Guía de Módulos de Fastest

Pablo Rodriguez Monetti

2011

Capítulo 1

Client

Este módulo lógico agrupa los módulos que implementan la funcionalidad de los clientes (en el sentido del paradigma Cliente-Servidor) del sistema Fastest. Se optó por dividir a Client, separando en dos módulos diferentes, aspectos de la lógica de negocios de aquellos de la lógica de presentación. También hay un tercer submódulo, AbstractOrder, que brinda una interfaz que abstrae la implementación del patrón de diseño Command.

1.1. ClientBusinessLogic

Aquí se agrupan los módulos que se encargan de las tareas de negocio del lado del cliente del sistema. Tales aspectos se separan en dos módulos, de acuerdo a si involucra funciones de gestión y comunicación en el sistema o a otras funciones propias del dominio de la aplicación y que tienen que ver con el proceso de testing propiamente dicho.

1.1.1. ClientManagement

Subsistema encargado de administrar la comunicación entre clientes y servidores de Fastest y de manejar el flujo de control interno en los clientes.

Communication

Módulo lógico que agrupa a aquellos otros relacionados con la comunicación entre clientes y servidores del sistema y con el registro de servidores.

CServersControl Módulo físico que mantiene y gestiona el almacén con la configuración de cada servidor de cómputo del sistema. Sus elementos son de tipo ServerConfig. El módulo está basado en el patrón de diseño Singleton por lo que debe crearse una única instancia de él.

CServersConfigLoader Módulo físico que tiene la responsabilidad de cargar la configuración de los servidores de cómputo en la única instancia del CServersControl, desde el servidor de datos o desde un archivo de configuración. Oculta la implementación del algoritmo que realiza tal tarea, que actualmente consiste en la lectura de un archivo de configuración.

DServerConfig Módulo físico que oculta la estructura de datos y los algoritmos que implementan la configuración sobre el servidor de datos del sistema. Estos datos, los imprescindibles para que el cliente se conecte con el servidor de datos, se almacenan en un archivo de configuración. El módulo DServerConfig está basado en el patrón de diseño Singleton por lo que debe crearse una única instancia de él.

ServerConfig Módulo físico que oculta la estructura de datos y los algoritmos que implementan la configuración necesaria para conectarse con un servidor (tanto de datos como de cómputo), como ser su nombre, dirección IP y puerto sobre el que está esperando una nueva conexión.

ServerConfigRepository Módulo físico que oculta la estructura de datos y los algoritmos que implementan un agregado o repositorio de instancias de ServerConfig.

ServerName Módulo físico que representa el nombre de Servidor de Cómputo del Sistema. Se usa por comodidad en la descripción del sistema y no se implementa.

SocketWriter Módulo físico encargado de escribir información en un flujo de salida(en este caso un Socket) relacionada con la generación de casos de prueba. Cualquier modificación en el protocolo de comunicación Cliente/Servidor de Fastest implicará cambios en este módulo. El módulo oculta detalles del protocolo Cliente/Servidor subyacente así como también los algoritmos y estructuras de datos empleados para cumplir con su funcionalidad.

ServiceMediator Módulo físico cuya funcionalidad es establecer la conexión directa (a través de una interfaz de socket) de distintos componentes de la aplicación cliente del sistema con los servidores de cómputo, codificando los pedidos que puedan realizar y decodificando los resultados antes de enviárselos de regreso. Interactúa con el módulo SocketWriter para enviar la información pertinente a los servidores. El módulo oculta los algoritmos que implementan tales funciones. Debe crearse una nueva instancia de este módulo para cada solicitud de servicio que un componente del cliente del sistema efectúe.

ImplicitInvocation

Es el módulo lógico que contiene aquellos módulos relacionados con la invocación implícita de subrutinas y con los componentes sobre los que se pueden invocar tales subrutinas.

Events Módulo lógico que agrupa aquellos módulos que implementan eventos. Events se divide en módulos de acuerdo a los distintos tipos de eventos que pueden lanzar un componente del sistema. Para agregar un nuevo evento a Fastest se debe definir un nuevo heredero de Event_.

1. AbstractionFuncLoaded. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando

se haya cargado la función de abstracción necesaria para abstraer casos de prueba concretos. El módulo es heredero del módulo Event_ dado que representa un evento particular.

2. AllTCasesRequested. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se ordene la generación de todos los casos de prueba. Se lanzará una instancia de este evento por cada árbol de prueba que haya sido generado previamente. El módulo es heredero del módulo Event_ dado que representa un evento particular.

3. AllTCasesGenerated. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se haya concluido el intento de generación de todos los casos de prueba solicitados. El módulo es heredero del módulo Event_ dado que representa un evento particular.

4. AllTTreesGenerated. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se haya concluido el intento de generación de todos los árboles de prueba solicitados. El módulo es heredero del módulo Event_ dado que representa un evento particular.

5. Event_. Módulo físico que oculta la estructura de datos y los algoritmos que implementan un evento.

6. NotTClassLeavesFounded. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se ha recorrido un árbol de pruebas para buscar sus hojas y no se ha encontrado ninguna. El módulo es heredero del módulo Event_ dado que representa un evento particular.

7. OutputAbstracted. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado después de abstraer la salida concreta que resulta de ejecutar un caso de prueba de cierta operación. El módulo es heredero del módulo Event_ dado que representa un evento particular.

8. RefinementFuncLoaded. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se haya cargado la función de refinamiento necesaria para refinar casos de prueba abstractos. El módulo es heredero del módulo Event_ dado que representa un evento particular.

9. SpecLoaded. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se haya cargado la especificación del programa a testear. El módulo es heredero del módulo Event_ dado que representa un evento particular.

10. TCaseChecked. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado después de haber analizado la correspondencia entre un caso de prueba abstracto y la salida abstracta que corresponde a su ejecución junto con la especificación de la

operación. El módulo es heredero del módulo Event_ dado que representa un evento particular.

11. TCaseExecuted. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado después de haber ejecutado un caso de prueba concreto de cierta operación a testear. El módulo es heredero del módulo Event_ dado que representa un evento particular.

12. TCaseGenerated. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado después de haberse terminado el intento de generación del caso de prueba abstracto de cierta clase de prueba de una operación particular. El módulo es heredero del módulo Event_ dado que representa un evento particular.

13. TCaseRefined. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado después de haber refinado un caso de prueba abstracto. El módulo es heredero del módulo Event_ dado que representa un evento particular.

14. TCaseRequested. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado al solicitar el cálculo de un caso de prueba para cierta clase de prueba. El módulo es heredero del módulo Event_ dado que representa un evento particular.

15. TCaseStrategySelected. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado después de que el usuario selecciona la estrategia de generación de casos de prueba que desea. El módulo es heredero del módulo Event_ dado que representa un evento particular.

16. TTreeGenerated. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado después de generarse el árbol de prueba de cierta operación. El módulo es heredero del módulo Event_ dado que representa un evento particular.

17. PruneTTreeRequested. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se ordena la poda del arbol de prueba. El módulo es heredero del módulo Event_ dado que representa un evento particular.

18. PruneTClassRequested. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado cuando se ordena la poda de una clase de prueba. El módulo es heredero del módulo Event_ dado que representa un evento particular.

19. PrunningResult. Módulo físico que oculta la estructura de datos y los algoritmos que implementan el evento que debe ser lanzado anunciando el analisis de factibilidad de la poda de una clase de prueba. El módulo es heredero del módulo Event_ dado que representa un evento particular.

EventAdmin Módulo físico cuya funcionalidad corresponde a la del administrador de eventos de la aplicación Cliente del sistema, y que permite tanto leer un archivo de configuración de invocación implícita par construir la tabla de eventos como anunciar un evento y llamar a las subrutinas interesadas en él. El módulo está basado en el patrón de diseño Singleton por lo que debe crearse una única instancia de él. Oculta las estructuras de datos y los algoritmos para implementar tales tareas.

EventName Módulo físico que representa el nombre de un evento del Sistema. Se usa por comodidad en la descripción del sistema y no se implementa.

EventTable Módulo físico que oculta la estructura de datos y los algoritmos que implementan un agregado o repositorio de instancias de Subscriptor.

IIComponent Interfaz que oculta la estructura de datos y los algoritmos que implementan un componente del sistema que puede estar interesado en eventos de invocación implícita. En principio, las instancias de los herederos de éste módulo corresponden a los componentes mostrados en la última figura del documento Introducción a la Arquitectura del Sistema Fastest. Cada instancia mantiene una referencia a la interfaz de usuario del sistema para poder interactuar con ella en caso de ser necesario. En principio solo un heredero de IIComponent, Controller, será responsable de actuar sobre la interfaz de usuario, por lo que será el único de ellos que haga uso de la mencionada referencia.

IIOrder Módulo físico que hereda de AbstractOrder y que contiene una implementación concreta de tal módulo. Tal implementación es utilizada en todo el subsistema de Invocación Implícita para referirse a las subrutinas de ciertos módulos, que deben ser invocadas en respuesta al anuncio de algún evento de interés. IIOrder mantiene en su estado una referencia a la instancia del componente (módulo IIComponent) sobre el que se invocará la subrutina, el nombre de la subrutina a invocar, y una referencia al evento anunciado.

MethodName Módulo físico que representa el nombre de un método de componente de invocación implícita. Se usa por comodidad en la descripción del sistema y no se implementa.

Subscriptor Módulo físico que oculta las estructuras de datos y los algoritmos que implementan la subscripción de una subrutina de cierto módulo para ser invocada al lanzarse determinado evento.

Controller

Módulo físico que tiene la responsabilidad de mantener referencias a:

- árboles de pruea previamente calculados
- casos de prueba previamente calculados
- especificación cargada por el usuario
- especificación con los esquemas expandidos

- lista de operaciones a ser testeadas, junto con las tácticas que se le aplicarán a cada una
- cantidad de clases de prueba para las que se está intentando obtener un caso de prueba
- cantidad de clases de prueba que se están analizando para verificar que son o no son vacías
- valores asignados a variables definidas en definiciones axiomáticas
- variables, definidas en definiciones axiomáticas, que esperan un valor por parte del usuario
- para cada variable, definida en alguna definición axiomática, los predicados, de defincciones axiomáticas, que la contienen
- las variables de tipo no básico, definidas en definiciones axiomáticas, contenidas en cada predicado
- la lista de declaraciones de constantes auxiliares que se introducen para definir el valor de alguna variable definida en alguna definición axiomáticas
- la lista de variables, entre las dadas en definiciones axiomáticas y que esperan un valor del parte del usuario, cuyos valores no pueden ser seteadas via un comando setaxdef debido a que aparecen en el predicado de alguna clase de prueba previamente podada
- etc...

Este módulo oculta las estructuras de datos y algoritmos para llevar a cabo las tareas mencionadas. Controller hereda del módulo IIComponent ya que resulta ser una implementación particular de él: puede interesarse en eventos de invocación implícita.

1.1.2. Testing

Módulo lógico que agrupa aquellos módulos relacionados con el proceso de testing funcional basado en especificaciones Z.

ClientTCaseChecking

Módulo lógico que agrupa a aquellos otros vinculados a la comprobación de correspondencia entre un caso de prueba abstracto de cierta operación, la operación, y la salida abstracta que surge de ejecutar el caso de prueba abstracto. Dado que dicha comprobación se realiza en servidores de cómputo, este módulo sólo es responsable de la solicitud de tales servicios.

TCaseCheckerClient Módulo físico cuya funcionalidad es manejar las solicitudes, a servidores de cómputo, de comprobación de correspondencias entre casos de prueba abstracto de operacion, las correspondientes operaciones, y las salidas abstracta que surgen de ejecutar los casos de prueba abstractos. Cada

solicitud se realiza en un thread diferente para favorecer cuestiones de performance, ejecutando en cada thread la subrutina run() del módulo TCaseCheckerClientRunner. El módulo TCaseCheckerClient oculta los algoritmos que implementan la funcionalidad comentada y hereda del módulo IIComponent ya que resulta ser una implementación particular de él.

TCaseCheckerClientRunner Módulo físico que se encarga de solicitar, a un servidor de cómputo, una comprobación de correspondencia entre un caso de prueba abstracto de una operación, la operación, y la salida abstracta que surge de ejecutar el caso de prueba abstracto. El módulo devuelve el resultado a través del anuncio de un evento de tipo TCaseChecked y oculta los algoritmos que implementan las tareas mencionadas.

