Specification of the VDM-SL/VDM++ Toolbox Specification Manager

Author: Henrik Voss,
Jeppe Nyløkke Jørgensen
The Institute of Applied Computer Science

Date: March 8, 2013

Doc. Id.: IFAD-VDM-28

Type:	Report
Status:	Under Development
Availability:	Confidential
Copyright:	©IFAD

Document History

V1.0	First version.
V1.1	The version used in release 3.0 of the VDM-SL Toolbox.
V1.2	Requirements added.
V1.3	Updated to include specification manager for VDM++ as well.
V1.4	Pretty printed version.

Note:

Contents

1 Introduction

This document describes the design and specification of the VDM-SL and VDM^{++} Toolbox Specification Manager, written in VDM^{++}. TO BE WRITTEN.

2 System Requirements

2.1 General Description

The motivation for the Specification Manager is basically the introduction of the GUI (Graphical User Interface) to the IFAD VDM-SL Toolbox. With the GUI, two different interfaces are available for the toolbox, namely the GUI and the traditional ascii interface. The main purposes of the specification manager can be stated as:

- Keep track of the status for each module in the current project,
- Decide legal actions for each module depending on the status of the module,
- Take care of error handling,
- Form the interface to the internals of the toolbox,
- Save the project in its current status, making it possible to quit the session and later restart at the same point.

The Specification Manager should be viewed as an internal part of the toolbox that is not directly related to the end user. Instead, it is related to the developer of the toolbox. The functionality of the toolbox wanted by the developer should be accessed via the Specification Manager.

It is not possible to foresee all possible requirements from the developer to the internals of the toolbox. Instead, the Specification Manager should offer a suitable interface to the internals of the toolbox, eventually including a basic set of functions (parsing, type checking, code generation, pretty printing). It should also be possible to define different interfaces depending on the developers need (VDM-SL Toolbox, VDM^{++} Toolbox, the Specification Animator). This could be stated as calling conventions or calling structure to the internals of the toolbox.

2.2 Definitions

Core definitions common to VDM-SL and VDM^{++}:

Project: Consists of one or more files containing specification.

File: For VDM-SL a file can contain one or more modules in case of a structured specification. In case of a flat specification, a file contains one or more definitions blocks. For VDM^{++}

a file contain one or more classes. Syntax checking and pretty printing are performed at files.

Status: Status for files concerns syntax checking and pretty printing and status for modules/classes concerns type checking and code generation. Status can be `None`(Action has not been performed), `OK`(Action has been performed successfully) or `Error`(Action has been performed with error).

Definitions specific to VDM-SL:

Module: Contains the VDM-SL specification. In case of a flat specification, this specification is turned into a single module called "DefaultMod". Type checking and code generation is performed on modules.

Session: Describes the type of specification as either *flat* or *structured*. Flat and structured specifications cannot be parsed in the same session.

Definitions specific to VDM^{++}:

Class Contains the VDM^{++} specification. Type checking and code generation is performed on classes.

2.3 Static Requirements

The information that the Specification Manager must contain and operate on is described as the *state* of the Specification Manager. This section describes the requirements to this state.

The core state should contain:

1. The project name,
2. The names of files and modules in current project,
3. Status for each file and module/class,
4. AST for each module/class,
5. Tag information for the code generator,
6. Error and warning messages,
7. Options,
8. Dependency information (for VDM-SL this is imported modules and for VDM^{++} this is super and subclasses and uses and used by classes).

For the VDM-SL Toolbox furthermore:

1. Session type

For the VDM^{++} Toolbox furthermore:

1. State for the type checker.

2.4 Functional Requirements Stated as Use cases

Actors Two actors to the use cases were identified:

API: The Application Program Interface, i.e. the user of Specification Manager from the interface.

Toolbox: The internal toolbox functionalities.

Use Case no. 1: Handling commands to the Toolbox

Introduction

This use case handles all possible commands that can be given to the toolbox. This includes actions on files and modules/classes, i.e. parsing and pretty printing of files and type checking, code generation and "processing"[1] of modules/classes.

Type Concrete

Relations Extends

Initialisation

Activated by API

Actors

API

Preconditions

Specification Manager must be instantiated.

Description

All issued commands are passed to the Toolbox, i.e. both known and unknown commands. Furthermore, the following commands should be handled:

- Syntax check and pretty printing of a sequence of files.
- Type checking and code generation of a sequence of modules/classes.
- "Processing" a sequence of files.

Commands not covered by the above (e.g. commands dedicated to special purposes) must be described in an `extends` use case.

It does not affect the funcional behaviour of the Specification Manager whether the issued command is correct or not, or whether the command results in errors or not. Under all circumstances is the command passed to the Toolbox, and the result is always simply returned to the caller.

[1] Syntax and type checking of a module/class and modules/classes it depends on

Exeptions

None.

Postconditions

None.

Use Case no. 2: Extracting information from the Specification Manager

Introduction

This information concerns allowed actions on files and modules/classes, which modules/classes a file contains, extracting ASTs etc.

Type Concrete

Relations None

Initialisation

Activated by API or Toolbox

Actors

API, Toolbox

Preconditions

None

Description

An actor can extract:

1. Allowed actions on files and modules/classes *(Exception: File/module/class for action is not defined)*,
2. Compare session type (VDM-SL Toolbox only),
3. Status for a file or module*(Exception: File/module/class for status not defined)*,
4. Get file name from file id and vice versa *(Exception: File id/name not defined)*,defined files and modules, correspondence between files and modules/classes *(Exception: Name not defined)*,
5. ASTs,
6. Current project name*(Exception: Project name has not been set)*,
7. Next, previous, first and last error/warning *(Exception: The requested error/warning is not defined)* for the following commands: Syntax check, type check, code generation and pretty printing.
8. the type checker state for the VDM^{++} Toolbox.

Exeptions

File/module/class for action is not defined: Give message to log.

File/module/class for status is not defined: Return nil,

File id/name not defined: Return 0/nil,

Name not defined: Return an empty set,

Project name has not been set: Return nil,

The requested error/warning is not defined: Ignore.

Postconditions

None

Use Case no. 3: Updating the state of the Specification Manager.

Introduction

When the Toolbox receives a command, this command possibly will update the state of the Specification Manager. This use case describes these commands.

Type Concrete

Relations Extends

Initialization

Activated by Toolbox.

Actors

Toolbox

Preconditions

The API has issued the initial command and the Specification Manager has passed the command to the Toolbox.

Description

The state for the Specification Manager must be updated with regards to:

1. Adding/removing of files,
2. Status for files and modules/classes,
3. ASTs and tag information for code generator,
4. Errors and warnings,
5. Options.
6. The type checker state (only for the VDM^{++} Toolbox).

Exeptions

Postconditions

Updated state for Specification Manager.

Use Case no. 4: Project handling (New, Open, Save, Save As)

Introduction

This use case handles changing of project.

Type Concrete

Relations None

Initialization

Activated by API

Actors

API

Preconditions

None.

Description

The actor should be able to explicit set the name of current project, create a new project, save current project and open a named project. Furthermore should the actor be able to detect if there is unsaved changes in the current project. *(Exception: Project cannot be saved or loaded).*

Exeptions

Project cannot be saved or loaded: Give message to log

Postconditions

None.

