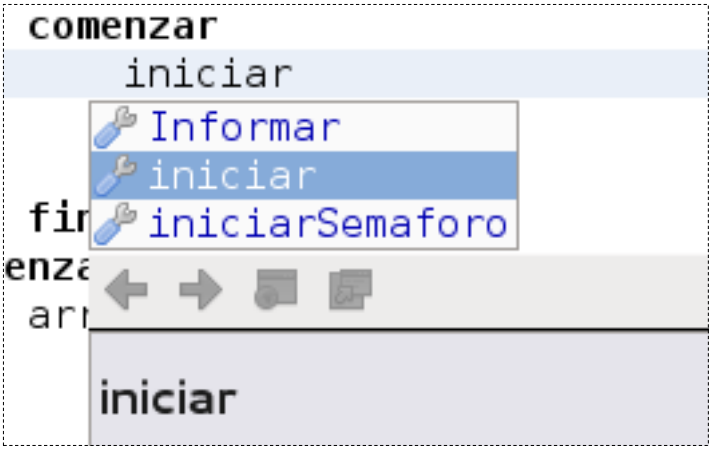
- ## Realce de sintaxis

El código fuente del proyecto debe realzarse, permitiendo que el usuario pueda distinguir rápidamente los diferentes elementos léxicos del lenguaje (palabras reservadas, cadenas de texto, símbolos, etc).



- Auto- completado

El entorno debe permitir invocar, un auto-completado de código que permita ver los distintos elementos del lenguaje y recibir rápida ayuda sobre los mismo.



- Demarcación de errores

El entorno debe verificar la sintaxis del código fuente del proyecto mientras el usuario escribe, e indicar de forma clara y visible los errores que va cometiendo, al igual que sugerir pistas de para solucionarlos.

- Editor visual de programas

El entorno debe facilitar su uso por usuarios sin conocimientos previos sobre algorítmica. Para ello es necesario la codificación mediante métodos visuales y no solamente mediante la edición manual de código.

- Navegación de código por hiperenlaces

El entorno debe permitir navegar por el código fuente a través de hiperenlaces en los identificadores, hasta la definición de los mismos.

- Demarcación de uso de identificadores

Al posicionarse sobre un identificador el entorno debe resaltar todas las instancia del mismo dentro del código fuente. Esto permite identificar rápidamente los usos del mismo.

- Plantillas de auto-completado de código

En entorno debe proporcionar un conjunto de plantillas de código, que permitan al usuario escribir rápidamente definiciones válidas dentro del lenguaje.

- Re-factoreo de código

El entorno debe permitir extraer porciones de código repetitivas y reestructurarlas de forma acorde a las buenas prácticas de programación.

- Advertencias y sugerencias acerca del código

El entorno debe proporcionar advertencias acerca de malas prácticas de programación detectadas en el código fuente y sugerencias sobre como corregir las mismas.

- Características de la ejecución de programas

En este conjunto de características, encontramos las mínimas e indispensables para la ejecución de proyectos de DaVinciConcurrente, así como para la configuración de los elementos que intervienen en la misma. Las características a implementar son:

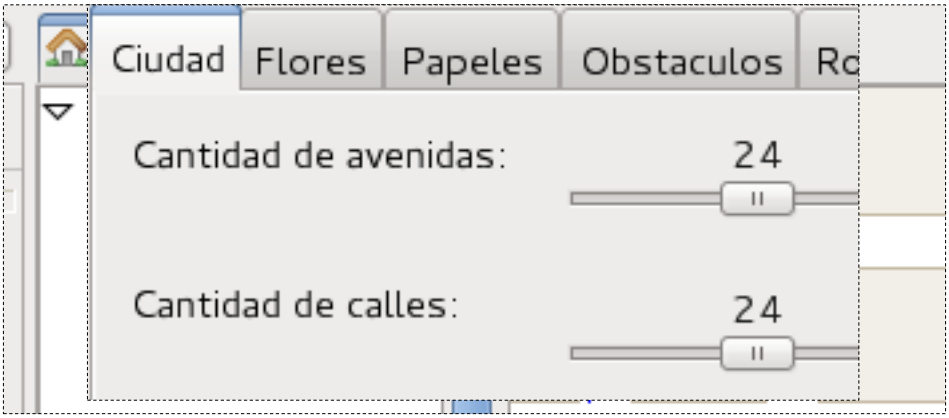
- Visualización de la ejecución
- Configuración de la ciudad
- Configuración del interprete
- Configuración de la entrada salida

Se han definido también un conjunto de características a implementar en futuros trabajos. Estas son:

- Repetición de la última ejecución
- Editor de la traza de la última ejecución

- Visualización de la ejecución

Dado que la ejecución de programas en DVC, se basa en el movimiento de robots en una ciudad, se necesita de algún sistema de visualización de la ciudad y las variaciones que se producen en la misma producto de la ejecución del código. El entorno debe proveer mecanismos para dicha



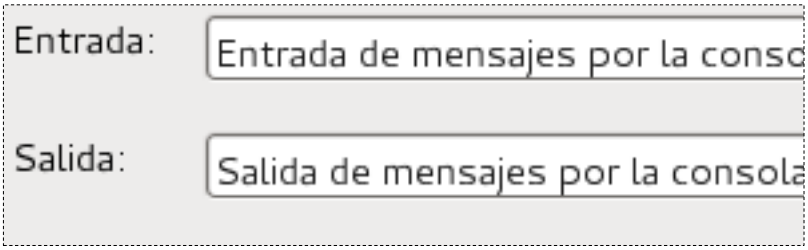
visualización, contemplando los siguientes requerimientos:

- Deben poder distinguirse las avenidas y las calles.
- Deben poder distinguirse la ubicación y cantidad de flores en las esquinas.
- Deben poder distinguirse la ubicación y cantidad de papeles en las esquinas.
- Deben poder distinguirse la ubicación de obstáculos en las esquinas.
- Deben poder distinguirse la ubicación, dirección y estado de los robots.

- ## Configuración de la ciudad

Para poder plantear distintos tipos de problemas a resolver, deben poder ser configurados distintos aspectos y elementos de la ciudad. El entorno debe permitir configurar:

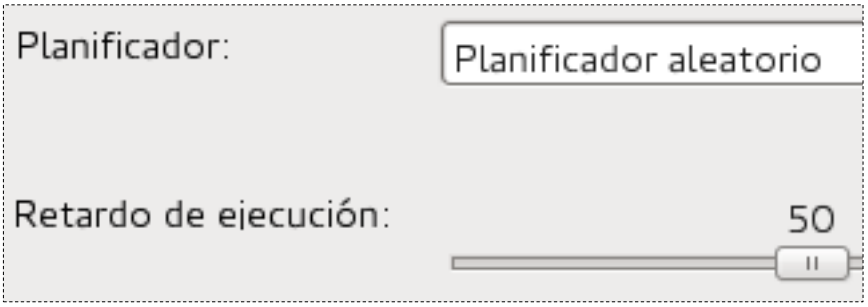
- Cantidad de calles y avenidas.
- Ubicación y cantidad de flores en la ciudad.
- Ubicación y cantidad de papeles en la ciudad.
- Ubicación y cantidad de obstáculos en la ciudad.
- Cantidad de flores inicial en la mochila de los robots.
- Cantidad de papeles inicial en la mochila de los robots.



- Configuración del intérprete

Para poder experimentar con distintas estrategias de planificación, deben poder ser configurados distintos aspectos del intérprete. El entorno debe permitir configurar:

- Tiempo de retardo entre la ejecución de instrucciones.
- Planificador de tareas a utilizar.



- Configuración de la entrada salida

Para poder experimentar con distintos mecanismos de entrada-salida de mensajes, el entorno debe permitir configurar:

- Canal de entrada de mensajes.
- Canal de salida de mensajes.

- Repetición de la última ejecución

El entorno debe permitir repetir exactamente la traza de la última ejecución del proyecto, como así también conservarla para su análisis posterior, particularmente cuando dicha traza ha dado lugar a fallos de sincronización o violación de la exclusión mutua.

- Editor de trazas

El entorno debe permitir la creación de trazas en forma manual (desde cero o mediante modificaciones de una traza guardada), a fin de provocar errores de sincronización o violación de la

exclusión mutua que
mediante un enfoque
fuente.

han sido detectados
deductivo del código

- ## Características de la depuración de programas

Dentro de la depuración se han definido las siguientes características a implementar:

- Ejecución de proyectos paso a paso

Se han definido también un conjunto de características a implementar en un futuro. Estas son:

- Visualización de variables
- Visualización de hilos
- Visualización de pila de llamadas
- Puntos de interrupción

- ## Ejecución de proyectos paso a paso

El entorno debe permitir pausar la ejecución en cualquier momento, y continuarla paso a paso a través de las sentencias del proyecto.

- ## Visualización de variables

El entorno debe proveer un cuadro de dialogo en el cual visualizar las variables que se encuentran en el contexto de ejecución actual, junto con sus valores actuales.

- ## Visualización de hilos

El entorno debe proveer un cuadro de dialogo en el cual visualizar cada uno de los hilos que se encuentran en ejecución y el estado de los mismos.

- ## Visualización de pila de llamadas

El entorno debe proveer un cuadro de dialogo en el cual visualizar la pila de llamadas actual.

- ## Puntos de interrupción

El entorno debe proveer la posibilidad de marcar distintos puntos, dentro del código fuente del proyecto, para detener la ejecución del mismo, y así poder examinar el contexto actual de ejecución.

- ## Formato del proyecto

Como todo proyecto implementado sobre la plataforma NetBeans RCP, DaVinci Concurrente EDI

se divide en módulos, cada uno de los cuales implementa una o más de las características requeridas por el presente trabajo. A continuación repasaremos cada uno de los módulos, detallando de cada uno, sus funciones, que características implementa, un diagrama de dependencias sobre el resto de los módulos, un diagrama de las clases que lo componen, y uno o más diagramas de secuencia sobre las funcionalidades principales. Cada uno de los diagramas que veremos en cada módulo, son simples y no pretenden cubrir todo lo implementado en cada módulo, dadas las limitaciones de espacio en el presente documento. De lo contrario, solo pretenden mostrar y clarificar los más relevante de cada módulo.

Los módulos se han dividido en cuatro grupos según las características que implementa cada uno:

- Módulos que brindan las características básicas del entorno
- Módulos que brindan las características de la edición de código
- Módulos que brindan las características de la ejecución de código
- Módulos que brindan las características de la depuración de código

• Módulos que brindan las características básicas del entorno

En este apartado, iremos recorriendo cada uno de los módulos que implementan las características básicas del entorno, propuestas para este trabajo. De cada módulo, veremos que características implementa, de que módulos del proyecto depende, que clases implementa, cual es la responsabilidad de cada clase, alguna secuencia significativa del módulo, si existen errores conocidos del mismo, y los trabajos futuros que ya se han definido.

Los módulos son:

- DaVinciConcurrenteEDI
- DaVinciConcurrenteProyecto
- DaVinciConcurrentePlantillaProyecto

• DaVinciConcurrenteEDI

Este módulo representa la aplicación DaVinci Concurrente EDI. El mismo contiene la siguiente información:

- Nombre de la aplicación.
- Información de licenciamiento de la aplicación.
- Imagen de la pantalla de inicio y de acerca de.
- Referencia a los módulos de la plataforma NetBeans RCP que incluye.
- Referencia a los módulos de desarrollo propio que incluye.
- Archivos de configuración de la maquina virtual de java y de auditoria.

De este módulo no se presentan diagramas de ningún tipo, ya que no incluye definiciones de clases, ni ningún tipo de código ejecutable. Por el contrario, solo contiene archivos de definición y configuración.

• Trabajos futuros sobre el módulo.

A continuación se detallan los trabajos futuros sobre el módulo:

- Personalización de la pantalla de inicio: para esta tarea se debe escoger un logotipo identificativo de lenguaje DaVinci Concurrente y diseñar una imagen de inicio que incluya al mismo.

- Creación de un pantalla de bienvenida: la mayoría de los entornos de desarrollo, muestran al iniciar por primera vez, una pantalla de bienvenida que permite acceder a tutorias y ejemplos al usuario novel o casual. Se puede crear una pantalla similar que permita una demostración sobre las capacidades del lenguaje y del entorno al usuario que toma contacto por primera vez con la herramienta.

• DaVinciConcurrenteProyecto

Para la plataforma NetBeans RCP, un proyecto es un directorio con cierto contenido especial que lo caracteriza. En el caso de DaVinci Concurrente, un proyecto es un directorio que contiene cuatro archivos:

1. Programa.dvc: contiene el código fuente del programa. Su extensión, son las siglas de **D**a**V**inci **C**oncurrente.

2. Configuracion.cdc: contiene la configuración de la ciudad. Su extensión, son las siglas de Configuración **d**e **C**iudad.

3. Configuracion.cdi: contiene la configuración del interprete. Su extensión, son las siglas de Configuración **d**e **I**nterprete.

4. Configuracion.ces: contiene la configuración de entrada-salida. Su extensión, son las siglas de Configuración de **E**ntrada-**S**alida.

Este módulo implementa las clases necesarias para dotar a la plataforma de las siguientes características:

- Formato de proyectos: como enunciamos anteriormente, un proyecto es un directorio que contiene cuatro archivos.

- Detección de proyectos: cuando el desde el entorno se intenta abrir un proyecto previamente guardado en disco, se debe detectar que directorios son proyectos y cuales no.

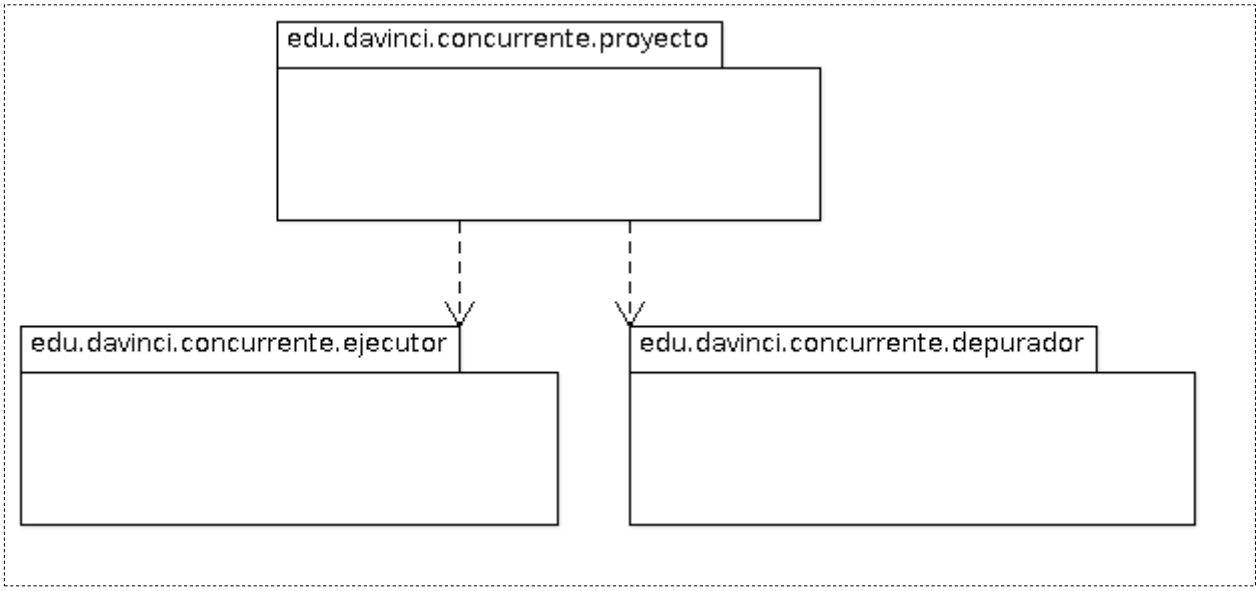
- Apertura de proyectos: cuando se abre un proyecto previamente guardado, el entorno instancia las clases que representan un proyecto y todas la información necesaria del mismo.

- Representación de proyectos: los proyectos abiertos en el entorno, son representados en forma de nodos de un árbol de la vista de proyectos del mismo.

A continuación veremos algunos diagramas que representan las dependencias del módulo, que clases lo componen y sus responsabilidades, y como implementa la funcionalidad requerida.

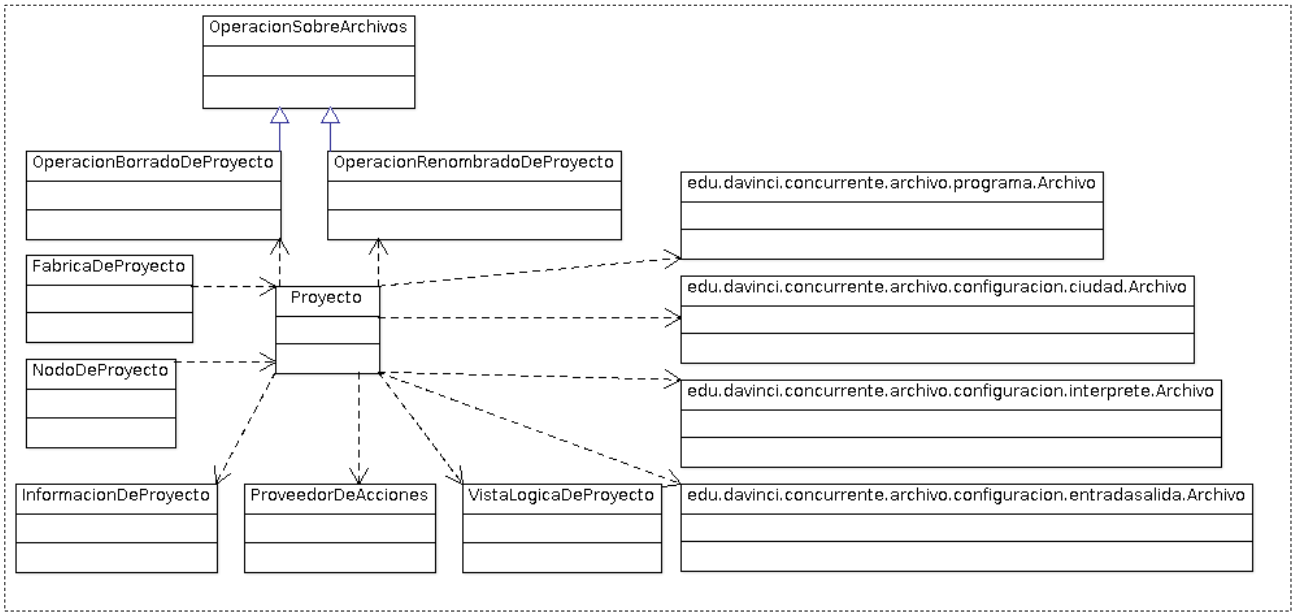
• Diagrama de dependencias

En el presente diagrama se puede visualizar de que módulos depende directamente el presente. No se incluyeron en el mismo, aquellos que se alcanzan por transitividad para simplificar el mismo.



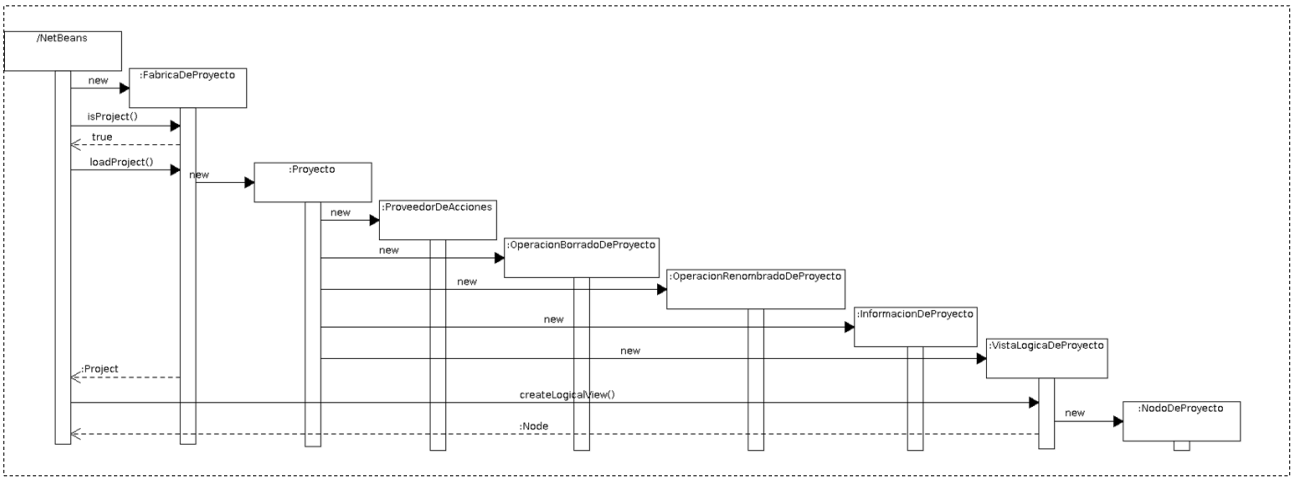
- Diagrama de clases

Este diagrama representa las clases que contiene el presente módulo y sus distintas relaciones, al igual que algunas relaciones con clases implementadas en otros módulos.