ClientPrunning

Módulo lógico que agrupa a aquellos otros relacionados con la poda de clases de prueba y cuyas instancias siempre corren del lado del cliente. Dado que la poda de clases de prueba se realiza en servidores de cómputo, este módulo es sólo responsable de la solicitud de tales servicios y no de su realización.

PrePruner Módulo físico que permite determinar si una clase de prueba es vacía por tener en su predicado una contradicción booleana, esto es, un predicado de la forma $p \wedge \neg\, p$. El módulo oculta el algoritmo que implementa esta funcionalidad.

TClassPruneClient Módulo físico cuyas instancias (aunque se asume que existirá sólo una) se interesan en los eventos de tipo PruneTClassRequested, anunciados por una instancia de TPrunning, cada uno de los cuales es una solicitud de poda de una clase de prueba. TClassPruneClient se encarga de manejar estas solicitudes procesando cada una en un thread diferente para favorecer cuestiones de performance, ejecutando la subrutina run() del módulo TClassPruneClientRunner en cada nuevo thread. El módulo TClassPruneClient oculta los algoritmos que implementan dicha funcionalidad. Hereda del módulo IIComponent ya que resulta ser una implementación particular de él.

TClassPruneClientRunner Módulo físico que se encarga de solicitar, a un servidor de cómputo, el análisis del predicado de una clase de prueba, para verificar si es o no es una contradicción. En caso que lo sea la clase de prueba podrá podarse del árbol de pruebas al que pertenece. Cada solicitud debe hacerse a través de una nueva instancia de ServiceMediator, que es la que mantiene el secreto de cómo se lleva a cabo la comunicación con los servidores. El módulo TClassPruneClientRunner devuelve el resultado a través del anuncio de un evento de tipo PrunningResult, y oculta los algoritmos que implementan las tareas mencionadas.

TPrunning Módulo físico cuyas instancias (aunque se asume que existirá sólo una) manejan eventos de tipo PruneTTreeRequested y anuncian eventos de tipo PruneTClassRequested. El primer evento tiene como parámetro un árbol de pruebas que se quiere podar, el segundo evento tiene lleva una clase de prueba

cuyo predicado se quiere analizar para determinar si es o no una contracción. Cada clase de prueba será una clase de prueba del árbol de pruebas o una modificación de una de ellas, reemplazando definiciones axiomáticas por los valores que les corresponden y agregando declaraciones de constantes que puedan estar contenidas en estos valores. Estas modificaciones se realizarán en este módulo TPrunning, el cual oculta los algoritmos y estructuras de datos necesarios para cumplir con la funcionalidad comentada.

ClientTCaseGeneration

Módulo lógico que agrupa a aquellos otros módulos relacionados a la generación de casos de prueba abstractos del lado del cliente, es decir, módulos que hacen las solicitudes, a servidores de cómputo del sistema, del cálculo de casos de prueba abstractos.

TCaseGenClient Módulo físico que se encarga de manejar las solicitudes, a servidores de cómputo, de generación de casos de prueba abstractos. Estas solicitudes son indicadas en cada evento de tipo TCaseRequested que se anuncie en el sistema, y son procesadas por TCaseClient en un thread diferente para favorecer cuestiones de performance, ejecutando la subrutina run() del módulo TCaseGenClientRunner en cada nuevo thread. El módulo TCaseGenClient oculta los algoritmos que implementan dicha funcionalidad. Hereda del módulo IIComponent ya que resulta ser una implementación particular de él.

TCaseGenClientRunner Módulo físico que se encarga de solicitar, a un servidor de cómputo, la generación de un caso de prueba abstracto para cierta clase de prueba de un árbol de prueba de una operación. Cada solicitud debe hacerse a través de una nueva instancia de ServiceMediator, que es la que mantiene el secreto de cómo se lleva a cabo la comunicación con los servidores. El módulo TCaseGenClientRunner devuelve el resultado a través del anuncio de un evento de tipo TCaseGenerated, y oculta los algoritmos que implementan las tareas mencionadas.

TClassExtractor Módulo físico que se encarga de procesar eventos de tipo AllTCasesRequested (aquellos que tienen como parámetro un arbol de prueba, junto con el nombre de la operación) lanzando, para cada uno, sucesivos eventos de tipo TCaseRequested, uno para cada clase de prueba que sea hoja del árbol correspondiente. Cada evento TCaseRequested que TClassExtractor anuncia tendrá como parámetros el nombre de la operación y la clase de prueba que corresponda. No se procesan las ramas que aparezcan podadas en los árboles. El módulo TClassExtractor oculta los algoritmos que implementan estas tareas. Hereda del módulo IIComponent ya que resulta ser una implementación particular de él.

ClientAbstraction

Módulo lógico que agrupa elementos del lado del cliente vinculados a la abstracción de casos de prueba a nivel de implementación en casos de prueba a nivel de especificación.

TCaseNodeAdderClient Módulo físico que se encarga de manejar las solicitudes para agregar a los casos concretos el codigo necesario para capturar la salida de las variables. Estas solicitudes son indicadas en cada evento de tipo TCaseAddCaptureCodeRequested que se anuncie en el sistema, y son procesadas por TCaseNodeAdderClient en un thread diferente para favorecer cuestiones de performance, ejecutando la subrutina run() del módulo TCaseNodeAdderClientRunner en cada nuevo thread. El módulo TCaseNodeAdderClient oculta los algoritmos que implementan dicha funcionalidad.

TCaseNodeAdderClientRunner Módulo físico que se encarga de interactuar con los módulos adecuados para agregar el código de captura a un caso concreto en particular. De acuerdo al lenguaje objetivo crea el heredero correcto de ScriptTestWriter y le pasa como parámetro las variables que deben ser monitoreadas para que se agregue el código de captura. Oculta los algoritmos y las estructuras de datos empleados para tal fin.

TCaseAbsClient Módulo físico que se encarga de manejar las solicitudes de abstraccion de casos concretos. Estas solicitudes son indicadas en cada evento de tipo TCaseAbstractRequested que se anuncie en el sistema, y son procesadas por TCaseAbsClient en un thread diferente para favorecer cuestiones de performance, ejecutando la subrutina run() del módulo TCaseAbsClientRunner en cada nuevo thread. El módulo TCaseAbsClient oculta los algoritmos que implementan dicha funcionalidad.

TCaseAbsClientRunner Módulo físico que se encarga de interactuar con los módulos adecuados para completar la abstracción de un caso concreto. Básicamente solicita al módulo XMLParser para que parsee los archivos XML que deben haber sido generados en la etapa de ejecucion y finalmente le pasa los resultados del parseo al módulo ASTAbstraction para que complete el proceso. Oculta los algoritmos y estructuras de datos empleados para cumplir con su función.

ManipulersFactory Interfaz creada siguiendo los lineamientos del patrón de diseño AbstractFactory que tendrá un heredero por cada lenguaje que soporte Fastest en la etapa de abstracción. Se utiliza para evitar inconsistencias relacionadas con el lenguaje objetivo.

ManipulersFactoryC Módulo físico que hereda de ManipulersFactory y se encarga de crear los objetos necesarios para la abstracción relacionados exclusivamente con el lenguaje de programación C.

AbstractionLawRepository Módulo físico que almacena las leyes de abstracción cargadas al sistema como instancias de objetos AbstractionLaw. Oculta las estructuras de datos empleadas.

TCaseExecution

Módulo lógico que agrupa a aquellos otros módulos vinculados a la ejecución de casos de prueba.

ConcreteOutput Módulo físico que oculta la estructura de datos y los algoritmos que implementan el resultado concreto (a nivel de implementacíón), que se obtiene de ejecutar un caso de prueba.

ConcreteTCase Módulo físico que oculta la estructura de datos y los algoritmos que implementan un caso de prueba concreto (a nivel de implementación).

Executor Módulo físico cuya responsabilidad es ejecutar casos de prueba sobre el programa a ser testeado. Oculta las estructuras de datos y los algoritmos necesarios para implementar la mencionada tarea. Hereda del módulo IIComponent ya que resulta ser una implementación particular de él.

TCaseRefinement

Módulo lógico que agrupa elementos vinculado al refinamiento de casos de prueba abstractos en casos de prueba concretos.

TCaseRefClient Módulo físico que se encarga de manejar las solicitudes de refinamiento de casos de prueba abstractos. Estas solicitudes son indicadas en cada evento de tipo TCaseRefineRequested que se anuncie en el sistema, y son procesadas por TCaseRefClient en un thread diferente para favorecer cuestiones de performance, ejecutando la subrutina run() del módulo TCaseRefClientRunner en cada nuevo thread. El módulo TCaseRefClient oculta los algoritmos que implementan dicha funcionalidad. Hereda del módulo IIComponent ya que resulta ser una implementación particular de él.

TCaseRefClientRunner Módulo físico que se encarga de solicitar, a un servidor de cómputo, el refinamiento de un caso de prueba a un lenguaje en particular. Este módulo se encargará de parsear el archivo con la ley de refinamiento y posteriormente solicitará al módulo RefineAST el refinamiento en cuestión. Finalmente anuncia el caso refinado a través de un evento de tipo TCaseRefined.

ConcreteTCase Módulo físico que abstrae un caso refinado. Almacena el codigo en el lenguaje de programacion indicado con la inicializacion de las variables que se indicaron en la ley de refinamiento, asi como tambien almacena informacion relacionada al caso abstracto del cual refinado, el lenguaje que se empleo para su refinamiento, el nombre de la ley de refinamiento, etc. Oculta los algoritmos y estructuras de datos utilizados para su implementacion.

ConstantGenerator Módulo físico que se encarga de generar constantes de tipos primitivos de un lenguaje de programacion a partir de un objeto de CZT. Los resultados serán además almacenados en la instancia única de ConstantStore para su posterior uso en el proceso de abstracción. Oculta los algoritmos empleados para la implementacion.

ConstantStore Módulo físico que se encarga de mapear expresiones de CZT con constantes de tipos primitivos de los lenguajes de programacion soportados por Fastest en la etapa de refinamiento. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

RefineAST Módulo físico que se encarga de el refinamiento propiamente dicho. Parte de una ley de refinamiento parseada y un caso abstracto y va interactuando con un heredero adecuado del modulo RefineExpr hasta obtener una instancia de ConcreteTCase. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

RefineExpr Interfaz de la cual heredarán los módulos que se encargan de obtener el código en un lenguaje de programación en particular necesario para inicializar las variables indicadas en las reglas de refinamiento que reciben como parámetro. Tendrá un heredero por cada lenguaje de programación que soporte Fastest en esta étapa.

RefineExprC Modulo físico que implementa la interfaz RefineExpr. Se encarga de generar código C para inicializar las variables que se le indican en la regla de refinamiento que se le pasa como parámetro. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

RefineExprJava Modulo físico que implementa la interfaz RefineExpr. Se encarga de generar código Java para inicializar las variables que se le indican en la regla de refinamiento que se le pasa como parámetro. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

RefLawRepository Modulo físico que almacena las leyes de refinamiento sin parsear cargadas en Fastest. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

TCaseAssignment Modulo físico que almacena código de inicialización de una variable en un lenguaje de programación. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

TCRL_Parser Modulo físico que se encarga de parsear una ley de refinamiento. Este módulo fue creado por Java Compiler Compiler a partir del archivo tcrl.jj presente en el sistema. Toma un archivo y devuelve el AST resultante de parsear dicho archivo. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

TCRL_PreProcessor Modulo físico que se encarga de reemplazar el uso de sinonimos en un objeto de tipo TCRL_AST por los objetos correspondientes. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

TCRL_Parser Modulo físico que se encarga de reemplazar los sinonimos en una ley de refinamiento parseada por las expresiones con las cuales se corresponden. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

RefinementUtils Modulo físico que brinda funcionalidades comunes al proceso de refinamiento. Oculta los algoritmos y estructuras de datos empleadas para la implementacion.

TTreeModules

Módulo lógico que agrupa a aquellos otros relacionados con los árboles de prueba y su generación. Se divide de acuerdo a las distintas tácticas que pueden aplicarse para generar árboles de prueba, a los módulos que implementen Visitantes (patrón de diseño Visitor) para aplicar operaciones sobre ellos y a otros módulos relacionados.