Use Case no. 5: Configuring the current project

Introduction

This use case handles adding and removing of files to the current project

Type Concrete

Relations None

Initialization

Activated by API

Actors

API

Preconditions

None.

Description

Files should be added *(Exception: File exists)* or removed *(Exception: File does not exist)* from the current project.

Exeptions

File exists: Nothing should be done on the state.

File does not exist: Nothing should be done on the state.

Postconditions

None.

Use Case no. 6: Commands for the Specification Animator to the VDM-SL Toolbox

Introduction

This use case extends use case no. 1

Type Concrete

Relations Extends use case no. 1

Initialization

Actors

API

Preconditions

Description

Commands needed by the Specification Animator that should be passed via the Specification Manager to the VDM-SL Toolbox. These commands include:

1. VDM-SL expressions parsed to abstract syntax trees,
2. Semantic value converted to string,
3. Parsing and evaluation of a VDM-SL value to a semantic value,
4. Extract defined functions and operations into a string,
5. Extract defined extended explicit operations and functions into a map of ASTs,
6. Comparing types.

Exeptions

None.

Postconditions

None.

Use Case no. 7: Dependecy information for the VDM^{++} Toolbox

Introduction

This use case deals with the dependency information (super and subclasses, uses and used by).

Type Concrete

Relations Extends use case no. 3

Initialization

Activated by the VDM^{++} Toolbox.

Actors

Toolbox

Preconditions

Description

Toolbox can insert dependency information for each class in the Specification Manager.

Toolbox can extract either dependency or inheritance information for a named class *(Exeption: Class does not exist).*

Exeptions

Class does not exist: Return empty information string.

Postconditions

Possibly updated state for dependency informatin.

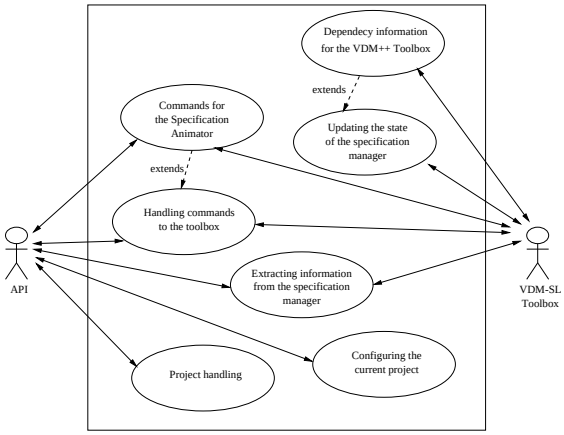

Figure 1: Use case diagram for Specification Manager

2.5 Document Requirements

Developers Manual to Specification Manager This manual must describe how the Specification Manager is to be used by the developer. This includes the creation, how to get access to the toolbox internals, how to add extra functionalites.

3 Class Structure

This section describes the overall **design** of the Specification Manager and the structure of its specification. Furthermore the specification **style** used is described in Section 3.2.

3.1 Design

The overall ideas in the design of the Specification Manager is the following:

- All calls to the internals of the Toolbox must go via the Specification Manager. The BaseTools class is designed for this purpose.
- The Specification Manager is to be used for both an ascii and graphical user interface. The Interface and Errors classes are abstract in the specification in order to make them independant of the kind of interface.

- The full functionality of the Toolbox can be accessed from the ToolMediator class. A restricted access to the Toolbox should be provided by the ToolKit class, forming an initial version API.
- The Repository contains the status information for the current project in the Toolbox and is updated from the Toolbox.
- The Options class contains the options for the Toolbox.

The design can be seen in figure 2.

3.2 Specification Style

The specification is described as a combination of imperative and functional parts. That is, methods and functions are used in combination.

Each class is described in a separate section by means of:

- An introduction to the class.
- An OMT class diagram giving the interface to the class. This diagram will usually have been automatically generated by the reverse VDM^{++} to OMT generator.
- The VDM^{++} specification of the class.

The following name convention are used in the specification:

- All class, type, function, value and method names starts with a capital letter.
- All local variable names start with a lower case letter. An exception is when a variable clashes a key word. In this case the variable will start with an upper case letter.
- For variables of type sequence, maps, sets, records, tuples and object reference the name might have a postfix being a underscore concatenated with the letter:

  sequence: l

  map: m

  set: s

  record: r

  tuple: t

  object reference: o

  Underscores do not appear in variable names otherwise.

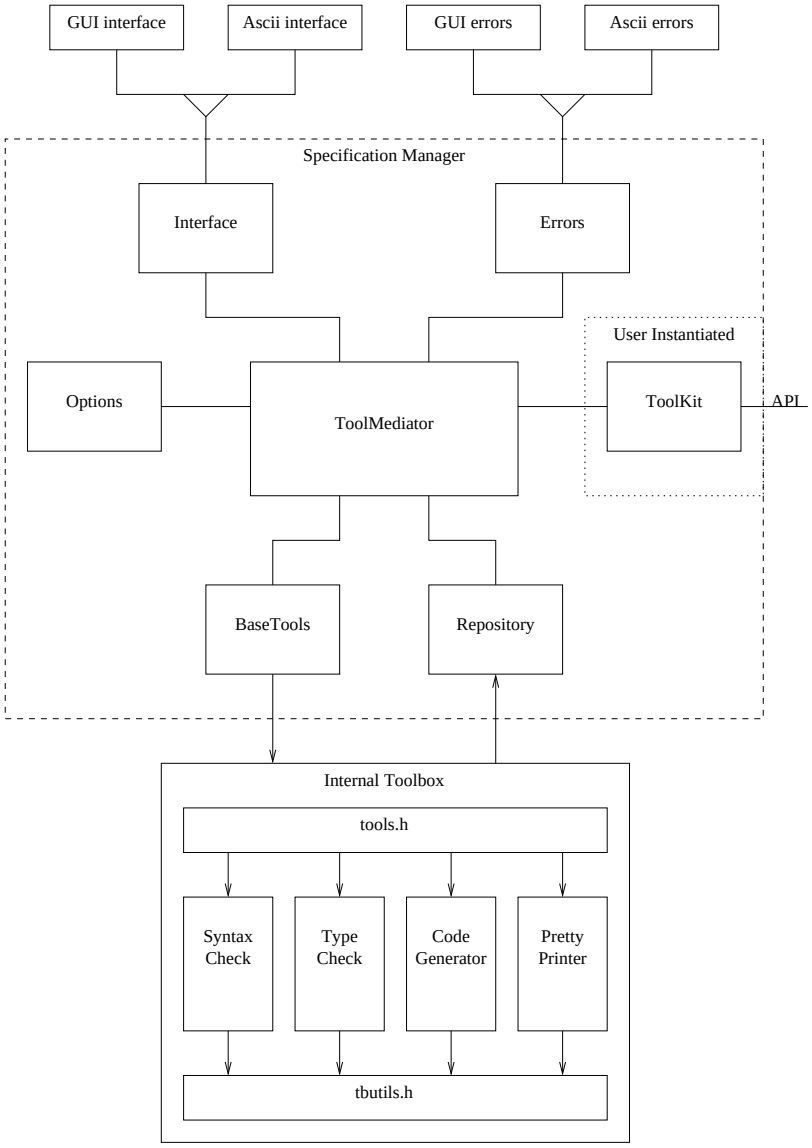

Figure 2: Context diagram and desing overview for Specification Manager

3.3 Toolkit

The classes in this section forms the coupling between the user, the internals of the Specification Manager and the functionalities of the Toolbox. This is achived using the *Mediator* behaviour pattern as described in [EJ95]. The Toolkit class is the public interface to the methods in the class ToolMediator. The *intension* behind this is that not all methods in the ToolMediator class should be made public. Only methods that the user should be able to access should be defined in the Toolkit class. In *reality* this is implemented in another way in order to reduce the number of methods. For each class that the ToolMediator refers to there is a method returning a reference to the actual class. All functionalities for the class can then be accessed by using this reference. This reduces the number of methods in the classes Toolkit and ToolMediator significantly, but on the other hand this makes all methods public.