Las clases implementadas por el módulo son:

- FabricaDeProyecto: contiene los métodos necesarios para realizar la detección de proyectos a partir de directorios y de la instanciación de proyectos, también a partir de directorios.
- Proyecto: representa un proyecto y registra las operaciones que se pueden realizar sobre el, la representación y la información del mismo, instanciando las clases adecuadas.
- NodoDeProyecto: determina la forma en que se visualiza el proyecto en el árbol de proyectos del entorno, el icono representativo, nombre, etc.
- InformacionDeProyecto: determina la forma en que se visualiza el proyecto en el dialogo de



apertura de proyectos del entorno, el icono representativo, nombre, etc.

- VistaLogicaDeProyecto: encargada de instanciar las clases de representación del proyecto (NodoDeProyecto) en el árbol del proyectos.
- ProveedorDeAcciones: determina que acciones se pueden realizar sobre un proyecto (borrarlo, re-nombrarlo, etc) y las ejecuta a demanda.
- OperacionesSobreArchivos: clase base de las operaciones sobre archivos o carpetas.
- OperacionBorradoDeProyecto: encargada de llevar adelante el borrado de un proyecto.
- OperacionRenombradoDeProyecto: encargada de llevar adelante el cambio de nombre de un proyecto.

• Diagrama de secuencia

El siguiente diagrama de secuencia, muestra la interacción de las clases del módulo y el entorno, para realizar la apertura de un proyecto.

• Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

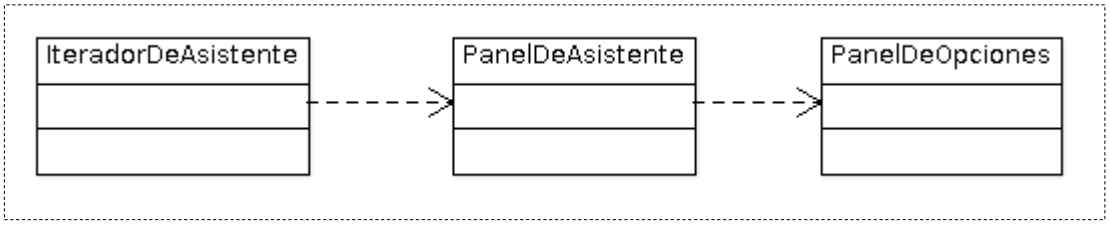
• Trabajos futuros sobre el módulo

La visualización de los archivos de configuración en forma de nodos del proyecto no es la más clara y adecuada. En un futuro se eliminara esta visualización y se accederá a la configuración a través de opciones del proyecto con ventanas emergentes modales.

Se plantea la inclusión de un nuevo archivo al proyecto, que sí se representara como un nodo del mismo. El mismo sera un archivo de texto simple en el cual se encontraran los detalles del ejercicio que debe resolver el alumno. De esta forma, será más fácil para el docente distribuir entre los alumnos los ejercicios, directamente en forma de proyectos, pre-configurados junto con el enunciado del mismo.

• DaVinciConcurrentePlantillaProyecto

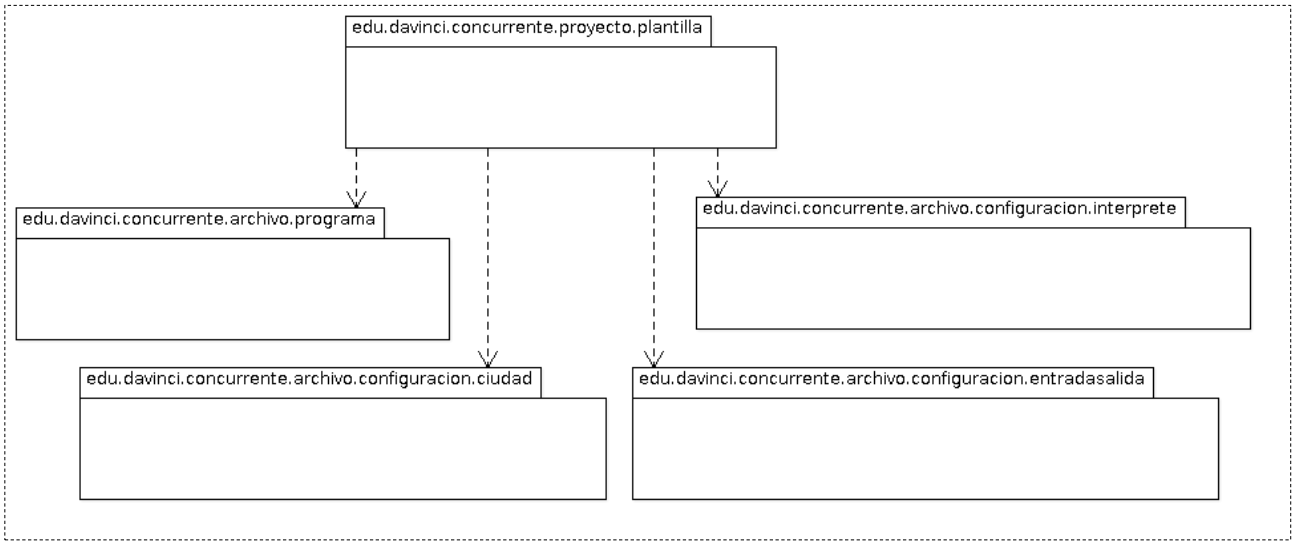
Para poder crear un nuevo proyecto dentro de la plataforma, es necesario de un asistente que nos

guie a 

paso paso, y una plantilla del mismo.

Dentro de las características que implementa este módulo, se encuentran:

- Creación de proyectos: para esto implementa el asistente y la plantilla necesaria para la creación de proyectos.



A continuación veremos algunos diagramas para profundizar más en el módulo.

• Diagrama de dependencias

Como vemos en el siguiente diagrama, el módulo depende del resto de los módulos que implementan cada uno de los archivos que conforman un proyecto de DaVinci Concurrente.

• Diagrama de clases

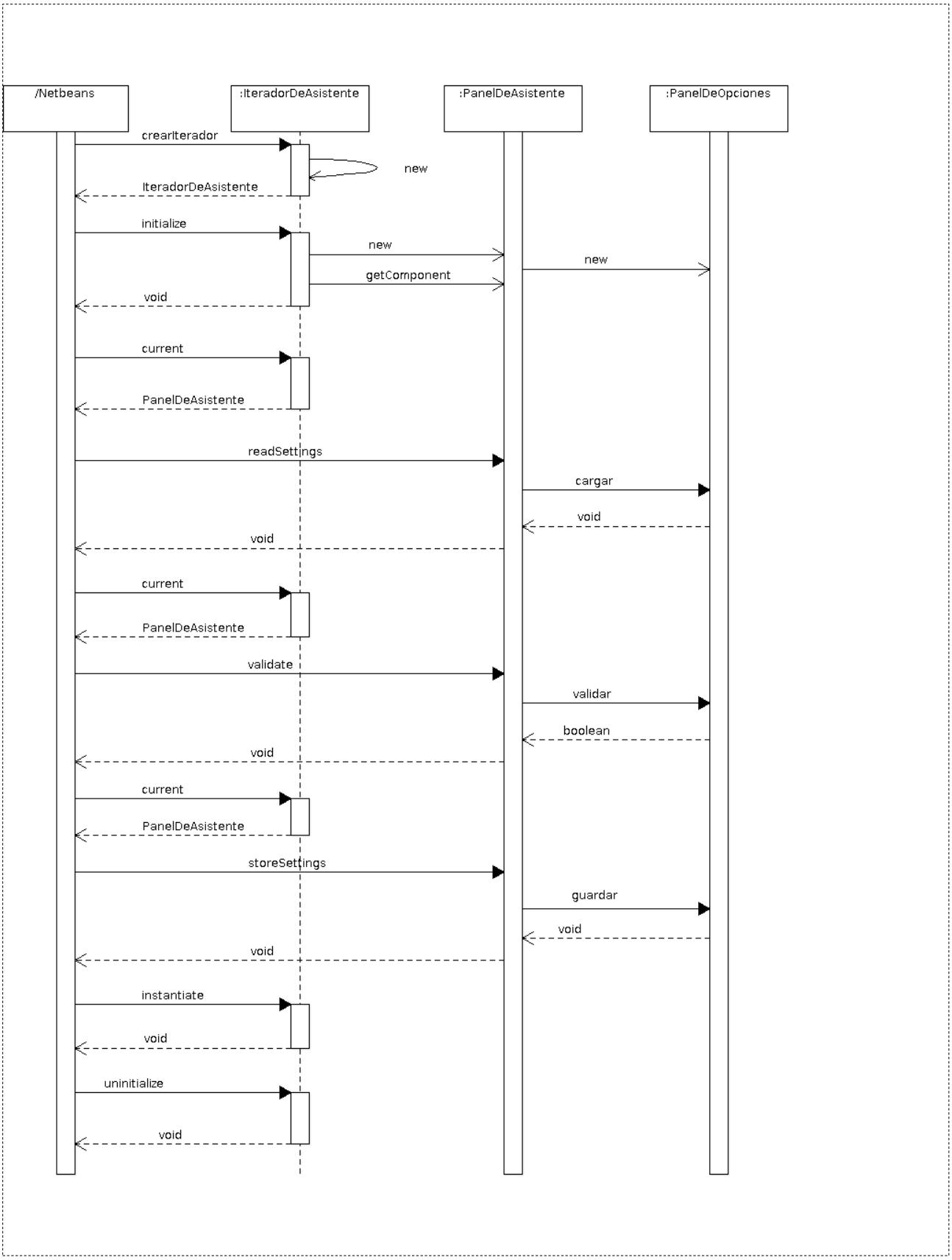
El siguiente diagrama muestra las clases implementadas por el módulo y sus relaciones.

Las clases son:

- IteradorDeAsistente: esta clase es responsable del esta actual del asistente para la creación de proyectos, cual es el panel actual, si hay uno siguiente, si hay un posterior, etc. Además es responsable de crear físicamente, en el disco, el proyecto cuando finaliza el asistente.
- PanelDeAsistente: esta clase es responsable de instanciar el PanelDeOpciones y validar que sus entradas sean validas.
- PanelDeOpciones: esta clase es el panel visual que muestra opciones al usuario al momento de la creación del proyecto.

- Diagrama de secuencia

En este diagrama se puede ver la interacción entre las clases del módulo y el entorno, en el



momento de la creación de un proyecto.

- ## Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- ## Trabajos futuros sobre el módulo

No se han definidos aun trabajos futuros sobre el módulo.

- ## Módulos que brindan las características de la edición de código

En este apartado, iremos recorriendo cada uno de los módulos que implementan las características de la edición de código, propuestas para este trabajo. De cada módulo, veremos que características implementa, de que módulos del proyecto depende, que clases implementa, cual es la responsabilidad de cada clase, alguna secuencia significativa del módulo, si existen errores conocidos del mismo, y los trabajos futuros que ya se han definido.

Los módulos son:

- DaVinciConcurrenteArchivo
- DaVinciConcurrenteArchivoPrograma
- DaVinciConcurrenteAyuda
- DaVinciConcurrenteAnalisisLenguaje
- DaVinciConcurrenteRealceSintaxis
- DaVinciConcurrenteAutoCompletado
- DaVinciConcurrenteErroresSintaxis

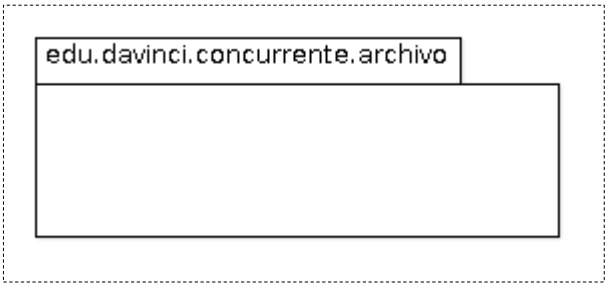
- ## DaVinciConcurrenteArchivo

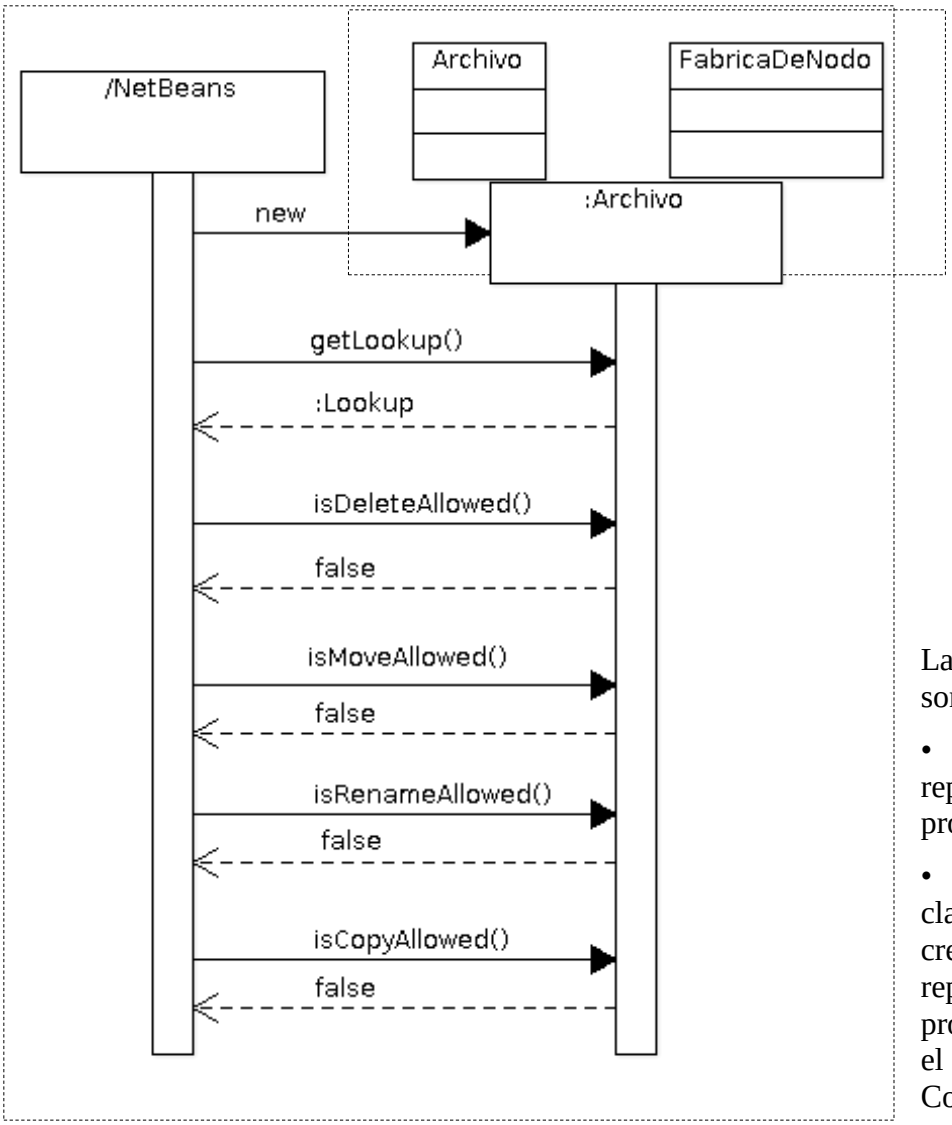
Este módulo no implementa ninguna característica en si mismo, pero define clases que sirven de base para los módulos que implementan los distintos archivos del proyecto.

A continuación veremos algunos diagramas para profundizar más en el módulo.

- ## Diagrama de dependencias

En el siguiente diagrama podemos ver que el módulo no tiene dependencias de ningún otro módulo del proyecto.





- Diagrama de clases

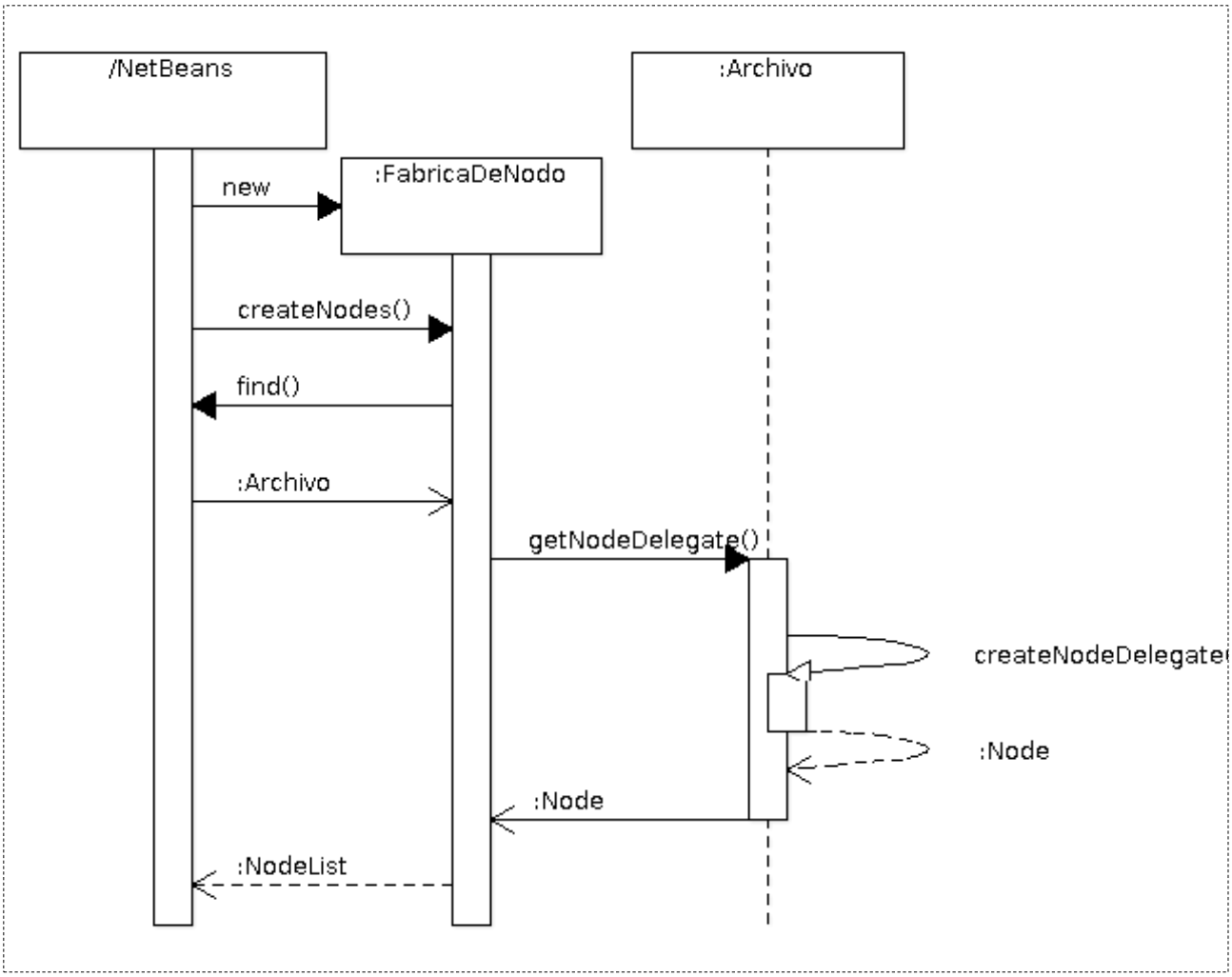
En el siguiente diagrama podemos ver las dos clases que implementa el módulo:

Las clases que implementa son:

- Archivo: esta clase representa un archivo del proyecto.
- FabricaDeNodo: esta clase es la encargada de crear los nodos que representan en el visor de proyectos a un archivo. En el caso de DaVinci Concurrente EDI, un archivo esta representado por un solo nodo.

- Diagrama de secuencia de clase Archivo

A continuación veremos un diagrama de secuencia que muestra la interacción típica entre el entorno y la clase Archivo:



- Diagrama de secuencia de clase FabricaDeNodo

En este migrada, vemos la secuencia que realiza el entorno en la creación de nodo de cada archivo:

- Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- Trabajos futuros sobre el módulo

No se han definidos aun trabajos futuros sobre el módulo.

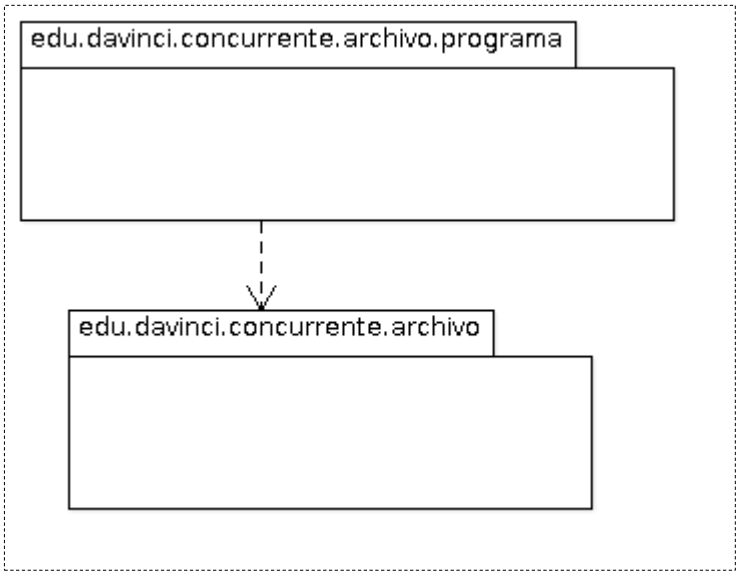
- DaVinciConcurrenteArchivoPrograma

Este módulo implementa las clases necesarias para la representación del archivo de programa, que contiene el código fuente del proyecto. Así mismo define las operaciones que pueden ser realizadas sobre el archivo.

