Especificación de Interfaces de los Módulos de Fastest

Pablo Rodriguez Monetti

Rosario, 2009

Generic Module	**AbstractIterator**(X)
exportsproc	first() next() hasNext(): Bool remove() currentElement(): X
comments	first() inicializa el elemento actual al primer elemento. next() avanza el elemento actual al siguiente elemento. hasNext() comprueba si hay más elementos. remove() elimina el elemento actual. currentElement() devuelve el elemento actual.

Module	**AbstractionFunc**

Module	**AbstractionFuncLoaded inherits from** Event
imports	AbstractionFunc
exportsproc	setAbstractionFunc(**i** AbstractionFunc) getAbstractionFunc(): AbstractionFunc

Module	**Abstractor inherits from** IIComponent
imports	AbstractionFunc, AbstractionFuncLoaded, AbstractTCase, ClientUI, ConcreteOutput,EventAdmin, OpName, OutputAbstracted, TCaseExecuted
exportsproc	manageEvent(**i** Event)
announceevents	makeResultAbstract(): OutputAbstracted
callonevents	tCaseExecuted::manageEvent(tCaseExecuted) abstractionFuncLoaded::manageEvent(abstractionFuncLoaded)
private	makeResultAbstract() abstractTCase: AbstractTCase abstractionFunc: AbstractionFunc opName: OpName
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de AbstractionFuncLoaded ni de TCaseExecuted. La subrutina abstractResult() es llamada desde manageEvent(), luego de que se establezca el valor de alguna/s de las variables instancia del módulo en base a alguno/s de los parámetros del evento que se le pasa como argumento. abstractResult() comprueba que todas estas variables estén seteadas, y de ser así, realiza la abstracción de la salida concreta correspondiente, anunciando a continuación el evento outputAbstracted. La implementación del anuncio del evento se realiza a través de una llamada a la subrutina announceEvent() de la instancia única de EventAdmin, pasando la apropiada instancia de OutputAbstracted como argumento. Si no están seteadas todas estas variables, abstractResult() regresa sin anunciar ningún evento.

Module	**AbstractOrder**
exportsproc	run()

Module	**AbstractOutput inherits from** OutputClass
imports	Todos los imports de AxPara
exportsproc	isAbstractOutput(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
comments	Es un interfaz que hereda de la interfaz OutputClass.

Module	**AbstractOutputImpl inherits from** AbstractOutput
imports	Todos los imports de AxPara
exportsproc	isAbstractOutput(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isAbstractOutput() determina si su argumento es un AbstractOutput. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Generic Module	**AbstractRepository**(X)
imports	AbstractIterator
exportsproc	addElement(**i** X) createIterator(): AbstractIterator(X)

Module	**AddTacticCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller, CZT, OpScheme, OpSchemeImpl, SpecUtils, Tactic
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**AbstractTCase inherits from** TClass
imports	Todos los imports de AxPara
exportsproc	isAbstractTCase(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
comments	Es una interfax que hereda de la interfaz TClass.

Module	**AbstractTCaseImpl inherits from** AbstractTCase
imports	Todos los imports de AxPara
exportsproc	isAbstractTCase(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isAbstractTCase() determina si su argumento es un AbstractTCase. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**AllTCasesGenerated inherits from** Event
imports	OpName

Module	**AllTCasesRequested inherits from** Event
imports	TTreeNode, OpName
exportsproc	setOpName(**i** OpName) getOpName(): OpName setTTree(**i** TClassNode) getTTree(): TClassNode

Module	**AllTTreesGenerated inherits from** Event
imports	OpName

Module	**AndOrPredDistributor**
imports	AndPred, NegPred, Pred, Pred2
exportsproc	visitAndPred(**i** AndPred): Pred visitNegPred(**i** NegPred): Pred visitPred2(**i** Pred2): Pred visitPred(**i** Pred): Pred
comments	Pred, AndPred, NegPred y Pred2 son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Pattern	**AndOrPredDistributorVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is AndOrPredDistributor Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** NegPred ElementoConcreto3 **is** Pred ElementoConcreto4 **is** Pred2
comments	Pred, AndPred, NegPred, Pred2, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**AndPredClausesExtractor**
imports	AndPred, AbstractIterator, ConcreteRepository, OrPred, Pred
exportsproc	visitAndPred(**i** AndPred): Pred visitPred(**i** Pred): Pred
comments	Pred y AndPred son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Pattern	**AndPredClausesExtractorVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is AndPredClausesExtractor Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** Pred
comments	Pred, AndPred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**AndPredClausesRemover**
imports	AndPred, OrPred, Pred, VarNameRepository
exportsproc	visitAndPred(**i** AndPred): Pred visitPred(**i** Pred): Pred
private	varNamesRep: VarNameRepository
comments	Pred y AndPred son módulos que heredan de CZT de acuerdo a la estructura de modulos. Debe pasarse una instancia de VarNameRepository al constructor de instancias de este módulo de tal forma de poder establecer el valor de la variable varNamesRep.

Pattern	**AndPredClausesRemoverVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is AndPredClausesRemover Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** Pred
comments	Pred, AndPred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**AndPredSimplifier**
imports	AbstractIterator, AbstractRepository, AndPred, AndPredClausesExtractor, AndPredClausesRemover, AndPredVisitor, OrPred, Pred, PredInOrVerifier, PredVisitor, SpecUtils
exportsproc	visitAndPred(**i** AndPred): Pred visitPred(**i** Pred): Pred
comments	Pred y AndPred son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Pattern	**AndPredSimplifierVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is AndPredSimplifier Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** Pred
comments	AndPred, Pred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**ApplyCommand inherits from** Command
imports	AxPara, ClientTextUI, Controller, PruneAnalizer, PruneUtils, TClass, TreePruner, TypecheckingUtils
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**AxDef inherits from** AxPara
imports	Todos los imports de AxPara
exportsproc	isAxDef(**i** AxPara): Bool getMyAxPara(): AxPara setMyAxPara(**i** AxPara) Todos los de AxPara
comments	Es una interfaz que hereda de la interfaz AxPara.

Module	**AxDefImpl inherits from** AxDef
imports	Todos los imports de AxPara
exportsproc	isAxDef(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
private	myAxPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isAxDef() determina si su argumento es un AxDef. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**AxParaRepository is** AbstractRepository(AxPara)

Module	**CEPTactic inherits from** Tactic
imports	CEPTacticInfo, OpScheme, TClass
exportsproc	applyTactic(**i** TClass): List(TClass) setOriginalOp(**i** OpScheme) getOriginalOp(): OpScheme setTacticInfo(**i** CEPTacticInfo) getTacticInfo(): CEPTacticInfo
private	originalOp: OpScheme tacticInfo: CEPTacticInfo
comments	La subrutina applyTactic() devuelve la lista de instancias de TClass de la forma [a1,b1,a2,b2,...,an,bn] que se obtienen de aplicar la táctica a la clase de prueba que se pasa como argumento y donde para cada i entre 1 y n, ai es la clase de prueba en forma compacta y bi es la misma clase de prueba pero en forma expandida. Si la táctica no genera nuevas clases de prueba para la clase de prueba dada, la subrutina devuelve la lista vacía.

Module	**CEPTacticInfo inherits from** TacticInfo
imports	OutputClassRepository
exportsproc	setOutputClassRepository(**i** OutputClassRepository) getOutputClassRepository(): OutputClassRepository
private	outputClassRepository: OutputClassRepository
comments	La instancia outputClassRepository debe ser una partición del espacio de salida de la operación para la que quiere generarse un árbol de pruebas. El constructor de instancias de este módulo debe invocar a la subrutina setTacticName() pasando como argumento el nombre de la táctica asociada, que en este caso es Cause Effect Propagation.

Module	**Client**
comprises	ClientBusinessLogic ClientPresentation Util

Module	**ClientBusinessLogic**
comprises	ClientManagement Testing

Module	**ClientGUI inherits from** ClientUI
imports	EventAdmin
exportsproc	run()

Module	**ClientManagement**
comprises	Communication ImplicitInvocation Controller

Module	**ClientPresentation**
comprises	Commands ClientGUI ClientTextUI ClientUI TCaseStrategyParsers

Module	**ClientPrunning**
comprises	PrePruner TClassPruneClient TClassPruneClientRunner TPrunning

Module	**ClientTCaseChecking**
comprises	TCaseCheckerClient TCaseCheckerClientRunner

Module	**ClientTCaseGeneration**
comprises	TCaseGenClient TCaseGenClientRunner TClassExtractor

Module	**ClientTextUI inherits from** ClientUI
imports	BufferedReader, Command, CServersConfigLoader, EventAdmin, PrintWriter
exportsproc	main() processCommand(**i** String, **i** String) readAction() getVersion(): String getBanner(): String
comments	La subrutina processCommand() crea y ejecuta (llamando a su subrutina run()) la instancia de Command correspondiente al comando indicado por el String que se recibe como primer argumento junto con los parámetros indicados por el String que se recibe como segundo argumento. Se supone que tales instancias de String son creadas en función del comando tipeado por el usuario en el intérprete de comandos de la interfaz de usuario en modo texto. La subrutina readAction() imprime el prompt del intérprete de comandos y espera por un nuevo comando del usuario. Las subrutinas getVersion() y getBanner() devuelven, respectivamente, un String con la versión actual de Fastest y un String con el banner que se muestra al inicio de la ejecución del sistema usando la interfaz en modo texto.

Module	**ClientUI**
imports	EventAdmin, Controller
exportsproc	setMyController(**i** Controller) getMyController(): Controller

Module	**Command**
imports	ClientTextUI
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	Commands
comprises	AddTacticCommand
	ApplyCommand
	Command
	EvalCommand
	GenAllTCasesCommand
	GenAllTTCommand
	LoadElimTheoremsCommand
	LoadSpecCommand
	PruneBelowCommand
	PruneFromCommand
	PruneTreeCommand
	ResetCommand
	SelOpCommand
	SetAxDefCommand
	SearchTheoremsCommand
	SetFiniteModelsCommand
	ShowAxDefsCommand
	ShowAxDefValuesCommand
	ShowHelpCommand
	ShowLoadedOpsCommand
	ShowTacticsCommand
	ShowSchCommand
	ShowSelOpsCommand
	ShowSpecCommand
	ShowTTCommand
	ShowVersionCommand
	UnPruneCommand
	UnSelAllOpsCommand
	UnSelOpCommand

Module	Common
comprises	OpName
	RepositoryModules
	ZAbstraction

Module	Communication
comprises	CServersControl
	CServersConfigLoader
	DServerConfig
	ServerConfig
	ServerConfigRepository
	ServerName
	ServiceMediator

Module	**CompleteTCaseStrategy inherits from** TCaseStrategy
imports	AbstractTCase, AxPara, Decl, Expr, EvaluationResp, NameList, NormalTypeAndFM, RefExpr, SchemeEvaluator, Spec, SpecUtils, SetExpr, TClass, TypeFMsGenVisitor, VarDecl, VarDeclList, VarValueMap, ZDeclList, ZExprList, ZFactory
exportsproc	generateAbstractTCase(**i** Spec, **i** TClass): AbstractTCase setFMSize(**i** Int) getFMSize(): Int

Module	**CompleteTCaseStrategyParser inherits from** TCaseStrategyParser
imports	TCaseStrategy
exportsproc	parse(**i** String, **i** TCaseStrategy): Bool

Module	**CompServer**
comprises	ServerTCaseChecking ServerTCaseGeneration ServerManagement

Module	**ConcreteIterator inherits from** AbstractIterator
exportsproc	first() next() hasNext(): Bool remove() currentElement(): X
comments	first() inicializa el elemento actual al primer elemento. next() avanza el elemento actual al siguiente elemento. hasNext() comprueba si hay más elementos. remove() elimina el elemento actual. currentElement() devuelve el elemento actual.