3.3.1 Class ToolKit

The Toolkit class is the fundamental class in the Specification Manager in the sense that this class is the one that instantiates the rest of the Specification Manager. This class forms the interface to the ToolMediator class. The Toolkit class must contain at least one initialization method in order to set up object references to the abstract classes as defined by the user (see section 4) and instantiate the proper BaseTools class (4.4).

class
ToolKit is subclass of *ProjectTypes*
instance variables
$mediator : ToolMediator := \text{new } ToolMediator\,();$

The *Init* method sets up the standard set of basetools specified in class BaseTools
operations
public
$Init : Interface \times Errors \times Errors \xrightarrow{o} ()$
$Init\,(interface, err, exprerr) \triangleq mediator.$
$Init(interface, err, exprerr)\,;$
public
$GetErrors : () \xrightarrow{o} Errors$
$GetErrors\,() \triangleq mediator.$
$GetErrors()\,;$
public
$GetExprErrors : () \xrightarrow{o} Errors$
$GetExprErrors\,() \triangleq mediator.$
$GetExprErrors()\,;$
public
$GetOptions : () \xrightarrow{o} Options$
$GetOptions\,() \triangleq mediator.$
$GetOptions()\,;$
public

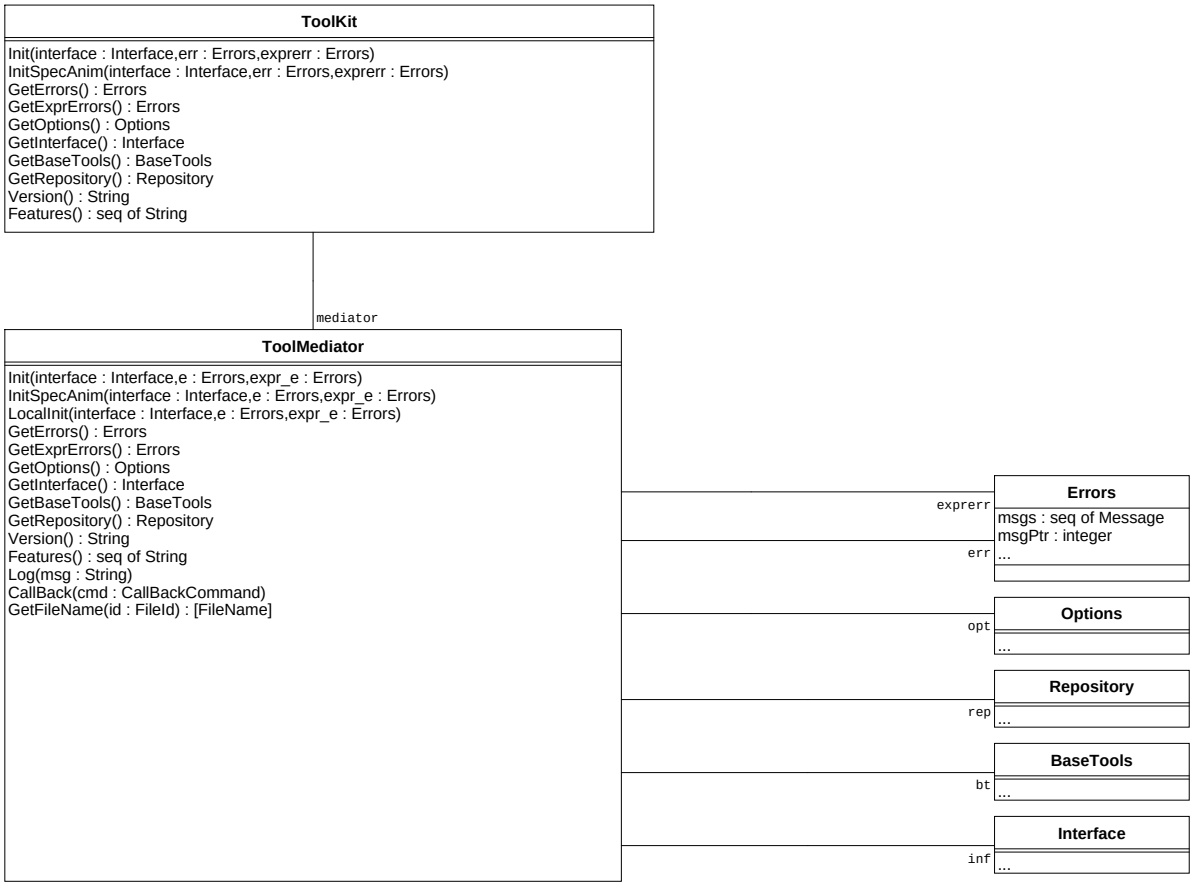

Figure 3: The ToolKit and ToolMediator classes

GetInterface : () $\xrightarrow{o}$ *Interface*
GetInterface () $\triangleq$ *mediator*.
GetInterface() ;
public
GetBaseTools : () $\xrightarrow{o}$ *BaseTools*
GetBaseTools () $\triangleq$ *mediator*.
GetBaseTools() ;
public
GetRepository : () $\xrightarrow{o}$ *Repository*
GetRepository () $\triangleq$ *mediator*.
GetRepository() ;
public
GetUMLTool : () $\xrightarrow{o}$ *UMLTool*
GetUMLTool () $\triangleq$ *mediator*.
GetUMLTool() ;

About the Toolbox version and features.
public
Version : () $\xrightarrow{o}$ *String*
Version () $\triangleq$ *mediator*.
Version() ;
public
Features : () $\xrightarrow{o}$ *String**
Features () $\triangleq$ *mediator*.
Features()
end
ToolKit

3.3.2 Class ToolMediator

The ToolMediator class is the central class in the Specification Manager. This class contains references to the different ToolColleague subclasses and makes it possible for each class to access methods in other classes. The ToolMediator class must contain initialisation methods corresponding to initialisation methods in class Toolkit.