A continuación veremos algunos diagramas para profundizar más en el módulo.

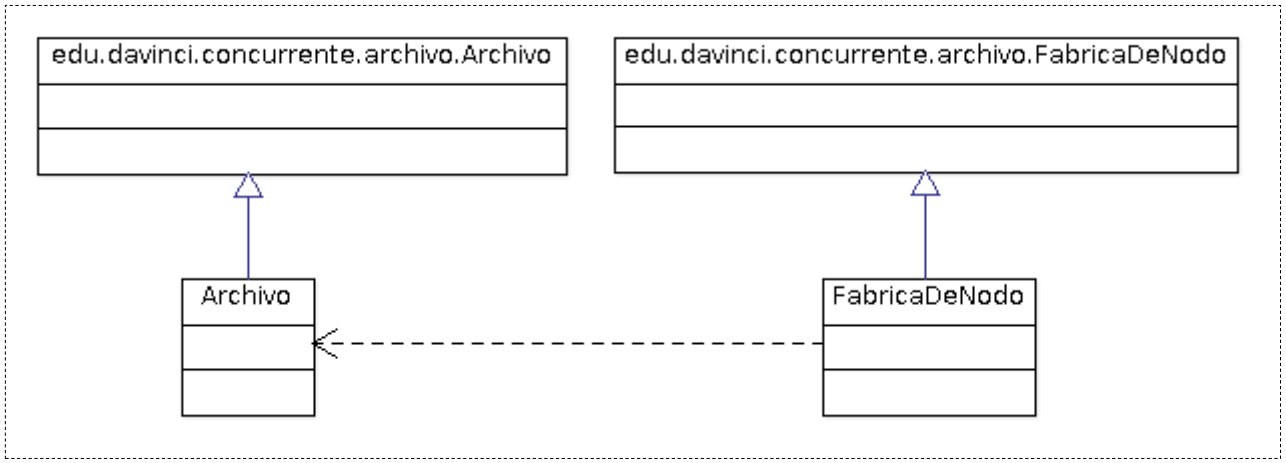
- Diagrama de dependencias

En el siguiente diagrama, podemos visualizar que este módulo depende solo del módulo DaVinciConcurrenteArchivo de este proyecto.

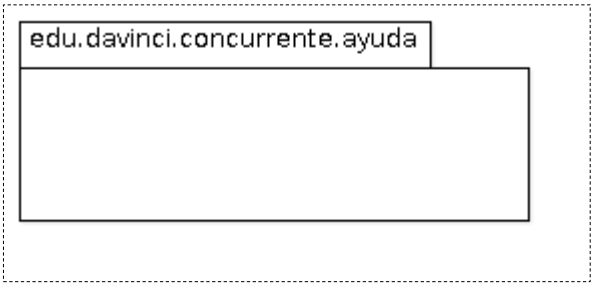


- Diagrama de clases

En el siguiente diagrama de clases, se pueden ver las dos clases que implementa el módulo, las cuales heredan directamente de las clases del módulo DaVinciConcurrenteArchivo, sin agregar nigua tipo de funcionalidad extra.



- Errores conocidos del módulo



No se han encontrado aun errores sobre el módulo.

- Trabajos futuros sobre el módulo

No se han definidos aun trabajos futuros sobre el módulo.

- DaVinciConcurrenteAyuda

Este módulo implementa la ayuda del entorno, y brinda servicios para que el módulo de auto-completado muestre documentación de cada uno de los elementos que presenta. Se encuentra desarrollado sobre el sistema JavaHelp, que es el estándar para dotar de ayuda a las aplicaciones que corren sobre la plataforma Java. En este sistema la ayuda consta de algunos archivos de configuración de la tabla de contenidos, indexación, visualización, etc., y archivos en formato html que dan el contenido de a la misma.

En su interior contiene tres directorios:

- raíz: en este directorio están contenidos los archivos de configuración de JavaHelp, y una única clase utilitaria.
- contenido: en este directorio se encuentran los archivos html, que representan las paginas de la ayuda.
- diagramas: en este directorio se encuentran los diagramas de la sintaxis de DaVinci Concurrente en formato de imagen.

Dentro de las características que implementa este módulo, se encuentran:

- Ayuda

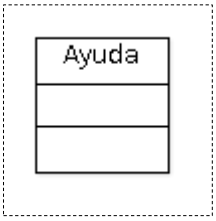
A continuación veremos algunos diagramas para ilustrar el contenido del presente módulo.

- Diagrama de dependencias

Este módulo no tiene dependencias de ningún otro módulo del proyecto.

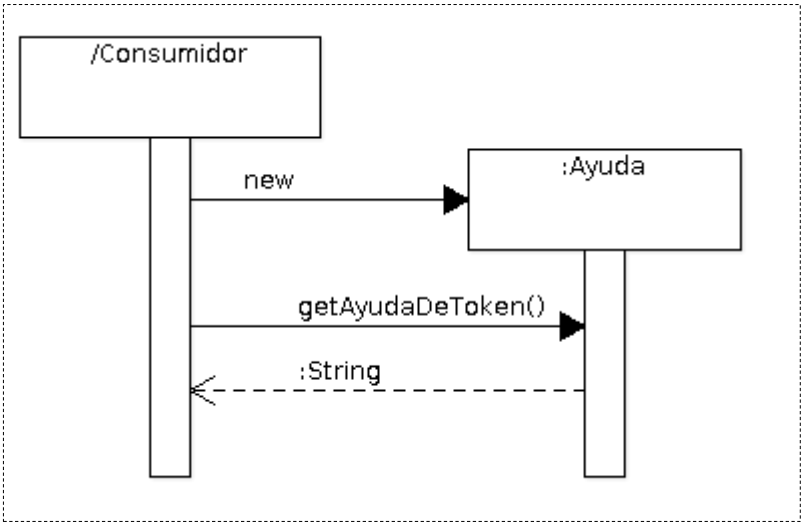
- Diagrama de clases

El módulo solo implementa una clase utilitaria, que permite obtener la documentación asociada a un elemento léxico.



- Diagrama de secuencia

En el siguiente diagrama de secuencia simple, vemos como la clase utilitaria Ayuda, brinda servicios para obtener documentación de un elemento léxico en particular.



- Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- Trabajos futuros sobre el módulo

Paulatinamente, trabajando de forma conjunta con las cátedras involucradas de la universidad, se irán incorporando tutorias y ejemplos del lenguaje a la ayuda del entorno.

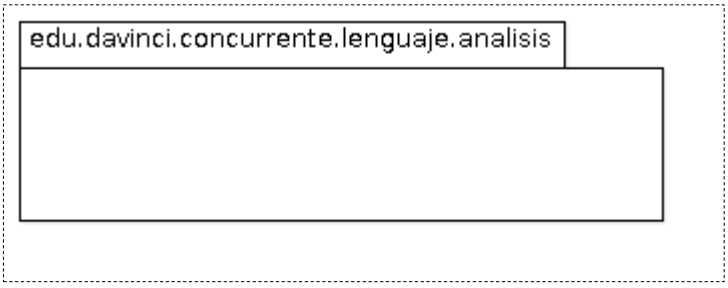
- DaVinciConcurrenteAnalisisLenguaje

Este módulo es utilizado por otros, para realizar la división en componentes léxicos del código fuente del programa y la detección de errores en el mismo.

A continuación veremos algunos diagramas para profundizar más en el módulo.

- Diagrama de dependencias

Este módulo no tienen ningún tipo de dependencias sobre otros módulos del proyecto.

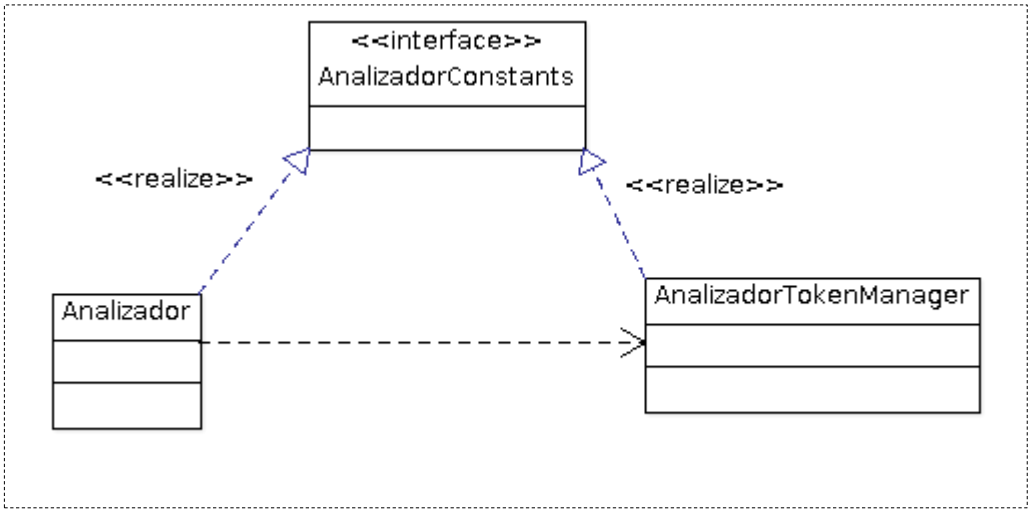


- Diagrama de clases

Si bien este módulo contiene un montón de clases, todas ellas son generadas por la herramienta JavaCC a través de la definición de la gramática del lenguaje.

En el siguiente diagrama, solo se representan las tres clases principales generadas y sus relaciones.

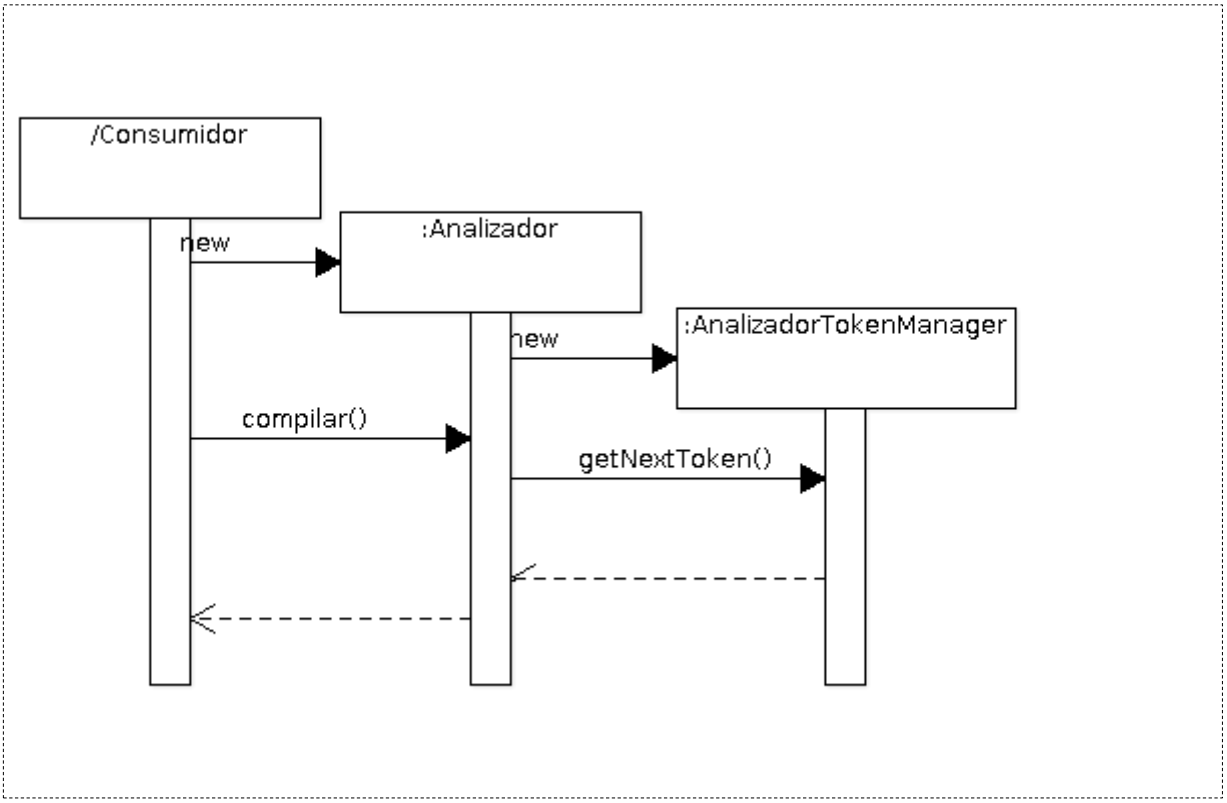
Las clases son:



- AnalizadorConstants: esta clase contiene definiciones de constantes, principalmente de cada elemento léxico del lenguaje.
- AnalizadorTokenManager: esta clase es responsable de realizar el análisis léxico del código fuente y la división del mismo en elementos léxicos.
- Analizador: esta clase es responsable de realizar el análisis sintáctico del código fuente, permite también la detección de errores en el mismo.

- Diagrama de secuencia

El siguiente diagrama de secuencia muestra de forma simplificada la forma en que el analizador realiza el análisis del código fuente.



- Errores conocidos del módulo

Errores en tiempo de ejecución pueden producirse en el módulo si es detectado un carácter invalido en el código fuente.

Dado que la definición de la gramática no define como elementos léxicos a los comentarios, sino que los ignora, no pueden distinguirse estos para por ejemplo, el realce de sintaxis de los mismos.

- Trabajos futuros sobre el módulo

Este módulo debe ser fusionado con el interprete del lenguaje. Su existencia es simplemente justificada, por diferencias necesarias en la gramática a la definición original del lenguaje.

- DaVinciConcurrenteRealceSintaxis

Este módulo se encarga de realzar de distintos colores, cada una de las categorías de elementos léxicos encontrados en el código fuente de un proyecto. Para esta tarea, se vale de los servicios del módulo de análisis del lenguaje.

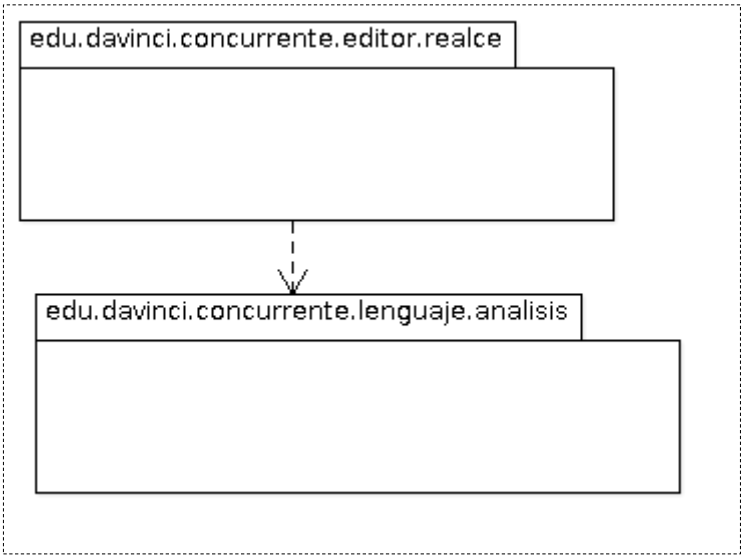
Dentro de las características que implementa este módulo, se encuentran:

- Realce de sintaxis

A continuación veremos algunos diagramas para profundizar más en el módulo.

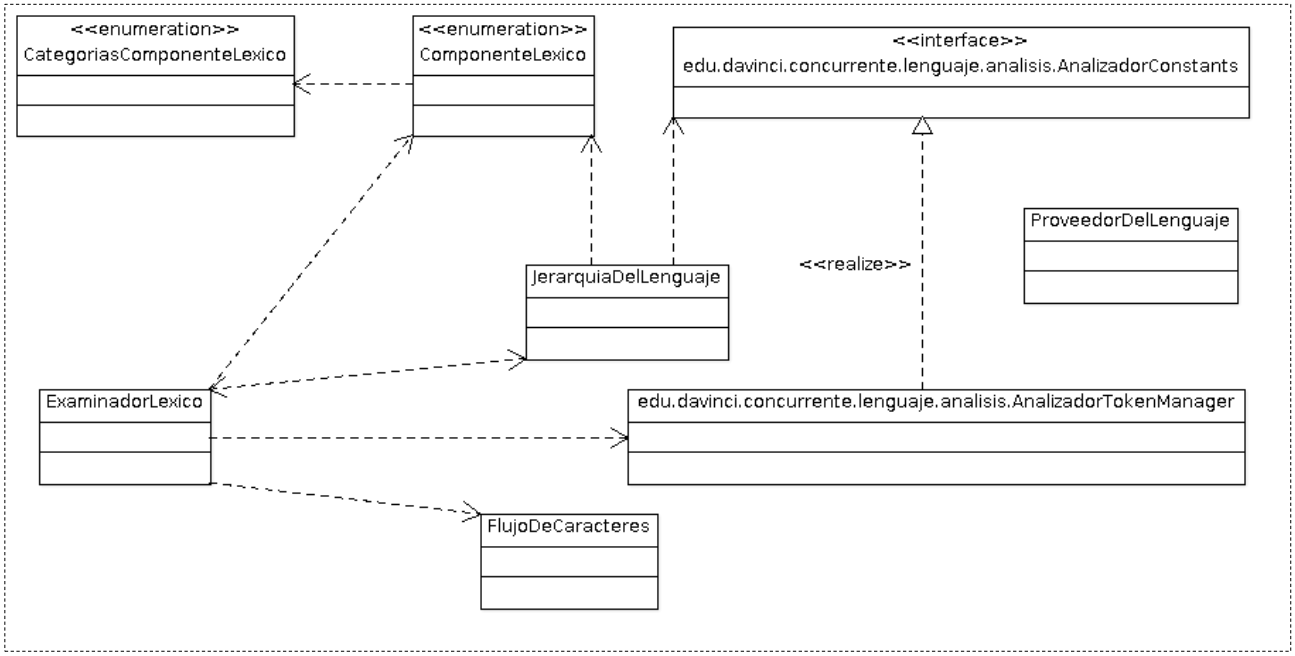
- Diagrama de dependencias

El siguiente diagrama de dependencias, muestra como este módulo depende del módulo de análisis del lenguaje.



- Diagrama de clases

En el diagrama de clases que esta a continuación, pueden apreciarse todas las clases implementadas en el módulo, y algunas de sus relaciones con clases del módulo de análisis del lenguaje.



Las clases son:

- CategoriasComponenteLexico: esta enumeración define cada una de las categorías de componentes léxicos que reconoce el realzador (palabras reservadas, símbolos, etc).

- ComponenteLexico: esta enumeración define cada uno de los elementos léxicos que componen la gramática del lenguaje.

- ExaminadorLexico: extrae del código fuente del proyecto, cada uno de los elementos léxicos, utilizando los servicios del módulo de análisis de lenguaje.

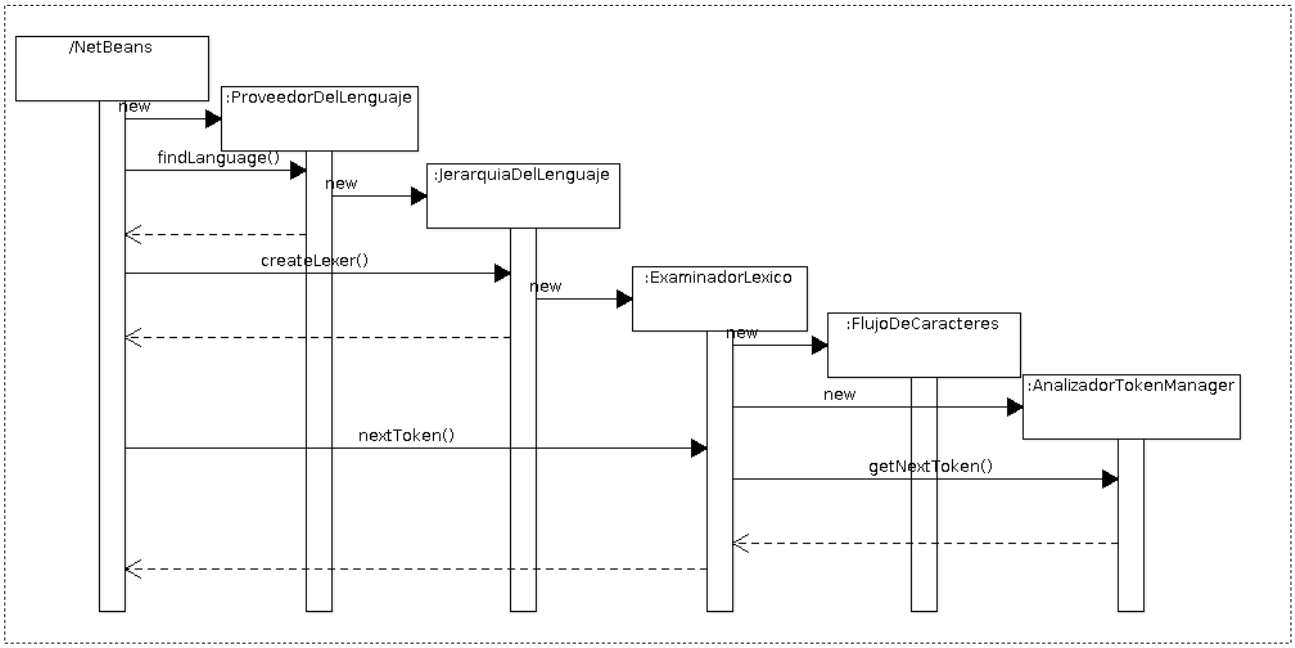
- JerarquiaDelLenaguaje: esta clase factoría, tiene la responsabilidad de crear el analizador

léxico.