Module	**ContainsTermVerifier**
imports	SpecUtils, Term, TermVisitor
exportsproc	visitTerm(**i** Term): Bool

Pattern	**ContainsTermVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is ContainsTerm Elemento **is** Term ElementoConcreto1 **is** Term
comments	Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**Controller inherits from** IIComponent
imports	AbstractIterator, AbstractRepository, AbstractTCase, AllTCasesGenerated, AllTCasesRequested, AllTTreesGenerated, AxPara, ClientUI, ClientTextUI, ConcreteRepository, ContainsTermVerifier, Event, Expr, Event_, EventAdmin, FastestResetted, IIComponent, NotTClassLeavesFounded, OpName, OpNameRepository, Pred, PruneTClassRequested, PruneTTreeRequested, PruneUtils, PrunningResult, RefExpr, SectionManager, Spec, Scheme, SchemeTTreeFinder, SpecUtils, Tactic, TCaseGenerated, TCaseNodeAdder, TCaseRequested, TClass, TClassNode, TreePruner, TTreeGenerated, TTreeStrategy, ZDeclList
exportsproc	manageEvent(**i** Event) setMyClientUI(**i** ClientUI) setOriginalSpec(**i** Spec) getOriginalSpec(): Spec setUnfoldedSpec(**i** Spec) getUnfoldedSpec(): Spec setOpsToTestRep(**i** OpNameRepository) getOpsToTestRep(): OpNameRepository setLoadedOpRep(**i** OpNameRepository) getLoadedOpRep(): OpNameRepository setOpTTreeStrategyMap(**i** List(record{OpName, TTreeStrategy})) getOpTTreeStrategyMap(): List(record{OpName, TTreeStrategy}) setOpTacticMap(**i** List(record{OpName, TacticRepository})) getOpTacticMap(): List(record{OpName, TacticRepository}) setOpTTreeMap(**i** List(record{OpName, TClassNode})) getOpTTreeMap(): List(record{OpName, TClassNode}) setTypeCheckerManager(**i** SectionManager) getTypeCheckerManager(): SectionManager setMaxEval(**i** Int) getMaxEval(): Int setAxDefsRequired(**i** List(record{VarName, Expr})) getAxDefsRequired(): List(record{VarName, Expr}) setAxDefsValues(**i** List(record{RefExpr, Expr})) getAxDefsValues(): List(record{RefExpr, Expr}) setBasicAxDefs(**i** List(record{String, List(String)})) getBasicAxDefs(): List(record{String, List(String)}) setAxDefsRequiredPreds(**i** List(record{VarName, List(Pred)})) getAxDefsRequiredPreds(): List(record{VarName, List(Pred)}) setAxDefsPredVars(**i** List(record{Pred, List(VarName)})) getAxDefsPredVars(): List(record{Pred, List(VarName)}) setAuxiliarDecls(**i** List(record{VarName, ZDeclList})) getAuxiliarDecls(): List(record{VarName, ZDeclList}) setBlockedAxDefs(**i** List(VarName)) getBlockedAxDefs(): List(VarName)
comments	La subrutina manageEvent() deberá lanzar una excepción si la instancia de Event que recibe como argumento no es una instancia de los eventos de interés para este módulo.

Module	**CServersControl inherits from** ServerConfigRepository
imports	AbstractRepository, AbstractIterator, ServerConfig, ServerConfigRepository
exportsproc	getInstance(): CServersControl getRandomServerConfig(): ServerConfig
private	compServersControl: CServersControl
comments	getInstance() debe ser una subrutina estática, es decir, que no pueda ser invocada para un objeto particular de la clase que el módulo implemente, sino solo invocada a nivel de tal clase. Esta subrutina tiene que devolver el objeto compServersControl, antes realizando su creación de ser necesario. El constructor del módulo no puede ser una subrutina pública. getRandomServerConfig() devuelve una instancia de ServerConfig, aleatoria, del repositorio compServersControl.

Pattern	**CServersControlSinglPatt**
based on	Singleton
because	Debe haber exactamente una instancia de este módulo y debe ser accesible a los clientes desde un punto de acceso conocido.
where	Singleton is CServersControl

Module	**CServersConfigLoader**
imports	CompServersControl, DServerConfig, ServerConfig, ServerName
exportsproc	loadCompServersConfig()
comments	loadCompServersConfig() carga información de los servidores de cómputo en la única instancia del CServersControl, desde el servidor de datos o desde un archivo de configuración (por el momento es de la segunda manera).

Module	**CZT**
comprises	CZTVisitors UniqueZLive

Module	**CZTCloner**
imports	Term
exportsproc	Term visitTerm(**i** Term)
comments	Term es un módulo que hereda de CZT de acuerdo a la estructura de modulos.

Pattern	**CZTClonerVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is CZTCloner Elemento **is** Term ElementoConcreto1 **is** Term
comments	Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**CZTReplacer**
imports	VarNameRepository, Pred
exportsproc	visitAndPred(**i** AndPred): AndPred visitPred(**i** Pred): Pred
private	varNamesRep: VarNameRepository
comments	Pred y AndPred son módulos que heredan de CZT de acuerdo a la estructura de modulos. Debe pasarse una instancia de VarNameRepository al constructor de instancias de este módulo de tal forma de poder establecer el valor de la variable varNamesRep.

Pattern	**CZTReplacerVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is CZTReplacer Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** Pred
comments	Pred, AndPred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**CZTVisitors**
comprises	AndOrPredDistributor AndPredClausesExtractor AndPredClausesRemover AndPredSimplifier CZTCloner CZTReplacer ImpliesPredRemover NegPredDistributor OrPredRemover PredInOrVerifier PredRemover PrimeVarsMaker StringToNumReplacer WordsFinder

Module	**DNFTactic inherits from** Tactic
imports	DNFTacticInfo, OpScheme, TClass
exportsproc	applyTactic(**i** TClass): List(TClass) setOriginalOp(**i** OpScheme) getOriginalOp(): OpScheme setTacticInfo(**i** DNFTacticInfo) getTacticInfo(): DNFTacticInfo
private	originalOp: OpScheme tacticInfo: DNFTacticInfo
comments	La subrutina applyTactic() devuelve la lista de instancias de TClass de la forma [a1,b1,a2,b2,...,an,bn] que se obtienen de aplicar la táctica a la clase de prueba que se pasa como argumento y donde para cada i entre 1 y n, ai es la clase de prueba en forma compacta y bi es la misma clase de prueba pero en forma expandida. Si la táctica no genera nuevas clases de prueba para la clase de prueba dada, la subrutina devuelve la lista vacía.

Module	**DNFTacticInfo inherits from** TacticInfo
comments	El constructor de instancias de este módulo debe invocar a la subrutina setTacticName() pasando como argumento el nombre de la táctica asociada, que en este caso es Disjunctive Normal Form.

Module	**DServerConfig**
imports	File, ServerConfig
exportsproc	getInstance(): DServerConfig readConfigFile() setConfigFile(**i** File) getConfigFile(): File setServerConfig(**i** ServerConfig) getServerConfig(): ServerConfig
private	dataServerConfig: DServerConfig serverConfig: ServerConfig
comments	getInstance() debe ser una subrutina estática, es decir, que no pueda ser invocada para un objeto particular de la clase que el módulo implemente, sino solo invocada a nivel de tal clase. Esta subrutina tiene que devolver el objeto dataServerConfig, antes realizando su creación de ser necesario. El constructor no puede ser una subrutina pública.

Pattern	**DServerConfigSinglPatt**
based on	Singleton
because	Debe haber exactamente una instancia de este módulo y debe ser accesible a los clientes desde un punto de acceso conocido.
where	Singleton is DServerConfig

Module	**DesignPatterns**
comprises	AndOrPredDistributorVisPatt
	AndPredClausesExtractorVisPatt
	AndPredClausesRemoverVisPatt
	AndPredRemoverVisPatt
	AndPredSimplifierVisPatt
	ContainsTermVerifierVisPatt
	CServersControlSinglPatt
	CZTClonerVisPatt
	CZTReplacerVisPatt
	DServerConfigSinglPatt
	EventAdminCommandPatt
	EventAdminSinglPatt
	FiniteModelCreatorVisPatt
	ImpliesPredRemoverVisPatt
	IteratorPatt
	NegPredDistributorVisPatt
	OrPredRemoverVisPatt
	ParamExtractorVisPatt
	PredInOrVerifierVisPatt
	PredRemoverVisPatt
	PrimeVarsMakerVisPatt
	StringToNumReplacerVisPatt
	TacticManagerSinglPatt
	TCaseStrategyPatt
	TTreeCompositePatt
	TTreeStrategyPatt
	TTreeVisPatt
	TypeFMsGenVisPatt
	WordsFinderVisPatt

Module	**EvalCommand inherits from** Command
imports	ClientTextUI, Expr, Pred, SpecUtils, UniqueZLive
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**Evaluation**
comprises	EvaluationResp
	NormalTypeAndFM
	SchemeEvaluator
	VarValueMap

Module	**EvaluationResp**
exportsproc	setLog(**i** String) getLog(): String setResult(**i** Bool) getResult(): Bool

Module	**Event**
imports	EventName
exportsproc	setEventName(**i** EventName) getEventName(): EventName

Module	**EventAdmin**
imports	AbstractRepository, AbstractIterator, File, ClientUI, Event, IIComponent, IIOrder, EventName, Subscriptor
exportsproc	getInstance(**i** File, **i** ClientUI): EventAdmin getInstance(): EventAdmin announceEvent(**i** Event) readFile() setFile(**i** File) getFile(**i** File)
private	file: File myClientUI: ClientUI eventAdmin: EventAdmin eventTable: EventTable
comments	Las subrutinas getInstance() y getInstance(**i** File, **i** ClientUI) y la variable de instancia eventAdmin son estáticas, es decir, tiene sentido hablar de ellas a nivel de módulo pero no a nivel de una instancia del módulo. Todas las demás subrutinas y variables no deben ser estáticas. La subrutina getInstance(**i** File, **i** ClientUI) debe llamarse una sola vez y antes que cualquier otra subrutina del módulo, caso contrario se lanzará una excepción. Esto se debe a la necesidad de construir la instancia eventAdmin en una única ocasión. El constructor del módulo no puede ser una subrutina pública pues se desea que solo pueda ser invocado por getInstance(**i** File, **i** ClientUI). La subrutina readFile() carga desde el archivo indicado por la instancia file la tabla de eventos indicada por la instancia eventTable. En esta tabla se listan las ternas (evento, componente interesado en el evento, subrutina del componente a ser llamada al lanzarse el evento). Además de completar la tabla de eventos, la subrutina readFile() tiene la responsabilidad de crear cada uno de los componentes indicados en la tabla, y luego establecer en cada uno de ellos la referencia a clientUI. La subrutina announceEvent() debe invocar a cada método interesado en el evento que se pasa como argumento, es decir, a cada método indicado indicado por los subscriptores listados en la instancia eventTable. Cada invocación debe hacerse en un thread diferente por cuestiones de performance.