Public interface to ToolMediator provided to the client through class ToolKit.
class
ToolMediator is subclass of *ProjectTypes*
instance variables
inf : *Interface*;
bt : *BaseTools* := new *BaseTools* ();
rep : *Repository* := new *Repository* ();
opt : *Options* := new *Options* ();
err : *Errors*;
uml : *UMLTool* := new *UMLTool* ();
exprerr : *Errors*;

operations

public

$Init : Interface \times Errors \times Errors \xrightarrow{o} ()$
$Init\,(interface, e, expr\_e) \triangleq$
($inf := interface$;
$err := e$;
$exprerr := expr\_e$;
$bt.SetMediator(\mathsf{self})$;
$rep.SetMediator(\mathsf{self})$;
$opt.SetMediator(\mathsf{self})$;
$err.SetMediator(\mathsf{self})$;
$exprerr.SetMediator(\mathsf{self})$;
$uml.SetMediator(\mathsf{self})$
);

public

$GetErrors : () \xrightarrow{o} Errors$
$GetErrors\,() \triangleq$
return err;

public

$GetExprErrors : () \xrightarrow{o} Errors$
$GetExprErrors\,() \triangleq$
return $exprerr$;

public

$GetOptions : () \xrightarrow{o} Options$
$GetOptions\,() \triangleq$
return opt;

public

$GetInterface : () \xrightarrow{o} Interface$
$GetInterface\,() \triangleq$
return inf;

public

$GetBaseTools : () \xrightarrow{o} BaseTools$
$GetBaseTools\,() \triangleq$
return bt;

public

$GetRepository : () \xrightarrow{o} Repository$
$GetRepository\,() \triangleq$
return rep;

public

$GetUMLTool : () \xrightarrow{o} UMLTool$
$GetUMLTool\,() \triangleq$
return uml;

public

$Version : () \xrightarrow{o} String$
$Version\,() \triangleq$
is not yet specified;

public

$Features : () \xrightarrow{o} String^*$
$Features\,() \triangleq$
is not yet specified;

public

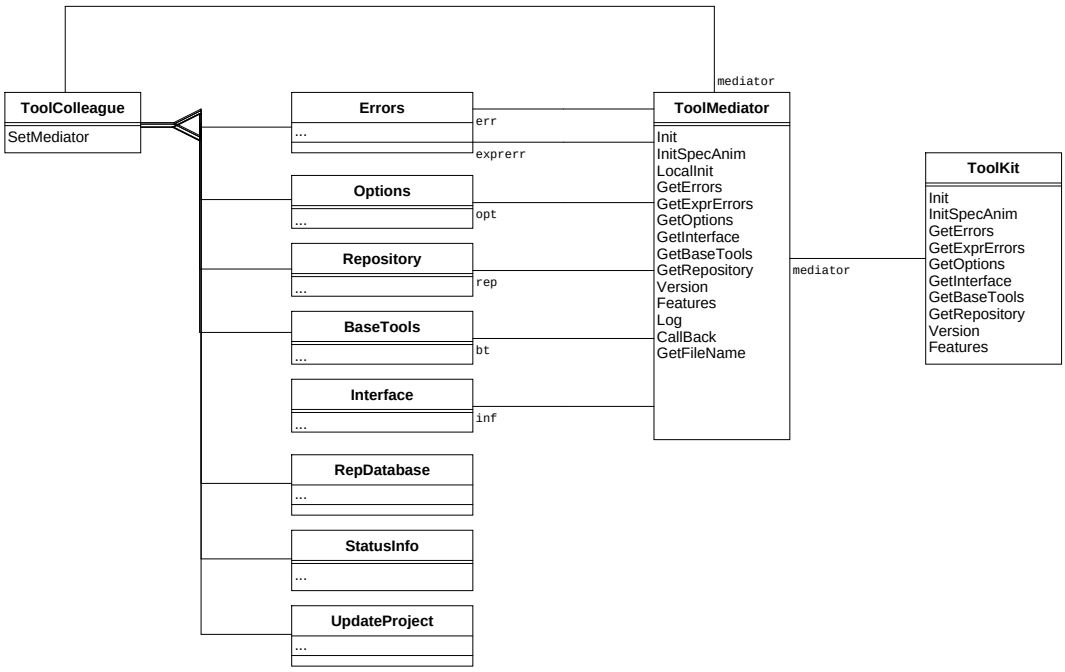

Figure 4: ToolColleague inheritance and associations for ToolMediator.

$Log : String \xrightarrow{o} ()$
$Log\,(msg) \triangleq inf.$
$\quad Log(msg)$;

public

$CallBack : CallBackCommand \xrightarrow{o} ()$
$CallBack\,(cmd) \triangleq inf.$
$\quad CallBack(cmd)$;

public

$GetFileName : FileId \xrightarrow{o} [FileName]$
$GetFileName\,(id) \triangleq rep.$
$\quad GetFileName(id)$

end
ToolMediator

3.3.3 Class ToolColleague

This class is superclass for all classes that need an object reference to the ToolMediator class as part of the instance variable.

class
ToolColleague is subclass of *ProjectTypes*
instance variables
protected $mediator : ToolMediator$;

operations
public
$SetMediator : ToolMediator \xrightarrow{o} ()$
$SetMediator\,(m) \triangleq$
$mediator := m$
end
ToolColleague

SetMediator MUST be called before any other methods in the sub classes of ToolColleague (is called in the initialisation statement of ToolMediator).

3.4 Repository Interface

The repository subsystem of the Specification Manager contains information about status for files and modules/classes and the corresponding AST's. The Repository class forms the interface to all repository related classes and instantiates the classes of the subsystem. The associations between the Repository class and the subsystem classes can be seen in figure 5.

3.4.1 Class Repository

The primary role for the Repository class is to delegate the different repository calls to the prober subsystem class and in this way form a uniform interface to the caller independant of the subsystem structure.
class
Repository is subclass of *ToolColleague*
instance variables
$repdb : RepDatabase := \text{new } RepDatabase\,();$
$cgrepos : CGRepository := \text{new } CGRepository\,();$
$session : UpdateSes := \text{new } NoneSes\,();$
$status : StatusInfo := \text{new } StatusInfo\,();$
$project : UpdateProject := \text{new } UpdateProject\,();$
$depend : Dependency := \text{new } Dependency\,();$
$old\_ses : SessionType;$
$fileStateSaved : \mathbb{B} := \text{true};$
$packagePrefix : FileName^* := [];$