- FlujoDeCaracteres: esta clase permite leer los caracteres del código fuente al módulo de análisis del lenguaje.
- ProveedorDelLenaguje: tiene la responsabilidad de crear la jerarquía del lenguaje de acuerdo al tipo de archivo que reconoce.

• Diagrama de secuencia

El siguiente diagrama de secuencia muestra la forma en la que se realiza el realzado de la sintaxis del código fuente.



• Errores conocidos en el módulo

El módulo no puede reconocer los comentarios en el código fuente, dado que el analizador léxico no reconoce a los comentarios como elementos léxicos.

• Trabajos futuros sobre el módulo

No se han definido aun trabajos futuros sobre el módulo.

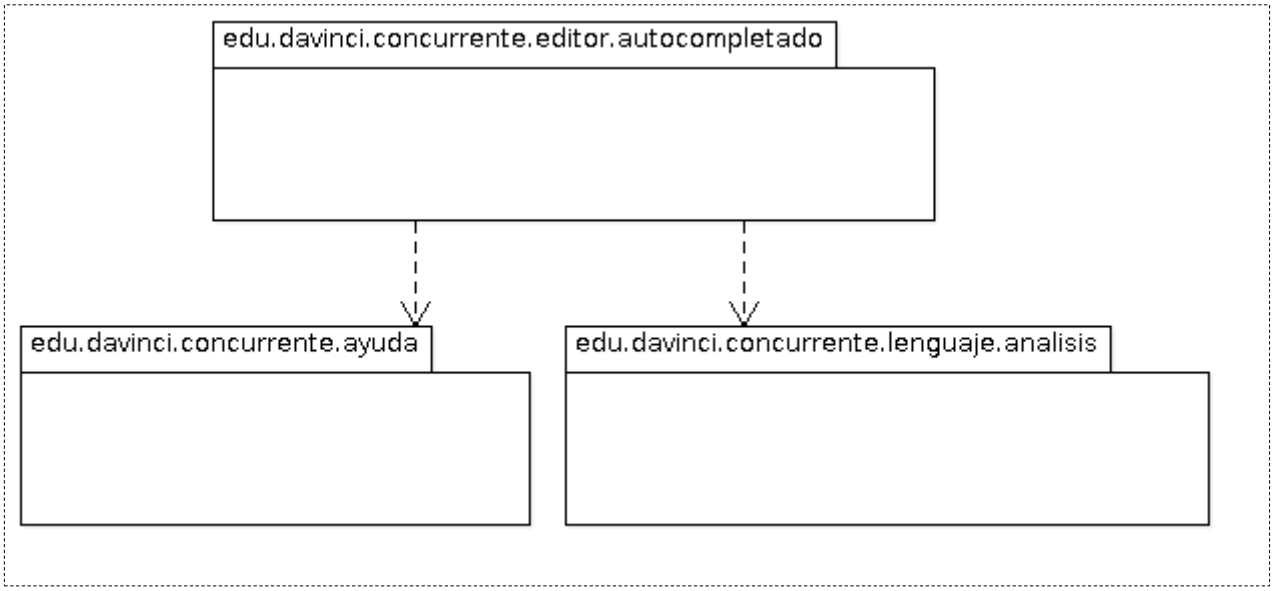
• DaVinciConcurrenteAutoCompletado

Este módulo se encarga de permitir al programador invocar el servicio de auto-completado de código, a través de una atajo de teclado. Extrae los elementos definidos en el lenguaje (palabras reservadas, variables, etc) y los presenta para su elección, junto con la documentación asociada.

Características implementadas por el módulo:

- Auto-completado

A continuación veremos algunos diagramas para profundizar más en el módulo.

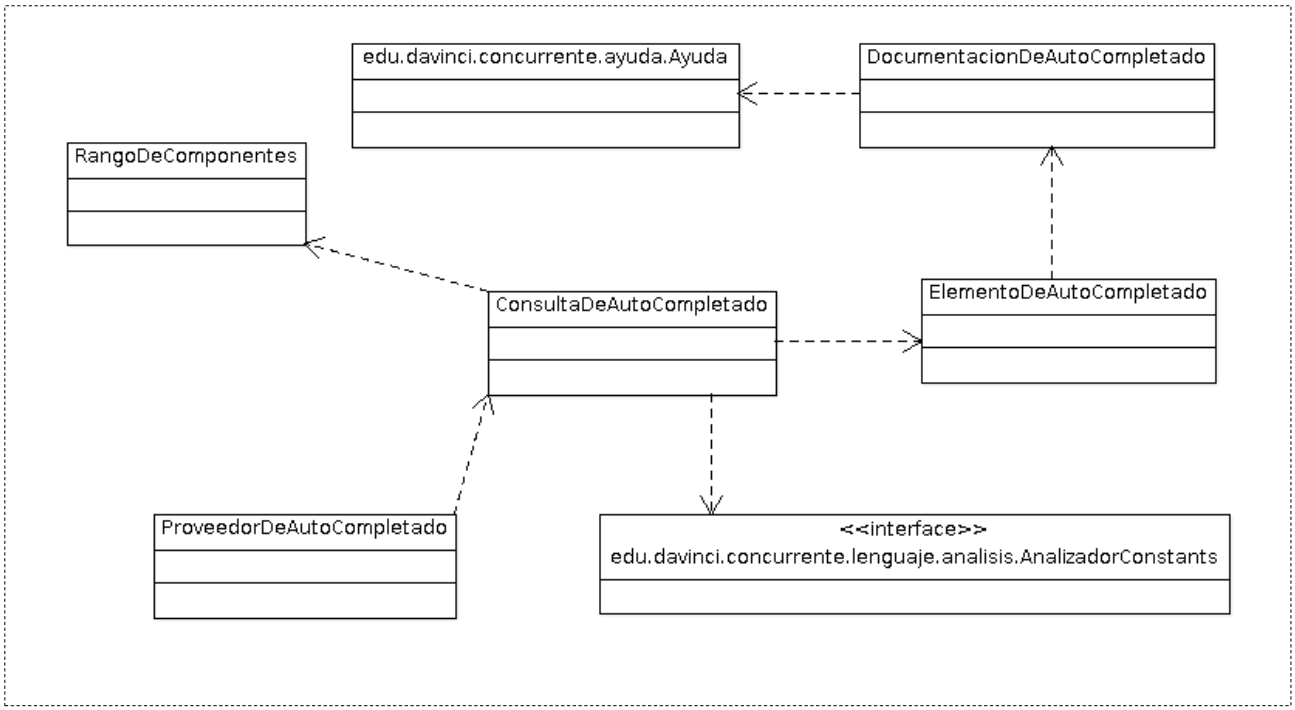


- Diagrama de dependencias

El siguiente diagrama muestra como este módulo depende del módulo de análisis del lenguaje, para extraer los elementos del mismo, y como depende de la ayuda para mostrar documentación de los elementos.

- Diagrama de clases

El siguiente diagrama de clases muestra las clases implementadas por el módulo:

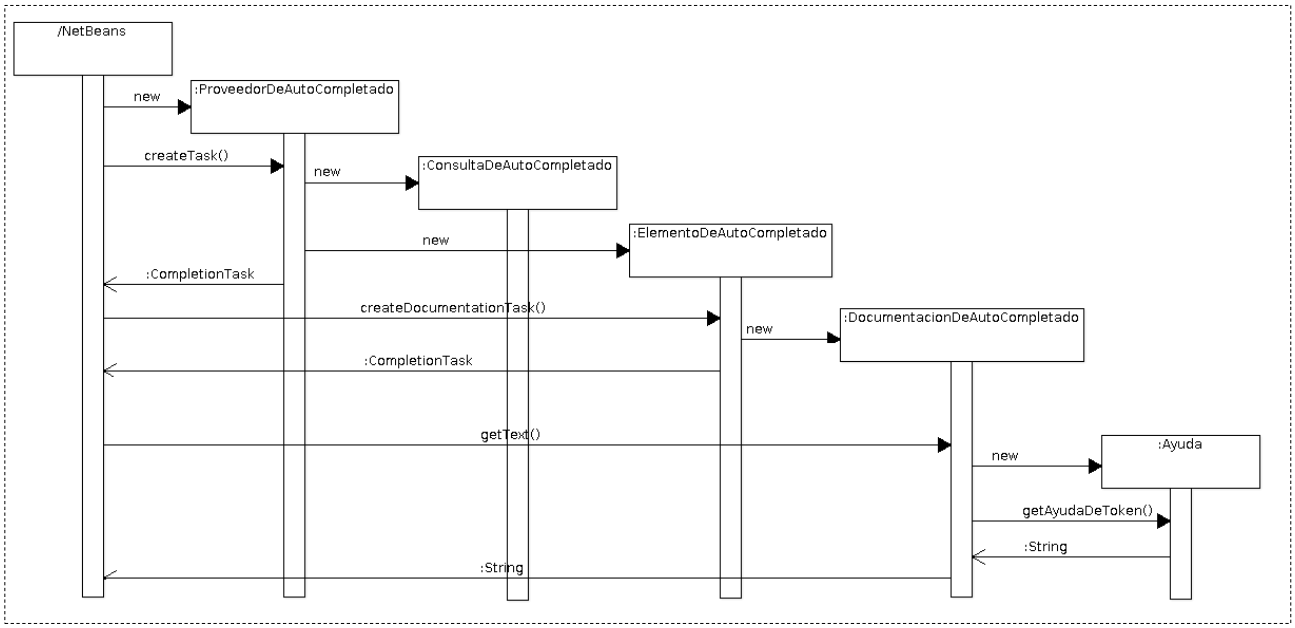


Las clases son:

- RangoDeComponentes: dado que los elementos del lenguaje se definen por números ordinales y se encuentran agrupados por categorías, esta clase permite representar cada uno de los rangos de elementos de cada categoría.
- ProveedorDeAutoCompletado: esta clase factoría permite la creación de la consulta de auto completado.
- ConsultaDeAutoCompletado: recibe la consulta de auto-completado, junto con el contexto actual, y retorna la lista de posibles candidatos a auto completar.
- ElementoDeAutoCompletado: representa un elemento de la lista de auto-completado.
- DocumentacionDeAutoCompletado: representa la documentación, asociada a un elemento de auto completado.

Diagrama de secuencia

En el siguiente diagrama de secuencia, se puede ver la forma en la que se realiza el auto completado de código.



Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

Trabajos futuros sobre el módulo

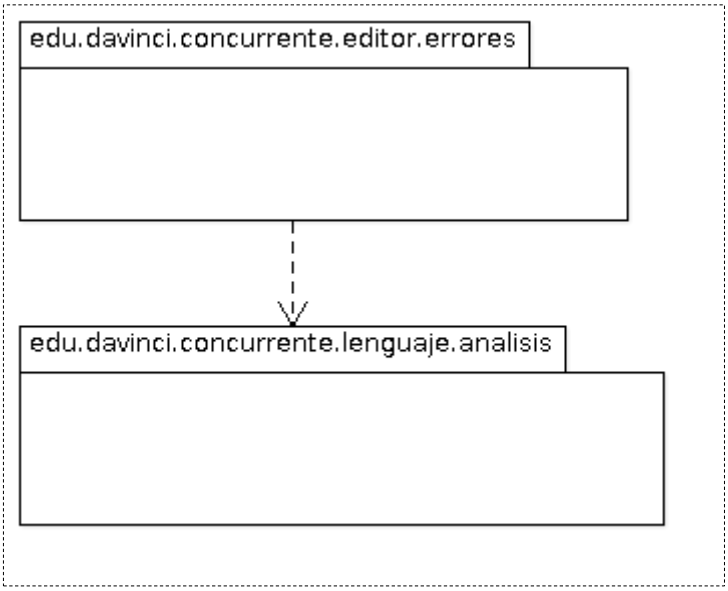
El auto completado actual, se realiza sobre los elementos estáticos del lenguaje (palabras reservadas, funciones predefinidas, etc). El módulo debe permitir también el auto completado de elementos definidos por el programador (variables, procesos, hilos, etc).

DaVinciConcurrenteErroresSintaxis

Este módulo, permite detectar errores en el código fuente del proyecto, mientras el programador esta escribiendo. No solo brinda la indicación del error, sino que también, información sobre las

posibles soluciones al mismo.

Características implementadas por el módulo:



- Demarcación de errores

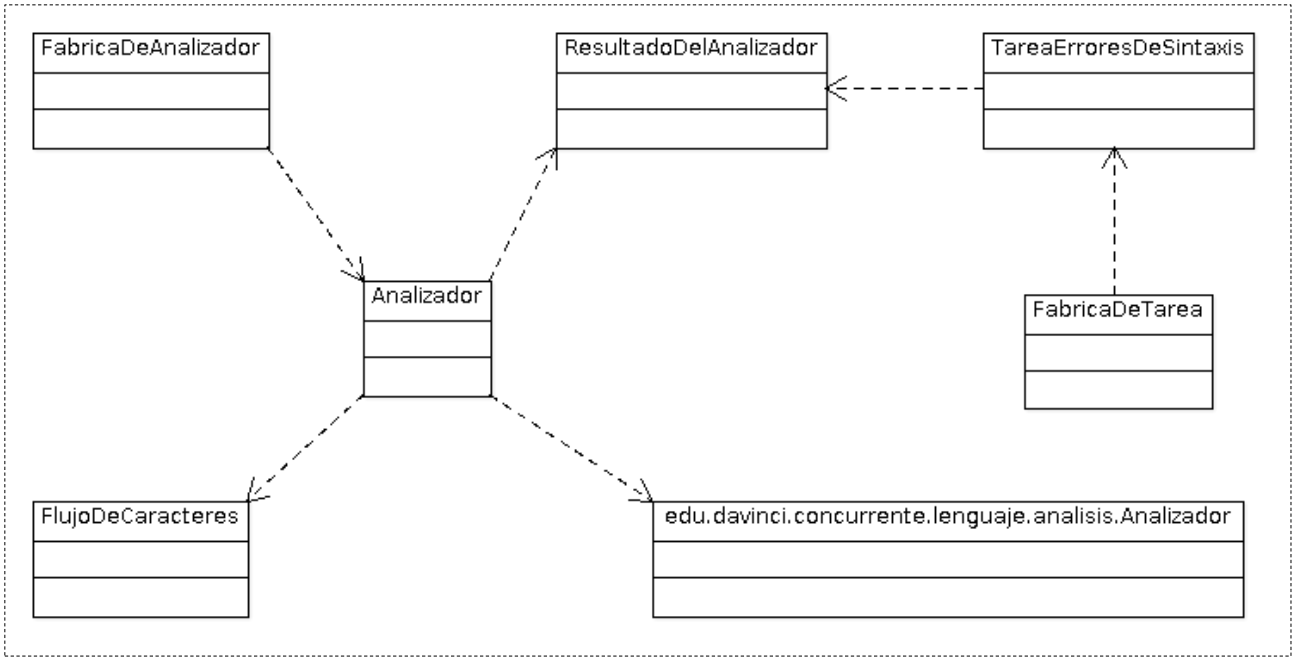
A continuación veremos algunos diagramas para profundizar más en el módulo.

- ## Diagrama de dependencias

El siguiente diagrama, muestra como este módulo depende del módulo de análisis del lenguaje.

- ## Diagrama de clases

En el siguiente diagrama, podemos ver las clases implementadas en el módulo y sus relaciones.

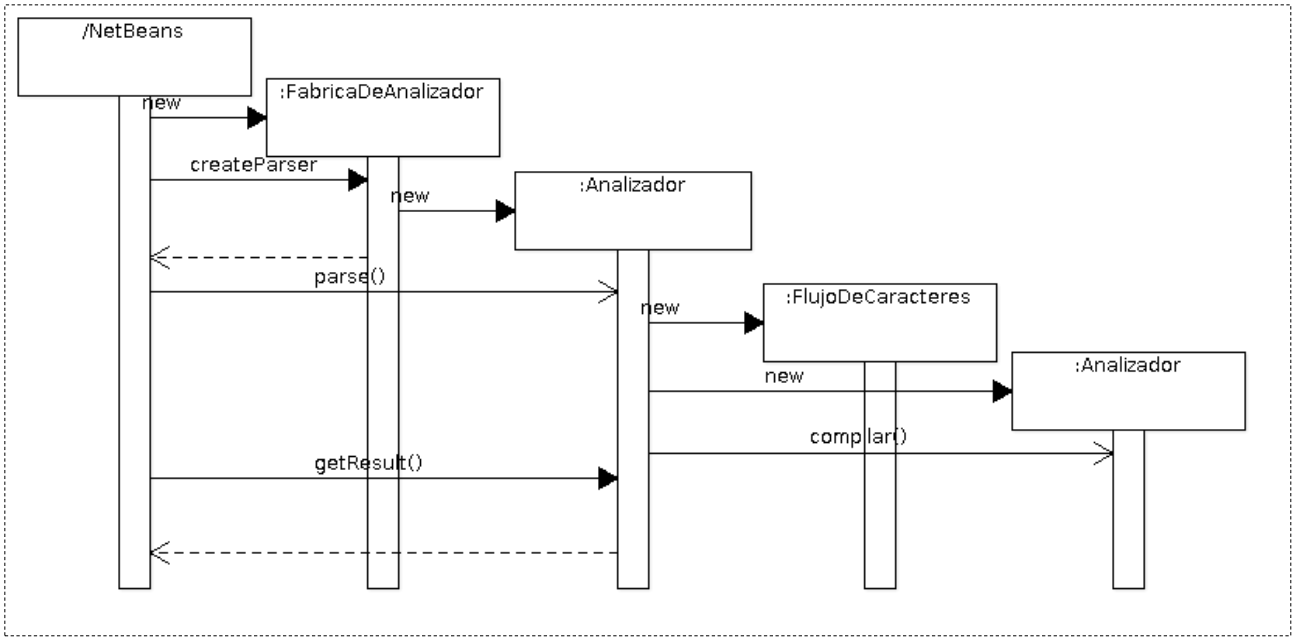


Las clases son:

- FabricaDeAnalizador:esta clase factoría, es la responsable de crear el analizador del código fuente.
- Analizador: es la clase responsable de realizar el análisis del código (a través del módulo de análisis del lenguaje) y retornar los resultados del mismo.
- ResultadoDeAnalizador: representa el resultado del análisis, conteniendo en el todos los errores encontrados.
- FabricaDeTarea: clase factoría que permite crear la tarea de demarcado de errores encontrados.
- TareaErroresDeSintaxis: tarea en segundo plano, que demarca los errores encontrados, a partir del resultado del análisis de código.

• Diagrama de secuencia

En el siguiente diagrama se puede ver como se realiza la tarea de detección de errores en el código.



- Errores conocidos del módulo

La detección de errores no se inicia, hasta que no se haya corrido el programa por lo menos una vez y de error de compilación.

En algunas ocasiones los errores detectados y corregidos, siguen demarcados hasta que se guarde el archivo de código fuente del proyecto.

- Trabajos futuros sobre el módulo

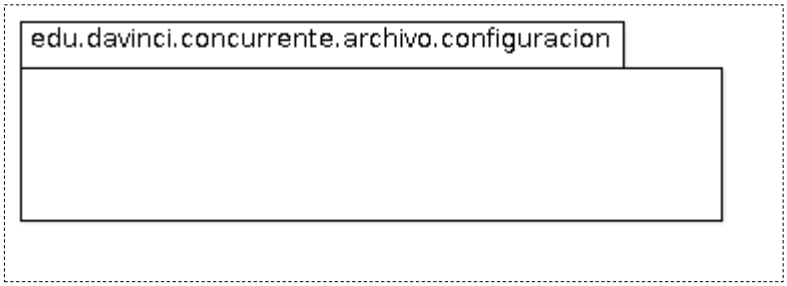
Por limitaciones del analizador de código, este módulo solo muestra el primer error encontrado en el código fuente. A futuro se estudiara la posibilidad y conveniencia de marcar más de un error al mismo tiempo dentro del código fuente.

- Módulos que brindan las características de la ejecución

En este apartado, iremos recorriendo cada uno de los módulos que implementan las características de la ejecución de programas, propuestas para este trabajo. De cada módulo, veremos que características implementa, de que módulos del proyecto depended, que clases implementa, cual es la responsabilidad de cada clase, alguna secuencia significativa del módulo, si existen errores conocidos del mismo, y los trabajos futuros que ya se han definido.