Pattern	**EventAdminCommandPatt**
based on	Command
because	Queremos tener módulos relacionados que sólo difieran en su comportamiento. Las estrategias permiten configurar un módulo con un determinado comportamiento de entre muchos posibles.
where	Orden is AbstractOrder OrdenConcreta is IIOrder Invocador is EventAdmin Receptor is Component Client is EventAdmin

Pattern	**EventAdminSinglPatt**
based on	Singleton
because	Debe haber exactamente una instancia de este módulo y debe ser accesible a los clientes desde un punto de acceso conocido.
where	Singleton is EventAdmin

Module	**Events**
comprises	AbstractionFuncLoaded AllTCasesGenerated AllTCasesRequested AllTTreesGenerated Event NotTClassLeavesFounded OutputAbstracted RefFunctionLoaded SpecLoaded TCaseChecked TCaseExecuted TCaseGenerated TCaseRefined TCaseRequested TCaseStrategySelected TTreeGenerated TTreeRequested

Module	**EventTable is** AbstractRepository(Subscriptor)

Module	**EventName**
comments	En su lugar se usa el módulo String.

Module	**Executor inherits from** IIComponent

Module	**Fastest**
comprises	Client
	Common
	CompServer
	DServer

Module	**FiniteModel**
imports	Expr
exportsproc	initialize()
	hasNext(): Bool
	next(): Expr
	getNormalizedType(): Expr
	getFMSize(): Int
comments	Este módulo no se implementa, solo brinda una interfaz a los módulos que heredan de él.

Module	**FiniteModelCreator**
imports	AxPara, ApplExpr, Freetype, FiniteModel, GivenPara, ProdExpr, PowerExpr, RefExpr, TypeFMsGenMap
exportsproc	getExprMap(): TypeFMsGenMap
	visitAxPara(**i** AxPara): FiniteModel
	visitApplExpr(**i** ApplExpr): FiniteModel
	visitFreetype(**i** Freetype): FiniteModel
	visitGivenPara(**i** GivenPara): FiniteModel
	visitPowerExpr(**i** PowerExpr): FiniteModel
	visitProdExpr(**i** ProdExpr): FiniteModel
	visitRefExpr(**i** RefExpr): FiniteModel
	visitSetExpr(**i** SetExpr): FiniteModel
	visitExpr(**i** Expr): FiniteModel
	getExprMap():
private	size: Int
	typeFMsGenMap: TypeFMsGenMap
comments	size y typeFMsGenMap los toma como argumento el constructor del módulo. typeFMsGenMap contiene las correspondencias entre cada tipo y el generador de modelo finito del tal tipo.

Pattern	**FiniteModelCreatorVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is FiniteModelCreator Elemento **is** Term ElementoConcreto1 **is** AxPara ElementoConcreto2 **is** ApplExpr ElementoConcreto3 **is** Freetype ElementoConcreto4 **is** GivenPara ElementoConcreto5 **is** PowerExpr ElementoConcreto6 **is** ProdExpr ElementoConcreto7 **is** RefExpr ElementoConcreto8 **is** SetExpr
comments	AxPara, ApplExpr,Freetype, GivenPara, PowerExpr, ProdExpr, RefExpr, SetExprTerm y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**FiniteModelGeneration**
comprises	FiniteModel FiniteModelCreator FreetypeFiniteModel FromToFiniteModel GivenFiniteModel IntFiniteModelGeneration NatFiniteModelGeneration PFuncFiniteModel PowerFiniteModel ProdFiniteModel RelFiniteModel SeqFiniteModel SetFiniteModel TClassFiniteModel TFuncFiniteModel TypeFMsGenVisitor

Module	**FromToFiniteModel inherits from** FiniteModel
imports	Expr, NumExpr, RefExpr, ZExprList, ZFactory, ZName, ZNumeral
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr leftValue: Int rightValue: Int index: Int
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. leftValue y rightValue se reciben como argumentos en el constructor del módulo.

Module	**GenAllTCasesCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller, EventAdmin, Tactic, TTreeRequested, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**GenAllTTCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller, EventAdmin, Tactic, TTreeRequested, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**GivenIntFiniteModel inherits from** IntFiniteModel
imports	Expr, NumExpr, RefExpr, ZExprList, ZFactory, ZName, ZNumeral
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int

Module	**GivenIntFiniteModelParser**
imports	IntFiniteModel
exportsproc	parse(**i** List(String), **i** IntFiniteModel): Int

Module	**GivenNatFiniteModel inherits from** NatFiniteModel
imports	Expr, NumExpr, RefExpr, ZExprList, ZFactory, ZName, ZNumeral
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int

Module	**GivenNatFiniteModelParser**
imports	NatFiniteModel
exportsproc	parse(**i** List(String), **i** NatFiniteModel): Int

Module	**HorizDef inherits from** AxPara
imports	Todos los imports de AxPara
exportsproc	isHorizDef(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
private	axPara: AxPara
comments	Es una interfaz que hereda de la interfaz AxPara.

Module	**HorizDefImpl inherits from** HorizDef
imports	Todos los imports de AxPara
exportsproc	isHorizDef(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isHorizDef() determina si su argumento es un HorizDef. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**IIComponent**
imports	ClientUI, Event
exportsproc	manageEvent(**i** Event) setMyClientUI(**i** ClientUI) getMyClientUI(): ClientUI

Module	**IIOrder inherits from** AbstractOrder
imports	IIComponent, MethodName, Event
exportsproc	run()
	getImplicitParam(): IIComponent
	getMethodName(): MethodName
	getExplicitParam(): Event
	setImplicitParam(**i** IIComponent)
	setMethodName(**i** MethodName)
	setExplicitParam(**i** Event)
private	implicitParam: IIComponent
	methodName: MethodName
	explicitParam: Event

Module	**ImplicitInvocation**
comprises	Events
	EventAdmin
	EventName
	EventTable
	IIComponent
	IIOrder
	MethodName
	Subscriptor

Module	**ImpliesPredRemover**
imports	Pred, IffPred, ImpliesPred, Pred2, NegPred
exportsproc	visitIffPred(**i** IffPred): Pred
	visitImpliesPred(**i** ImpliesPred): Pred
	visitPred2(**i** Pred2): Pred
	visitNegPred(**i** NegPred): Pred
	visitPred(**i** Pred): Pred
comments	Pred, IffPred, ImpliesPred, NegPred y Pred2 son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Pattern	**ImpliesPredRemoverVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is ImpliesPredRemover Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** IffPred ElementoConcreto3 **is** ImpliesPred ElementoConcreto4 **is** NegPred ElementoConcreto5 **is** Pred ElementoConcreto6 **is** Pred2
comments	AndPred, IffPred, ImpliesPred, NegPred, Pred, Pred2, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**IntFiniteModel inherits from** FiniteModel
comments	Interfaz que hereda de la interfaz FiniteModel.

Module	**IntFiniteModelGeneration** GivenIntFiniteModel IntFiniteModel SeedsIntFiniteModel ZeroIntFiniteModel

Module	**IntFiniteModelParser**
imports	IntFiniteModel
exportsproc	parse(**i** List(String), **i** IntFiniteModel): Int

Module	**IterativeTCaseStrategy inherits from** TCaseStrategy
imports	AbstractTCase, AxPara, Expr, EvaluationResp, FiniteModelCreator, IntFiniteModel, NatFiniteModel, NumericConstsExtractor,RefExpr, SchemeEvaluator, Spec, TClass, TClassFiniteModel, ZeroIntFiniteModel, ZeroNatFiniteModel
exportsproc	generateAbstractTCase(**i** Spec, **i** TClass): AbstractTCase setFMSize(**i** Int) getFMSize(): Int setIntFiniteModel(**i** IntFiniteModel) getIntFiniteModel(): IntFiniteModel setNatFiniteModel(**i** NatFiniteModel) getNatFiniteModel(): NatFiniteModel

Module	**IterativeTTreeStrategy**
imports	SpecUtils, OpScheme, TacticRepository, TTreeNode, TClassNode
exportsproc	generateTTree(**i** OpScheme, **i** TacticRepository): TClassNode
comments	Este módulo no se implementa, solo sirve como interfaz para los módulos que heredan de él.

Pattern	**IteratorPatt**
based on	Iterator
because	Para acceder al contenido de los objetos agregados sin exponer su representación interna.
where	Iterador is AbstractIterator IteradorConcreto **is** ConcreteIterator Agregado **is** AbstractRepository AgregadoConcreto **is** ConcreteRepository

Module	**FreetypeFiniteModel inherits from** FiniteModel
imports	Expr, Name, ZBranchList, ZFactory
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr index: Int zBranchList: ZBranchList
comments	La instancia de normalizedType se una sola vez al construirse una instancia del módulo. El valor de size se recibe como argumento en el constructor del modulo.

Module	**FTTactic inherits from** Tactic
imports	AndPredSimplifier, AxPara, Branch, ContainsTermVerifier, CZTCloner, BranchList, Decl, DeclList, Expr, FreePara, Freetype, FreetypeList, MemPred, Name, NameList, OpScheme, Para, ParaList, ParseUtils, Pred, RefExpr, Sect, SetExpr, Spec, SpecUtils, StringSource, TClass, TClassImpl, Term, TextUI, UniqueZLive, VarDecl, ZBranchListImpl, ZDeclList, ZExprList, ZFactory, ZFactoryImpl, ZFreetypeListImpl, ZName, ZNameListImpl, ZLive, ZParaList, ZSect
exportsproc	applyTactic(**i** TClass): List(TClass) setOriginalOp(**i** OpScheme) getOriginalOp(): OpScheme parseArgs(String): Bool setTacticInfo(**i** SPTacticInfo) getTacticInfo(): SPTacticInfo
private	originalOp: OpScheme tacticInfo: SPTacticInfo
comments	La subrutina applyTactic() devuelve la lista de instancias de TClass de la forma [a1,b1,a2,b2,...,an,bn] que se obtienen de aplicar la táctica a la clase de prueba que se pasa como argumento y donde para cada i entre 1 y n, ai es la clase de prueba en forma compacta y bi es la misma clase de prueba pero en forma expandida. Si la táctica no genera nuevas clases de prueba para la clase de prueba dada, la subrutina devuelve la lista vacía. parseArgs() parsea el parámetro que se indica al agregar la táctica a la lista de tácticas para una operación. Este parámetro es el nombre de la variable que sea del tipo libre que quiere tenerse en cuenta para aplicar la táctica. Si éstos parámetros son correctos se devuelve true; caso contrario, se devuelve false. Para que lo sea, el nombre de variable no puede ser de una variable primada y la variable debe estar declarada en el esquema de operación como una de un tipo libre definido en la especificación cargada en Fastest.