operations
public

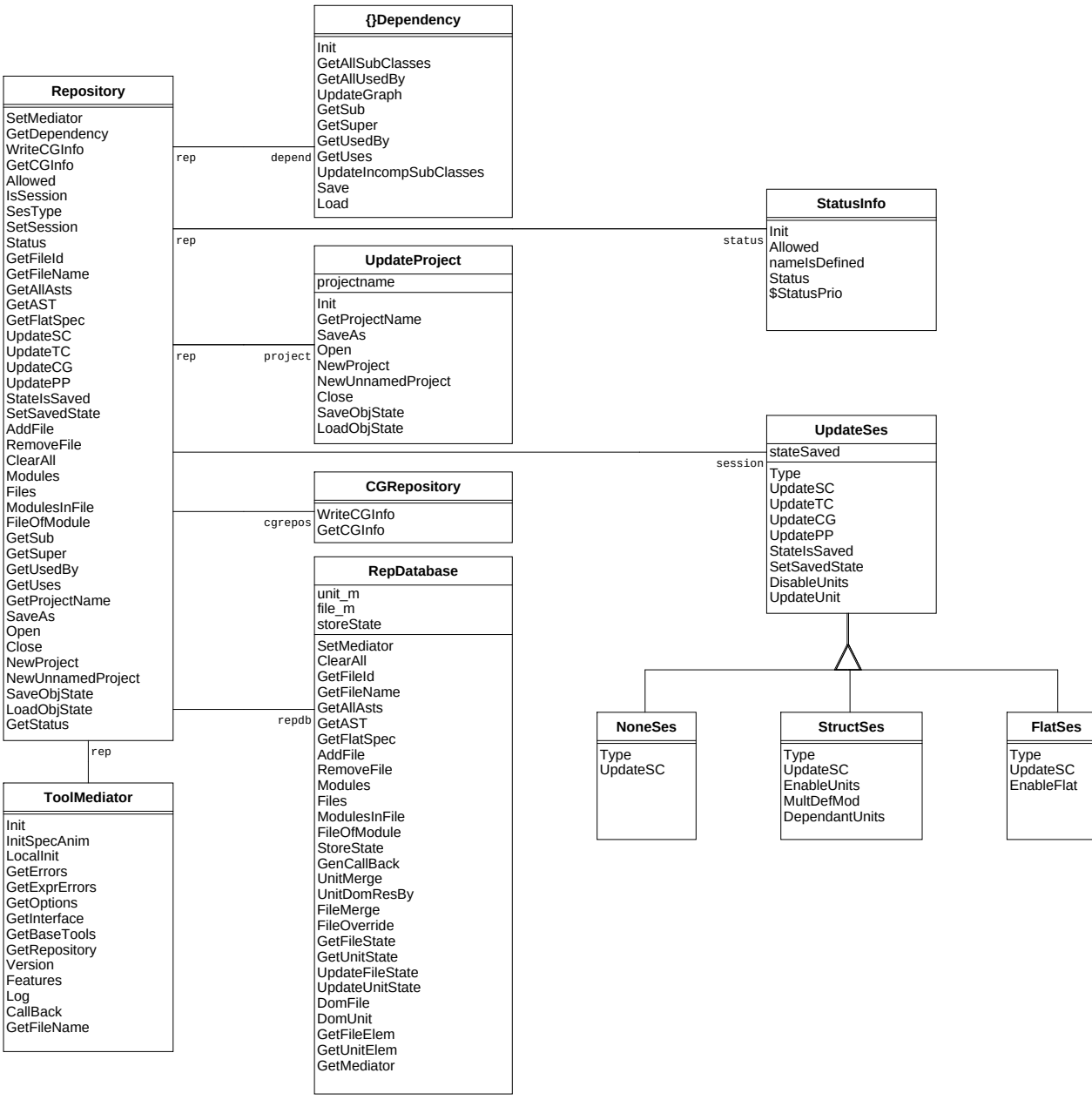

Figure 5: Associations for Repository class

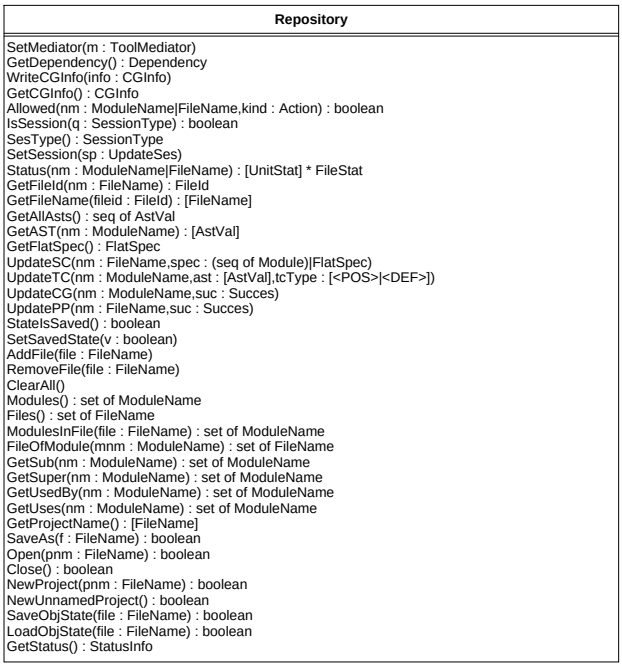

Figure 6: Interface to Repository class

$$SetMediator : ToolMediator \xrightarrow{o} ()$$
$$SetMediator\,(m) \triangleq$$
$$\begin{array}{ll}(& mediator := m;\\ & repdb.SetMediator(m)\,;\\ & status.Init(\mathsf{self}, repdb, m)\,;\\ & depend.Init(\mathsf{self})\,;\\ & project.Init(\mathsf{self}, repdb, m, depend)\\ );\end{array}$$

public

$$GetDependency : () \xrightarrow{o} Dependency$$
$$GetDependency\,() \triangleq$$
$$\mathsf{return}\ depend;$$

Interface to the CGRepository

public

$$WriteCGInfo : CGInfo \xrightarrow{o} ()$$
$$WriteCGInfo\,(info) \triangleq cgrepos.$$
$$WriteCGInfo(info)\,;$$

public

$$GetCGInfo : () \xrightarrow{o} CGInfo$$
$$GetCGInfo\,() \triangleq cgrepos.$$
$$GetCGInfo()\,;$$

Methods handling status and type of session

public

$Allowed : (ModuleName \mid FileName) \times Action \xrightarrow{o} \mathbb{B} \times \mathsf{char}^*$
$Allowed\,(nm, kind) \triangleq$
  def $rt = status.Allowed\,(nm, kind)$ in
  return rt;

public

$IsSession : SessionType \xrightarrow{o} \mathbb{B}$
$IsSession\,(q) \triangleq$
  def $sestype = session.Type\,()$ in
  return $q = sestype$;

public

$SesType : () \xrightarrow{o} SessionType$
$SesType\,() \triangleq$
  def $stp = session.Type\,()$ in
  return stp;

public

$DisableSession : () \xrightarrow{o} ()$
$DisableSession\,() \triangleq$
  (def $sestype = session.Type\,()$ in
    if $sestype \neq \text{NONE} \wedge sestype \neq \text{DISABLED}$
    then $old\_ses := sestype$;
    $session :=$ new $NoneSes\,()$;
    $session.Disable()$
  );

public

$EnableSession : () \xrightarrow{o} ()$
$EnableSession\,() \triangleq session.$
  $Enable()$;

public

$OldSession : () \xrightarrow{o} SessionType$
$OldSession\,() \triangleq$
  return old_ses;

The *DisableSession*, *EnableSession* and *OldSession* has to do with possible changing of session type in the same session. If all parsed files in the current project is going to be parsed again, it is possible to change the session type, and *DisableSession* is called. The current session type is stored in $old_s es$ and the session is set to *NoneSes* and is furthermore disabled. If the session type does not change (*OldSession*) by the parsing, *EnableSession* is called.

public

$SetSession : UpdateSes \xrightarrow{o} ()$
$SetSession\,(sp) \triangleq$
  $session := sp$;

SetSession sets the current type of session. Is only called from *NoneSes* class.

public

$Status : (ModuleName \mid FileName) \xrightarrow{o} [UnitStat] \times FileStat$
$Status\,(nm) \triangleq$
  def $rt = status.Status\,(nm)$ in
  return rt;

public

$GetStatus : () \xrightarrow{o} StatusInfo$
$GetStatus\ () \triangleq$
return *status*;