Los módulos son:

- DaVinciConcurrenteArchivoConfiguracion
- DaVinciConcurrenteArchivoConfiguracionInterprete
- DaVinciConcurrenteArchivoConfiguracionCiudad
- DaVinciConcurrenteArchivoConfiguracionEntradaSalida
- DaVinciConcurrenteInterprete
- DaVinciConcurrenteCiudad
- DaVinciConcurrenteEntradaSalida



- DaVinciConcurrenteEjecucion
- DaVinciConcurrenteEjecutor

• DaVinciConcurrenteArchivoConfiguracion

Este módulo implementa clases que son base, para cada uno de los módulos que definen los distintos archivos de configuración de un proyecto.

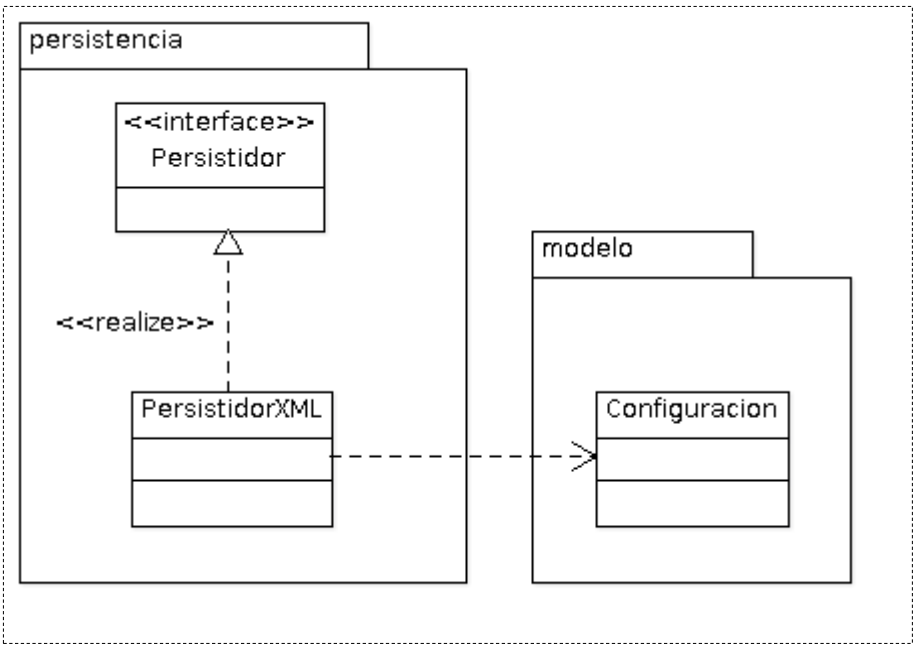
A continuación veremos algunos diagramas para profundizar más en el módulo.

• Diagrama de dependencias

El el siguiente diagrama, podemos ver que este módulo no depende de ningún otro paquete del presente proyecto.

• Diagrama de clases

En el siguiente diagrama, podemos ver las clases que implementa este módulo, y las relaciones entre las mismas.

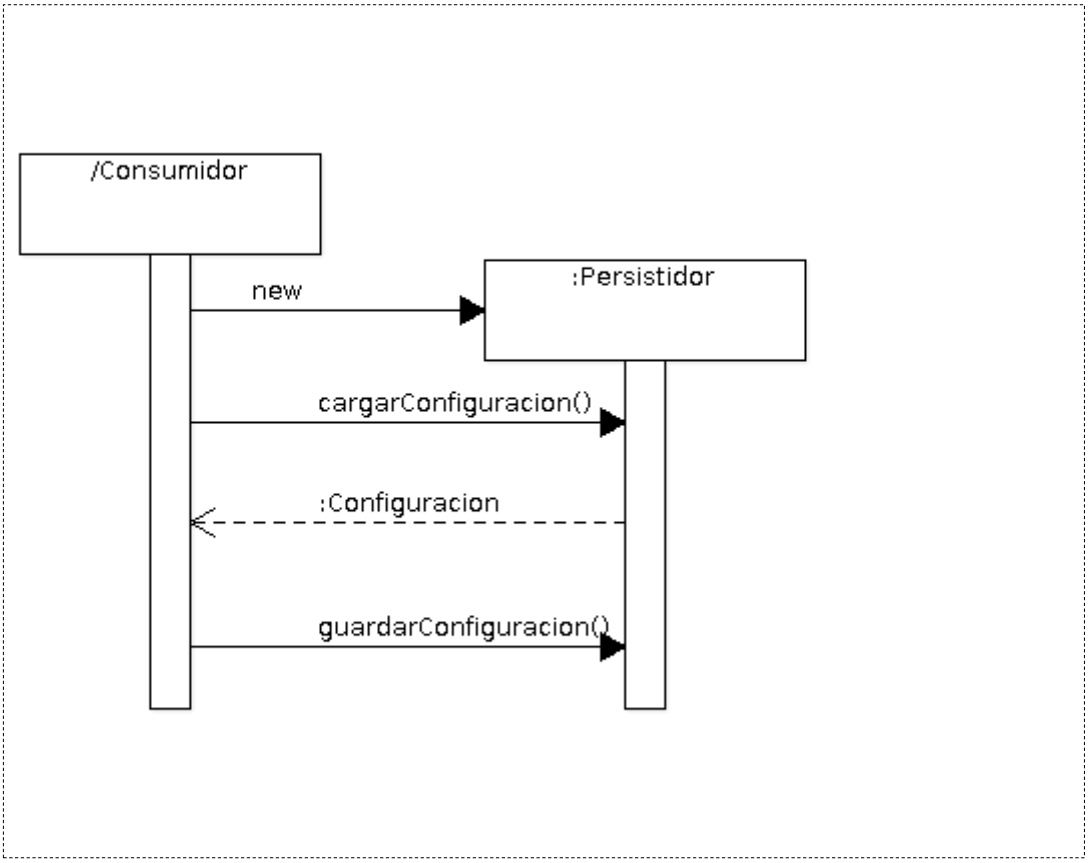


Las clases son:

- Persistidor: interface que define las operaciones mínimas que deben ser implementadas por cualquier persistidor de configuraciones.
- PersistidorXML: implementación de la interfaz Persistidor, que permite almacenar y recuperar en formato XML configuraciones.
- Configuracion: clase base que representa algún tipo de configuración.

• Diagrama de secuencia

En el siguiente diagrama podemos ver la forma en que se realiza la carga y almacenado de configuraciones.



- Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- Trabajos futuros sobre el módulo

No se han definido aun trabajos futuros sobre el módulo.

- DaVinciConcurrenteArchivoConfiguracionInterprete

Este módulo define e implementa las clases necesarias para realizar la configuración del interprete.

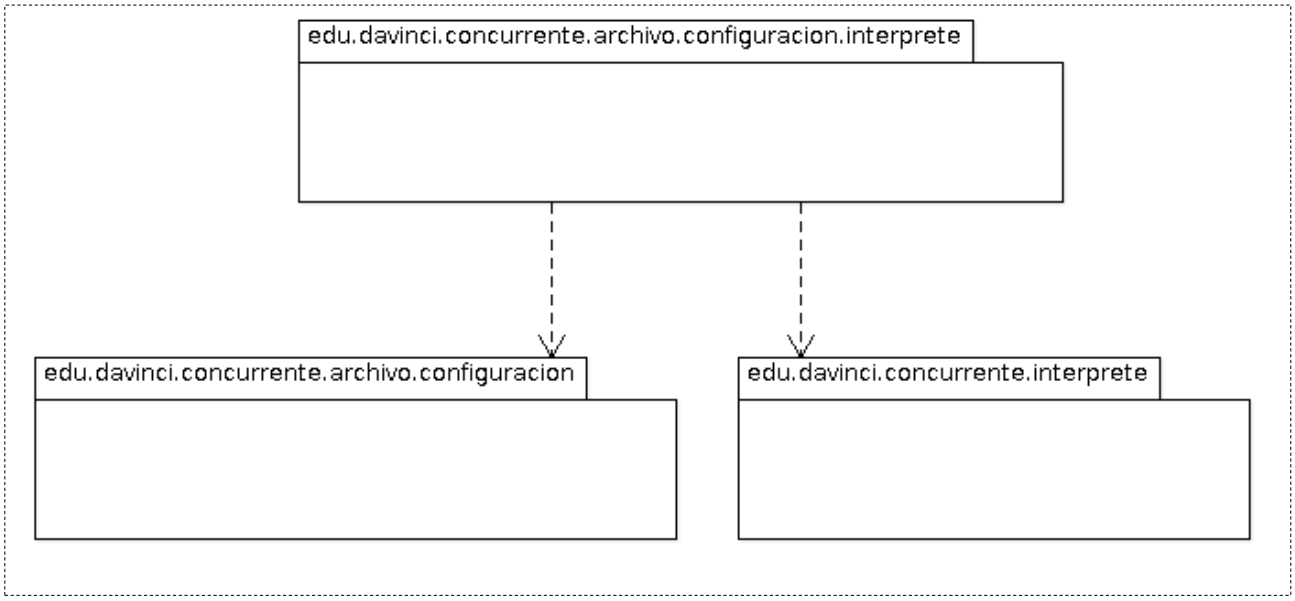
Dentro de las características que implementa este módulo, se encuentran:

- Configuración del interprete

A continuación veremos algunos diagramas para profundizar más en el módulo.

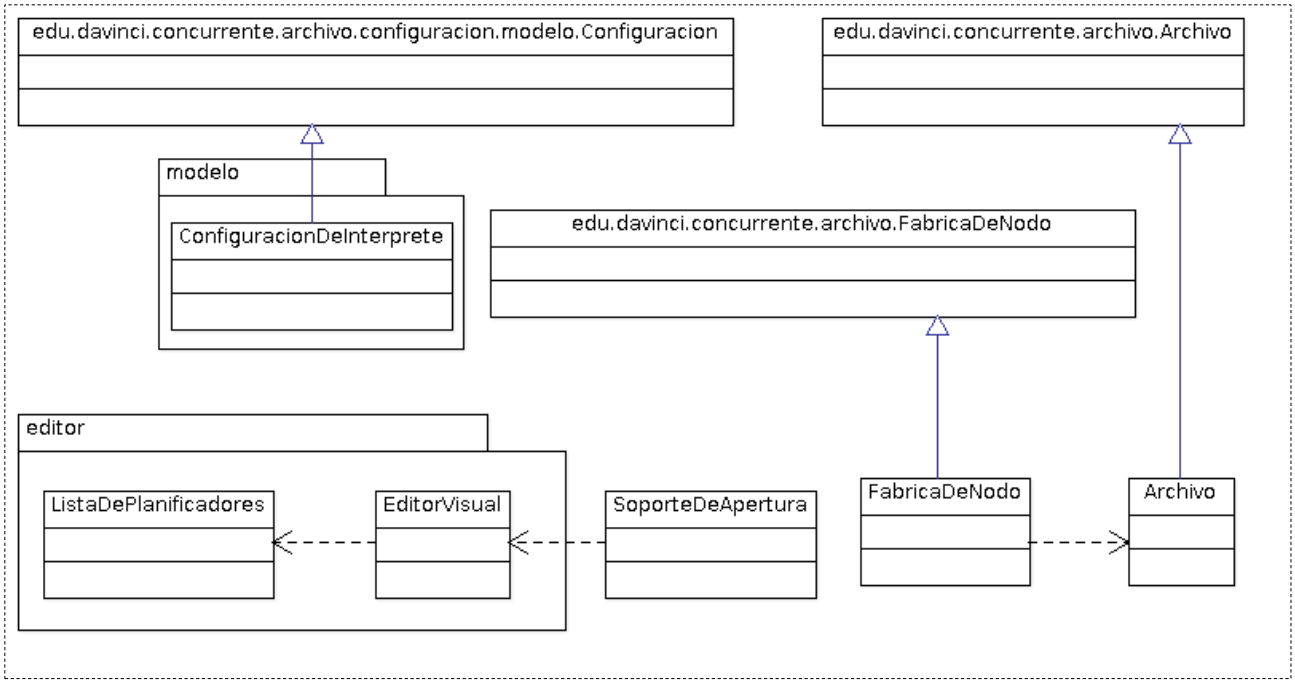
- Diagrama de dependencias

El siguiente diagrama permite visualizar que el módulo depende del módulo base de configuración y del módulo del interprete.



- Diagrama de clases

En el siguiente diagrama podemos visualizar las clases que implementa el módulo, sus relaciones, y relaciones con otras clases de los módulos de los que depende.



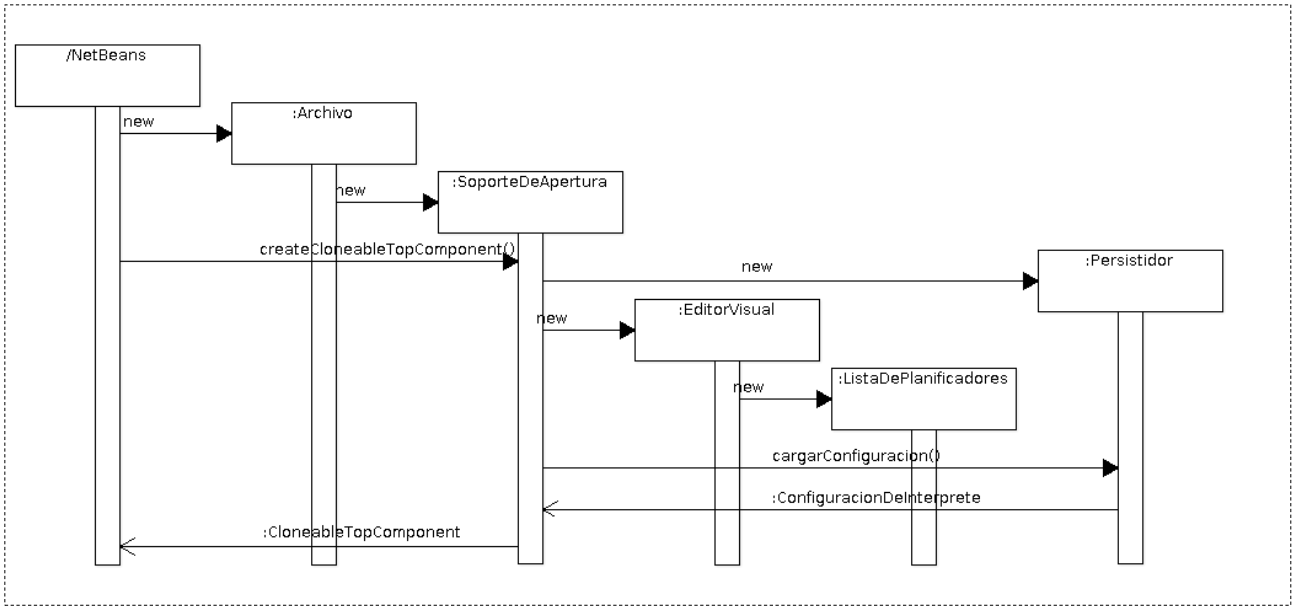
Las clases son:

- Archivo: clase que representa el archivo de configuración del interprete.
- FabricaDeNodo: clase responsable de la representación en forma de nodo del archivo dentro del proyecto.
- SoporteDeApertura: clase responsable de realizar la apertura del archivo y presentar el cuadro de dialogo de edición del mismo.
- EditorVisual: dialogo de edición de la configuración del interprete.
- ListaDePlanificadores: clase responsable de descubrir las implementaciones de planificadores presentes, y mostrarlas en forma de lista para la elección del usuario.

Cabe destacar que dentro de las opciones que permite configurar del interprete, se encuentra la elección del planificador de tareas a utilizar. Las implementaciones de planificadores disponibles, son descubiertas utilizando la ServiceLoader API de Java. Esto permite, que personas interesadas implementen nuevos planificadores y los puedan utilizar, sin modificar ninguna linea del código existente.

- ## Diagrama de secuencia

El siguiente diagrama de secuencia permite visualizar la forma en que se instancia a través del entorno, la pantalla de configuración del interprete.



- Errores conocidos del módulo

La única forma de guardar la configuración, es cerrando la ventana.

- Trabajos futuros sobre el módulo

La representación de configuraciones a través de nodos del proyecto, no ha sido la más adecuada. La misma se reemplazara a través de opciones, accesibles con el botón secundario sobre el nodo del proyecto.

Solo se permite elegir dentro de una lista, el planificador a utilizar. Debe permitirse también la configuración de otros parámetros dependientes del planificador. Por ejemplo, un planificador de cola circular, requiere especificar el tamaño del intervalo de tiempo (quantum) a utilizar.

- DaVinciConcurrenteArchivoConfiguracionCiudad

Este módulo define e implementa las clases necesarias para realizar la configuración de la ciudad.

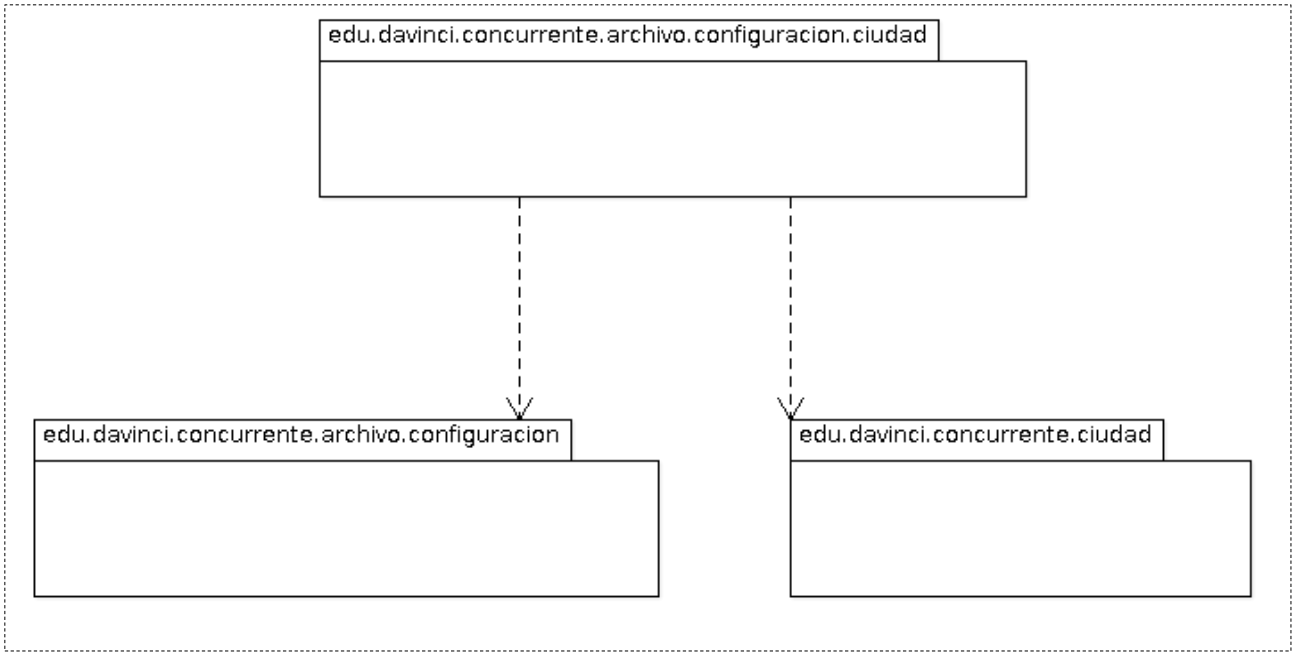
Dentro de las características que implementa este módulo, se encuentran:

- Configuración de la ciudad

A continuación veremos algunos diagramas para profundizar más en el módulo.

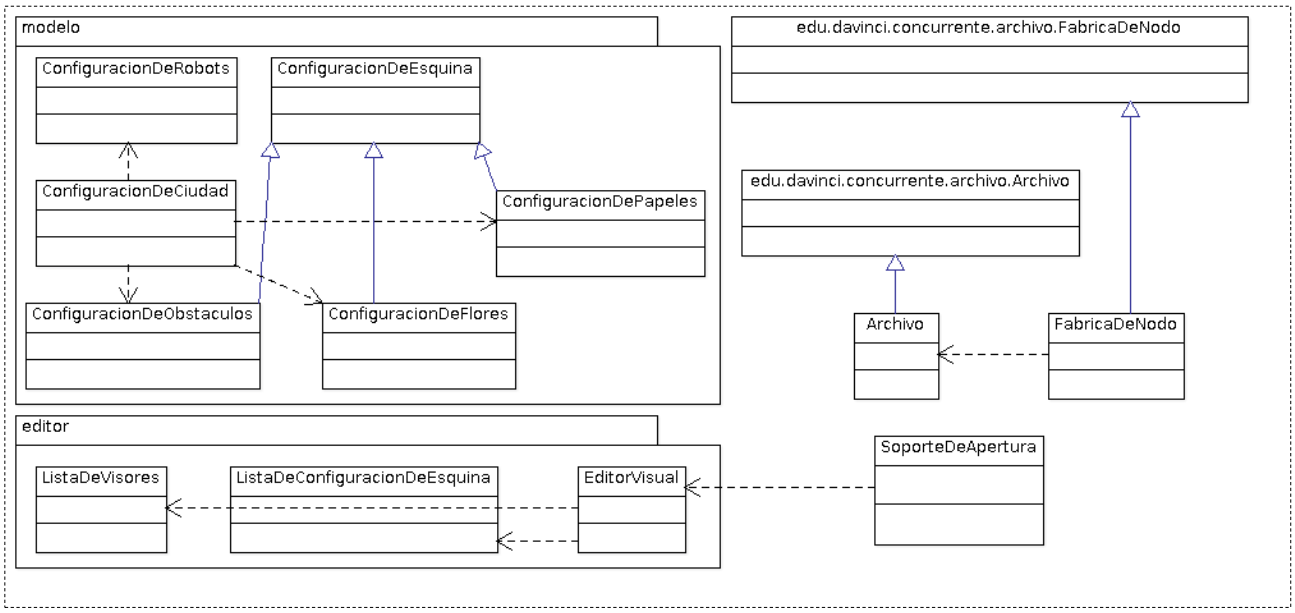
- Diagrama de dependencias

En el siguiente diagrama podemos observar como el presente módulo depende del módulo base de configuración y del módulo de la ciudad.