Tactics Módulo lógico cuyos submódulos implementan tácticas de generación de clases de prueba o información sobre éstas tácticas. Las nuevas tácticas que puedan agregarse al sistema deberán ser módulos herederos de Tactic.

1. AbstractTactic. Módulo físico que oculta la estructura de datos y los algoritmos que una táctica abstracta. Implementa los métodos de Tactic cuya implementación es la misma independientemente de la táctica específica. Este módulo es heredero del módulo Tactic dado que implementa algunas de sus funcionalidades.

2. CEPTactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica de proyección causa-efecto (Cause Effect Propagation). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

3. CEPTacticInfo. Módulo físico que mantiene información relacionada a la táctica de proyección causa-efecto aplicada. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

4. DNFIterator Módulo físico que facilita la implementación de la táctica Disjunctive Normal Form (DNF). Permite solicitarle los términos de la disyunción uno a uno. A través de este módulo se logra no tener que almacenar el predicado en DNF completo sino solamente una lista de predicados en DNF que, si se combinan en conjunción sus elementos, es equivalente al predicado en DNF. La lista de predicados en DNF es calculada con el modulo DNFPredExtractor.

5. DNFTactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica DNF (Disjunctive Normal Form). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

6. DNFTacticInfo. Módulo físico que mantiene información relacionada a la táctica DNF aplicada. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

7. FTTactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica de tipos libres (Free Types). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

8. FTTacticInfo. Módulo físico que mantiene información relacionada a la táctica de tipos libres, como ser el nombre de la variable a partir de cuyo tipo se van a generar nuevas clases de prueba y una referencia a la instancia de Freetype, en la especificación cargada en Fastest, que corresponde al tipo. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

9. SISETactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica Set In Set Extension (SISE). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

10. SISETacticInfo. Módulo físico que mantiene información relacionada a la táctica SISE, como ser el nombre de la variable a partir de la cual se van a generar nuevas clases de prueba y una referencia al conjunto definido por extensión al cual pertenece la variable antes mencionada. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

11. SSSETactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica Sub Set of Set Extension (SSSE). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

12. SSSETacticInfo. Módulo físico que mantiene información relacionada a la táctica SSSE, como ser el nombre de la variable a partir de la cual se van a generar nuevas clases de prueba y una referencia al conjunto definido por extensión del cual es subconjunto la variable antes mencionada. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

13. PSSETactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica Proper Subset of Set Extension (PSSE). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

14. PSSETacticInfo. Módulo físico que mantiene información relacionada a la táctica PSSE, como ser el nombre de la variable a partir de la cual se van a generar nuevas clases de prueba y una referencia al conjunto definido por extensión del cual es subconjunto la variable antes mencionada. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

15. SMTactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica de mutación de especificaciones Z (Specification Mutation). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

16. SMTacticInfo. Módulo físico que mantiene información relacionada a la táctica de mutación de especificaciones aplicada, como por ejemplo, la operación mutante que junto a la operación original permitirá aplicar la mencionada táctica. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

17. SPTactic. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la táctica de particiones estándar de operadores (Standard Partition). Este módulo es heredero del módulo Tactic dado que representa y permite aplicar un tipo de táctica particular.

18. SPTacticInfo. Módulo físico que mantiene información relacionada a la táctica de particiones estándar de operadores, como ser el predicado (de cierto esquema de operación) o el operador sobre el que se aplicó la partición. Oculta las estructuras de datos y algoritmos que implementan tal información y hereda del módulo TacticInfo.

19. StdPartition. Módulo físico que representa una partición estándar de un operador del lenguaje de especificaciones Z. Oculta los algoritmos y estructuras de datos que la implementan.

20. StdPartitionLoader. Módulo encargado de parsear el archivo de configuración que contiene la definición de las particiones estándar del sistema, y que las almacena como instancias de StdPartition en la instancia única de StdPartitionsControl que hay en el Sistema. StdPartitionLoader oculta los algoritmos y estructuras de datos que llevan a cabo la funcionalidad mencionada.

21. StdPartitionsControl. Módulo que permite agrupar una serie de instancias de StdPartition. Debe haber una única instancia de StdPartitionsControl en el sistema en cada momento.

22. Tactic. Interfaz que abstrae la estructura de datos y algoritmos que implementan una táctica y que permite aplicarla a una clase de prueba Z obteniendo las clases de prueba resultantes.

23. TacticInfo. Módulo físico que permite obtener información relevante de una táctica de testing aplicada, como ser su identificador de táctica y demás argumentos que se precisen. El módulo oculta las estructuras de datos y algoritmos que implementan tal información.

TTreeStrategies Módulo lógico que agrupa aquellos módulos que implementen estrategias de generación de árboles de prueba.

1. IterativeTTreeStrategy. Módulo físico que oculta la estructura de datos y los algoritmos que implementan la estrategia iterativa de generación de casos de prueba. Esta estrategia se basa en aplicar las tácticas de testing (de la lista dada) una a una por cada nuevo nivel que se vaya generando en el árbol de pruebas. Este módulo es heredero del módulo TTreeStrategy dado que representa una estrategia particular de generación de árboles de prueba.

2. TTreeStrategy.Interfaz que abstrae las estructuras de datos y el algoritmo que implementa una estrategia de aplicación de tácticas en la generación del árbol de prueba de una operación. Este módulo fue concebido en base al patrón de diseño Strategy. Las distintas estrategias que pueden aplicarse deberán ser implementadas por módulos que, a su vez, implementen esta interfaz. Dados un esquema de operación Z y una lista de tácticas de

testing, cada estrategia establece el criterio de selección de la táctica a aplicar, entre las de la lista de tácticas dada, en cada paso del proceso de construcción del árbol de prueba. Una estrategia podría ser aplicar las tácticas de la lista una a una por cada nuevo nivel del árbol de prueba, asumiendo que hay tantas tácticas como niveles posibles. Otra podría ser tener tantas tácticas como clases de prueba internas en el árbol de prueba que se va generando y aplicarlas una a una.

TTreeVisitors Módulo lógico que agrupa aquellos módulos que implementen visitantes (patrón de diseño Visitor) sobre árboles de prueba.

1. SchemeTTreeFinder. Módulo físico que permite recorrer un árbol de prueba para buscar una clase o caso de prueba particular en él. El esquema buscado debe estar dada/o por el nombre que se pasa como argumento en el contructor de este módulo. También se pasa como argumento un entero que representa el grado de expansión con el que se quiere expandir el esquema devuelto. Esto significa, que una vez encontrado el nodo del árbol para el esquema deseado, este se expanderá tantos niveles hacia arriba en el árbol como indique ese valor. El módulo hereda de TTreeVisitor dado que se implementa como un visitante particular.

2. TCaseNodeAdder. Módulo físico que permite agregar un nodo de tipo TCaseNode como hoja a un árbol de prueba. El constructor del módulo toma el nombre de la clase de prueba del nodo (de tipo TClassNode) padre del nuevo nodo y la instancia de AbstractTCase que corresponde al caso de prueba no expandido que será parte del contenido de este nuevo nodo. La otra parte, el caso de prueba expandido, se determina usando el caso de prueba expandido y la clase de prueba del nodo padre. El módulo hereda de TTreeVisitor dado que se implementa como un visitante particular.

3. TCaseNodeTextUIPrinter. Módulo físico que permite recorrer un árbol de prueba e imprimir todos sus casos de prueba en pantalla mientras se utilice la interfaz de usuario en modo texto. El módulo hereda de TTreeVisitor dado que se implementa como un visitante particular.

4. TClassLeavesFinder. Módulo físico que permite recorrer un árbol de prueba para obtener aquellas clases de prueba que sean hojas del árbol. Estas clases de pruebas son las que resultan de interés a la hora de querer obtener casos de prueba para la operación correspondiente. No se consideran las hojas de los subárboles podados y sí se considera una clase si todas las clases que cuelgan de ella están podadas. El módulo hereda de TTreeVisitor dado que se implementa como un visitante particular.

5. TClassNodeTextUIPrinter. Módulo físico que permite recorrer un árbol de prueba e imprimir todos sus casos de prueba en pantalla mientras se utilice la interfaz en modo texto. El módulo hereda de TTreeVisitor dado que se implementa como un visitante particular.

6. TTreeBelowPruner. Módulo físico que permite recorrer un árbol de prueba para buscar una clase de prueba, especificada a través de su nombre en el constructor del módulo, para marcar sus clases hijas como podadas o como no podadas, según se especifique en el constructor del módulo.

TTreeBelowPruner hereda de TTreeVisitor dado que se implementa como un visitante particular.

7. TTreeFromPruner. Módulo físico que permite recorrer un árbol de prueba para buscar una clase de prueba, especificada a través de su nombre en el constructor del módulo, según se especifique en el constructor del módulo. TTreeFromPruner hereda de TTreeVisitor dado que se implementa como un visitante particular.

8. TTreeTextUIPrinter Módulo físico que permite recorrer e imprimir un árbol de prueba en pantalla mientras se utilice la interfaz en modo texto. Lo que se imprime es el nombre de cada nodo del árbol, no el contenido de cada nodo. Si hay algún nodo que corresponde a una clase de prueba podada, no se imprimen los nodos que cuelgan de él. El módulo hereda de TTreeVisitor dado que se implementa como un visitante particular.

9. TTreeVisitor. Interfaz que abstrae los algoritmos que implementan un visitante de árboles de prueba. Permite aplicar una operación particular a los nodos de un árbol de prueba, teniendo de que tipo particular es cada uno.

TacticName Módulo físico que representa el nombre de una táctica de generación de árboles de prueba. Se usa por comodidad en la descripción del sistema y no se implementa.

TacticManager Módulo físico que se encarga de gestionar definiciones de tácticas de testing funcional, descargándolas desde el servidor de datos o bien agregando nuevas. Oculta los algoritmos que implementan tales tareas. El módulo está basado en el patrón de diseño Singleton por lo que puede crearse una única instancia de él.

TCaseNode Módulo físico que oculta la estructura de datos y los algoritmos que implementan un nodo de un árbol de prueba cuyo dato es un caso de prueba abstracto. Más precisamente, cada instancia de TCaseNode mantiene en su estado dos instancias de AbstractTCase que corresponden al caso de prueba abstracto asociado al nodo: una con los esquemas incluídos totalmente expandidos y la otra con los esquemas incluídos totalmente colapsados. Esto se hace por una cuestión de comodidad. El módulo es heredero de TTreeNode, pues representa un tipo particular de nodo árbol de prueba..

TClassNode Módulo físico que oculta la estructura de datos y los algoritmos que implementan un nodo de un árbol de prueba cuyo dato es una clase de prueba. En la intancia de TClassNode también debe indicarse si el árbol de prueba está podado en esa clase de pruena o no. El módulo TClassNode es heredero de TTreeNode pues representa un tipo particular de nodo de prueba.

TTreeGen Módulo físico que permite generar el árbol de prueba de una operación determinada de una especificación Z. Oculta las estructuras de datos y los algoritmos que implementan la tarea mencionada. Hereda del módulo IIComponent ya que resulta ser una implementación particular de él.

TTreeNode Interfaz que abstrae la estructura de datos y los algoritmos que implementan un nodo de un árbol de prueba.

1.2. ClientPresentation

Módulo lógico que contiene aquellos módulos vinculados a la lógica de presentación, es decir, aquellos módulos que implementan la interacción entre la lógica de negocio del sistema y el usuario.

1.2.1. Commands

Módulo lógico que agrupa a los módulos relacionados con los comandos que puede ejecutar el usuario de Fastest al usar la interfaz en modo texto del sistema.

AddTacticCommand

Módulo físico que implementa el comando de interfaz de usuario que permite agregar una nueva táctica (junto con sus parámetros, de ser necesario) a la lista de tácticas asociada a una de las operaciones a testear. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ApplyCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la posibilidad de aplicar un teorema de eliminación a una clase de prueba. Recibe como parámetro la clase en cuestión, el teorema de eliminación y los parámetros reales. Esto permite determinar si una clase de prueba es una clase vacía para luego podarla y evitar el intento de generación del caso de prueba en vano. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

Command

Interfaz que abstrae un comando de la interfaz de usuario de modo texto del sistema. Cada vez que se necesite agregar un nuevo comando de usuario, debe crearse un heredero del módulo Command.