Module	**FTTacticInfo inherits from** TacticInfo
imports	Freetype
exportsproc	setVar(**i** String) getVar(): String setFreetype(**i** Freetype) getFreetype(): Freetype
private	stdPartition: StdPartition realParamList: List(Term)

Module	**GivenFiniteModel inherits from** FiniteModel
imports	Expr, ZFactory, ZName
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr index: Int typeName: String
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. Los valores de size y de typeName se reciben como argumento en el constructor del modulo.

Module	**IterativeTCaseStrategyParser inherits from** TCaseStrategyparser
imports	TCaseStrategy
exportsproc	parse(**i** String, **i** TCaseStrategy): Bool

Module	**LoadElimTheoremsCommand inherits from** Command
imports	AbstractIterator, ClientTextUI, RWRule, RWRulesControl, RWRulesLoader, Theorem, TheoremsControl, TheoremsLoader
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**LoadSpecCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller, CZT, EventAdmin, OpSchemeImpl, UnfoldUtils, SpecLoaded, SpecUtils, Tactic, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ManualTCaseStrategy inherits from** TCaseStrategy
exportsproc	generateAbstractTCase(**i** Spec, **i** TClass): AbstractTCase setFMSize(**i** Int) getFMSize(): Int

Module	**MethodName**
comments	En su lugar se usa el módulo String.

Module	**NatFiniteModel inherits from** FiniteModel
comments	Interfaz que hereda de la interfaz FiniteModel.

Module	**NatFiniteModelParser**
imports	NatFiniteModel
exportsproc	parse(**i** List(String), **i** NatFiniteModel): Int

Module	**NegPredDistributor**
imports	Pred, Pred2, NegPred
exportsproc	visitNegPred(**i** NegPred): Pred visitPred2(**i** Pred2): Pred visitPred(**i** Pred): Pred
comments	Pred, NegPred y Pred2 son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Pattern	**NegPredDistributorVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is NegPredDistributor Elemento **is** Term ElementoConcreto1 **is** NegPred ElementoConcreto2 **is** Pred ElementoConcreto3 **is** Pred2
comments	NegPred, Pred, Pred2, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**NormalTypeAndFM**
imports	Expr, SetExpr
exportsproc	setNormalType(**i** Expr) getNormalType(): Expr setFM(**i** SetExpr) getFM(): SetExpr

Module	**OpName**
comments	Este módulo no se implementa y solo existe para que, al utilizarlo, el diseño sea más sugestivo. En la implementación del sistema, se usa una instancia de String cada vez que el diseño indique usar una instancia de él.

Module	**OpNameRepository is** AbstractRepository(OpName)

Module	**OpScheme inherits from** Scheme
imports	Todos los imports de AxPara
exportsproc	isOpScheme(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
comments	Es una interfaz que hereda de la interfaz Scheme.

Module	**OpSchemeImpl inherits from** Scheme
imports	Todos los imports de AxPara
exportsproc	isOpScheme(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isOpScheme() determina si su argumento es un OpScheme. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**OrPredRemover**
imports	AndPred, OrPred, Pred, VarNameRepository
exportsproc	visitAndPred(**i** AndPred): Pred visitOrPred(**i** OrPred): Pred visitPred(**i** Pred): Pred
private	varNamesRep: VarNameRepository
comments	Pred y AndPred son módulos que heredan de CZT de acuerdo a la estructura de modulos. Debe pasarse una instancia de VarNameRepository al constructor de instancias de este módulo de tal forma de poder establecer el valor de la variable varNamesRep.

Pattern	**OrPredRemoverVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is OrPredRemover Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** OrPred ElementoConcreto3 **is** Pred
comments	AndPred, OrPred, Pred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**OutputAbstracted inherits from** Event
imports	AbstractTCase, AbstractOutput, OpName
exportsproc	setOpName(**i** OpName) getOpName(): OpName setAbstractTCase(**i** AbstractTCase) getAbstractTCase(): AbstractTCase setAbstractOutput(**i** AbstractOutput) getAbstractOutput(): AbstractOutput

Module	**OutputClass inherits from** Scheme
imports	Todos los imports de AxPara
exportsproc	isOutputClass(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
comments	Interfaz que hereda de la interfaz Scheme.

Module	**OutputClassImpl inherits from** OutputClass
imports	Todos los imports de AxPara
exportsproc	isOutputClass(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isOutputClass() determina si su argumento es un OutputClass. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**OutputClassRepository is** AbstractRepository(OutputClass)

Module	**ParamExtractor**
imports	ApplExpr, ApplExprVisitor, Expr, ExprPred, ExprPredVisitor, ExprVisitor, MemPred, MemPredVisitor, Pred, PredVisitor, RefExpr, SetExpr, Term, TupleExpr, ZExprList
exportsproc	visitPred(**i** Pred): List(Term) visitExpr(**i** Expr): List(Term) visitMemPred(**i** MemPred): List(Term) visitApplExpr(**i** ApplExpr): List(Term) visitExprPred(**i** ExprPred): List(Term)

Pattern	**ParamExtractorVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is ParamExtractor Elemento **is** Term ElementoConcreto1 **is** ApplExpr ElementoConcreto2 **is** Expr ElementoConcreto3 **is** ExprPred ElementoConcreto4 **is** MemPred ElementoConcreto5 **is** Pred
comments	ApplExpr, Expr, ExprPred, MemPred, Pred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**PFuncFiniteModel inherits from** FiniteModel
imports	Expr, TupleExpr, ZExprList, ZFactory, ZName
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr leftFiniteModel: FiniteModel rightFiniteModel: FiniteModel
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. leftFiniteModel y rightFiniteModel se reciben como argumentos en el constructor del módulo.

Module	**PowerFiniteModel inherits from** FiniteModel
imports	Expr, ZExprList, ZFactory, ZName
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr fMGen: FiniteModel Indices internos
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. fMGen se recibe como argumento en el constructor del módulo.

Module	**PredInOrVerifier**
imports	Pred, OrPred, SpecUtils
exportsproc	visitOrPred(**i** OrPred): Bool visitPred(**i** Pred): Bool
comments	Pred y OrPred son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Pattern	**PredInOrVerifierVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is PredInOrVerifier Elemento **is** Term ElementoConcreto1 **is** OrPred ElementoConcreto2 **is** Pred
comments	OrPred, Pred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**PredRemover**
imports	VarNameRepository, Pred
exportsproc	visitAndPred(**i** AndPred): AndPred visitPred(**i** Pred): Pred
private	varNamesRep: VarNameRepository
comments	Pred y AndPred son módulos que heredan de CZT de acuerdo a la estructura de modulos. Debe pasarse una instancia de VarNameRepository al constructor de instancias de este módulo de tal forma de poder establecer el valor de la variable varNamesRep.

Pattern	**PredRemoverVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is PredRemover Elemento **is** Term ElementoConcreto1 **is** AndPred ElementoConcreto2 **is** Pred
comments	AndPred, Pred, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**PrimeVarsMaker**
imports	VarNameRepository, Term
exportsproc	setVarNameRep(**i** VarNameRepository) visitTerm(**i** Term): Term visitZName(**i** ZName): ZName
private	varNamesRep: VarNameRepository
comments	Term y ZName son módulos que heredan de CZT de acuerdo a la estructura de modulos. Debe pasarse una instancia de VarNameRepository al constructor de instancias de este módulo de tal forma de poder establecer el valor de la variable varNamesRep.

Pattern	**PrimeVarsMakerVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is PrimeVarsMaker Elemento **is** Term ElementoConcreto1 **is** Term ElementoConcreto2 **is** ZName
comments	Term, Visitor y ZName son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**ProdFiniteModel inherits from** FiniteModel
imports	Expr, TupleExpr, ZExprList, ZFactory
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr fMGenList: List(FiniteModel) Indices internos
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. fMGenList se recibe como argumento en el constructor del módulo.

Module	**PruneBelowCommand inherits from** Command
imports	ClientTextUI, Controller, TClassNode, TTreeBelowPruner
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**PruneFromCommand inherits from** Command
imports	ClientTextUI, Controller, TClassNode, TTreeFromPruner
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**PruneTreeCommand inherits from** Command
imports	AbstractRepository, ClientTextUI, Controller, EventAdmin, PruneTTreeRequested, PruneUtils, TClass
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**RefinementFunc**

Module	**RefinementFuncLoaded inherits from** Event
imports	RefinementFunc
exportsproc	setRefinementFunc(**i** RefinementFunc) getRefinementFunc(): RefinementFunc

Module	**Refinator inherits from** IIComponent
imports	AbstractTCase, ClientUI, ConcreteTCase, EventAdmin, OpName, RefinementFunc, RefinementFuncLoaded, , TCaseGenerated, TCaseRefined
exportsproc	manageEvent(**i** Event)
announceevents	refineTCase()::tCaseRefined
callonevents	refinamentFuncLoaded::manageEvent(refinamentFuncLoaded) tCaseGenerated::manageEvent(tCaseGenerated)
private	refineTCase() abstractTCase: AbstractTCase abstractionFunc: AbstractionFunc opName: OpName
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de RefinementFuncLoaded ni de TCaseGenerated. La subrutina refineTCase() es llamada desde manageEvent(), luego de que se establezca el valor de alguna/s de las variables instancia del módulo en base a alguno/s de los parámetros del evento que se le pasa como argumento. refineTCase() comprueba si todas estas variables estén seteadas, y de ser así, realiza el refinamiento del caso de prueba abstracto correspondiente, anunciando a continuación el evento tCaseRefined. La implementación del anuncio del evento se realiza a través de una llamada a la subrutina announceEvent() de la instancia única de EventAdmin, pasando la apropiada instancia de TCaseRefined como argumento. Si no están seteadas todas estas variables, refineTCase() regresa sin anunciar ningún evento.

Module	**RelFiniteModel inherits from** FiniteModel
imports	Expr
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr leftFiniteModel: FiniteModel rightFiniteModel: FiniteModel Indices internos
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. leftFiniteModel y rightFiniteModel se reciben como argumentos en el constructor del módulo.

Module	**RepositoryModules**
comprises	AbstractIterator AbstractRepository ConcreteRepository

Module	**ResetCommand inherits from** Command
imports	ClientTextUI, Controller, Tactic, TClassNode, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**Scheme inherits from** AxPara
imports	Todos los imports de AxPara
exportsproc	isScheme(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isScheme() determina si su argumento es un Scheme. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**Scheme inherits from** AxPara
imports	Todos los imports de AxPara
exportsproc	isScheme(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
comments	Interfaz que hereda de AxPara

Module	**SchemeEvaluator**
imports	EvaluationResp, TClass, VarValueMap, , ZLive
exportsproc	evalSchemeSat(**i** TClass, **i** VarValueMap): EvaluationResp
comments	La subrutina evalSchemeSat() requiere de los servicios del subsistema ZLive de CZT para evaluar el predicado que resulta de reemplazar cada variable por la expresión que tiene asignada en la instancia de VarValueMap.