Status returns a tuple consisting of status for the module and status for the corresponding file. If called with a filename the status for module is *nil*. Run-time error if called with an unknown name.

public

$IsAllTypeCorrect : () \xrightarrow{o} \mathbb{B}$
$IsAllTypeCorrect\ () \triangleq status.$
$IsAllTypeCorrect()$;

public

$IsSyntaxCorrect : FileName \xrightarrow{o} \mathbb{B}$
$IsSyntaxCorrect\ (nm) \triangleq status.$
$IsSyntaxCorrect(nm)$;

Interface to Repository Database

public

$GetFileId : FileName \xrightarrow{o} FileId$
$GetFileId\ (nm) \triangleq repdb.$
$GetFileId(nm)$;

public

$GetCmdLineFileId : () \xrightarrow{o} FileId$
$GetCmdLineFileId\ () \triangleq repdb.$
$GetIrregFileId(\text{CMDLINEFILEID})$;

public

$GetCgFileId : () \xrightarrow{o} FileId$
$GetCgFileId\ () \triangleq repdb.$
$GetIrregFileId(\text{CGFILEID})$;

public

$GetTcFileId : () \xrightarrow{o} FileId$
$GetTcFileId\ () \triangleq repdb.$
$GetIrregFileId(\text{TCFILEID})$;

public

$GetTmpFileId : () \xrightarrow{o} FileId$
$GetTmpFileId\ () \triangleq repdb.$
$GetIrregFileId(\text{TMPFILEID})$;

public

$GetStdInFileId : () \xrightarrow{o} FileId$
$GetStdInFileId\ () \triangleq repdb.$
$GetIrregFileId(\text{STDINFILEID})$;

public

$GetFileName : FileId \xrightarrow{o} [FileName]$
$GetFileName\ (fileid) \triangleq repdb.$
$GetFileName(fileid)$;

public

$GetAllAsts : () \xrightarrow{o} AstVal^*$
$GetAllAsts\,() \triangleq repdb.$
$\quad GetAllAsts() ;$

public
$GetJavaAsts : () \xrightarrow{o} AstVal^*$
$GetJavaAsts\,() \triangleq repdb.$
$\quad GetJavaAsts() ;$

public
$GetVDMAsts : () \xrightarrow{o} AstVal^*$
$GetVDMAsts\,() \triangleq repdb.$
$\quad GetVDMAsts() ;$

public
$GetAST : ModuleName \xrightarrow{o} [AstVal]$
$GetAST\,(nm) \triangleq repdb.$
$\quad GetAST(nm) ;$

public
$GetFlatSpec : () \xrightarrow{o} FlatSpec$
$GetFlatSpec\,() \triangleq repdb.$
$\quad GetFlatSpec() ;$

public
$SetFileTimestamp : FileName \times \mathbb{N} \xrightarrow{o} ()$
$SetFileTimestamp\,(f, t) \triangleq repdb.$
$\quad SetFileTimestamp(f, t) ;$

public
$GetFileTimestamp : FileName \xrightarrow{o} \mathbb{N}$
$GetFileTimestamp\,(f) \triangleq repdb.$
$\quad GetFileTimestamp(f) ;$

public
$FileModified : FileName \xrightarrow{o} \mathbb{B}$
$FileModified\,(f) \triangleq repdb.$
$\quad FileModified(f) ;$

public
$SetFileModified : FileName \xrightarrow{o} ()$
$SetFileModified\,(f) \triangleq repdb.$
$\quad SetFileModified(f, session) ;$

Interface to Update methods

public
$UpdateSC : FileName \times ((Module^*) \mid FlatSpec) \xrightarrow{o} ()$
$UpdateSC\,(nm, spec) \triangleq session.$
$\quad UpdateSC(\mathsf{self}, repdb, nm, spec) ;$

public
$UpdateTC : ModuleName \times [\text{POS} \mid \text{DEF}] \xrightarrow{o} ()$
$UpdateTC\,(nm, tcType) \triangleq session.$
$\quad UpdateTC(repdb, nm, tcType) ;$

public
$UpdateCG : ModuleName \times (\text{CPP} \mid \text{JAVA}) \times Succes \xrightarrow{o} ()$
$UpdateCG\,(nm, kind, suc) \triangleq session.$
$\quad UpdateCG(repdb, nm, kind, suc) ;$

```
public
        UpdatePP : FileName × Succes →o ()
        UpdatePP (nm, suc) ≜ session.
          UpdatePP(repdb, nm, suc) ;
public
        StateIsSaved : () →o B
        StateIsSaved () ≜
          return fileStateSaved;
public
        SetSavedFileState : B →o ()
        SetSavedFileState (v) ≜
          fileStateSaved := v;
public
        AddFiles : FileName-set →o ()
        AddFiles (files) ≜
          (   fileStateSaved := false;
              repdb.AddSetOfFiles(files)
          );
public
        AddTempFile : FileName →o TmpFileName
        AddTempFile (fnm) ≜ repdb.
          AddTempFile(fnm) ;
public
        GetTempFileName : FileName →o TmpFileName
        GetTempFileName (fnm) ≜ repdb.
          GetTempFileName(fnm) ;
public
        GetRealFileName : TmpFileName →o FileName
        GetRealFileName (tfnm) ≜ repdb.
          GetRealFileName(tfnm) ;
```

The method *RemoveFiles* remove all the files in the repository by calling the method *RemoveSetOfFiles*, and furthermore, the dependency information is updated, the session type is set to none in case all files are removed, and the inheritance tree is updated.