EvalCommand

Módulo físico que implementa el comando de interfaz de usuario que que permite evaluar expresiones y predicados Z. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

GenAllTCasesCommand

Módulo físico que implementa el comando de interfaz de usuario que que brinda la funcionalidad de generar (o intentar generar) los casos de prueba de todas las operaciones para las que previamente se generó un árbol de prueba. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

GenAllTTCommand

Módulo físico que implementa el comando de interfaz de usuario que que permite generar los árboles de prueba de todas las operaciones seleccionadas para ser testeadas. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

LoadRefinementLawCommand

Módulo físico que implementa el comando de interfaz de usuario que permite la carga a Fastest de una ley de refinamiento. Recibe como parámetros el nombre de la ley y el path del archivo en el cual se encuentra la ley. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

LoadSpecCommand

Módulo físico que implementa el comando de interfaz de usuario que permite realizar la carga en el sistema de una especificación Z escrita en Latex. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

PruneBelowCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la posibilidad de podar las clases de prueba, en un árbol de prueba, que cuelgan de una clase de prueba especificada por su nombre. Esto permite evitar el intento de generación de casos de prueba para aquellas clases podadas y para las que cuelgan de ellas. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

PruneFromCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la posibilidad de podar una clase de prueba, en un árbol de prueba, especificada por su nombre. Esto permite evitar el intento de generación de casos de prueba para aquellas clases podadas y las que cuelgan de ellas. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

PruneTreeCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la posibilidad de ordenar la poda del árbol de testing. Esto permite que el programa intente eliminar clases vacías. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

RefineOpCommand

Módulo físico que implementa el comando de interfaz de usuario que permite refinar casos abstractos de prueba. Recibe como parametros el nombre de una operacion o de una clase en particular, el nombre de la unidad que se esta testeando, el lenguaje al cual se refinará y el nombre de la ley de refinamiento. En caso de que se reciba el nombre de una operacion se solicita el refinamiento de todas las hojas de dicha operacion. En caso de que se reciba el nombre de

una clase de prueba, se solicita el refinamiento de la clase en cuestión. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ResetCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la posibilidad de remover toda la información cargada o generada por Fastest, como ser, una especificación Z, operaciones seleccionadas para ser testeadas, tácticas asociadas a una operación a ser testeadas, un árbol de prueba generado, etc. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

SearchTheoremsCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de buscar, para una clase de prueba, las posibles combinaciones de las expresiones contenidas en sus predicados que se ajustan a la firma de los teoremas de eliminación. Esto permite determinar cuáles son los teoremas de eliminación candidatos para aplicar a la clase de prueba. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

SelOpCommand

Módulo físico que implementa el comando de interfaz de usuario que permite seleccionar una operación a testear junto con la estrategia de aplicación de tácticas que le corresponde. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

SelPredCommand

Módulo físico que implementa el comando de interfaz de usuario que permite seleccionar un esquema como predicado. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

SetAxDefCommand

Módulo físico que implementa el comando de interfaz de usuario que permite establecer valores para las variables dadas en las definiciones axiomáticas. Se valida que el valor ingresado tenga el tipo adecuado respecto a las definiciones axiomáticas y que también se veriquen sus predicados. Además no podrá darse el valor para una variable que aparezca en el predicado de una clase de prueba previamente podada. SetAxDefCommand oculta algoritmos y estructuras de datos necesarias para lograr su propósito.

SetFiniteModelsCommand

Módulo físico que implementa el comando de interfaz de usuario que permite establecer la estrategia de generación de casos de prueba que será utilizada con la clase de prueba especificada. En particular se determinarán los modelos finitos a considerarse en el cálculo de casos de prueba. SetFiniteModelsCommand oculta algoritmos y estructuras de datos necesarias para lograr su propósito.

ShowAxDefCommand

Módulo físico que implementa el comando de interfaz de usuario que permite mostrar en pantalla las definiciones axiomáticas que hayan sido cargadas con una especificación. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowAxDefValuesCommand

Módulo físico que implementa el comando de interfaz de usuario que permite mostrar en pantalla los valores asignados a las variables definidas en las definiciones axiomáticas que hayan sido cargadas con una especificación. Estos valores pueden estar dados en los predicados de las definiciones o haber sido ingresados por el usuario. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowConcreteTCaseCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de presentar ante el usuario los casos refinados. Se pueden mostrar todos los casos refinados en la actual sesion de Fastest, los casos relacionados con una operacion en particular o un caso en especial. De acuerdo a la opcion que se setee los resultados se mostraran por pantalla o se alamacenaran en un archivo.Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowHelpCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de presentación de la ayuda de Fastest. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowLoadedOpsCommand

Módulo físico que implementa el comando de interfaz de usuario que que permite presentar en pantalla los nombres de las operaciones cargadas con la especificación actual. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowSchCommand

Módulo físico que implementa el comando de interfaz de usuario que permite presentar los esquemas que estén en el contexto de la herramienta. Esto es, mostrar esquemas de operación y clases y casos de prueba, en su totalidad, por operación o individualmente. La impresión puede hacerse en pantalla o sobre algún archivo especificado, con la posibilidad de crear un archivo en formato Latex que posteriormente puede ser compilado. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowSelOpsCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de presentación de los nombres de las operaciones seleccionadas para ser testeadas. Junto con el nombre, se imprime la estrategia de aplicación de tácticas y la lista de tácticas que le corresponde a cada una. ShowSelOpsCommand oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowSelPredsCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de presentación de los nombres de los esquemas seleccionados como predicados. ShowSelPredsCommand oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowSpecCommand

Módulo físico que implementa el comando de interfaz de usuario que permite presentar la especificación cargada en el contexto de la herramienta, tanto en su forma original como con sus esquemas expandidos. La impresión puede hacerse en pantalla o sobre algún archivo especificado, con la posibilidad de crear un archivo en formato Latex, que posteriormente puede ser compilado. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowTacicsCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de presentación de las distintas tácticas de testing disponibles en la herramienta. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowTTCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la posibilidad de presentar un árbol de prueba generado para determinada operación, o si se desea, de todos los árboles de prueba que hayan sido generados. La impresión puede hacerse en pantalla o sobre algún archivo especificado, con la posibilidad de crear un archivo latex que posteriormente puede ser compilado. ShowTTCommand oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

ShowVersionCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de presentación de la versión actual del Fastest. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

UnPruneCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la posibilidad de quitar la marca de poda de una clase de prueba en un árbol

de prueba. Esto permite revertir la poda de una clase, que pudo ordenarse incorrectamente. UnPruneCommand oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

UnSelAllOpsCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de quitar todas las operaciones de la lista de operaciones a testear. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

UnSelAllPredsCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de quitar la selección de todos los esquemas seleccionados como predicados. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

UnSelOpCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de quitar una operación de la lista de operaciones a testear. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

UnSelPredCommand

Módulo físico que implementa el comando de interfaz de usuario que brinda la funcionalidad de quitar la selección de un esquema seleccionado como predicado. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

1.2.2. TCaseStrategyParsers

Módulo lógico que agrupa a aquellos módulos que realizan algún parseo de la entrada del usuario al momento que este elige la estrategia de generación de casos de prueba.

CompleteTCaseStrategyParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia completa de generación de casos de prueba. El módulo oculta el algoritmo que implementa la mencionada tarea.

GivenIntFiniteModelParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia iterativa de generación de casos de prueba que en particular genera los modelos finitos para los enteros utilizando los valores que aparecen en la clase de prueba para la que se quiere encontrar un caso de prueba. GivenIntFiniteModelParser oculta el algoritmo que implementa la mencionada tarea.

GivenNatFiniteModelParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia iterativa de generación de casos de prueba que en particular genera los modelos finitos para los naturales utilizando los valores que aparecen en la clase de prueba para la que se quiere encontrar un caso de prueba. GivenNatFiniteModelParser oculta el algoritmo que implementa la mencionada tarea.

IntFiniteModelParser

Interfaz que abstrae el parseo de los parámetros que el usuario del sistema pasa al seleccionar la estrategia iterativa de generación de casos de prueba junto con determinada estrategia para generar los modelos finitos para el tipo de los enteros. Debe crearse un heredero de este módulo cada vez que se agrega una nueva estrategia al sistema.

IterativeTCaseStrategyParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia iterativa de generación de casos de prueba. El módulo oculta el algoritmo que implementa la mencionada tarea.

NatFiniteModelParser

Interfaz que abstrae el parseo de los parámetros que el usuario del sistema pasa al seleccionar la estrategia iterativa de generación de casos de prueba junto con determinada estrategia para generar los modelos finitos para el tipo de los enteros. Debe crearse un heredero de este módulo cada vez que se agrega una nueva estrategia al sistema.

SeedsIntFiniteModelParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia iterativa de generación de casos de prueba que en particular genera los modelos finitos para los enteros utilizando los valores que aparecen en la clase de prueba para la que se quiere encontrar un caso de prueba, y que además les agrega valores intermedios entre estos. SeedsIntFiniteModelParser oculta el algoritmo que implementa la mencionada tarea.

SeedsNatFiniteModelParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia iterativa de generación de casos de prueba que en particular genera los modelos finitos para los naturales utilizando los valores que aparecen en la clase de prueba para la que se quiere encontrar un caso de prueba, y que además les agrega valores intermedios entre estos. SeedsNatFiniteModelParser oculta el algoritmo que implementa la mencionada tarea.

TCaseStrategyParser

Interfaz que abstrae el parseo de los parámetros que el usuario del sistema pasa al seleccionar determinada estrategia de generación de casos de prueba. Debe crearse un heredero de este módulo cada vez que se agrega una nueva estrategia al sistema.

ZeroIntvParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia iterativa de generación de casos de prueba que en particular genera los modelos finitos para los enteros utilizando los valores que rodean al 0 en una cantidad solicitada. ZeroIntFiniteModelParser oculta el algoritmo que implementa la mencionada tarea.

ZeroNatFiniteModelParser

Módulo físico que brinda la funcionalidad de parseo de los parámetros que el usuario del sistema pasa al elegir la estrategia iterativa de generación de casos de prueba que en particular genera los modelos finitos para los naturales utilizando los valores mayores al 0 en una cantidad solicitada. ZeroNatFiniteModelParser oculta el algoritmo que implementa la mencionada tarea.

1.2.3. ClientGUI

Módulo físico encargado de implementar la interfaz gráfica de usuario. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito. Este módulo es heredero de ClientUI, dado que la interfaz gráfica es una interfaz de usuario particular.

1.2.4. ClientTextUI

Módulo físico encargado de implementar la interfaz de usuario en modo texto. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito. Este módulo es heredero de ClientUI, dado que la interfaz en modo texto es una interfaz de usuario particular.

1.2.5. ClientUI

Módulo físico encargado de brindar la funcionalidad general de las interfaces de usuario. Oculta algoritmos y estructuras de datos necesarios para lograr su propósito.

1.3. Util

Módulo lógico que agrupa aquellos módulos que brindan alguna funcionalidad general que es de utilidad para el sistema.

1.3.1. AbstractOrder

Interfaz cuya responsabilidad es dar la funcionalidad correspondiente a una orden, es decir, una unidad de cómputo donde se indica la petición de una llamada a procedimiento y el receptor de la llamada. El módulo fue concebido en base al patrón de diseño Command.

Capítulo 2

Common

Este módulo lógico incluye los módulos que representan elementos de software comunes a las aplicaciones Cliente y Servidor de Cómputo. El mismo se divide en los siguientes módulos lógicos.

2.1. RepositoryModules

Es el módulo lógico que agrupa módulos relacionados con repositorios abstractos de objetos y con iteradores que pueden definirse sobre estos repositorios.

2.1.1. AbstractIterator

Módulo físico que brinda la funcionalidad correspondiente a un iterador sobre agregados de elementos. Este módulo no se implementa, solo brinda una interfaz para los módulos que heredan de él. Es un módulo genérico cuyo parámetro formal corresponde al tipo de los elementos sobre los que se puede iterar.

2.1.2. AbstractRepository

Módulo físico que brinda la funcionalidad correspondiente a un agregado o repositorio abstracto de elementos. Este módulo no se implementa, solo brinda una interfaz para los módulos que heredan de él. Es un módulo genérico cuyo parámetro formal corresponde al tipo de los elementos que pueden almacenarse en el repositorio.