Module	**SchemeImpl inherits from** Scheme
imports	Todos los imports de AxPara
exportsproc	isScheme(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isScheme() determina si su argumento es un Scheme. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**SchemeTTreeFinder inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**SchName**
comments	En su lugar se usa el módulo String.

Module	**SchemeUnfolder**
imports	AbstractIterator, AbstractRepository, AndExpr, AxPara, Box, ConstDecl, CZTCloner, Decl, DeclList, DecorExpr, Expr, InclDecl, OrExpr, Para, ParaList, Pred, PrimeVarsMaker, RefExpr, SchExpr, Sect, Spec, Term, Visitor, ZFactory, ZParaList, ZSchText, ZSect
exportsproc	unfoldSpec(**i** Spec): Spec unfoldAxPara(**i** AxPara, **i** ZParaList) unfoldExpr(**i** Expr, **i** ZParaList): Expr unfoldExpr(**i** SchExpr, **i** ZParaList): SchExpr unfoldAndExpr(**i** AndExpr, **i** ZParaList): SchExpr unfoldOrExpr(**i** OrExpr, **i** ZParaList): SchExpr unfoldRefExpr(**i** RefExpr, **i** ZParaList): SchExpr

Module	**SearchTheoremsCommand inherits from** Command
imports	AbstractIterator, ClientTextUI, RWRule, RWRulesControl, RWRulesLoader, Theorem, TheoremsControl, TheoremsLoader
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**SeedsIntFiniteModel inherits from** IntFiniteModel
imports	Expr, NumExpr, RefExpr, ZExprList, ZFactory, ZName, ZNumeral
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int

Module	**SeedsIntFiniteModelParser**
imports	IntFiniteModel
exportsproc	parse(**i** List(String), **i** IntFiniteModel): Int

Module	**SeedsNatFiniteModel inherits from** NatFiniteModel
imports	Expr, NumExpr, RefExpr, ZExprList, ZFactory, ZName, ZNumeral
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int

Module	**SeedsNatFiniteModelParser**
imports	NatFiniteModel
exportsproc	parse(**i** List(String), **i** NatFiniteModel): Int

Module	**SelOpCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller, Tactic, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**SeqFiniteModel inherits from** FiniteModel
imports	Expr, NumExpr, ProdExpr, RefExpr, SetExpr. TupleExpr, ZExprList, ZFactory, ZName
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr fMGen: FiniteModel Indices internos
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. se reciben como argumentos en el constructor del módulo. El valor de size se establece en el constructor del modulo.

Module	**ServerConfig**
imports	InetAddress, ServerName
exportsproc	setServerName(**i** ServerName) getServerName(): ServerName setInetAddress(**i** InetAddress) getInetAddress(): InetAddress setPortNumber(**i** Int) getPortNumber(): Int

Module	**ServerConfigRepository is** AbstractRepository(ServerConfig)

Module	**ServerManagement**
comprises	ServerThread ServiceManager

Module	**ServiceMediator**
imports	InetAddress, Spec, TClass, TCaseStrategy
exportsproc	generateAbstractTCase(**i** Spec, **i** TClass, **i** TCaseStrategy): AbstractTCase checkCorresp(**i** Spec, **i** AbstractTCase, **i** AbstractOutput): Bool
privatecomments	Al ejecutar las subrutinas generateAbstractTCase() o checkCorresp() se establece una conexión con un servidor de cómputo a través de una interfaz de socket. El servidor de cómputo con el que se realizará la conexión se establece en el constructor de este módulo, a través de una instancia de InetAddress y el entero que corresponde al puerto que el servidor escuchará.

Module	**ServerName**
comments	En su lugar se usa el módulo String.

Module	**ServerTCaseChecking**
comprises	TCaseChecker

Module	**ServerTCaseGeneration**
comprises	Evaluation FiniteModelGeneration TCaseStrategies TCaseGen

Module	**ServerThread**
imports	AbstractTCase, AxPara, JaxbXmlReader, JaxbXmlWriter, SpecUtils, TCaseGen, TCaseStrategy, TClass, Term, XmlReader, XmlWriter
exportsproc	run() generateAbstractTCase() checkCorrespondence()

Module	**ServiceManager**
imports	ServerThread
exportsproc	main()
comments	La subrutina main() se encarga de atender los pedidos de conexión de los clientes y de brindar los servicios que ellos puedan solicitar. La comunicación se lleva a cabo a través de sockets. Cada servicio se procesa en un thread diferente, que ejecuta la subrutina run() de una instancia del módulo ServerThread.

Module	**SetAxDefCommand inherits from** Command
imports	ClientTextUI, Controller, CZTCloner, CZTReplacer, Expr, ExprPred, Para, ParaList, ParseUtils, Pred, RefExpr, SectionManager, Spec, StringToNumReplacer, Term, TypeCheckUtils, UniqueZLive, ZLive, ZName, ZExprList, ZFactory, ZFactoryImpl, ZParaList
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**SetFiniteModel inherits from** FiniteModel
imports	Expr, ZExprList
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int
private	size: Int normalizedType: Expr index: Int valueList: List(String)
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. El valor de valueList se recibe como argumento en el constructor del modulo.

Module	**SetFiniteModelsCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller, EventAdmin, TCaseStrategy, TCaseStrategyParser, TCaseStrategySelected, TClassLeavesFinder, TClass, TClassNode
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowAxDefsCommand inherits from** Command
imports	AxPara, Box, ClientTextUI, Controller, Para, ParaList, Sect, Spec, SpecUtils, ZParaList, ZSect
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowAxDefValuesCommand inherits from** Command
imports	ClientTextUI, Controller, Expr, RefExpr, Spec, SpecUtils
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowHelpCommand inherits from** Command
imports	ClientTextUI, Controller, Expr, RefExpr, Spec, SpecUtils
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowHelpCommand inherits from** Command
imports	ClientTextUI
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowLoadedOpsCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowSchCommand inherits from** Command
imports	ClientTextUI, Controller EventAdmin, Scheme, SchemeTTreeFinder, SpecUtils, Tactic, TCaseNodeTextUIPrinter, TClassNode, TClassNodeTextUIPrinter, TTreeRequested, TTreeStrategy, TTreeVisitor
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowSelOpsCommand inherits from** Command
imports	AbstractIterator, AbstractRepository, ClientTextUI, Controller, Tactic, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowSpecCommand inherits from** Command
imports	ClientTextUI, Controller, SpecUtils
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowTacticsCommand inherits from** Command
imports	ClientTextUI, Tactic
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowTTCommand inherits from** Command
imports	ClientTextUI, Controller,TClassNode, TTreeTextUIPrinter
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**ShowVersionCommand inherits from** Command
imports	ClientTextUI
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**SMTactic inherits from** Tactic
imports	SMTacticInfo, OpScheme, TClass
exportsproc	applyTactic(**i** TClass): List(TClass) setOriginalOp(**i** OpScheme) getOriginalOp(): OpScheme setTacticInfo(**i** SMTacticInfo) getTacticInfo(): SMTacticInfo
private	originalOp: OpScheme tacticInfo: SMTacticInfo
comments	applyTactic() lanzará una excepción si la instancia originalOp no tiene como operación mutante al OpScheme miembro de SMTacticInfo. La subrutina applyTactic() devuelve la lista de instancias de TClass de la forma [a1,b1,a2,b2,...,an,bn] que se obtienen de aplicar la táctica a la clase de prueba que se pasa como argumento y donde para cada i entre 1 y n, ai es la clase de prueba en forma compacta y bi es la misma clase de prueba pero en forma expandida. Si la táctica no genera nuevas clases de prueba para la clase de prueba dada, la subrutina devuelve la lista vacía.

Module	**SMTacticInfo inherits from** TacticInfo
imports	OpScheme
exportsproc	setMutantOp(**i** OpScheme) getMutantOp(): OpScheme
private	mutantOp: OpScheme
comments	La instancia mutantOp debe ser el esquema de operación mutante de la operación original para la que quiere generarse un árbol de pruebas. El constructor de instancias de este módulo debe invocar a la subrutina setTacticName() pasando como argumento el nombre de la táctica asociada, que en este caso es Specification Mutation.

Module	**SpecLoaded inherits from** Event
imports	Spec
exportsproc	setSpec(**i** Spec) getSpec(): Spec

Module	**SpecUtils**
imports	AxPara, DeclList, Pred, VarNameRepository, SchExpr, ZDeclList, VarDecl, ZName
exportsproc	getAxParaName(**i** AxPara): String getAxParaListOfDecl(**i** AxPara): DeclList getAxParaSchExpr(**i** AxPara): SchExpr getAxParaPred(**i** AxPara): Pred getVarNames(AxPara): VarNameRepository isThereTransition(**i** VarNameRepository): Bool isThereIO(**i** VarNameRepository): Bool andExprs(**i** SchExpr, **i** SchExpr): SchExpr orExprs(**i** SchExpr, **i** SchExpr): SchExpr makeDeclListUnion(**i** SchExpr, **i** SchExpr): SchExpr axParaSearch(**i** String, **i** ZParaList): AxPara insertZDeclList(**i** ZDeclList, **i** ZDeclList) insertVarDecl(**i** ZDeclList, **i** VarDecl, **i** Int): Bool containsName(**i** ZDeclList, **i** ZName): Bool andPreds(**i** Pred, **i** Pred): Pred orPreds(**i** Pred, **i** Pred): Pred createEqualPred(**i** String): Pred getCNF(**i** Pred): Pred getDNF(**i** Pred): Pred divideOrAxPara(**i** AxPara): AxParaRepository axParaToLatex(**i** AxPara): String areEqualTerms(**i** Term, **i** Term): Bool
private	lookUp(VarNameRepository, **i** VarName): Bool
comments	Los modulos DeclList, Pred, SchExpr, ZDeclList, VarDecl, ZName heredan de CZT de acuerdo a la estructura de modulos.

Module	**SPTactic inherits from** Tactic
imports	ContainsTermVerifier, OpScheme, ParamExtractor, SPTacticInfo, TClass
exportsproc	applyTactic(**i** TClass): List(TClass) setOriginalOp(**i** OpScheme) getOriginalOp(): OpScheme parseArgs(String): Bool setTacticInfo(**i** SPTacticInfo) getTacticInfo(): SPTacticInfo
private	originalOp: OpScheme tacticInfo: SPTacticInfo
comments	La subrutina applyTactic() devuelve la lista de instancias de TClass de la forma [a1,b1,a2,b2,...,an,bn] que se obtienen de aplicar la táctica a la clase de prueba que se pasa como argumento y donde para cada i entre 1 y n, ai es la clase de prueba en forma compacta y bi es la misma clase de prueba pero en forma expandida. Si la táctica no genera nuevas clases de prueba para la clase de prueba dada, la subrutina devuelve la lista vacía. parseArgs() parsea los parámetros que se indican al agregar la táctica a la lista de tácticas para una operación, como por ejemplo, el operador sobre el que se va a aplicar alguna partición estándar, y el término que lo contiene, dentro del predicado de la operación seleccionada. Si éstos parámetros son correctos se devuelve true; caso contrario, se devuelve false. Para que lo sean, debe haber una partición estándar cargada para el operador seleccionado y el término indicado debe estar contenido en la operación original.