- Diagrama de clases

El siguiente diagrama de clases permite ver la clases fundamentales implementadas en el módulo, sus relaciones, y las relaciones con los módulos de los que depende.



Las clases son:

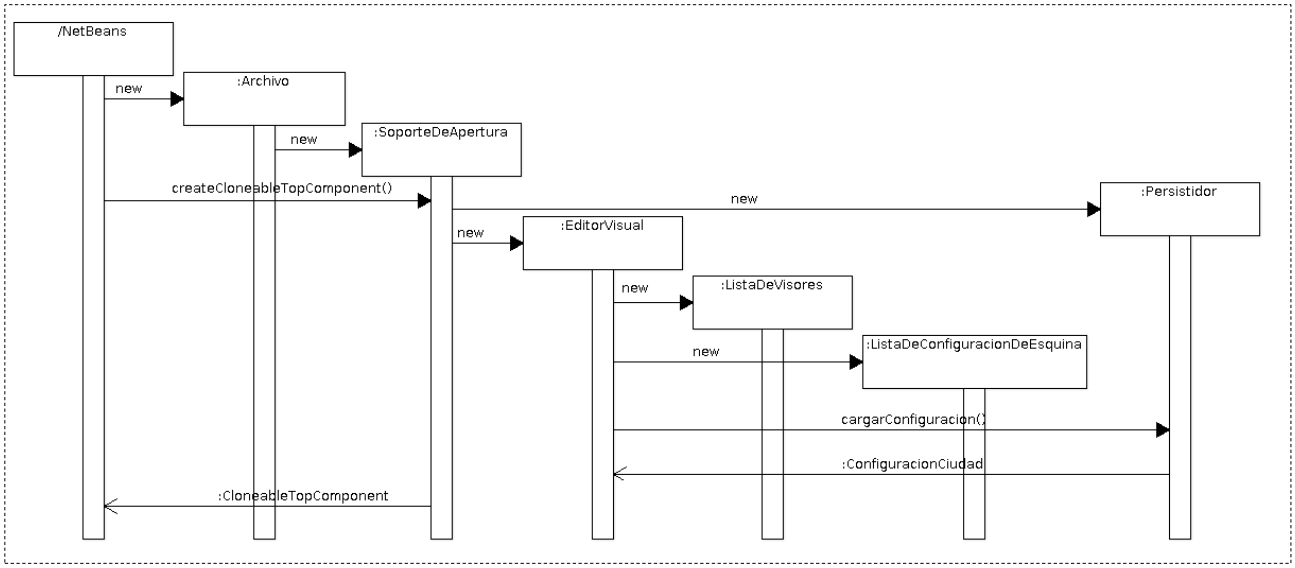
- ConfiguracionDeEsquina: clase base que representa la configuración de una esquina.
- ConfiguracionDePapeles: clase base que representa la configuración de los papeles de un esquina.
- ConfiguracionDeFlores: clase base que representa la configuración de las flores de un esquina.

- ConfiguracionDeObstaculos: clase base que representa la configuración de obstáculos de un esquina.
- ConfiguracionDeRobots: clase base que representa la configuración de robots.
- ConfiguracionDeCiudad: clase que representa la configuración de la ciudad.
- Archivo: clase que representa el archivo de configuración de la ciudad.
- FabricaDeNodo: clase responsable de la representación en forma de nodo del archivo de configuración de la ciudad.
- SoporteDeApertura: clase responsable de la apertura del cuadro de dialogo de configuración de la ciudad.
- ListaDeVisores: clase responsable de descubrir los visores de ciudad implementados y presentarlos en forma de lista para su elección.
- ListaDeConfiguracionesDeEsquina: clase responsable de mostrar las distintas configuraciones de esquinas en forma de lista.
- EditorVisual: cuadro de dialogo que permite la configuración de la ciudad.

Cabe destacar que dentro de las opciones que permite configurar de la ciudad, se encuentra la elección de los visores de ciudad a utilizar. Las implementaciones de visores disponibles, son descubiertas utilizando la ServiceLoader API de Java. Esto permite, que personas interesadas implementen nuevos visores y los puedan utilizar, sin modificar ninguna linea del código existente.

• Diagrama de secuencia

El siguiente diagrama de secuencia permite visualizar la forma en que se instancia a través del entorno, la pantalla de configuración de la ciudad.



• Errores conocidos del módulo

La única forma de guardar la configuración, es cerrando la ventana.

- Trabajos futuros sobre el módulo

La representación de configuraciones a través de nodos del proyecto, no ha sido la más adecuada. La misma se reemplazara a través de opciones, accesibles con el botón secundario sobre el nodo del proyecto.

Se debe permitir la configuración aleatoria de flores, papeles y obstáculos, en cuadrantes indicados.

- DaVinciConcurrenteArchivoConfiguracionEntradaSalida

Este módulo define e implementa las clases necesarias para realizar la configuración de la entrada y salida en tiempo de ejecución.

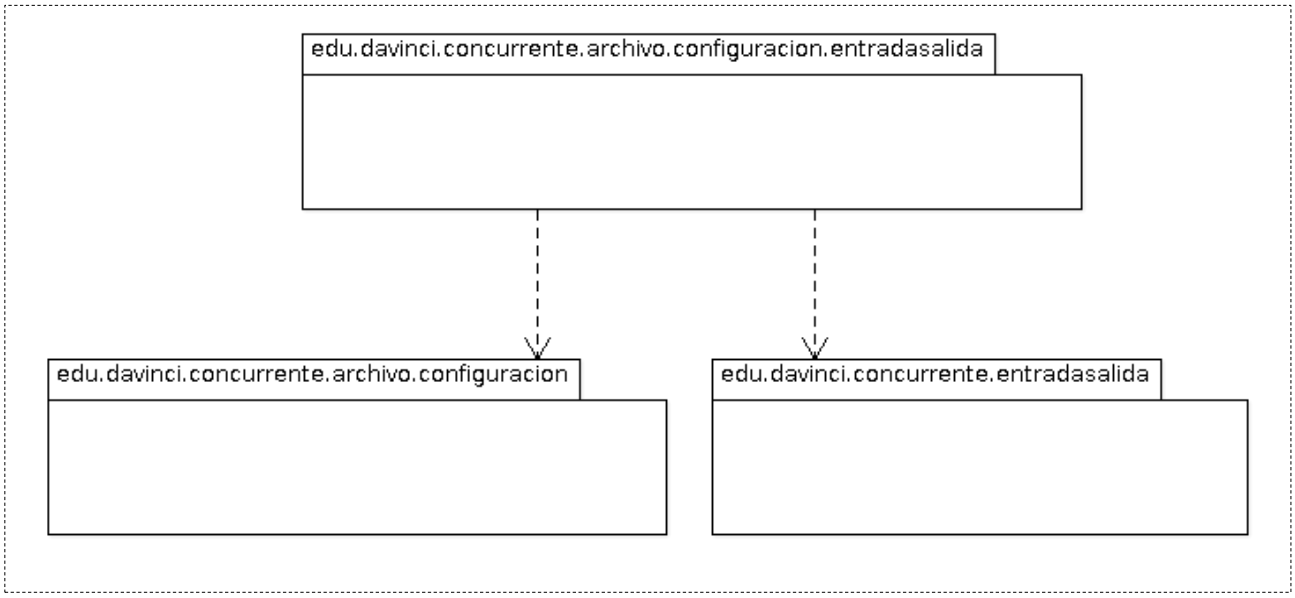
Dentro de las características que implementa este módulo, se encuentran:

- Configuración de la entrada salida

A continuación veremos algunos diagramas para profundizar más en el módulo.

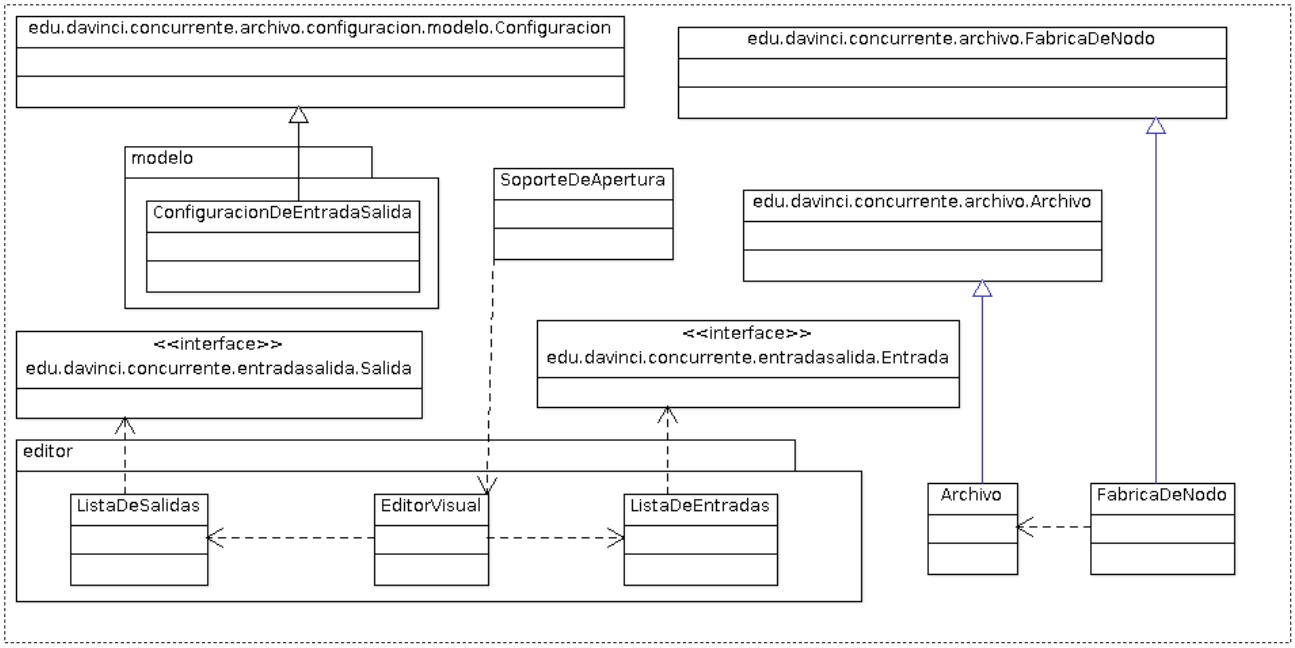
- Diagrama de dependencias

El siguiente diagrama permite visualizar que el módulo depende del módulo base de configuración y del módulo de entrada-salida.



- Diagrama de clases

El siguiente diagrama de clases permite ver la clases fundamentales implementadas en el módulo, sus relaciones, y las relaciones con los módulos de los que depende.



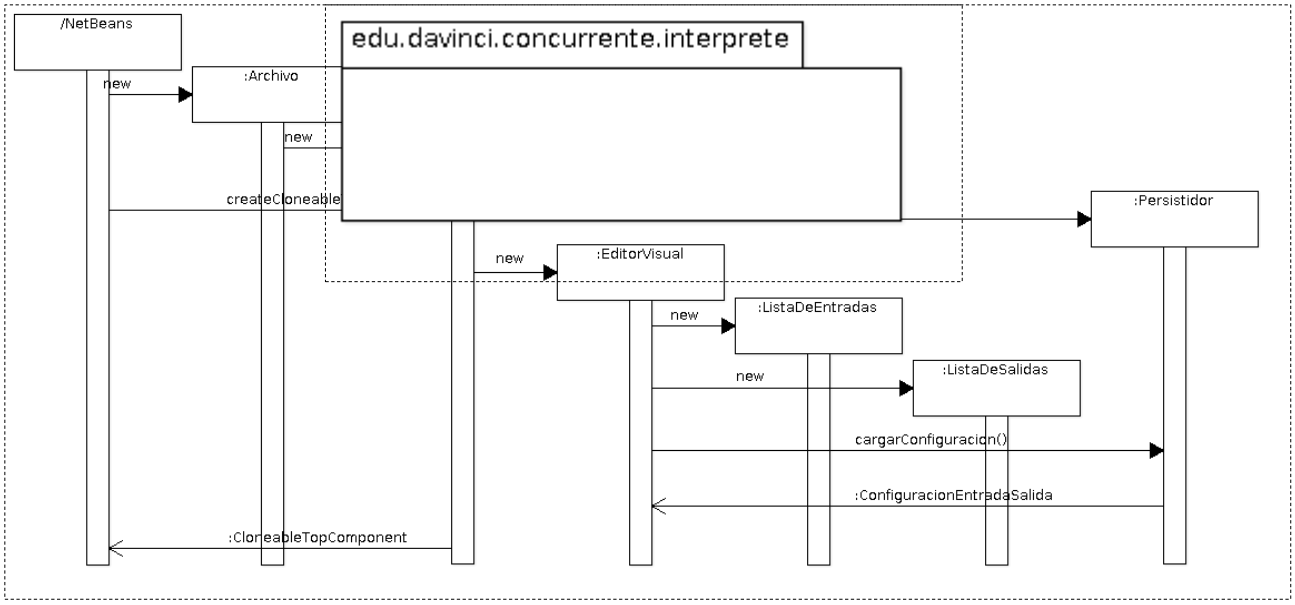
Las clases son:

- ConfiguracionDeEtradaSalida: clase que representa la configuración de entrada-salida.
- Archivo: clase que representa el archivo de configuración de la ciudad.
- FabricaDeNodo: clase responsable de la representación en forma de nodo del archivo de configuración de entrada-salida.
- SoporteDeApertura: clase responsable de la apertura del cuadro de dialogo de configuración de entrada-salida.
- ListaDeEntradas: clase responsable de descubrir implementaciones de entrada disponibles.
- ListaDeSalidas: clase responsable de descubrir implementaciones de salida disponibles.
- EditorVisual: cuadro de dialogo que permite la configuración de entrada-salida.

Cabe destacar que dentro de las opciones que permite configurar de entrada-salida, se encuentra la elección de las implementaciones a utilizar. Las implementaciones de entrada y de salida disponibles, son descubiertas utilizando la ServiceLoader API de Java. Esto permite, que personas interesadas implementen nuevos canales y los puedan utilizar, sin modificar ninguna linea del código existente.

• Diagrama de secuencia

El siguiente diagrama de secuencia, muestra la forma en que es instanciado el cuadro de dialogo de configuración de entrada-salida.



- Errores conocidos del módulo

La única forma de guardar la configuración, es cerrando la ventana.

- Trabajos futuros sobre el módulo

La representación de configuraciones a través de nodos del proyecto, no ha sido la más adecuada. La misma se reemplazara a través de opciones, accesibles con el botón secundario sobre el nodo del proyecto.

- DaVinciConcurrenteInterprete

Este módulo simplemente empaqueta, el interprete del lenguaje DaVinci Concurrente desarrollado por Daniel Aguil Mallea, de forma conveniente para ser utilizado dentro de la plataforma NetBeans RCP.

A continuación veremos algunos diagramas para profundizar más en el módulo.

- Diagrama de dependencias

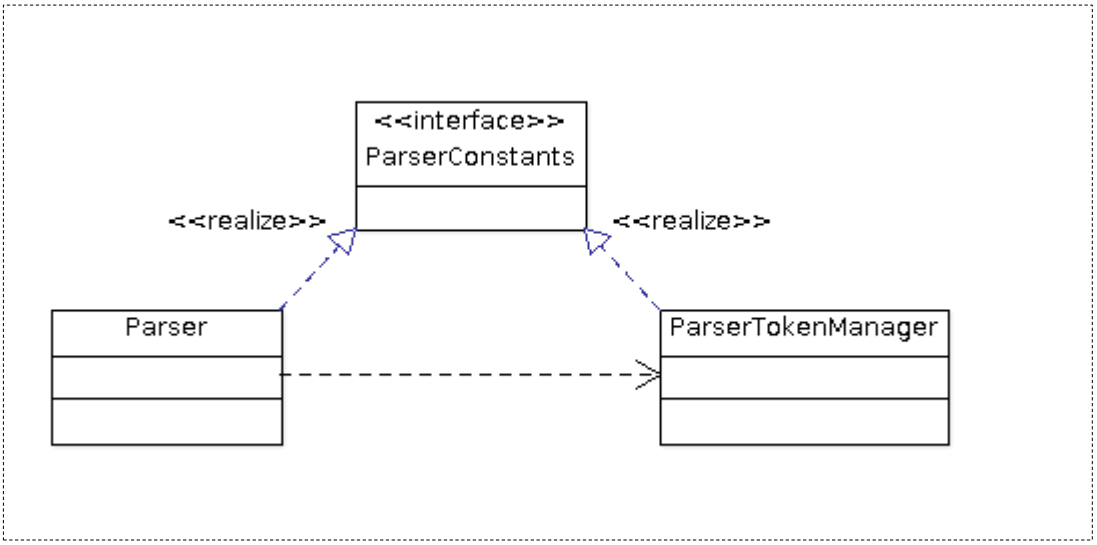
El siguiente diagrama muestra que este módulo, no depende de ningún otro de los módulos del proyecto.

- Diagrama de clases

Si bien este módulo contiene un montón de clases, todas ellas son generadas por la herramienta JavaCC a través de la definición de la gramática del lenguaje.

En el siguiente diagrama, solo se representan las tres clases principales generadas y sus relaciones.

Las

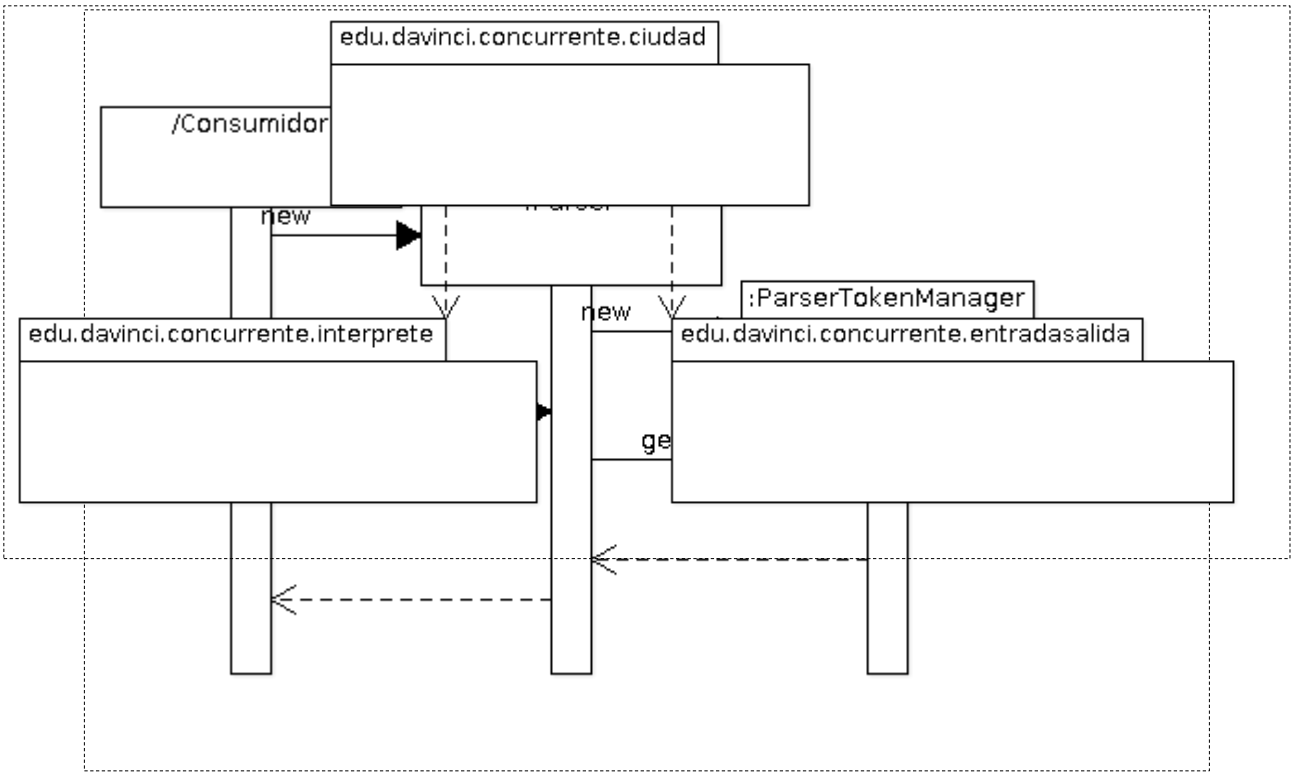


clases son:

- ParserConstants: esta clase contiene definiciones de constantes, principalmente de cada elemento léxico del lenguaje.
- ParserTokenManager: esta clase es responsable de realizar el análisis léxico del código fuente y la división del mismo en elementos léxicos.
- Parser: esta clase es responsable de realizar el análisis sintáctico del código fuente, permite también la detección de errores en el mismo.