2.1.3. ConcreteRepository

Hereda del módulo AbstractRepository ya que resulta ser una implementación particular de él. Oculta las estructuras de datos y los algoritmos que cumplen con la interfaz dada por AbstractRepository. Al heredar de un módulo genérico este módulo resulta ser también genérico.

2.2. ZAbstraction

Es un módulo lógico que incluye módulos ligados a construcciones del lenguaje de especificaciones Z.

2.2.1. CZT

Aquí se agrupan los distintos módulos relacionados con el proyecto CZT (Community Z Tools) y que son los que implementan las distintas construcciones del lenguaje de especificaciones Z, un parser de expresiones Z dadas en formato Latex, y un evaluador de predicados y expresiones Z.

CZTVisitors

Módulo lógico que agrupa aquellos módulos que permiten visitar construcciones en lenguaje Z aplicando cierta operación.

AndOrPredDistributor Módulo físico basado en el patrón de diseño Visitor que permite aplicar a predicados la propiedad distributiva de la conjunción respecto a la disyunción. El módulo oculta los algoritmos que implementan dicha tarea.

AndPredClausesExtractor Módulo físico basado en el patrón de diseño Visitor que permite obtener, de una conjunción de predicados, el repositorio con los predicados que la conforman. El módulo oculta los algoritmos que implementan dicha tarea.

AndPredClausesRemover Módulo físico basado en el patrón de diseño Visitor que permite remover de un predicado que sea una conjunción, todas las clausulas iguales a cierto predicado p. p es un predicado que se toma como argumento en el constructor del módulo. El módulo oculta los algoritmos y estructuras de datos que implementan dicha tarea.

AndPredSimplifier Módulo físico basado en el patrón de diseño Visitor que permite simplificar predicados aplicando las reglas de simplificación de idempotencia y de absorción de la conjunción. El módulo oculta los algoritmos que implementan dicha tarea.

AuxDeclGenerator Módulo físico basado en el patrón de diseño Visitor que permite construir recursivamente esquemas auxiliares que contengan declaraciones de variables de entrada o de estado sin primar y no tengan predicados. Estos esquemas auxiliares se pueden reemplazar por los originales cuando quiera hacerse referencia solamente a las declaraciones de variables de entrada. El módulo oculta los algoritmos que implementan dicha tarea.

ContainsTermInPredVerifier Módulo físico basado en el patrón de diseño Visitor que permite verificar si un término Z dado está contenido en el predicado de un esquema, pudiendo a su vez este esquema tener incluidos otros esquemas. El módulo oculta los algoritmos que implementan dicha tarea.

ContainsTermVerifier Módulo físico basado en el patrón de diseño Visitor que permite verificar si un término del lenguaje Z contiene a otro. El módulo oculta los algoritmos que implementan dicha tarea.

ContainsVarDeclVerifier Módulo físico basado en el patrón de diseño Visitor que permite verificar si una variable está declarada en un esquema, pudiendo estar este esquema incluyendo otros esquemas. El módulo oculta los algoritmos que implementan dicha tarea.

CZTCloner Módulo físico basado en el patrón de diseño Visitor que permite realizar la clonación (en términos de programación Java) de objetos de las diferentes clases definidas por CZT. El módulo oculta los algormitmos que implementan tal funcionalidad.

CZTReplacer Módulo físico basado en el patrón de diseño Visitor que permite reemplazar un término Z por otro en el contexto del término Z al que se aplica el visitante. El término original y el que reemplazará al original se deben pasar como parámetros al constructor del módulo. CZTReplacer oculta estructuras de datos y algoritmos que implementan la tarea mencionada.

DeclsExtractor Módulo físico basado en el patrón de diseño Visitor que permite calcular la lista de declaraciones del Valid Input Space (VIS) para una operación. Este módulo calcula la lista de declaraciones del VIS expandiendo solamente los esquemas que no estén seleccionados como operaciones ni como predicados. DeclsExtractor oculta estructuras de datos y algoritmos que implementan la tarea mencionada.

DNFPredExtractor Módulo físico basado en el patrón de diseño Visitor que permite calcular el predicado del Valid Input Space (VIS) para una operación. Este predicado lo calcula en forma de lista de predicados en DNF, con esta lista representando la conjunción de los predicados que la componen. Este módulo calcula la lista de predicados en DNF del VIS expandiendo solamente los esquemas que no estén seleccionados como operaciones ni como predicados. DNFPredExtractor oculta estructuras de datos y algoritmos que implementan la tarea mencionada.

ImpliesPredRemover Módulo físico basado en el patrón de diseño Visitor que permite transformar todas los predicados de la forma $a \Rightarrow b$ en predicados de la forma $\neg\ a \lor b$. El módulo oculta los algoritmos que implementan dicha tarea.

NegPredDistributor Módulo físico basado en el patrón de diseño Visitor que permite aplicar a predicados las Leyes de Morgan y la eliminación de la doble negación. El módulo oculta los algoritmos que implementan dicha tarea.

OpNamesExtractor Módulo físico basado en el patrón de diseño Visitor que permite recorrer una especificación y extrayendo en una lista los nombres de los esquemas que son esquemas de operación. El módulo oculta los algoritmos que implementan dicha tarea.

OrPredRemover Módulo físico basado en el patrón de diseño Visitor que permite remover de un predicado que sea una conjunción, todas las clausulas que sean disyunciones tales que alguno de los operandos que las conforman sea igual a cierto predicado p que se toma como argumento en el constructor del módulo. El módulo oculta los algoritmos que implementan dicha tarea.

OutputAtomicPredsRemover Módulo físico basado en el patrón de diseño Visitor que permite tomar un predicado en DNF y remover de él los predicados atómicos que contengan variables primadas o de salida. El módulo oculta los algoritmos que implementan dicha tarea.

ParamExtractor Módulo físico basado en el patrón de diseño Visitor que permite, dado un término del lenguaje Z, obtener una lista con sus parámetros. Por ejemplo, al término $a = b$ lo conforman a y b.El módulo oculta los algoritmos que implementan dicha tarea.

PredInOrVerifier Módulo físico basado en el patrón de diseño Visitor que permite determinar si el predicado al que se le aplica el visitante es una disyunción donde uno de sus operandos es el predicado que se pasa al constructor del módulo. El módulo oculta los algoritmos que implementan dicha tarea.

PredRemover Módulo físico basado en el patrón de diseño Visitor que permite remover de un predicado, que esté en forma normal disyuntiva, todos los literales (o negaciones de literales) que contengan la ocurrencia de alguno de los nombres de variables contenidos en la instancia de VarNameRepository que el visitante toma como parámetro. El módulo oculta los algoritmos y estructuras de datos que implementan dicha tarea.

PredSchemaUnfolder Módulo físico basado en el patrón de diseño Visitor que permite expandir los referencias a esquemas que aparezcan en un predicado, por los predicados de los esquemas referenciados, siempre y cuando estos esquemas no estén seleccionado como operaciones o como predicados. El módulo oculta los algoritmos y estructuras de datos que implementan dicha tarea.

PrimeVarsMaker Módulo físico basado en el patrón de diseño Visitor que permite primar todas las variables (es decir, concatenarles el caracter $'$) en la expresión o esquema Z visitado. El módulo oculta los algoritmos que implementan dicha tarea.

DeleteParenAnn Módulo físico basado en el patrón de diseño Visitor que permite borrar todas las anotaciones referentes a paréntesis que almacena un término. El módulo oculta los algoritmos que implementan dicha tarea.

ExpressionsExtractor Módulo físico basado en el patrón de diseño Visitor que permite extraer todas las expresiones bien formadas que integran un AST. El módulo oculta los algoritmos que implementan dicha tarea.

FuncApplExtractor Módulo físico basado en el patrón de diseño Visitor que permite extraer los operandos en una aplicación de función. El módulo oculta los algoritmos que implementan dicha tarea.

MemPredExtractor Módulo físico basado en el patrón de diseño Visitor que permite extraer los operandos en una aplicación de un operador de relación. El módulo oculta los algoritmos que implementan dicha tarea.

RefExprToArithmos Módulo físico basado en el patrón de diseño Visitor que permite reemplazar las referencias a cualquier objeto por referencias a números naturales. El módulo oculta los algoritmos que implementan dicha tarea.

StringToNumReplacer Módulo físico basado en el patrón de diseño Visitor que permite reemplazar en una expresión todas ocurrencias de constantes alfanuméricas por ocurrencias de constantes numéricas. Esto se lleva a cabo de manera tal que todas las ocurrencias de una misma constante alfanumérica se reemplazan por la misma constante numérica y que dos ocurrencias de constantes alfanuméricas diferentes se reemplazan por constantes numéricas diferentes. Esto significa que a medida que se recorre la expresión se va construyendo una función inyectiva entre constantes alfanuméricas y constantes numéricas. Esta función debe almacenarse como miembro de la instancia de StringToNumReplacer. El módulo oculta los algoritmos y estructuras de datos que implementan dicha tarea.

WordsFinder Módulo físico basado en el patrón de diseño Visitor que permite determinar si alguna de las variables contenidas en el repositorio de nombres de variables que se pasa como argumento al constructor de instancias de este módulo ocurre en la expresión o esquema Z visitado. El módulo oculta los algoritmos que implementan dicha tarea.

UniqueZLive

Módulo físico que encapsula una instancia del módulo ZLive (de CZT) con el objeto de que pueda accederse a ella como si fuera la única existente en el sistema.

2.2.2. AbstractOutput

Interfaz que abstrae la salida concreta (a nivel de especificación) que surge de ejecutar un caso de prueba. Este módulo hereda del módulo OutputClass. Esto se decidió en base a que cada salida abstracta es una clase de salida Z con la condición adicional de que su predicado debe estar compuesto por la conjunción de cero o más predicados de la forma variable=expresión.

2.2.3. AbstractOutputImpl

Módulo físico que implementa la interfaz AbstractOutput.

2.2.4. AbstractTCase

Módulo físico que oculta la estructura de datos y los algoritmos que implementan una clase de prueba del lenguaje Z. Este módulo hereda del módulo TClass. Esto se decidió en base a que cada caso de prueba Z es una clase de prueba Z con la restricción adicional de tener a su predicado compuesto por la conjunción de cero o más predicados de la forma variable=expresión.

2.2.5. AxDef

Interfaz que abstrae una definición axiomática del lenguaje Z. Este módulo hereda de AxPara (módulo del subsistema CZT) ya que cada instancia del módulo AxDef puede pensarse como una instancia de AxPara con una restricción sobre su contenido.

2.2.6. AxDefImpl

Módulo físico que implementa la interfaz AxDef.

2.2.7. AxParaRepository

Módulo físico que oculta la estructura de datos y los algoritmos que implementan un agregado o repositorio de instancias de AxPara

2.2.8. HorizDef

Interfaz que abstrae una definición horizontal del lenguaje Z. Este módulo hereda de AxPara (módulo del subsistema CZT) ya que cada instancia del módulo HorizDef es, en términos semánticos, una instancia de AxPara con una restricción sobre su contenido.

2.2.9. HorizDefImpl

Módulo físico que implementa la interfaz HorizDef.

2.2.10. OpNameRepository

Módulo físico que oculta la estructura de datos y los algoritmos que implementan un agregado o repositorio de instancias de OpName.

2.2.11. OpScheme

Interfaz que abstrae un esquema de operación del lenguaje Z. Este módulo hereda del módulo Scheme. Esto se decidió en base a que cada esquema de operación Z es un esquema Z con la condición de tener variables de entrada y salida o variables no primadas y primadas.

2.2.12. OpSchemeImpl

Módulo físico que implementa la interfaz OpScheme.

2.2.13. OutputClass

Interfaz que abstrae una clase de salida del lenguaje Z. Este módulo hereda del módulo Scheme. Esto se decidió en base a que cada clase de salida Z es un esquema Z con la restricción adicional de no tener variables de entrada, de salida ni no primadas.

2.2.14. OutputClassImpl

Módulo físico que implementa la interfaz OutputClass.

2.2.15. OutputClassRepository

Módulo físico que oculta la estructura de datos y los algoritmos que implementan un agregado o repositorio de instancias de OutputClass.