Module	**SPTacticInfo inherits from** TacticInfo
imports	CZT, StdPartition
exportsproc	setStdPartition(**i** StdPartition) getStdPartition(): StdPartition setRealParamList(**i** List(Term)) getRealParamList(): List(Term)
private	stdPartition: StdPartition realParamList: List(Term)

Module	**StdPartition**
imports	Pred
exportsproc	getPredList(): List(Pred) setPredList(**i** List(Pred)) getFormalParamList(): List(String) setFormalParamList(**i** List(String)) setDefinition(**i** String) getDefinition(): String setOperator(**i** String) getOperator(): String

Module	**StdPartitionLoader**
imports	CZT, StdPartition, StdPartitionsControl
exportsproc	loadStdPartitions()

Module	**StdPartitionsControl inherits from** StdPartitionsRepository
imports	AbstractRepository, AbstractIterator, StdPartition, StdPartitionRepository
exportsproc	getInstance(): StdPartitionsControl
private	stdPartitionsControl: StdPartitionsControl
comments	getInstance() debe ser una subrutina estática, es decir, que no pueda ser invocada para un objeto particular de la clase que el módulo implemente, sino solo invocada a nivel de tal clase. Esta subrutina tiene que devolver el objeto stdPartitionsControl, antes realizando su creación de ser necesario. El constructor del módulo no puede ser una subrutina pública.

Module	**StdPartitionsRepository is** AbstractRepository(StdPartition)

Module	**StringToNumReplacer**
imports	VarNameRepository, Pred
exportsproc	visitAndPred(**i** AndPred): AndPred visitPred(**i** Pred): Pred
private	varNamesRep: VarNameRepository
comments	Pred y AndPred son módulos que heredan de CZT de acuerdo a la estructura de modulos. Debe pasarse una instancia de VarNameRepository al constructor de instancias de este módulo de tal forma de poder establecer el valor de la variable varNamesRep.

Module	**Subscriptor**
imports	EventName, AbstractOrder, IIOrder
exportsproc	getEventName(): EventName getIIOrder(): IIOrder
private	eventName: EventName iIOrder: IIOrder
comments	El constructor de instancias de este módulo debe tomar como argumentos instancias de EventName y de IIOrder para luego guardarlas como los valores de las variables de instancia eventName y iIOrder.

Module	**SuggestiveTCaseStrategy inherits from** TCaseStrategy
exportsproc	generateAbstractTCase(**i** Spec, **i** TClass): AbstractTCase setFMSize(**i** Int) getFMSize(): Int

Module	**Tactic**
imports	OpScheme, TClass
exportsproc	applyTactic(**i** TClass): List(TClass) setOriginalOp(**i** OpScheme) getOriginalOp(): OpScheme setTacticInfo(**i** TacticInfo) getTacticInfo(): TacticInfo
comments	Este módulo no se implementa, solo brinda una interfaz a los módulos que heredan de él. La subrutina applyTactic() devuelve la lista de instancias de TClass de la forma [a1,b1,a2,b2,...,an,bn] que se obtienen de aplicar la táctica a la clase de prueba que se pasa como argumento y donde para cada i entre 1 y n, ai es la clase de prueba en forma compacta y bi es la misma clase de prueba pero en forma expandida. Si la táctica no genera nuevas clases de prueba para la clase de prueba dada, la subrutina devuelve la lista vacía.

Module	**Tactics**
comprises	CEPTactic CEPTacticInfo DNFTactic DNFTacticInfo FTTactic FTTacticInfo SMTactic SMTacticInfo SPTactic SPTacticInfo Tactic TacticInfo TacticName TacticRepository

Module	**TacticInfo**
imports	TacticName
exportsproc	setTacticName(**i** TacticName) getTacticName(): TacticName
comments	La instancia de TacticName permite reconocer a que tipo de táctica nos referimos, por ejemplo, DNF, mutación de especificaciones, etc.

Module	**TacticManager**
imports	DServerConfig
exportsproc	getInstance(): TacticManager addTacticDef(**i** File, **i** TacticName) getTacticDef(**i** TacticName): File
private	tacticManager: TacticManager
comments	La subrutina getTacticDef() realiza la transferencia del archivo indicado por su primer argumento y devuelve información del archivo en una instancia de File. getInstance() debe ser una subrutina estática, es decir, que no pueda ser invocada para un objeto particular de la clase que el módulo implemente, sino solo invocada a nivel de tal clase. Esta subrutina tiene que devolver el objeto tacticManager, antes realizando su creación de ser necesario. El constructor no puede ser una subrutina pública.

Pattern	**TacticManagerSinglPatt**
based on	Singleton
because	Debe haber exactamente una instancia de este módulo y debe ser accesible a los clientes desde un punto de acceso conocido.
where	Singleton is TacticManager

Module	**TacticName**
comments	En su lugar se usa el módulo String.

Module	**TacticRepository is** AbstractRepository(Tactic)

Module	**TCaseAbstraction**
comprises	AbstractionFunc Abstractor

Module	**TCaseChecked inherits from** Event
imports	AbstracTCase, OpName
exportsproc	setOpName(**i** OpName) getOpName(): OpName setAbstractTCase(**i** AbstractTCase) getAbstractTCase(): AbstractTCase setResult(**i** Bool) getResult(): Bool

Module	**TCaseChecker**
imports	AbstractOutput, AbstractTCase, OpScheme
exportsproc	testCorrespondence(**i** Spec, **i** OpName, **i** AbstractTCase, **i** AbstractOutput): Bool
comments	La subrutina testCorrespondence() determina si se corresponden el caso de prueba abstracto, el esquema de operación, y la salida abstracta representados por las instancias de AbstractTCase, OpScheme y AbstractOutput que recibe como argumentos, respectivamente.

Module	**TCaseCheckerClient inherits from** IIComponent
imports	AbstractTCase, AbstractOutput, ClientUI, Event, EventAdmin, IIComponent, OpName, OutputAbstracted, Spec,SpecLoaded, TCaseChecked, TCaseCheckerClientRunner
exportsproc	manageEvent(**i** Event)
callonevents	specLoaded::manageEvent(specLoaded) outputAbstracted::manageEvent(outputAbstracted)
announceevents	checkCorresp()::tCaseChecked
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de SpecLoaded ni de OutputAbstracted. Cuando se llama a manageEvent() y el argumento que se le pasa es una instancia de SpecLoaded, la instancia de Spec que el evento specLoaded tiene como parámetro debe guardarse en la variable spec. Cuando se llama a manageEvent() y el argumento que se le pasa es una instancia de outputAbstracted, se comprobará si la variable instancia spec está seteada. De ser así, se creará una instancia de TCaseCheckerClientRunner, pasando como argumentos de su constructor el valor de spec y los de las instancias de opName, specTCase y abstractOutput que son parámetros del evento outputAbstracted y a continuación se llamará a la subrutina run() del objeto creado. Esta subrutina se deberá correr en un nuevo thread y así manageEvent() podrá regresar. Si al llamar a manageEvent(), pasando como argumento una instancia de OutputAbstracted, la variable spec no está seteada, la subrutina regresará inmediatamente.

Module	**TCaseCheckerClientRunner**
imports	AbstractTCase, AbstractOutput, CServersControl, EventAdmin, InetAddress, OpName, ServerConfig, ServiceMediator, Spec, TCaseChecked
exportsproc	run()
announceevents	run()::TCaseChecked
private	spec: Spec opName: OpName abstractTCase: AbstractTCase abstractOutput: AbstractOutput
comments	Las variables instancia listadas se establecen al momento de crear una instancia de este módulo. Al llamar a la subrutina run() se deberá solicitar a uno de los servidores de cómputo la comprobación de la correspondencia entre el caso de prueba abstracto indicado por abstractTCase, la operación indicada por opName y spec, y la salida abstracta indicada por abstractOutput. Para esto se llamará a la subrutina checkCorresp() de un nueva instancia de ServiceMediator pasando los argumentos necesarios. El valor booleano devuelto por checkCorresp() será establecido como parámetro de un evento de tipo TCaseChecked que TCaseCheckerClientRunner deberá anunciar.

Module	**TCaseGenClient inherits from** IIComponent
imports	ClientUI, Event, GivenIntFiniteModel, GivenNatFiniteModel, IIComponent, IterativeTCaseStrategy, OpName, Spec, SpecLoaded, TCaseGenClientRunner, TCaseRequested, TCaseStrategy, TCaseStrategySelected, TClass
exportsproc	manageEvent(**i** Event)
callonevents	specLoaded::manageEvent(specLoaded) tCaseRequested::manageEvent(tCaseRequested) tCaseStrategySelected::manageEvent(tCaseStrategySelected)
private	spec: Spec tCaseStrategy: TCaseStrategy
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de SpecLoaded, de TCaseRequested, ni de TCaseStrategySelected. Cuando se llama a manageEvent() y el argumento que se le pasa es una instancia de SpecLoaded, la instancia de Spec que el evento specLoaded tiene como parámetro debe guardarse en la variable spec. Cuando se llama a manageEvent() y el argumento que se le pasa es una instancia de TCaseRequested, se comprobará si la variable instancia spec está seteada. De ser así, se creará una instancia de TCaseGenClientRunner, pasando como argumentos de su constructor el valor de spec y los de las instancias de OpName, TClass y TCaseStrategy (las dos primeras son parámetros del evento TCaseStrategy mientras que la tercera puede ser la que se crea por defecto salvo que se haya lanzado un evento TCaseStrategySelected, en cuyo caso se toma el parámetro que viene con él) y a continuación se llamará a la subrutina run() del objeto creado. Esta subrutina se deberá correr en un nuevo thread y así manageEvent() podrá regresar. Si al llamar a manageEvent() pasando como argumento una instancia de TCaseRequested y la variable spec no está seteada, la subrutina regresará inmediatamente. Cuando se llama a manageEvent() y el argumento que se le pasa es una instancia de TCaseStrategy, se guarda en el estado del módulo la asociación entre el nombre de la clase de prueba y la estrategia de generación de casos de prueba que indican los parámetros del evento.

Module	**TCaseGenClientRunner**
imports	AbstractTCase, CServersControl, EventAdmin, OpName, ServerConfig, ServiceMediator, Spec, TCaseStrategy, TClass, TTreeGenerated
exportsproc	run()
announceevents	run()::tTreeGenerated
private	spec: Spec opName: OpName tClass: TClass tCaseStrategy: TCaseStrategy
comments	Las variables instancia listadas se establecen al momento de crear una instancia de este módulo. Al llamar a la subrutina run() se deberá solicitar a uno de los servidores de cómputo la generación del caso de prueba asociado a la especificación spec y a la clase de prueba tClass, haciendo uso de la estrategia de generación de modelos finitos tCaseStrategy. Para esto se llamará a la subrutina generateAbstractTCase() de una nueva instancia de ServiceMediator pasando los argumentos necesarios. La instancia de AbstractTCase devuelta por generateAbstractTCase() será establecida como parámetro de un evento de tipo TCaseGenerated que TCaseGenClientHelper deberá anunciar.