- Diagrama de secuencia

El siguiente diagrama de secuencia muestra de forma simplificada la forma en que el analizador realiza el análisis del código fuente.



- Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- Trabajos futuros sobre el módulo

Este módulo debe ser fusionado con el módulo de análisis del lenguaje.

- DaVinciConcurrenteCiudad

Este módulo implementa las clases necesarias para representar cada uno de los elementos de la ciudad, y permitir la visualización de los mismos, y sus variaciones en la ejecución de proyectos.

Dentro de las características que implementa este módulo, se encuentran:

- Visualización de la ejecución

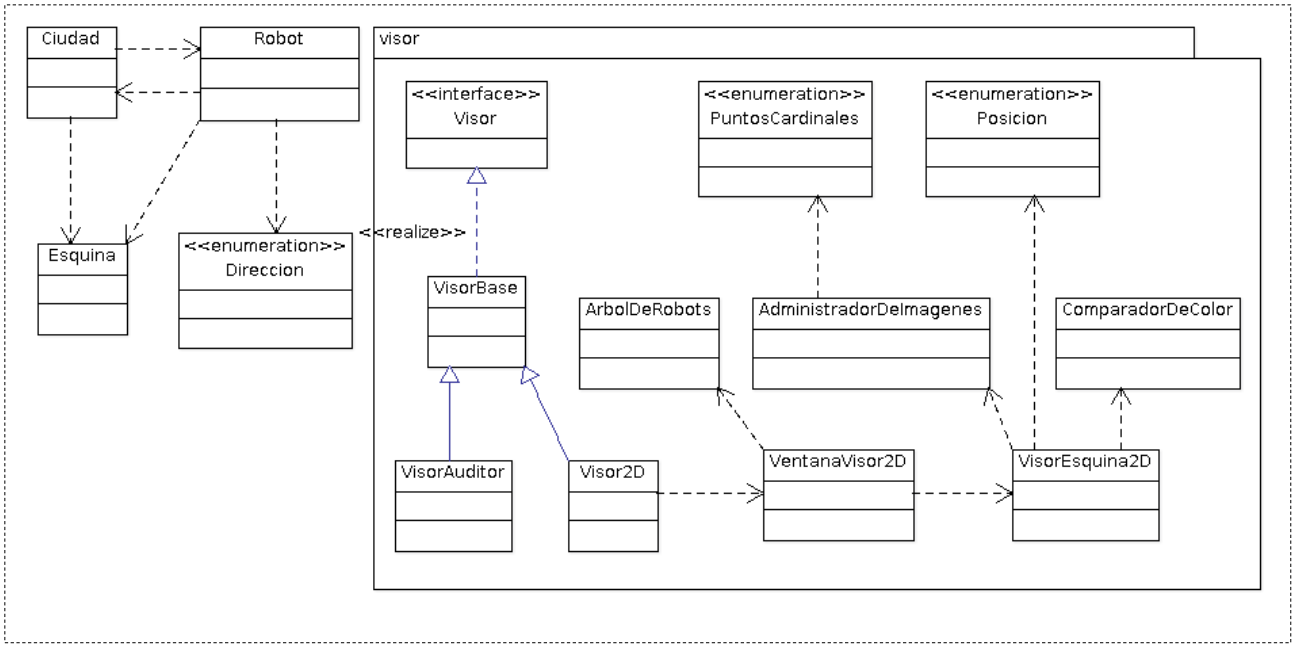
A continuación veremos algunos diagramas para profundizar más en el módulo.

- Diagrama de dependencias

El siguiente diagrama de dependencias permite ver como este módulo depende tanto del módulo del interprete, como así también del módulo de entrada-salida.

- Diagrama de clases

En el siguiente diagrama podemos ver las clases implementadas por el módulo, y las relaciones entre las mismas:

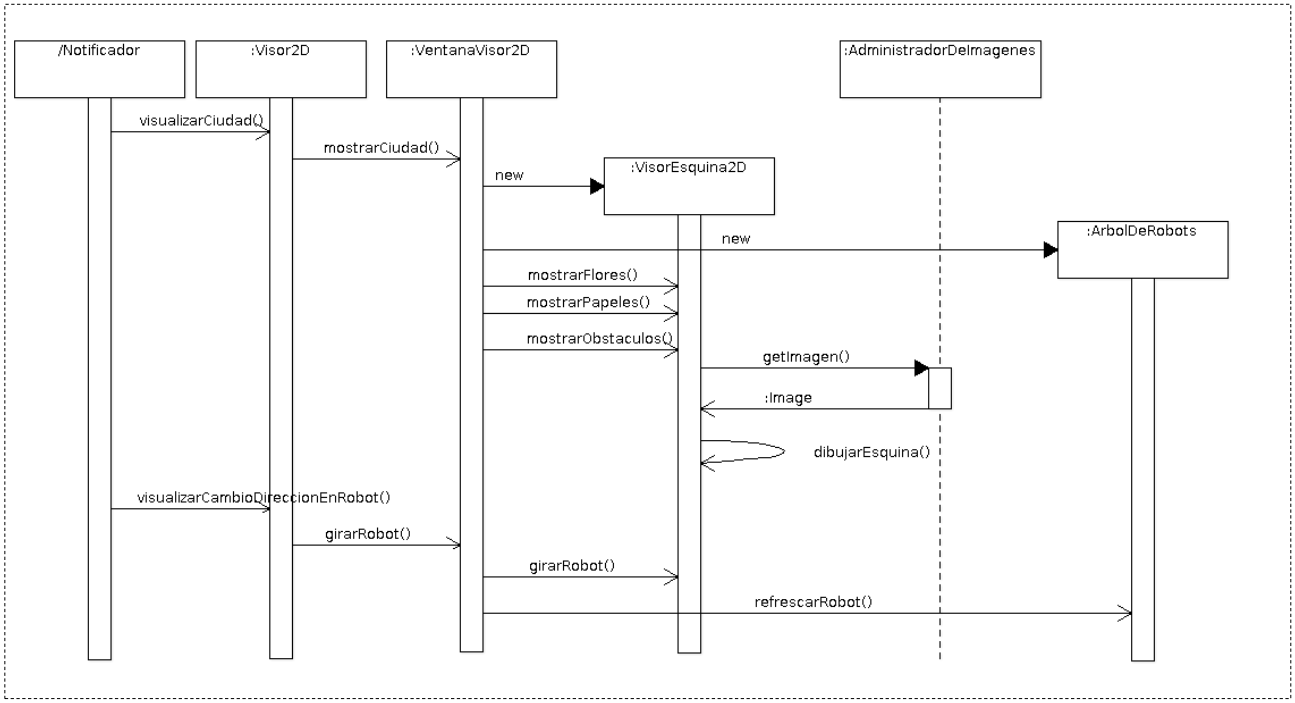


Las clases son:

- Direccion: enumeración que representa la dirección u orientación (norte, sur, etc) actual de un robot.
- Esquina: clase que representa el estado de una esquina de la ciudad.
- Robot: clase que representa un robot.
- Ciudad: clase que representa la ciudad.
- Visor: clase que representa una forma de visualización sobre la ciudad.
- VisorBase: clase de utilidad, que brinda mayor facilidad de implementar visores de la ciudad.
- VisorAuditor: clase que permite la visualización de la ciudad y su actividad en forma de auditor.
- Visor2D: clase que permite la visualización de la ciudad y su actividad en forma gráfica de dos dimensiones.
- VentanaVisor2D: ventana que enmarca la visualización de la ciudad en forma gráfica.
- ArbolDeRobots: parte de la visualización gráfica de la ciudad, que muestra el esta de cada uno de los robots en forma de árbol.
- VisorEsquina2D: parte de la visualización gráfica de la ciudad, que muestra los acontecimientos de la ciudad.
- PuntosCardinales: enumeración que representa cada uno de los puntos cardinales, por los que pasa el robot, al girar.
- Posicion: enumeración que representa cada una de las posiciones por las que pasa un robot cuando se mueve.
- AdministradorDeImagenes: clase que sirve de cache de imágenes generadas en la representación gráfica.
- ComparadorDeColor: clase que permite compara colores.

- Diagrama de secuencia

El siguiente diagrama de secuencia, muestra la interacción con el visualizador gráfico de la ciudad.



- Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- Trabajos futuros sobre el módulo

No se han definido aun trabajos futuros sobre el módulo.

- DaVinciConcurrenteEntradaSalida

Este módulo implementa las clases necesarias para la entrada y salida desde un proyecto en tiempo de ejecución.

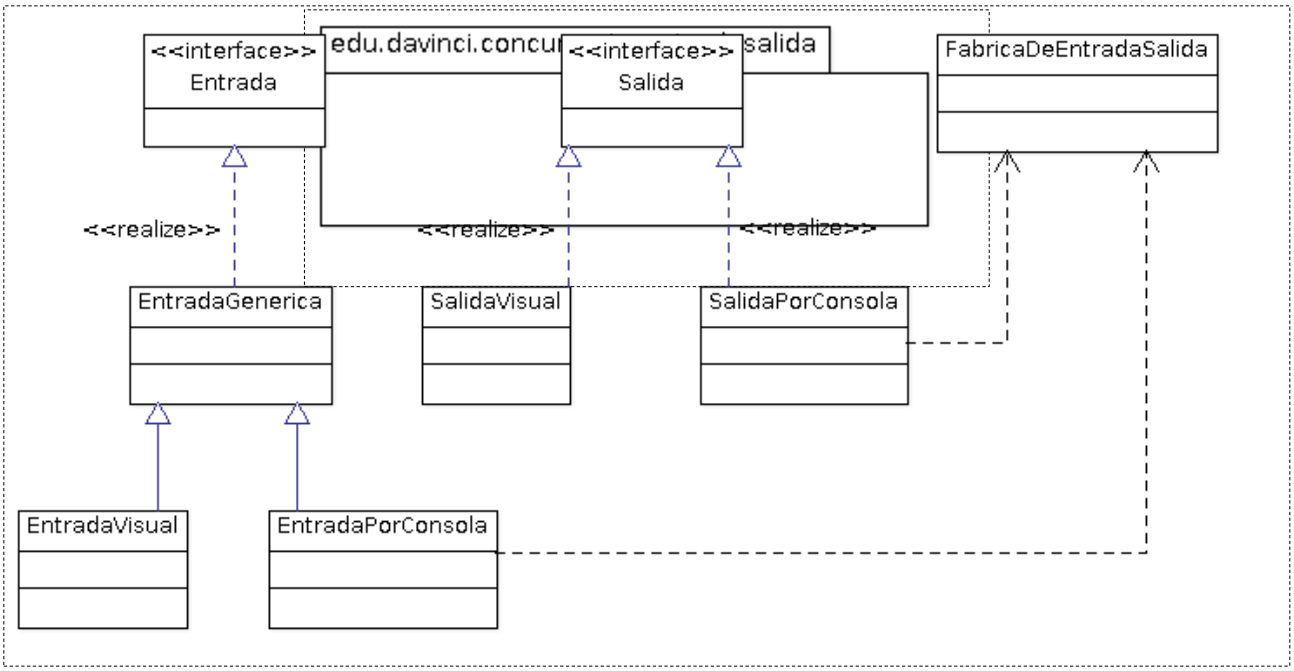
Dentro de las características que implementa este módulo, se encuentran:

- Visualización de la ejecución

A continuación veremos algunos diagramas para profundizar más en el módulo.

- Diagrama de dependencias

El siguiente diagrama muestra que este módulo, no depende de ningún otro de los módulos del proyecto.



- Diagrama de clases

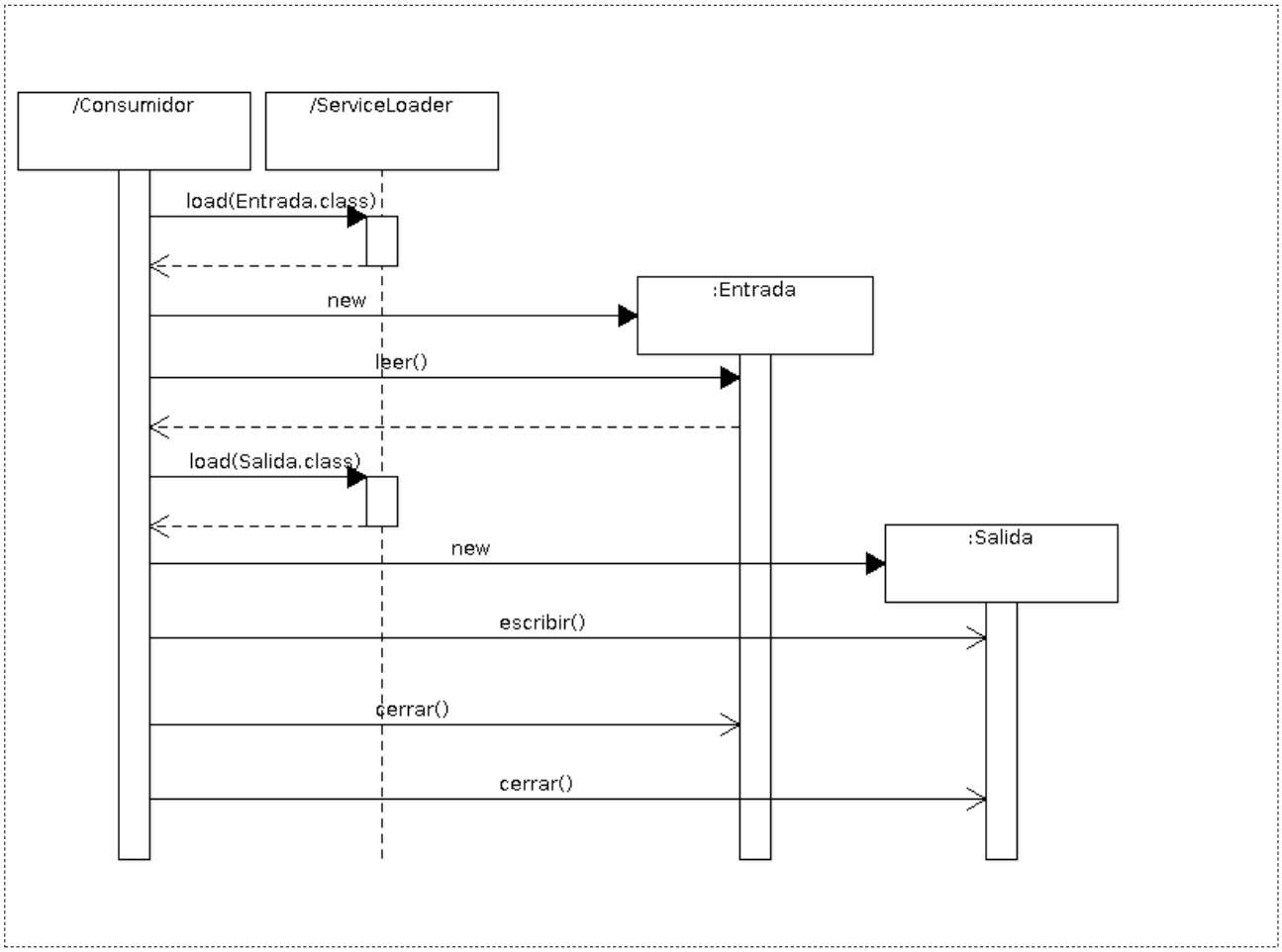
En el siguiente diagrama podemos ver las clases implementadas por el módulo, y las relaciones entre las mismas:

Las clases son:

- Entrada: interface que define los métodos necesarios para cualquier implementación de una entrada.
- EntradaGenerica: clase base de implementación de entrada.
- EntradaVisual: implementación de entrada a través de cuadros de dialogo.
- EntradaPorConsola: implementación de entrada a través de consola de comandos.
- Salida: interface que define los métodos necesarios para cualquier implementación de una salida.
- SalidaVisual: implementación de salida a través de cuadros de dialogo.
- SalidaPorConsola: implementación de salida a través de consola de comandos.
- FabricaDeEntradaSalida: clase factoría que permite obtener, las entrada y salidas estándares del entorno.

- Diagrama de secuencia

El siguiente diagrama de secuencia permite visualizar la forma en que son descubiertas las implementaciones de entrada y salida, y la instanciación de las mismas.



- Errores conocidos del módulo

No se han encontrado aun errores en el módulo.

- Trabajos futuros sobre el módulo

No se han definido aun trabajos futuros sobre el módulo el módulo.

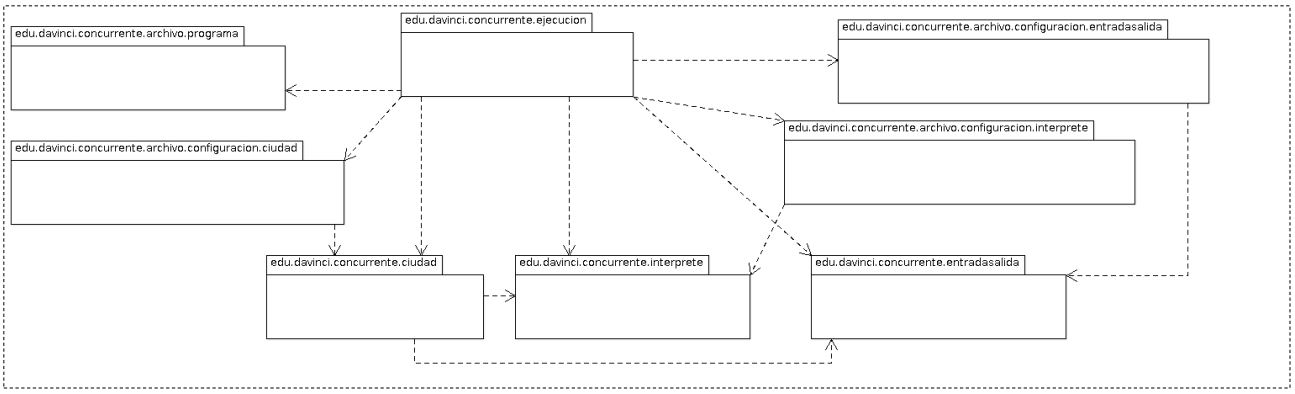
- DaVinciConcurrenteEjecucion

Este módulo es el encargado de aplicar las configuraciones establecidas por el usuario para el interprete, la ciudad y la entrada-salida. Sus servicios son utilizados tanto por el módulo de ejecución de proyectos, como por el módulo de depuración.

A continuación veremos algunos diagramas para profundizar más en el módulo.

- Diagrama de dependencias

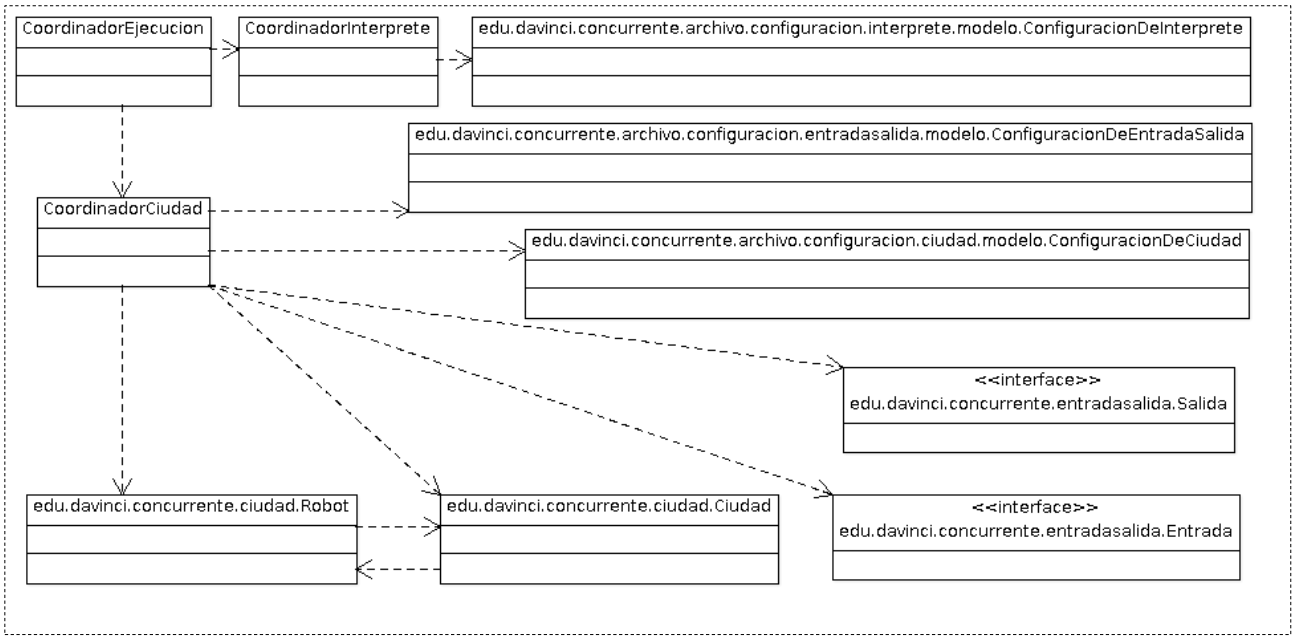
En el siguiente diagrama, podemos ver que el módulo depende de todos los módulos que



implementan el interprete, la ciudad, la entrada-salida, y los módulos que implementan la configuración de los mismos.