The *StateIsSaved* method is called from the interface when the project is (re)configured. In version 2 of the ProjectFile (see class *StateStore*) this method calls *StateIsSaved* in class *UpdateSession*. In version 3 of the ProjectFile only file names are stored in the project file, so instead the *FileStateIsSaved* is called in the *UpdateSession* class. In this case we are only interested in adding and removing of files. So we have different levels of change of state even that we are only subscribing to the change of files.

```
public
        RemoveFiles : FileName-set →o ()
        RemoveFiles (files) ≜
          (   dcl modules : ModuleName-set := {};
              fileStateSaved := false;
              if card files > 0
              then (   for all file ∈ files
```

```
                    do def mods = repdb.ModulesInFile (file) in
                       modules := modules ∪ mods;
                    depend.Remove(modules) ;
                    def nofiles = repdb.RemoveSetOfFiles (files) in
                    if nofiles
                    then def newses = new NoneSes () in
                         session := newses;
                    def - = mediator.GetBaseTools ().InhTree () in
                    skip
                )
    );
public
    ClearAll : () ->o ()
    ClearAll () ≜
      def baseTool = mediator.GetBaseTools () in
      (   baseTool.InitToolbox() ;
          cgrepos := new CGRepository ();
          session := new NoneSes ();
          repdb.ClearAll() ;
          depend.Init(self)
      );
public
    AllModules : () ->o ModuleName-set
    AllModules () ≜ repdb.
      AllModules() ;
public
    VDMModules : () ->o ModuleName-set
    VDMModules () ≜ repdb.
      VDMModules() ;
public
    JavaModules : () ->o ModuleName-set
    JavaModules () ≜ repdb.
      JavaModules() ;
public
    Files : () ->o FileName-set
    Files () ≜ repdb.
      Files() ;
public
    ModulesInFile : FileName ->o ModuleName-set
    ModulesInFile (file) ≜ repdb.
      ModulesInFile(file) ;
public
    FileOfModule : ModuleName ->o FileName-set
    FileOfModule (mnm) ≜ repdb.
      FileOfModule(mnm) ;
public
    ParsedFiles : () ->o FileName-set
    ParsedFiles () ≜ repdb.
      ParsedFiles() ;
public
```

```
        GetPackageOfFile : FileName × char →o FileName
        GetPackageOfFile (file, sepChar) △
          let path = MakePath (file, sepChar),
              packageFile = RemovePrefix (packagePrefix, path) in
          return dirname (packageFile, sepChar);
public
        GetFilesAndPackages : char →o FileName →m (FileName × FileName)-set
        GetFilesAndPackages (sepChar) △
          let files = Files () in
          if files = {}
          then return {↦}
          elseif card files = 1
          then let {f} = files,
                   p = MakePath (f, sepChar),
                   bn = basename (p) in
               return {mk_FileName ("") ↦ {mk_ (bn, f)}}
          else let paths = {mk_ (MakePath (f, sepChar), f) | f ∈ files},
                   prefix = CommonPrefix ({p | mk_ (p, -) ∈ paths}),
                   packagePaths = {mk_ (RemovePrefix (prefix, p), q) |
                                   mk_ (p, q) ∈ paths} in
               (   dcl packageMap : FileName →m (FileName × FileName)-set := {↦};
                   packagePrefix := prefix;
                   for all mk_ (p, q) ∈ packagePaths
                   do let package = dirname (p, sepChar),
                          file = basename (p),
                          properFile = if file = mk_FileName ([])
                                       then q
                                       else file in
                      if package ∈ dom packageMap
                      then packageMap(package) := packageMap (package)∪{mk_ (properFile, q)}
                      else packageMap(package) := {mk_ (properFile, q)};
                   return packageMap
               )
functions
public
        MakePath : FileName × char→FileName*
        MakePath (fname, sepChar) △
          let rawname = fname.nm,
              leadingSep = hd rawname = sepChar,
              sepInds =
                  (if leadingSep
                   then [1]
                   else []) ↷
                  [i | i ∈ inds rawname · rawname (i) = sepChar] ↷
                  [len rawname + 1],
              dirs = [rawname (1, ..., sepInds (1) − 1)] ↷
                     [rawname (sepInds (j) + 1, ..., sepInds (j + 1) − 1) | j ∈ inds sepInds ·
j < len sepInds] in
          [mk_FileName (dirs (i)) | i ∈ inds dirs · dirs (i) ≠ []];
public
```

$CommonPrefix : FileName^*\text{-set}\tilde{\to} FileName^*$
$CommonPrefix\,(paths) \triangleq$
  if $\exists\, p \in paths \cdot p = []$
  then $[]$
  else let $heads = \{\mathsf{hd}\ p \mid p \in paths\}$ in
    if card $heads \neq 1$
    then $[]$
    else let $\{p\} = heads$ in
      $[p] \frown CommonPrefix\,(\{\mathsf{tl}\ p \mid p \in paths\})$;
public
$RemovePrefix : FileName^* \times FileName^* \tilde{\to} FileName^*$
$RemovePrefix\,(pref, path) \triangleq$
  if $pref = [] \vee path = []$
  then $path$
  elseif hd $pref =$ hd $path$
  then $RemovePrefix\,(\mathsf{tl}\ pref, \mathsf{tl}\ path)$
  else $path$;
public
$dirname : FileName^* \times \mathsf{char} \tilde{\to} FileName$
$dirname\,(dirs, sepChar) \triangleq$
  if len $dirs \leq 1$
  then mk_$FileName\,([])$
  else let $restDir = dirname\,(\mathsf{tl}\ dirs, sepChar)$ in
    mk_$FileName\,((\mathsf{hd}\ dirs).nm \frown [sepChar] \frown restDir.nm)$;
public
$basename : FileName^* \tilde{\to} FileName$
$basename\,(dirs) \triangleq$
  if $dirs = []$
  then mk_$FileName\,([])$
  else $dirs\,(\mathsf{len}\ dirs)$

Dependency information for list boxes

operations
public
$GetSub : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetSub\,(nm) \triangleq depend.$
  $GetSub(nm)$;
public
$GetSuper : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetSuper\,(nm) \triangleq depend.$
  $GetSuper(nm)$;
public
$GetUsedBy : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetUsedBy\,(nm) \triangleq depend.$
  $GetUsedBy(nm)$;
public
$GetUses : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetUses\,(nm) \triangleq depend.$
  $GetUses(nm)$;

Process information

public

$IsCyclic : ModuleName \xrightarrow{o} \mathbb{B}$
$IsCyclic\,(nm) \triangleq depend.$
$\quad IsCyclic(nm)\,;$

public

$OrderOfProcess : ModuleName \xrightarrow{o} ModuleName\text{-set}^*$
$OrderOfProcess\,(nm) \triangleq depend.$
$\quad OrderOfProcess(nm)\,;$

Open, Close, Save and New Project

public

$GetProjectName : () \xrightarrow{o} [FileName]$
$GetProjectName\,() \triangleq project.$
$\quad GetProjectName()\,;$

public

$SaveAs : FileName \xrightarrow{o} \mathbb{B}$
$SaveAs\,(f) \triangleq project.$
$\quad SaveAs(f)\,;$

public

$Open : FileName \xrightarrow{o} \mathbb{B}$
$Open\,(pnm) \triangleq project.$
$\quad Open(pnm)\,;$

public

$NewUnnamedProject : () \xrightarrow{o} \mathbb{B}$
$NewUnnamedProject\,() \triangleq project.$
$\quad NewUnnamedProject()\,;$

public

$SaveObjState : FileName \xrightarrow{o} \mathbb{B}$
$SaveObjState\,(file) \triangleq project.$
$\quad SaveObjState(file)\,;$

public

$LoadObjState : FileName \xrightarrow{o} \mathbb{B}$
$LoadObjState\,(file) \triangleq project.$
$\quad LoadObjState(file)$

end
Repository

3.4.2 Class CGRepository

class
CGRepository is subclass of *ProjectTypes*
operations
public

$WriteCGInfo : CGInfo \xrightarrow{o} ()$
$WriteCGInfo\,(info) \triangleq$
$\quad$ is not yet specified;

WriteCGInfo writes the CGInfo, info, on the file system. The name of the CGInfo file is stored in the options class.

public

GetCGInfo : () $\xrightarrow{o}$ *CGInfo*
GetCGInfo () $\triangleq$
is not yet specified

end
CGRepository

GetCGInfo reads a the CGInfo from the file system. The name of the CGInfo file is stored in the options class.

3.5 Database for Repository

The database for the repository is the class that contains the file and module/class status information as part of the object state.