Module	**TCaseExecuted inherits from** Event
imports	AbstractTCase, ConcreteOutput, OpName
exportsproc	setOpName(**i** OpName) getOpName(): OpName setAbstractTCase(**i** AbstractTCase) getAbstractTCase(): AbstractTCase setConcreteOutput(**i** ConcreteOutput) getConcreteOutput(): ConcreteOutput

Module	**TCaseGen**
imports	AbstractTCase, Spec, TClass, TCaseStrategy
exportsproc	generateAbstractTCase(**i** Spec, **i** TClass, **i** TCaseStrategy): AbstractTCase
comments	La subrutina generateAbstractTCase() genera (o intenta generar) un caso de prueba abstracto para la clase de prueba que recibe como argumento. El caso de prueba abstracto es devuelto en una instancia de AbstractTCase. Para llevar a cabo su tarea, esta subrutina llama a la subrutina generateAbstractTCase() de la instancia de TCaseStrategy que recibió como argumento.

Module	**TCaseGenerated inherits from** Event
imports	AbstractTCase, OpName, TClass
exportsproc	setOpName(**i** OpName) getOpName(): OpName setAbstractTCase(**i** AbstractTCase) getAbstractTCase(): AbstractTCase setTClass(**i** TClass) getTClass(): TClass

Module	**TCaseExecution**
comprises	ConcreteOutput ConcreteTCase Executor

Module	**TCaseNode inherits from** TTreeNode
imports	AbstractTCase, TacticInfo, TTreeNodeRepository, TTreeVisitor
exportsproc	setChildren(**i** TTreeNodeRepository) getChildren(): TTreeNodeRepository) setDadNode(**i** TClassNode) getDadNode(): TClassNode acceptVisitor(**i** TTreeVisitor) setTacticInfo(**i** TacticInfo) getTacticInfo(): TacticInfo setValue(**i** Scheme) getValue(): AbstractTCase setUnfoldedValue(**i** Scheme) getUnfoldedValue(): AbstractTCase
comments	La subrutina getChildren() es reimplementada para devolver un TTreeNodeRepository nulo. setChildren() es reimplementada para lanzar una excepción en caso de ser llamada. Las subrutinas setTacticInfo() y getTacticInfo() se reimplementan para, respectivamente, lanzar una excepción en caso de ser llamada y devolver siempre null (del lenguaje de programación a utilizar).

Module	**TCaseNodeAdder inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**TCaseNodeTextUIPrinter inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**TCaseRequested inherits from** Event
imports	OpName, TCaseStrategy, TClass
exportsproc	setOpName(**i** OpName) getOpName(): OpName setTClass(**i** TClass) getTClass(): TClass
comments	El evento lleva como parámetros el nombre de la operación y la clase de prueba de la que quiere obtenerse un caso de prueba, indicada por una instancia de TClass.

Module	**TCaseRefined inherits from** Event
imports	AbstractTCase, ImplTCase, OpName
exportsproc	setOpName(**i** OpName) getOpName(): OpName setAbstractTCase(**i** AbstractTCase) getAbstractTCase(): AbstractTCase setConcreteTCase(**i** ConcreteTCase) getConcreteTCase(): ConcreteTCase

Module	**TCaseRefined inherits from** Event
imports	AbstractTCase, ImplTCase, OpName
exportsproc	setOpName(**i** OpName) getOpName(): OpName setAbstractTCase(**i** AbstractTCase) getAbstractTCase(): AbstractTCase setConcreteTCase(**i** ConcreteTCase) getConcreteTCase(): ConcreteTCase

Module	**TCaseRefinement**
comprises	RefinementFunc Refinator

Module	**TCaseStrategies**
comprises	CompleteTCaseStrategy IterativeTCaseStrategy ManualTCaseStrategy SuggestiveTCaseStrategy TCaseStrategy

Module	**TCaseStrategy**
imports	AbstractTCase, Spec, TClass
exportsproc	generateAbstractTCase(**i** Spec, **i** TClass): AbstractTCase setFMSize(**i** Int) getFMSize(): Int

Pattern	**TCaseStrategyPatt**
based on	Strategy
because	Queremos tener módulos relacionados que sólo difieran en su comportamiento. Las estrategias permiten configurar un módulo con un determinado comportamiento de entre muchos posibles.
where	Estrategia is TCaseStrategy EstrategiaConcreta1 is CompleteTCaseStrategy EstrategiaConcreta2 is IterativeTCaseStrategy EstrategiaConcreta3 is ManualTCaseStrategy EstrategiaConcreta3 is SuggestiveTCaseStrategy Contexto is TCaseGen

Module	**TCaseStrategyParser**
imports	TCaseStrategy
exportsproc	parse(**i** String, **i** TCaseStrategy): Bool

Module	**TCaseStrategyParsers**
comprises	CompleteTCaseStrategyParser GivenIntFiniteModelParser GivenNatFiniteModelParser IntFiniteModelParser IterativeTCaseStrategyParser NatFiniteModelParser SeedsIntFiniteModelParser SeedsNatFiniteModelParser TCaseStrategyParser ZeroIntFiniteModelParser ZeroNatFiniteModelParser

Module	**TCaseStrategySelected inherits from** Event
imports	TCaseStrategy
exportsproc	setTCaseStrategy(**i** TCaseStrategy) getTCaseStrategy(): TCaseStrategy setTClassName(**i** String) getTClassName(): String

Module	**TClass inherits from** Scheme
imports	Todos los imports de AxPara
exportsproc	isTClass(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
comments	Interfaz que hereda de Scheme.

Module	**TClassExtractor inherits from** IIComponent
imports	AbstractIterator, AbstractRepository, Event, EventAdmin, OpName, TCaseRequested,TClass, TClassLeavesFinder, TClassNode, TTreeGenerated
exportsproc	manageEvent(**i** Event)
announceevents	manageEvent()::tCaseRequested
callonevents	tTreeGenerated::manageEvent(tTreeGenerated)
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de AllTCasesRequested.

Module	**TClassFiniteModel**
imports	AxPara, Decl, DeclList, Expr, FiniteModel, FiniteModelCreator, NameList, RefExpr, SetExpr, VarDecl, ZDeclList, ZExprList, ZFactory
exportsproc	initialize() hasNext(): Bool next(): VarValueMap
private	size: Int index: Int varList: ZExprList fMGenList: List(FiniteModel)
comments	El constructor del módulo recibe una instancia de FiniteModelCreator.

Module	**TClassImpl inherits from** Scheme
imports	Todos los imports de AxPara
exportsproc	isTClass(**i** AxPara): Bool setMyAxPara(**i** AxPara) getMyAxPara(): AxPara setSchName(**i** SchName) getSchName(): SchName Todos los de AxPara
private	axPara: AxPara
comments	Cada subrutina heredada de AxPara se reimplementa para llamar a la subrutina con el mismo nombre del objeto myAxPara. isTClass() determina si su argumento es un TClass. La subrutina setMyAxPara() guarda en la variable myAxPara el valor de su argumento.

Module	**TClassLeavesFinder inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**TClassNode inherits from** TTreeNode
imports	Scheme, TacticInfo, TClass, TTreeNode, TTreeNodeRepository
exportsproc	setChildren(**i** TTreeNodeRepository) getChildren(): TTreeNodeRepository) setDadNode(**i** TClassNode) getDadNode(): TClassNode acceptVisitor(**i** TTreeVisitor) setTacticInfo(**i** TacticInfo) getTacticInfo(): TacticInfo setValue(**i** Scheme) getValue(): TClass setUnfoldedValue(**i** Scheme) getUnfoldedValue(): TClass setPruned(**i** Bool) isPruned(): Bool

Module	**TClassNodeTextUIPrinter inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**TClassPruneClient inherits from** IIComponent
exportsproc	manageEvent(**i** Event)
callonevents	specLoaded::manageEvent(specLoaded) pruneTClassRequested::manageEvent(pruneTClassRequested)
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de SpecLoaded ni de PruneTClassRequested. Cuando se llama a manageEvent() y el argumento que se le pasa es una instancia de SpecLoaded, la instancia de Spec que el evento specLoaded tiene como parámetro debe guardarse en la variable spec. Cuando se llama a manageEvent() y el argumento que se le pasa es una instancia de PruneTClassRequested, se comprobará si la variable instancia spec está seteada. De ser así, se creará una instancia de TClassPruneClientRunner, a continuación se llamará a la subrutina run() del objeto creado. Esta subrutina se deberá correr en un nuevo thread y así manageEvent() podrá regresar.

Module	**TClassPruneClientRunner**
exportsproc	run()
announceevents	run()::prunningResult

Module	**TClassRepository is** AbstractRepository(TClass)

Module	**Testing**
comprises	ClientPrunning
	ClientTCaseChecking
	ClientTCaseGeneration
	TCaseAbstraction
	TCaseExecution
	TCaseRefinement
	TTreeModules

Module	**TFuncFiniteModel inherits from** FiniteModel
imports	Expr, SetExpr, TupleExpr, ZExprList, ZFactory, ZName
exportsproc	initialize()
	hasNext(): Bool
	next(): Expr
	getNormalizedType(): Expr
	getFMSize(): Int
private	size: Int
	normalizedType: Expr
	leftFiniteModel: FiniteModel
	rightFiniteModel: FiniteModel
	Indices internos
comments	La instancia de normalizedType se crea una sola vez al construirse una instancia del módulo. leftFiniteModel y rightFiniteModel se reciben como argumentos en el constructor del módulo.

Module	**TPrunning inherits from** IIComponent
imports	
exportsproc	manageEvent(**i** Event)
callonevents	pruneTTreeRequested::manageEvent(pruneTTreeRequested)
	prunningResult::manageEvent(prunningResult)
announceevents	manageEvent()::pruneTClassRequested
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de PruneTTreeRequested ni de PrunningResult.

Pattern	**TTreeCompositePatt**
based on	Composite
because	Se quiere representar una jerarquía de objetos parte-todo y que los clientes sean capaces de obviar las diferencias entre composiciones de objetos y objetos individuales. Así, los clientes tratarán a todos los objetos de la estructura compuesta de manera uniforme.
where	Componente is TTreeNode
	Hoja1 **is** TCaseNode
	Compuesto **is** TClassNode

Module	**TTreeGen inherits from** IIComponent
imports	ClientUI, Event, EventAdmin, OpName, Spec, SpecLoaded, SpecUtils, TacticRepository, TTreeGenerated,TTreeRequested, TTreeStrategy
exportsproc	manageEvent(**i** Event)
announceevents	generateTTree()::tTreeGenerated
callonevents	specLoaded::manageEvent(specLoaded) tTreeRequested::manageEvent(tTreeRequested)
private	generateTTree() spec: Spec tacticList: List(Tactic) tTreeStrategy: TTreeStrategy opName: OpName
comments	La subrutina manageEvent() debe lanzar una excepción si su argumento no es una instancia de SpecLoaded ni de TTreeRequested. La subrutina generateTTree() es llamada desde manageEvent(), luego de que se establezca el valor de alguna/s de las variables instancia del módulo en base a alguno/s de los parámetros del evento que se le pasa como argumento. generateTTree() comprueba que todas estas variables estén seteadas, y de ser así, realiza la generación del árbol de pruebas correspondiente, anunciando a continuación el evento tTreeGenerated. La implementación del anuncio del evento se realiza a través de una llamada a la subrutina announceEvent() de la instancia única de EventAdmin, pasando la apropiada instancia de TTreeGenerated como argumento. Si no están seteadas todas estas variables, generateTTree() regresa sin anunciar ningún evento.