- Diagrama de clases

En el siguiente diagrama podemos ver las clases que implementa el módulo y las interacciones con las clases de los módulos de los que depende.

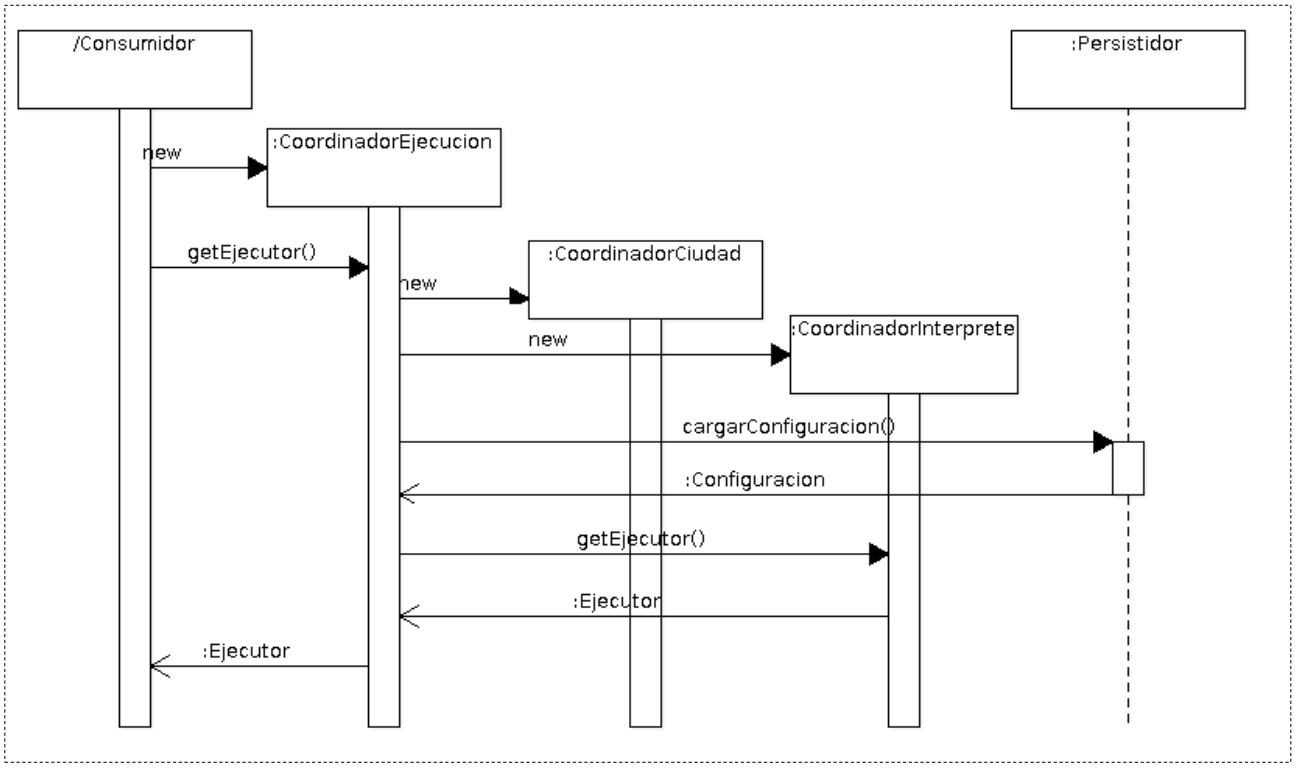


Las clases son:

- CoordinadorDeCiudad: esta clase se encarga de instanciar todas las clases relacionadas con la ciudad y la entrada-salida, y de aplicar la configuración de las mismas.
- CoordinadorDeInterprete: esta clase se encarga de instanciar toas las clases relacionadas con el interprete y de aplicar su configuración.
- CoordinadorDeEjecucion: esta clase se encarga de instanciar al resto de los coordinadores y proveer un entorno listo para la ejecución de proyectos.

- Diagrama de secuencia

En el siguiente diagrama se puede visualizar la forma en que se coordinan las tareas entre las clases del módulo.



- Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- Trabajos futuros sobre el módulo

No se han definido aun trabajos futuros sobre el módulo.

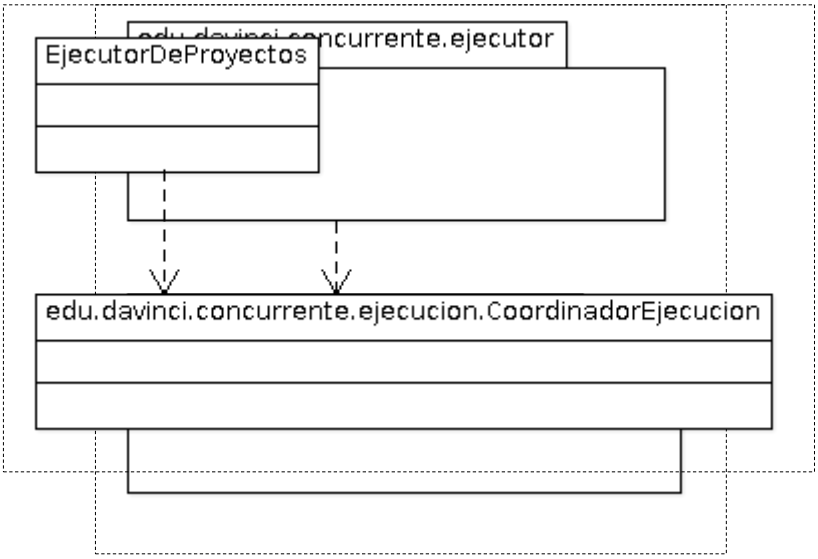
- DaVinciConcurrenteEjecutor

Este módulo es el encargado de realizar la ejecución de proyectos, utilizando los servicios del módulo de ejecución, del cual depende.

A continuación veremos algunos diagramas para profundizar más en el módulo.

- Diagrama de dependencias

En el siguiente diagrama podemos visualizar como este módulo depende del módulo de ejecución.



- Diagrama de clases

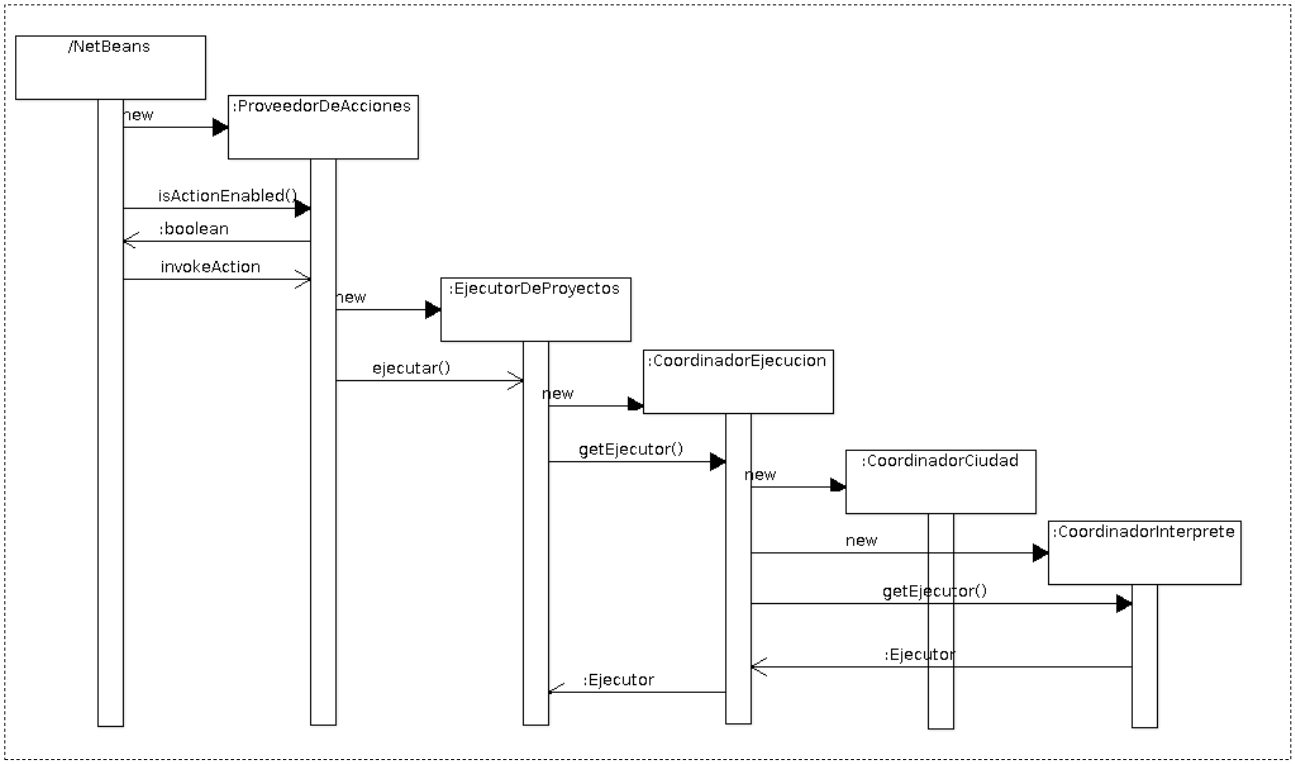
En este diagrama podemos visualizar la única clase que implementa este módulo y su interacción con una clase del módulo del cual depende.

Las clase que implementa, es:

- EjecutorDeProyectos: esta clase es la encargada de realizar la ejecución de proyectos.

- Diagrama de secuencia

En el siguiente diagrama de secuencia podemos visualizar la forma en la que se realiza la ejecución de proyectos.



- ## Errores conocidos del módulo

No se han encontrado aun errores sobre el módulo.

- ## Trabajos futuros sobre el módulo

No se han definido aun trabajos futuros sobre el módulo.

- ## Módulos que brindan las características de la depuración

En este apartado, iremos recorriendo cada uno de los módulos que implementan las características de la depuración de programas, propuestas para este trabajo. De cada módulo, veremos que características implementa, de que módulos del proyecto depende, que clases implementa, cual es la responsabilidad de cada clase, alguna secuencia significativa del módulo, si existen errores conocidos del mismo, y los trabajos futuros que ya se han definido.

Los módulos son:

- DaVinciConcurrenteDepurador

- ## DaVinciConcurrenteDepurador

Este módulo implementa las clases necesarias para la depuración de proyectos.

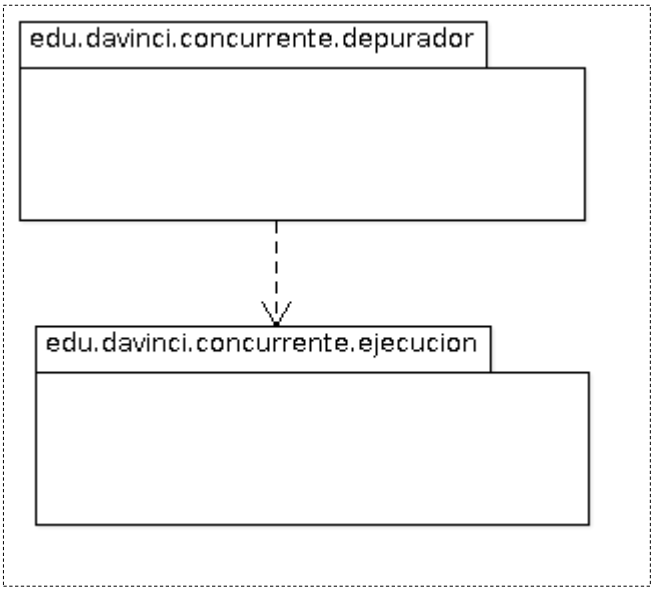
Dentro de las características que implementa este módulo, se encuentran:

- Ejecución de proyectos paso a paso

A continuación veremos algunos diagramas para profundizar más en el módulo.

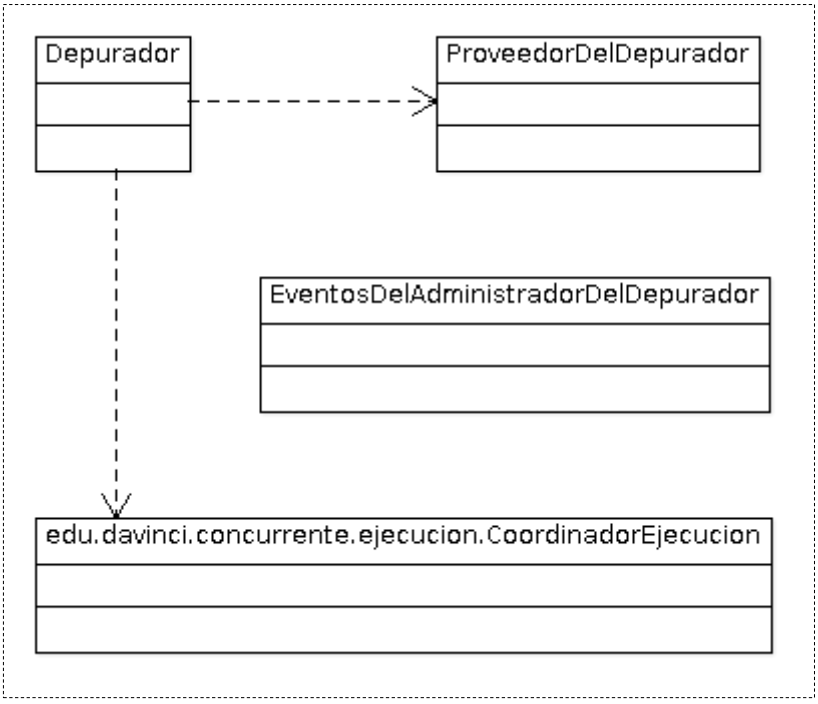
- Diagrama de dependencias

En el siguiente diagrama se puede visualizar como este módulo, depende del módulo de servicios de ejecución.



- Diagrama de clases

En el siguiente diagrama podemos ver las clases que implementa el módulo y las relaciones con las clases del módulo del que depende.



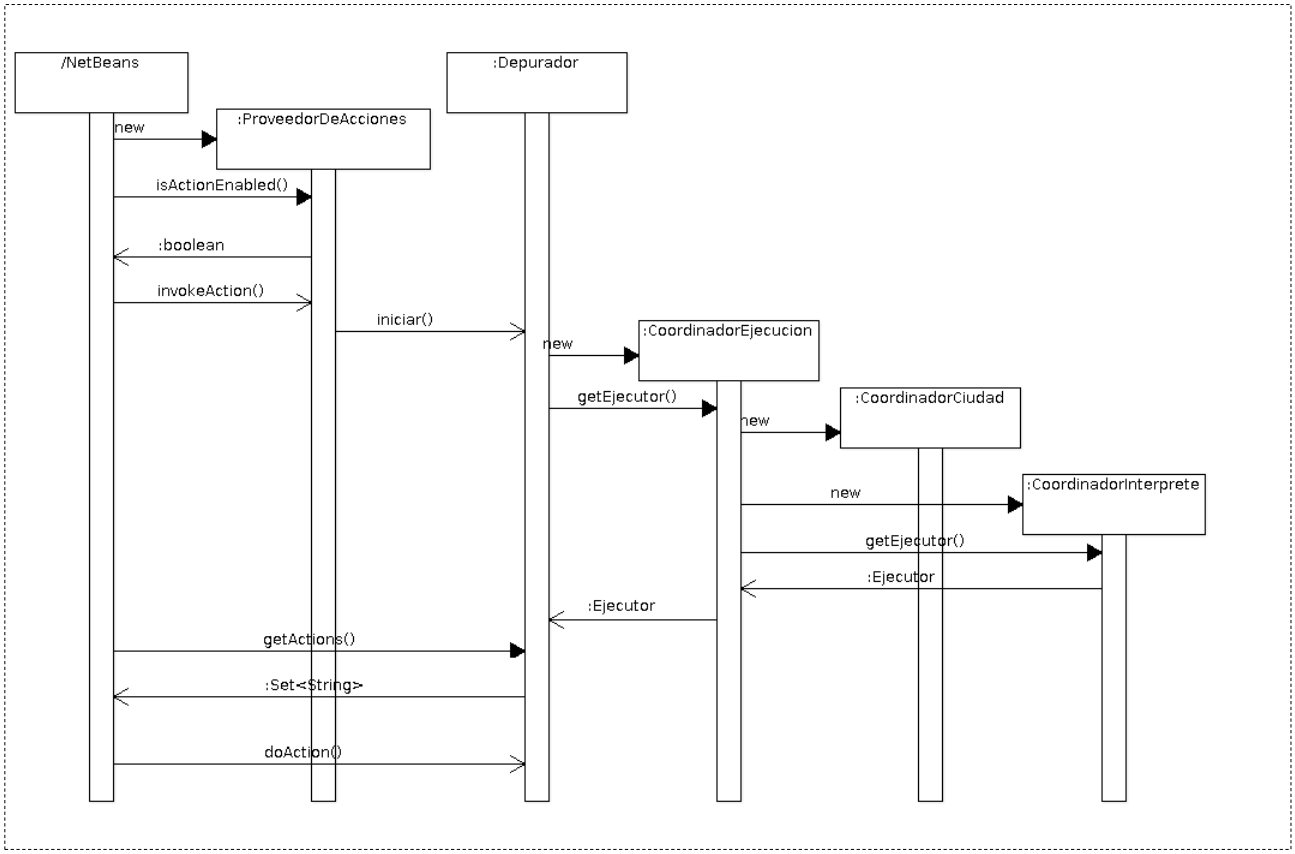
Las clases son:

- ProveedorDelDepurador: clase factoría del depurador.

- Depurador: clase encargada de la depuración de proyectos.
- EventosDelAdministradorDelDepurador: clase que permite interceptar eventos que se producen durante la depuración de código.

• Diagrama de secuencia

En el siguiente diagrama podemos visualiza la forma en la que se lleva adelante la depuración de código.



• Errores conocidos del módulo

En la depuración de código, no se esta visualizando la linea actual que se esta ejecutando.

• Trabajos futuros sobre el módulo

Se deben implementar todas las características faltantes para la depuración de código:

- Cuadro de dialogo de visualización de variables contextuales y sus valores.
- Cuadro de dialogo de visualización de hilos.
- Cuadro de dialogo de visualización de pila de llamadas.
- Implementación de puntos de interrupción.

3. Bibliografía utilizada

• C

hampredonde, Raul, y Armando De Guisti. *Design and Implementation of Visual Da Vinci.* La Plata, Argentina: III Congreso Argentino en Ciencias de la Computación, 1997.
• .....Feierherd, Guillermo; Depetris, Beatriz; Jerez, Marcela. *Una Evaluación sobre la incorporación temprana de algorítmica y programación en el ingreso a Informática.* El Calafate, Santa Cruz: VII Congreso Argentino de Ciencias de la Computación, 2001.
• .....Stallings, William. *Sistemas Operativos.* Madrid: Prentice Hall, 1997.
• .....Tanenbaoum, Andrew S. y Woodhoull AlbertS. . *Sistemas Operativos: Diseño e implmentación.* Mexico: Prentice Hall, 1997.
• .....Javier Garcia de Jalon y otros. *Aprenda C como si estuviera en primero*. Universidad de Navarra, 1998.
• .....Javier Garcia de Jalon y otros. *Aprenda C++ como si estuviera en primero*. Universidad de Navarra, 1998.
• .....Bruce Eckel . *Thinking In C++* . Prentice Hall, 2000.
• .....Jeff Ferguson y otros. *La Biblia de C#*. Anaya, 2003.
• .....Bruce Eckel. *Piensa en JAVA*. Madrid, Pearson, 2007.
• .....James Gosling y otros. *The Java Language Specification, Java SE 7 Edition*. Oracle, 2011.
• .....Steve Holzner . *Eclipse Cookbook*. O'Reilly , 2004.
• .....Tim Boudreau y otros. *Plugging into the NetBeans Platform* . Prentice Hall, 2007.
• .....Adam Myatt . *Pro NetBeans 6 IDE, Rich Client Platform Edition* . Apress, 2008.
• .....Heiko Bock. *The Definitive Guide to NetBeans Platform 7*. Apress.
• .....Sun Microsystems, Inc. *JavHelp 2.0 System User's Guide.* Sun Microsystems, Inc, 2004.

4. Productos utilizados

- Java
- JavaCC
- NetBeans
- ArgoUML
- LibreOffice