2.2.16. Scheme

Interfaz que abstrae un esquema del lenguaje Z. Este módulo hereda de AxPara (módulo del subsistema CZT) ya que cada instancia del módulo Scheme es, en términos semánticos, una instancia de AxPara con una restricción adicional sobre su contenido.

2.2.17. SchemeImpl

Módulo físico que implementa la interfaz Scheme.

2.2.18. SchemeUnfolder

Módulo físico cuyo funcionalidad consiste en realizar la expansión de esquemas de una especificación Z. Oculta la estructura de datos y los algoritmos que implementan tal tarea.

2.2.19. SchName

Módulo físico que representa el nombre de un esquema. Se usa por comodidad en la descripción del sistema y no se implementa.

2.2.20. SpecUtils

Módulo físico que brinda funcionalidades útiles en el manejo de especificaciones Z, como ser funciones que calculan el Valid Input Space (VIS) de una operación o que partiendo de una especificación y un nombre de operación obtienen el correspondiente esquema de operación . El módulo oculta los algoritmos que implementan tales tareas.

2.2.21. TClass

Interfaz que abstrae una clase de prueba del lenguaje Z. Este módulo hereda del módulo Scheme. Esto se decidió en base a que cada clase de prueba Z es un esquema Z con la restricción no tener variables de entrada, de salida ni primadas.

2.2.22. TClassImpl

Módulo físico que implementa la interfaz TClass.

2.2.23. TClassRepository

Módulo físico que oculta la estructura de datos y los algoritmos que implementan un agregado o repositorio de instancias de TClass.

2.2.24. UtilSymbols

Módulo físico que oculta la estructura de datos y los algoritmos que implementan algunos símbolos comunes del lenguaje de especificaciones Z y que se utilizan frecuentemente en el código de la herramienta. Por ejemplo, se puede obtener el símbolo para los números naturales, el de los enteros, el de las flechas que correspondan a distintos tipos de funciones, etc.

2.2.25. VarName

Módulo físico que representa el nombre de una variable. Se usa por comodidad en la descripción del sistema y no se implementa.

2.2.26. VarNameRepository

Módulo físico que oculta la estructura de datos y los algoritmos que implementan un agregado o repositorio de instancias de VarName.

2.2.27. VISGen

Módulo físico que permite calcular el VIS (Valid Input Space) de un esquema de operación Z. Lo calcula teniendo en cuenta y no expandiendo los esquemas seleccionados como operaciones y los esquemas seleccionados como predicados. Oculta los algoritmos que implementan tal tarea.

2.3. OpName

Módulo físico que representa el nombre de una operación. Se usa por comodidad en la descripción del sistema y no se implementa.

2.4. Regex

Módulo lógico que nuclea a los módulos que brindan funcionalidades que involucran expresiones regulares.

2.4.1. RegExUtils

Módulo físico que brinda funcionalidades generales relativas a las expresiones regulares. Permite el agregados de caracteres de escape a una cadena de caracteres así como también la construcción de ciertas expresiones. Oculta los algoritmos empleados.

2.5. Util

Módulo lógico que agrupa módulos que brindan funcionalidades generales a la aplicación.

2.5.1. MathUtils

Módulo físico que brinda funcionalidades relativas a operaciones matemáticas, tales como el cálculo de permutaciones y conjuntos de partes de conjuntos. Oculta los algoritmos empleados.

Capítulo 3

CompServer

Módulo lógico que agrupa la funcionalidad y los datos correspondientes a la aplicación Servidor de Cómputo del sistema Fastest. Se decidió dividir al módulo separando unidades de cómputo que ofrecen servicios de cálculo de casos de prueba de aquellas que verifican la correspondencia de casos de prueba con la salida que resulta de ejecutarlos, de aquellas que se encargan de podar el árbol de testing y de aquellas que tienen que ver con el manejo de solicitudes de servicios de aplicaciones Cliente.

3.1. ServerTCaseChecking

Módulo lógico que contiene módulos que implementan la verificación de correspondencias entre casos de pruebas abstractas con las salidas abstracta que les corresponden.

3.1.1. TCaseChecker

Módulo físico que provee funcionalidad para comprobar si cierto caso de prueba abstracto produce el resultado esperado en relación a una especificación dada. Oculta implementación de algoritmos.

3.2. ServerTCaseGeneration

Módulo lógico que agrupa a aquellos módulos encargados de implementar la generación de casos de prueba abstractos. Se divide en módulos de acuerdo a si tienen relación con modelos finitos, con la generación de casos de prueba propiamente dicha, o con la evaluación de predicados de esquemas Z.

3.2.1. Evaluation

Módulo lógico que contiene aquellos módulos relacionados con la evaluación de predicados de esquemas del lenguaje Z.

EvaluationResp

Módulo físico que representa un par de elementos (r,l), donde r es una instancia de Bool y l es una instancia de String. Este módulo es utilizado por SchemeEvaluator para devolver el resultado y un log de resumen de la evaluación realizada. El módulo EvaluationResp oculta las estructuras de datos y los algoritmos que implementan al par.

NormalTypeAndFM

Módulo físico que representa un par de elementos (t,fm), donde t es una instancia del módulo Expr y fm es una instancia del módulo SetExpr. El módulo NormalTypeAndFM oculta las estructuras de datos y los algoritmos que implementan al par.

SchemeEvaluator

Módulo físico que permite evaluar si el predicado de una clase de prueba es satisfecho por un conjunto dado de asignaciones de variables a expresiones. Esta evaluación se lleva a cabo utilizando el procedimiento evalSchemeSat(), que toma como parámetros la clase de prueba (instancia de TClass) y la instancia de VarExprMap que mapea nombres de variables a expresiones y devuelve una instancia de EvaluationResp con la respuesta de la evaluación y un log de resumen de la misma. El módulo SchemeEvaluator oculta las estructuras de datos y los algoritmos que implementan al par.

3.2.2. VarValueMap

Módulo físico cuyas instancias mapean instancias del módulo RefExpr a instancias del módulo Expr, que corresponden a nombres de variables y a valores, respectivamente. El módulo se implementa utilizando la colección Map de Java instanciada en RefExpr y en Expr.

3.2.3. FiniteModels

Módulo lógico que agrupa a aquellos otros que participan en la generación de modelos finitos para tipos del lenguaje Z.

IntFiniteModels

Módulo lógico que agrupa a aquellos otros módulos que participan en la generación de modelos finitos para el tipo de los enteros en el lenguaje Z.

GivenIntFiniteModel Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito, asociado al tipo el lenguaje Z que corresponde a los enteros, y que en particular construye el conjunto de valores con los de una lista dada. El módulo es heredero de IntFiniteModel ya que implementa un modelo finito para los enteros.

IntFiniteModel Interfaz que abstrae al modelo finito asociado al tipo del lenguaje Z que corresponde a los números enteros.

SeedsIntFiniteModel Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito, asociado al tipo del lenguaje Z que corresponde a los enteros, y que en particular construye el conjunto de valores con los de una lista dada y agregando valores intermedios entre cada par consecutivo de estos. El módulo es heredero de IntFiniteModel ya que implementa un modelo finito para los enteros.

ZeroIntFiniteModel Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito, asociado al tipo del lenguaje Z que corresponde a los enteros, y que en particular considera una cantidad indicada de valores que rodean al cero. El módulo es heredero de IntFiniteModel ya que implementa un modelo finito para los enteros.

NatFiniteModels

Módulo lógico que agrupa a aquellos otros módulos que participan en la generación de modelos finitos para el tipo de los naturales en el lenguaje Z.

GivenNatFiniteModel Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito, asociado al tipo del lenguaje Z que corresponde a los naturales, y que en particular construye el conjunto de valores con los de una lista dada. El módulo es heredero de NatFiniteModel ya que implementa un modelo finito para los naturales.

NatFiniteModel Interfaz que abstrae al modelo finito asociado al tipo del lenguaje Z que corresponde a los números naturales.

SeedsNatFiniteModel Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito, asociado al tipo del lenguaje Z que corresponde a los naturales, y que en particular construye el conjunto de valores con los de una lista dada y agregando valores intermedios entre cada par consecutivo de estos. El módulo es heredero de NatFiniteModel ya que implementa un modelo finito para los naturales.

ZeroNatFiniteModel Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo del lenguaje Z que corresponde a los naturales, y que en particular considera una cantidad indicada de valores que son consecutivos y mayores al cero. El módulo es heredero de NatFiniteModel ya que implementa un modelo finito para los naturales.

FiniteModel

Interfaz que abstrae la estructura de datos y algoritmos que implementan un modelo finito. Un modelo finito es una estructura que se asocia a un tipo particular del lenguaje Z y, haciendo las veces de un iterador, puede solicitarsele, de a uno, los distintos elementos que conforman un subconjunto finito del tipo asociado. Este módulo se utiliza en el contexto de la estrategia IterativeTCaseStrategy, y como ésta sugiere, las instancias de FiniteModel no ocultan en su estado el modelo finito completo del tipo asociado, sino que sólo mantienen

la mínima información necesaria para saber qué elemento del modelo debe devolverse la próxima vez que se solicite uno. Tiene que haber un módulo heredero de FiniteModel por cada tipo diferente del lenguaje Z.

FiniteModelCreator

Módulo físico basado en el patrón de diseño Visitor que permite recorrer una especificación de forma tal de ir creando, y agregando a una estructura que forma parte del estado del módulo, modelos finitos (instancias que heredan de FiniteModel), uno por cada tipo que se defina globalmente en la especificación.

FreetypeFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado a un tipo libre de datos del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa modelo finito particular.

FromToFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo del lenguaje Z que corresponde a un rango de valores y que se indica de la forma '$a \,..\, b$'. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

GivenFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado a un tipo básico de datos del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

PFuncFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo de las funciones parciales del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

PowerFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo potencia del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

ProdFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo producto cartesiano del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

RelFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo de las relaciones del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

SeqFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo de las secuencias del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

SetFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo de los conjuntos enumerados del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

TClassFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado a una clase de prueba para la cual se quiere obtener un caso de prueba. TClassFiniteModel permite ir obteniendo, una a una, las distintas asignaciones de valores a variables de la clase de prueba asociada. Para obtener estos valores, el módulo utiliza los modelos finitos que están asociados a los tipos de las variables, solicitando valores a cada uno, según corresponda, para obtener una nueva asignación. Para evitar confusiones, vale la pena aclarar que éste módulo no hereda de FiniteModel.

TFuncFiniteModel

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan un modelo finito asociado al tipo de los conjuntos del lenguaje Z. El módulo es heredero de FiniteModel ya que implementa un modelo finito particular.

3.2.4. TypeFiniteModelMap

Módulo físico cuyas instancias mapean instancias del módulo Expr a instancias del módulo FiniteModel, que corresponden a nombres de tipos de variables y a modelos finitos, respectivamente. El módulo se implementa utilizando la colección Map de Java instanciada en Expr y en FiniteModel.

3.2.5. TypeFMsMap

Módulo físico cuyas instancias mapean instancias del módulo Expr a instancias del módulo NormalTypeAndFM, que corresponden a nombres de tipos de

variables y a pares (tipo normalizado, modelo finito), respectivamente. El módulo se implementa utilizando la colección Map de Java instanciada en Expr y en FiniteModel.

TypeFMsGenVisitor

Módulo físico basado en el patrón de diseño Visitor que permite obtener los modelos finitos y los tipos normalizados de todos los tipos que se definen globalmente en una especificación. Al aplicar este visitante a una especificación, la misma no resulta modificada pero se devuelve una instancia del módulo TypeFMsMap, que es la que contiene la correspondencia entre instancias de Expr, que denoten tipos, e instancias de NormalTypeAndFM, que denoten un par con el tipo normalizado (dado por una instancia de Expr) y el modelo finito del tipo (dado por una instancia de SetExpr). Por el momento, este módulo permite generar modelos finitos para tipos básicos (instancias de GivenPara), tipos libres no inductivos (instancias de Freetype) y abreviaturas que no hagan referencia a esquemas (instancias de AxPara que se correspondan a definiciones horizontales que no sean esquemas ni hagan referencia a esquemas). TypeFMsGenVisitor oculta la implementación de los algoritmos y las estructuras necesarias para llevar a cabo la tarea descripta.