Module	**TTreeGenerated inherits from** Event
imports	TTreeNode, OpName
exportsproc	setOpName(**i** OpName) getOpName(): OpName setTTree(**i** TClassNode) getTTree(): TClassNode

Module	**TTreeModules**
comprises	TacticName Tactics TTreeStrategies TTreeVisitors TacticManager TCaseNode TClassNode TTreeGen TTreeNode

Module	**TTreeNode**
imports	Scheme, TacticInfo, TTreeNodeRepository, TTreeVisitor
exportsproc	setChildren(**i** TTreeNodeRepository) getChildren(): TTreeNodeRepository setDadNode(**i** TTreeNode) getDadNode(): TTreeNode acceptVisitor(**i** TTreeVisitor) setTacticInfo(**i** TacticInfo) getTacticInfo(): TacticInfo setValue(**i** Scheme) getValue(): Scheme setUnfoldedValue(**i** Scheme) getUnfoldedValue(): Scheme
comments	Este módulo no se implementa, solo brinda una interfaz a los módulos que heredan de él.

Module	**TTreeNodeRepository is** AbstractRepository(TTreeNode)

Module	**TTreeBelowPruner inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**TTreeFromPruner inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**TTreeRequested inherits from** Event
imports	OpName, Tactic, TTreeStrategy
exportsproc	setOpName(**i** OpName) getOpName(): OpName setTacticList(**i** List(Tactic)) getTacticList(): List(Tactic) setTTreeStrategy(**i** TTreeStrategy) getTTreeStrategy(): TTreeStrategy

Module	**TTreeStrategies**
comprises	IterativeTTreeStrategy TTreeStrategy

Module	**TTreeStrategy**
imports	SpecUtils, OpScheme, TacticRepository, TTreeNode, TClassNode
exportsproc	generateTTree(**i** OpScheme, **i** List(Tactic)): TClassNode
comments	Este módulo no se implementa, solo sirve como interfaz para los módulos que heredan de él.

Pattern	**TTreeStrategyPatt**
based on	Strategy
because	Queremos tener módulos relacionados que sólo difieran en su comportamiento. Las estrategias permiten configurar un módulo con un determinado comportamiento de entre muchos posibles.
where	Estrategia is TTreeStrategy EstrategiaConcreta1 is IterativeTTreeStrategy Contexto is TTreeGen

Module	**TTreeTextUIPrinter inherits from** TTreeVisitor
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)

Module	**TTreeVisitor**
imports	TCaseNode, TClassNode
exportsproc	visitTClassNode(**i** TClassNode) visitTCaseNode(**i** TCaseNode)
comments	Este módulo no se implementa, solo brinda una interfaz a los módulos que heredan de él.

Pattern	**TTreeVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is TTreeVisitor VisitanteConcreto1 is SchemeTTreeFinder VisitanteConcreto2 is TCaseNodeAdder VisitanteConcreto3 is TCaseNodeTextUIPrinter VisitanteConcreto4 is TClassLeavesFinder VisitanteConcreto5 is TClassNodeTextUIPrinter VisitanteConcreto6 is TTreeTextUIPrinter VisitanteConcreto7 is TTreeBelowPruner VisitanteConcreto8 is TTreeFromPruner Elemento **is** TTreeNode ElementoConcreto1 **is** TClassNode ElementoConcreto2 **is** TCaseNode

Module	**TTreeVisitors**
comprises	SchemeTTreeFinder TCaseNodeAdder TCaseNodeTextUIPrinter TClassLeavesFinder TClassNodeTextUIPrinter TTreeBelowPruner TTreeFromPruner TTreeTextUIPrinter TTreeVisitor

Module	**TypeFMsGenMap**
imports	Expr, FiniteModel
exportsproc	put(**i** Expr, **i** FiniteModel) get(**i** Expr): FiniteModel containsKey(**i** Expr): Bool containsValue(**i** FiniteModel): Bool remove(**i** Expr)
comments	El módulo se implementa usando la coleccion Map instanciada en los tipos Expr y FiniteModel de la API de Java.

Module	**TypeFMsMap**
imports	Expr, NormalTypeAndFM
exportsproc	put(**i** Expr, **i** NormalTypeAndFM) get(**i** Expr): NormalTypeAndFM containsKey(**i** Expr): Bool containsValue(**i** NormalTypeAndFM): Bool remove(**i** Expr)
comments	El módulo se implementa usando la coleccion Map instanciada en los tipos Expr y NormalTypeAndFM de la API de Java.

Module	**TypeFMsGenVisitor**
imports	ApplExpr, AxPara, Expr, Freetype, GivenPara, NormalTypeAndFM, ProdExpr, PowerExpr, RefExpr, SetExpr, Term, TypeFMsMap
exportsproc	visitTerm(**i** Term): NormalTypeAndFM visitAxPara(**i** AxPara): NormalTypeAndFM visitGivenPara(**i** GivenPara): NormalTypeAndFM visitFreetype(**i** Freetype): NormalTypeAndFM visitRefExpr(**i** RefExpr): NormalTypeAndFM visitApplExpr(**i** ApplExpr): NormalTypeAndFM visitPowerExpr(**i** PowerExpr): NormalTypeAndFM visitProdExpr(**i** ProdExpr): NormalTypeAndFM visitSetExpr(**i** SetExpr): NormalTypeAndFM visitExpr(**i** Expr): NormalTypeAndFM visitSetExpr(**i** SetExpr): NormalTypeAndFM getExprMap(): TypeFMsMap
private	fMSize: Int exprMap: TypeFMsMap

Pattern	**TypeFMsGenVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is TypeFMsGenVisitor Elemento **is** Term ElementoConcreto1 **is** AxPara ElementoConcreto2 **is** ApplExpr ElementoConcreto3 **is** Freetype ElementoConcreto4 **is** GivenPara ElementoConcreto5 **is** PowerExpr ElementoConcreto6 **is** ProdExpr ElementoConcreto7 **is** RefExpr ElementoConcreto8 **is** SetExpr
comments	AxPara, ApplExpr,Freetype, GivenPara, PowerExpr, ProdExpr, RefExpr, SetExprTerm, Term y Visitor son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**UnSelAllOpsCommand inherits from** Command
imports	AbstractRepository, ClientTextUI, Controller, Tactic, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**UnSelOpCommand inherits from** Command
imports	AbstractRepository, ClientTextUI, Controller, Tactic, TTreeStrategy
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**Util**
comprises	AbstractOrder

Module	**UtilSymbols**
exportsproc	primeSymbol(): String naturalSymbol():String relationSymbol():String partialFunctionSymbol():String totalFunctionSymbol():String emptySetSymbol():String leftAngleSymbol():String rightAngleSymbol():String
comments	Deben ser todas subrutinas estáticas, es decir, que no puedan ser invocadas para una instancia particular del módulo, sino solo invocada a nivel de módulo.

Module	**UniqueZLive**
imports	ZLive
exportsproc	getInstance(): ZLive
private	zLive: ZLive
comments	getInstance() debe ser una subrutina estática, es decir, que no pueda ser invocada para un objeto particular de la clase que el módulo implemente, sino solo invocada a nivel de tal clase. Esta subrutina tiene que devolver el objeto zLive, antes realizando su creación de ser necesario. El constructor del módulo no puede ser una subrutina pública.

Module	**UnPruneCommand inherits from** Command
imports	ClientTextUI, Controller, TClassNode, TTreeFromPruner
exportsproc	run(**i** ClientTextUI, **i** String)
comments	La subrutina run() ejecuta la orden correspondiente asumiendo que se está trabajando con la interfaz de usuario en modo texto.

Module	**VarName**
comments	En su lugar se usa el módulo String.

Module	**VarNameRepository is** AbstractRepository(VarName)

Module	**VarValueMap**
imports	Expr, RefExpr
exportsproc	put(**i** RefExpr, **i** Expr) get(**i** RefExpr): Expr containsKey(**i** RefExpr): Bool containsValue(**i** Expr): Bool remove(**i** Expr)
comments	El módulo se implementa usando la coleccion Map instanciada en los tipos RefExpr y Expr de la API de Java.

Module	**VISGen**
imports	AbstractRepository, AbstractIterator, AxPara, CZTCloner, Decl, DeclList, NameList, Pred, PredRemover, VarDecl, ZDeclList
exportsproc	generateVIS(**i** OpScheme): TClass

Module	**WordsFinder**
imports	VarNameRepository, Term
exportsproc	visitTerm(**i** Term): Term visitZName(**i** ZName): ZName
private	varNamesRep: VarNameRepository
comments	Term y ZName son módulos que heredan de CZT de acuerdo a la estructura de modulos. Debe pasarse una instancia de VarNameRepository al constructor de instancias de este módulo de tal forma de poder establecer el valor de la variable varNamesRep.

Pattern	**WordsFinderVisPatt**
based on	Visitor
because	Las clases que definen la estructura de objetos no cambian frecuentemente pero es posible necesitar que se definan nuevas operaciones sobre la estructura. Este patrón nos permite mantener juntas operaciones relacionadas definiéndolas en una clase.
where	Visitante is Visitor VisitanteConcreto is WordsFinder Elemento **is** Term ElementoConcreto1 **is** Term ElementoConcreto2 **is** ZName
comments	Term, Visitor y ZName son módulos que heredan de CZT de acuerdo a la estructura de modulos.

Module	**ZAbstraction**
comprises	CZT
	AbstractOutput
	AbstractOutputImpl
	AbstractTCase
	AxDef
	AxDefImpl
	AxParaRepository
	HorizDef
	HorizDefImpl
	OpNameRepository
	OpScheme
	OpSchemeImpl
	OutputClass
	OutputClassImpl
	OutputClassRepository
	Scheme
	SchemeImpl
	SchemeUnfolder
	SchName
	SpecUtils
	TClass
	TClassImpl
	TClassRepository
	UtilSymbols
	VarName
	VarNameRepository
	VISGen

Module	**ZeroIntFiniteModel inherits from** IntFiniteModel
imports	Expr, NumExpr, RefExpr, ZExprList, ZFactory, ZName, ZNumeral
exportsproc	initialize()
	hasNext(): Bool
	next(): Expr
	getNormalizedType(): Expr
	getFMSize(): Int

Module	**ZeroIntFiniteModelParser**
imports	NatFiniteModel
exportsproc	parse(**i** List(String), **i** NatFiniteModel): Int

Module	**ZeroNatFiniteModel inherits from** NatFiniteModel
imports	Expr, NumExpr, RefExpr, ZExprList, ZFactory, ZName, ZNumeral
exportsproc	initialize() hasNext(): Bool next(): Expr getNormalizedType(): Expr getFMSize(): Int

Module	**ZeroNatFiniteModelParser**
imports	NatFiniteModel
exportsproc	parse(**i** List(String), **i** NatFiniteModel): Int