3.2.6. TCaseStrategies

Este es un módulo lógico que contiene a aquellos módulos que representan distintas estrategias de generación de modelos finitos.

CompleteTCaseStrategy

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan una estrategia particular de generación de casos de prueba. La estrategia consiste en primero calcular los modelos finitos de TODOS los tipos que se definen globalmente en la especificación. Luego, los de los tipos de las variables de la clase de prueba para la que se quiere encontrar un caso de prueba. Y por último, se encarga de iterar sobre las asignaciones (tomadas de los modelos finitos de los tipos) de valores a variables de la clase de prueba, hasta encontrar una que satisfaga el predicado de la clase de prueba analizada. Para hacer la evaluación de cada asignación de valores se llama al procedimiento evalSchemeSat, del módulo SchemeEvaluator. CompleteTCaseStrategy hereda del módulo TCaseStrategy, dado que implementa una estrategia particular de generación de casos de prueba.

IterativeTCaseStrategy

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan una estrategia particular de generación de casos de prueba. La estrategia consiste en ir generando asignaciones de valores a las variables de la clase de prueba para la que se quiere obtener un caso de prueba hasta que alguna de ellas satisgafa el predicado de dicha clase de prueba. Para hacer la evaluación de cada asignación de valores se llama al procedimiento evalSchemeSat, del módulo SchemeEvaluator. A diferencia de la estrategia CompleteTCaseStrategy, IterativeTCaseStrategy no necesita generar el modelo finito completo para cada tipo

de variable de la clase de prueba ni para cada tipo definido globalmente en la especificación. Si bien esta estrategia es un poco menos eficiente teniendo en cuenta el orden temporal de complejidad, desde el punto de vista espacial la mejora es notable. IterativeTCaseStrategy hereda del módulo TCaseStrategy, dado que implementa una estrategia particular de generación de casos de prueba.

ManualTCaseStrategy

Módulo físico que oculta las estructuras de datos y los algoritmos que implementan una estrategia manual de generación de modelos finitos. Esto es, aquella estrategia que requiere que el usuario ingrese directamente los modelos finitos para los tipos de las variables de la clase de prueba Z analizada. Este módulo hereda de TCaseStrategy ya que representa una estrategia particular.

TCaseStrategy

Interfaz que abstrae las estructuras de datos y los algoritmos que implementan una estrategia de generación de casos de prueba. Este módulo está basado en el patrón de diseño Strategy. La generación de casos de prueba se hace a nivel de clase de prueba y no por operación ya que puede resultar conveniente que el encargado de testing realice alguna indicación para cada clase de prueba en función de su predicado. Es necesario definir un nuevo heredero de este módulo para agregar al sistema una nueva estrategia de generación de casos de prueba.

3.2.7. TCaseGen

Módulo físico que permite generar, o intentar generar, un caso de prueba abstracto dada una clase de prueba y la especificación que la contiene. Ocultas estructuras de datos y los algoritmos que implementan la tarea mencionada.

3.3. ServerManagement

3.3.1. ServerThread

Módulo físico sobre el que se atiende una solicitud de servicio particular. El procedimiento principal de este módulo es run(), en el cual se realizan solicitudes al módulo SocketReader para conocer la operación requerida así como también los parámetros necesarios para llevar a cabo la misma.Se regresa al cliente el resultado del servicio solicitado.

3.3.2. SocketReader

Módulo físico encargado de leer información en un flujo de entrada(en este caso un Socket) relacionada con la generación de casos de prueba. Cualquier modificación en el protocolo de comunicación Cliente/Servidor de Fastest implicará cambios en este módulo. El módulo oculta detalles del protocolo Cliente/Servidor subyacente así como también los algoritmos y estructuras de datos empleados para cumplir con su funcionalidad.

3.3.3. ServiceManager

Módulo físico encargado de iniciar el funcionamiento del Servidor de Cómputo correspondiente, atendiendo pedidos de servicio de los clientes. Que se puedan atender varios servicios en simultáneo se debe a que cada uno se corre en un thread diferente, invocando al procedimiento run() del módulo ServerThread. ServiceManager oculta las estructuras de datos y los algoritmos que implementan las tareas mencionadas.

3.4. ServerPrunning

Módulo lógico que contiene módulos que implementan la poda del árbol de testing

3.4.1. PruneUtils

Módulo físico que brinda funcionalidades comunes a los módulos que intervienen en el proceso de poda. Se encarga entre otras cosas de reemplazar parámetros formales por reales, de la búsqueda de cadenas dentro del predicado de las clases, etc. El módulo oculta los algoritmos que implementan tales tareas.

3.4.2. TheoremsLoader

Módulo físico que se encarga de parsear el archivo de configuración que contiene la definición de los teoremas de eliminación, creando instancias de Theorem y cargándolas en la única instancia de TheoremsControl. Además tiene la responsabilidad de aplicar las reglas de reescritura sobre los predicados de los teoremas. El módulo oculta los algoritmos que implementan tales tareas.

3.4.3. TheoremsControl

Módulo físico que almacena instancias de Theorem. Debe existir una única instancia por lo cual se emplea el patrón de diseño Singleton. El módulo oculta las estructuras de datos que emplea.

3.4.4. Theorem

Módulo físico que abstrae un teorema de eliminación. Almacena el nombre del teorema, los parámetros formales, el predicado e información relativa a los operadores especiales de Fastest. El módulo oculta las estructuras de datos empleadas para almacenar dicha información.

3.4.5. Variable

Módulo físico que abstrae una variable de un teorema de eliminación. Almacena información relativa al nombre de la variable y su tipo. El módulo oculta las estructuras de datos empleadas para almacenar dicha información.

3.4.6. PruneAnalizer

Módulo físico que se encarga de analizar la factibilidad de podar una clase de prueba aplicándole un teorema de eliminación y en caso afirmativo ordena la poda de dicha clase. El método principal del módulo es analize() que recibe el nombre de un teorema de eliminación, una clase de prueba y los parámetros reales. Este modulo era empleado en la primera version de la poda y en la actual ya es obsoleto. El módulo oculta el algoritmo que se encarga de coordinar los pasos necesarios para realizar su tarea.

3.4.7. TreePruner

Módulo físico que se encarga de la poda, propiamente dicha, del árbol de testing. El módulo ofrece métodos para podar una sola clase, para podar las clases descendientes de una clase en particular y para podar un árbol completo. Para este último caso interactúa con el módulo TheoremsChecker para que le obtenga una lista con los teoremas cuya expresion regular matchea con la de la clase analizada y los parámetros correspondientes y luego invoca al módulo TypeChecker para el correspondiente chequeo de tipos. El módulo oculta los algoritmos empleados para cumplir con sus funcionalidades.

3.4.8. TheoremsChecker

Modulo fisico que se encarga de buscar teoremas cuya expresion regular matchea con el predicado de una clase que se le pasa como parametro. Devuelve el teorema con las correspondientes expresiones de la clase que matchearon. El módulo oculta los algoritmos empleados para cumplir con sus funcionalidades.

3.4.9. TypeChecking

Módulo lógico que agrupa a aquellos módulos físicos encargados de realizar el chequeo de tipos entre las expresiones contenidas en los predicados de las clases de prueba y los parámetros formales de los teoremas de eliminación.

3.4.10. ParamExaminerPred

Módulo físico que se encarga de extraer todas las expresiones contenidas en el predicado de una clase de prueba que recibe como parámetro, calcular las permutaciones de esos elementos y verificar cuales de esas combinaciones se adaptan a la firma de los teoremas de eliminación. Posee un método para obtener una lista con las combinaciones de expresiones que se ajustan a la firma de un teorema en particular y otro para obtener una lista con todas las combinaciones para todos los teoremas de eliminación. Este módulo se empleaba en la primera versión de la poda pero en la versión actual es obsoleto. Oculta los algoritmos y estructuras de datos que emplea para cumplir con su funcionalidad.

3.4.11. SubTyping

Módulo físico que se encarga de aplicar el subtipado, determinando cuando el tipo de una expresión es subtipo o no del tipo de otra expresión. Posee una var-

iedad de métodos con diferentes firmas para adaptarse a diferentes situaciones. Oculta los algoritmos empleados para cumplir con su funcionalidad.

3.4.12. PrunningWorkRepository

Modulo físico que almacena el resultado del chequeo de tipos para un conjunto determinado de parametros, una firma de teorema y un arbol de testing en particular. Este módulo se empleaba en la primera versión de la poda pero en la versión actual es obsoleto. Oculta las estructuras de datos empleadas para cumplir con su funcionalidad.

3.4.13. TheoremSignature

Modulo fisico que abstrae la firma de un teorema. Oculta los algoritmos y estructuras de datos que emplea para cumplir con su funcionalidad.

3.4.14. TypeCheckingUtils

Modulo fisico que brinda funcionalidades generales para los modulos encargados del chequeo de tipos. Oculta los algoritmos y estructuras de datos que emplea para cumplir con su funcionalidad.

3.4.15. TypeChecker

Modulo fisico que dado un teorema, una clase de prueba y un conjunto de cadenas que representan expresiones de la clase que se pasarán como parámetros al teorema, se encarga de parsear las cadenas y realizar un chequeo de tipos sobre la firma del teorema. Oculta los algoritmos y estructuras de datos que emplea para cumplir con su funcionalidad.

3.4.16. Rewriting

Módulo lógico que agrupa a aquellos módulos encargados de reescribir los predicados de los teoremas de eliminación de una manera alternativa.

3.4.17. RWRules

Módulo lógico que contiene a los módulos físicos que abstraen reglas de reescritura de predicados (basándose en conmutatividad de operadores, asociatividad, equivalencia de predicados, etc.), al módulo de carga del archivo de configuración de reglas de reescritura y al módulo que los almacena.

3.4.18. RWEquality

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de igualdad, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.19. RWIntersection

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de intersección, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.20. RWUnion

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de unión, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.21. RWNotEqual

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de desigualdad, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.22. RWGreaterThan

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de desigualdad, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.23. RWLessThan

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de desigualdad, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.24. RWGreaterThanEqual

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de desigualdad, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.25. RWLessThanEqual

Módulo físico que se encarga de aplicar la propiedad de conmutatividad del operador de desigualdad, devolviendo la expresión original y la que se obtiene de alterar el orden de los operandos.

3.4.26. RWRule

Interfaz de la cual heredan todas las reglas de reescritura.

3.4.27. RWRuleOperator

Interfaz de la cual heredan todas las reglas de reescritura relacionadas con propiedades de operadores, tales como conmutatividad y asociatividad. Extiende la interfaz RWRule.

3.4.28. RWRuleLaw

Interfaz de la cual heredan todas las reglas de reescritura relacionadas con leyes matemáticas. Extiende la interfaz RWRule.

3.4.29. RWRuleLawImpl

Módulo físico que implementa la interfaz RWRuleLaw. Los 2 métodos más importantes son match() y rewrite(). El método match() se emplea para determinar si un predicado satisface la ley y el método rewrite() devuelve la expresión alternativa. Oculta los algoritmos y estructuras de datos empleados.

3.4.30. RWRulesLoader

Módulo físico que se encarga de parsear el archivo de configuración que contiene la definición de las reglas de reescritura, creando instancias de RWRule y cargándolas en la única instancia de RWRulesControl. El módulo oculta los algoritmos que implementan tales tareas.

3.4.31. RWRulesControl

Módulo físico que almacena instancias de RWRule. Debe existir una única instancia por lo cual se emplea el patrón de diseño Singleton. El módulo oculta las estructuras de datos que emplea.

3.4.32. Operators

Módulo lógico que agrupa a los operadores que implementan los operadores de Fastest, así como también otros módulos afines.

3.4.33. AnythingOperator

Módulo físico que implementa el operador anything de Fastest. El operador se emplea para poder representar en un predicado en formato de texto una cadena que puede estar conformada por cualquier caracter. Recibe una clase y un predicado en forma de texto y devuelve un valor booleano que determina si en el predicado de la clase se encuentra una linea que satisface la expresion una expresión regular que se crea a partir del predicado. Oculta los algoritmos empleados para satisfacer su funcionalidad.

3.4.34. EvalOperator

Módulo físico que implementa el operador eval de Fastest. El operador se emplea para evaluar el valor de verdad de un predicado que solo contiene constantes numéricas. Oculta los algoritmos empleados para satisfacer su funcionalidad.

3.4.35. SomewhereOperator

Módulo físico que implementa el operador somewhere de Fastest. El operado se emplea para determinar si en un determinado ámbito se encuentra una cadena de caracters. Oculta los algoritmos empleados para satisfacer su funcionalidad.

3.4.36. OperatorAnalizer

Módulo físico que se encarga de coordinar los operadores de Fastest. Analiza qué operador está involucrado en un predicado determinado y llama al operador en cuestión. Oculta los algoritmos que emplea.

3.4.37. SpecialLine

Módulo físico que almacena información acerca de un predicado atómico que contiene un operador propio de Fastest. Almacena el nombre del teorema al que pertenece, el predicado en cuestión y el operador involucrado. Oculta las estructuras de datos empleadas.

3.5. ServerAbstraction

Módulo lógico que agrupa los módulos encargados de la abstracción de casos de prueba.

3.5.1. AbstractionLaw

Módulo físico que almacena toda la información relacionada a una ley de abstracción. Oculta las estructuras de datos empleadas.

3.5.2. AbstractionRule

Módulo físico que almacena toda la información relacionada a una reglar de abstracción. Oculta las estructuras de datos empleadas.

3.5.3. AbstractionUtils

Módulo físico que contiene funcionalidades comunes a todo el proceso de abstracción. Oculta los algoritmos y estructuras de datos empleados.

3.5.4. VarComposition

Almacena la información que se utiliza en los casos en los que se abstraen más de una variable de la implementación a una variable de la especificación. Oculta las estructuras de datos empleadas.

3.5.5. Parser

Módulo lógico que agrupa a los módulos que se encargan de parsear las leyes de abstracción.

3.5.6. TRALLexer

Módulo físico generado a partir de la herramienta ANTLR que se encarga del análisis lexicográfico del archivo TRAL. Se modifica recompilando el archivo TRAL.g. Oculta los algoritmos y las estructuras de datos empleadas para cumplir con su objetivo.

3.5.7. TRALParser

Módulo físico generado a partir de la herramienta ANTLR que se encarga del parseo propiamente dicho del archivo TRAL. Se modifica recompilando el archivo TRAL.g. Oculta los algoritmos y las estructuras de datos empleadas para cumplir con su objetivo.

3.5.8. ABstractTCaseGeneration

Módulo lógico que agrupa a los módulos que se encargan de la abstracción del caso concreto.

3.5.9. ASTAbstraction

Módulo físico que se encarga de obtener una caso abstracto a partir de un caso concreto, una ley de abstracción y una lista con los valores capturados para las variables monitoreadas. Interactúa con el módulo VarAbstraction. Oculta los algoritmos empleados para cumplir con su objetivo.

3.5.10. VarAbstraction

Módulo físico que se encarga de la abstracción de una regla de abstracción en particular utilizando una lista con los valores capturados de la implementación que fueron monitoreados. Oculta los algoritmos empleados para tal fin.

3.5.11. Types

Módulo lógico que agrupa a los módulos que abstraen las variables con sus correspondientes tipos.

3.5.12. ImplTypes

Módulo lógico que agrupa a los módulos que abstraen las variables de la implementación con su información de tipos.

3.5.13. ImplNode

Interfaz de la cual heredarán los módulos que almacenan información de tipo de las variables de la implementación. Cualquier nuevo tipo de la implementación soportado debe agregarse como heredero de este módulo.

3.5.14. ImplNodeDB

Módulo físico que almacena toda la información de tipos de una variable de la implementación que se implementa como una base de datos. Oculta las estructuras de datos empleadas.

3.5.15. ImplNodeFile

Módulo físico que almacena toda la información de tipos de una variable de la implementación que se implementa como un archivo. Oculta las estructuras de datos empleadas.

3.5.16. ImplNodePLType

Módulo físico que almacena toda la información de tipos de una variable de la implementación que se implementa como un tipo primitivo del lenguaje. Oculta las estructuras de datos empleadas.

3.5.17. ImplNodePointer

Módulo físico que almacena toda la información de tipos de una variable de la implementación que se implementa como un puntero en C. Oculta las estructuras de datos empleadas.

3.5.18. ImplNodeScreen

Módulo físico que almacena toda la información necesaria para capturar información de la salida estándar. Oculta las estructuras de datos empleadas.

3.5.19. ImplNodeStructure

Módulo físico que almacena toda la información de tipos de una variable de la implementación que se implementa como una estructura. En C asumimos que es un variable de tipo struct y en Java asumimos que es una clase. Oculta las estructuras de datos empleadas.

3.5.20. ImplNodeList

Módulo físico que almacena toda la información de tipos de una variable de la implementación que se implementa como una lista. Oculta las estructuras de datos empleadas.

3.5.21. ImplNodeEnumeration

Módulo físico que se utiliza para mapear elementos de un tipo primitivo del lenguaje de implementación a elementos de un tipo enumerado en la especificación.

3.5.22. ImplNodeDBColumn

Módulo físico que almacena la información de tipos de una columna en una base de datos. Oculta las estructuras de datos empleadas.

3.5.23. ImplNodeField

Módulo físico que almacena la información de tipos de un campo de una estructura. Oculta las estructuras de datos empleadas.

3.5.24. SpecTypes

Módulo lógico que agrupa a los módulos que abstraen las variables de la especificación con su información de tipos.

3.5.25. SpecNode

Interfaz de la cual heredarán los módulos que almacenan información de tipo de las variables de la especificación. Cualquier nuevo tipo de la especificación soportado debe agregarse como heredero de este módulo.

3.5.26. SpecNodeBasicType

Módulo físico que almacena toda la información de tipos de una variable de la especificación de tipo básico. Oculta las estructuras de datos empleadas.

3.5.27. SpecNodeFreeType

Módulo físico que almacena toda la información de tipos de una variable de la especificación de tipo libre. Oculta las estructuras de datos empleadas.

3.5.28. SpecNodeNat

Módulo físico que almacena toda la información de tipos de una variable de la especificación de tipo natural. Oculta las estructuras de datos empleadas.

3.5.29. SpecNodeInt

Módulo físico que almacena toda la información de tipos de una variable de la especificación de tipo entero. Oculta las estructuras de datos empleadas.

3.5.30. SpecNodeBinRelation

Interfaz de la cual heredarán los módulos que almacenen información de tipos de una relación binaria.

3.5.31. SpecNodeFun

Módulo físico que almacena toda la información de tipos de una variable de la especificación que es una función total. Oculta las estructuras de datos empleadas.

3.5.32. SpecNodePFun

Módulo físico que almacena toda la información de tipos de una variable de la especificación que es una función parcial. Oculta las estructuras de datos empleadas.

3.5.33. SpecNodeRel

Módulo físico que almacena toda la información de tipos de una variable de la especificación que es una relación. Oculta las estructuras de datos empleadas.

3.5.34. SpecNodeSeq

Módulo físico que almacena toda la información de tipos de una variable de la especificación que es una secuencia. Oculta las estructuras de datos empleadas.

3.5.35. SpecNodeCartesianProd

Módulo físico que almacena toda la información de tipos de una variable de la especificación que es un producto cartesiano. Oculta las estructuras de datos empleadas.

3.5.36. SpecNodeFun

Módulo físico que almacena toda la información de tipos de una variable de la especificación que es un conjunto de partes. Oculta las estructuras de datos empleadas.

3.5.37. Capture

Módulo físico que agrupa a los módulos involucrados en el proceso de captura de las variables de la implementación.

3.5.38. CapturedVar

Interfaz de la cual heredarán los módulos que almacenen el valor de una variable de la implementación.

3.5.39. CapturedVarChar

Módulo físico que almacena el valor de una variable que se implementa como un caracter. Oculta las estructuras de datos empleadas.

3.5.40. CapturedVarDB

Módulo físico que almacena el valor de una variable que se implementa como una base de datos. Oculta las estructuras de datos empleadas.

3.5.41. CapturedVarFile

Módulo físico que almacena el valor de una variable que se implementa como un archivo. Oculta las estructuras de datos empleadas.

3.5.42. CapturedVarInt

Módulo físico que almacena el valor de una variable que se implementa como un entero. Oculta las estructuras de datos empleadas.

3.5.43. CapturedVarArray

Módulo físico que almacena el valor de una variable que se implementa como un arreglo. Oculta las estructuras de datos empleadas.

3.5.44. CapturedVarList

Módulo físico que almacena el valor de una variable que se implementa como una lista. Oculta las estructuras de datos empleadas.

3.5.45. CapturedVarPointer

Módulo físico que almacena el valor de una variable que se implementa como un puntero en C. Oculta las estructuras de datos empleadas.

3.5.46. CapturedVarReal

Módulo físico que almacena el valor de una variable que se implementa como un real. Oculta las estructuras de datos empleadas.

3.5.47. CapturedVarString

Módulo físico que almacena el valor de una variable que se implementa como un string. Oculta las estructuras de datos empleadas.

3.5.48. CapturedVarStructure

Módulo físico que almacena el valor de una variable que se implementa como una estructura. Oculta las estructuras de datos empleadas.

3.5.49. CapturedVarScreen

Módulo físico que almacena la información necesaria de una variable que debe ser capturada en la salida estándar.

3.5.50. XMLParser

Módulo físico que se encarga de parsear los archivos XML que generaron los casos concretos a los que se les agregó código de captura. Toma como parámetro el path del archivo y devuelde una lista con instancias de objetos herederos de CapturedVar que contienen los valores de las variable monitoreadas. Oculta los algoritmos y estructuras de datos empleadas.

Capítulo 4

DesignPatterns

Módulo lógico que agrupa a todos los patrones de diseño utilizados en el sistema.

4.1. AndOrPredDistributorVisPatt

Patrón de diseño basado en el patrón Visitor.

4.2. AndPredClausesExtractorVisPatt

Patrón de diseño basado en el patrón Visitor.

4.3. AndPredClausesRemoverVisPatt

Patrón de diseño basado en el patrón Visitor.

4.4. AndPredSimplifierVisPatt

Patrón de diseño basado en el patrón Visitor.

4.5. ContainsTermVerifierVisPatt

Patrón de diseño basado en el patrón Visitor.

4.6. CServersControlSinglPatt

Patrón de diseño basado en el patrón Singleton.

4.7. CZTClonerVisPatt

Patrón de diseño basado en el patrón Visitor.

4.8. CZTReplacerVisPatt

Patrón de diseño basado en el patrón Visitor.

4.9. DServerConfigSinglPatt

Patrón de diseño basado en el patrón Singleton.

4.10. EventAdminCommandPatt

Patrón de diseño basado en el patrón Command.

4.11. EventAdminSinglPatt

Patrón de diseño basado en el patrón Singleton.

4.12. FiniteModelCreatorVisPatt

Patrón de diseño basado en el patrón Visitor.

4.13. ImpliesPredRemover

Patrón de diseño basado en el patrón Visitor.

4.14. IteratorPatt

Patrón de diseño basado en el patrón Iterator.

4.15. NegPredDistributorVisPatt

Patrón de diseño basado en el patrón Visitor.

4.16. OrPredRemoverVisPatt

Patrón de diseño basado en el patrón Visitor.

4.17. ParamExtractorVisPatt

Patrón de diseño basado en el patrón Visitor.

4.18. PredInOrVerifierVisPatt

Patrón de diseño basado en el patrón Visitor.

4.19. PredRemoverVisPatt

Patrón de diseño basado en el patrón Visitor.

4.20. PrimeVarsMakerVisPatt

Patrón de diseño basado en el patrón Visitor.

4.21. StringToNumReplacerVisPatt

Patrón de diseño basado en el patrón Visitor.

4.22. TacticManagerSinglPatt

Patrón de diseño basado en el patrón Singleton.

4.23. TCaseStrategyPatt

Patrón de diseño basado en el patrón Strategy.

4.24. TTreeCompositePatt

Patrón de diseño basado en el patrón Composite.

4.25. TTreeStrategyPatt

Patrón de diseño basado en el patrón Strategy.

4.26. TTreeVisPatt

Patrón de diseño basado en el patrón Visitor y que se aplica para aplicar distintas operaciones a los árboles de prueba.

4.27. TypeFMsGenVisPatt

Patrón de diseño basado en el patrón Visitor.

4.28. WordsFinderVisPatt

Patrón de diseño basado en el patrón Visitor.