3.5.1 Class RepDatabase

The core data in the RepDatabase class is the two maps for file status and module/class status. These maps contain object references to the FileStatus and VDMUnitElem classes as can be seen from figure 7

class
RepDatabase is subclass of *ToolColleague*
instance variables
unit_m : *UnitState* := $\{\mapsto\}$;
file_m : *FileState* := $\{\mapsto\}$;
storeState : *StateType* := mk_($\{\mapsto\}, \{\mapsto\}$);

storeState is used to save the current object state for files and modules/classes in order to compute changes in the state, needed by the interface.

values
minFileId = 10;
predefFileIds = $\{$CMDLINEFILEID $\mapsto$ 1,
CGFILEID $\mapsto$ 2,
TCFILEID $\mapsto$ 3,
TMPFILEID $\mapsto$ 4,
STDINFILEID $\mapsto$ 5$\}$

operations
public
RepDatabase : () $\xrightarrow{o}$ *RepDatabase*
RepDatabase () $\triangleq$
(*Init*() ;
return self
);

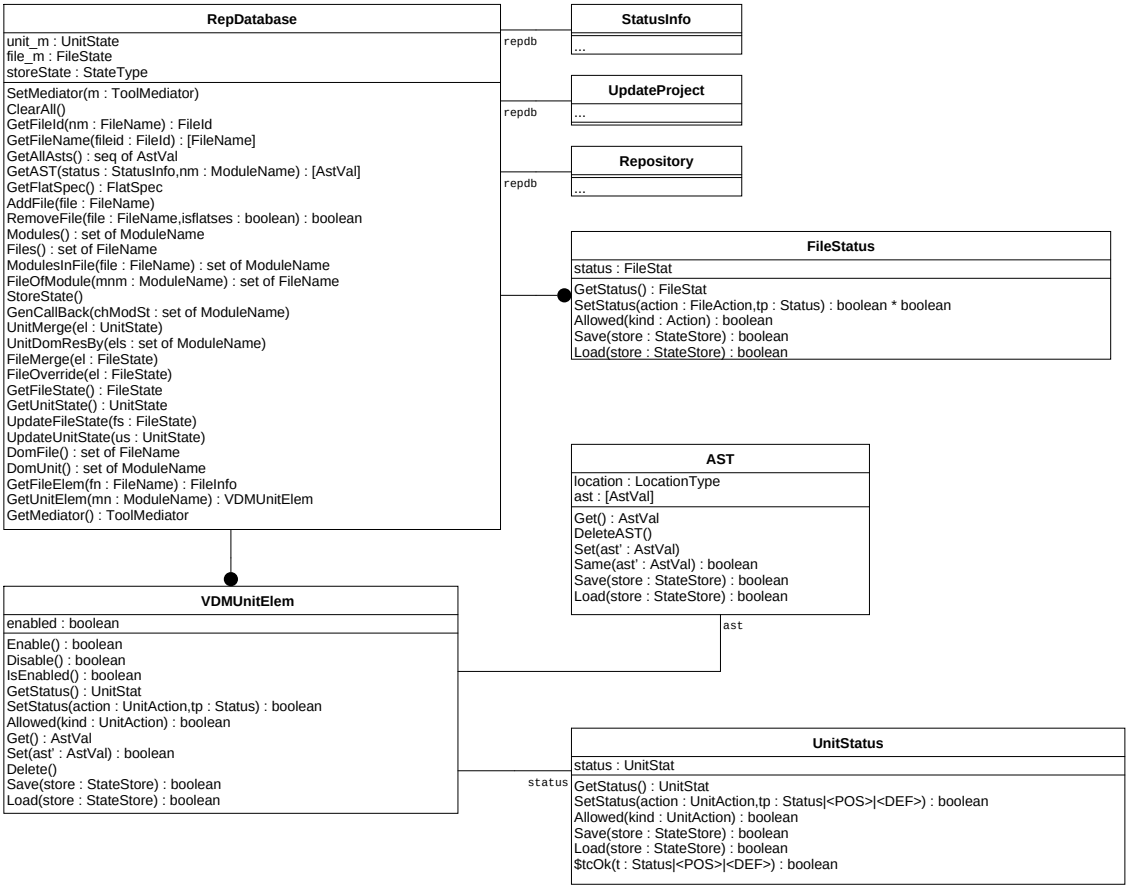

Figure 7: Associations and Interface for database related classes.

```
public
        SetMediator : ToolMediator ->o ()
        SetMediator (m) ≜
          mediator := m;
public
        ClearAll : () ->o ()
        ClearAll () ≜
          (   mediator.CallBack(mk_ProjectTypes`ClearAll ()) ;
              RemTempFiles() ;
              unit_m := {↦};
              file_m := {↦};
              storeState := mk_({↦}, {↦})
          );
```

ClearAll is called whenever the state is to be cleared, i.e. by change of project.

```
public
        GetFileId : FileName ->o FileId
        GetFileId (nm) ≜
          (   dcl fid : FileId := 0;
              Lock() ;
              if nm ∈ dom file_m
              then let mk_(-,-,fileid,-) = file_m (nm) in
                     fid := fileid;
              UnLock() ;
              return fid
          );
public
        GetIrregFileId : IrregFileIds ->o FileId
        GetIrregFileId (ireg) ≜
          return predefFileIds (ireg);
public
        GetFileName : FileId ->o [FileName]
        GetFileName (fileid) ≜
          (   dcl filenm : [FileName] := nil ;
              Lock() ;
              if fileid ≥ minFileId
              then for all fnm ∈ dom file_m
                     do  cases  file_m (fnm):
                           mk_(-,-,(fileid),-) → filenm := fnm,
                           others → skip
                         end
              else filenm := mk_FileName ("");
              UnLock() ;
              return filenm
          );
public
        GetAllAsts : () ->o AstVal*
        GetAllAsts () ≜
          (   dcl asts : AstVal* := [];
```

```
                Lock() ;
                for all un ∈ rng unit_m
                do def ena = un.IsEnabled () in
                   if ena
                   then def unAst = un.Get () in
                          asts := asts ↷ [unAst];
                UnLock() ;
                return asts
             );
public
          GetJavaAsts : () →o AstVal*
          GetJavaAsts () △
             (  dcl asts : AstVal* := [];
                Lock() ;
                for all md ∈ dom unit_m
                do if isJava (md)
                   then let un = unit_m (md) in
                          if un.IsEnabled ()
                          then asts := asts ↷ [un.Get ()];
                UnLock() ;
                return asts
             );
public
          GetVDMAsts : () →o AstVal*
          GetVDMAsts () △
             (  dcl asts : AstVal* := [];
                Lock() ;
                for all md ∈ dom unit_m
                do if ¬ isJava (md)
                   then let un = unit_m (md) in
                          if un.IsEnabled ()
                          then asts := asts ↷ [un.Get ()];
                UnLock() ;
                return asts
             );
public
          GetAST : ModuleName →o [AstVal]
          GetAST (nm) △
             (  dcl astval : [AstVal] := nil ;
                Lock() ;
                if nm ∈ dom unit_m
                then let unit : VDMUnitElem = unit_m (nm) in
                       astval := unit.Get ();
                UnLock() ;
                return astval
             )
          pre  nm ∈ dom unit_m ;
public
```

```
        GetFlatSpec : () ^o→ FlatSpec
        GetFlatSpec () △
          (   Lock() ;
              let {defmodname} = dom unit_m,
                  {defmodunit} = rng unit_m,
                  ast = defmodunit.Get () in
              let fs = mk_FlatSpec (defmodname, ast) in
              (   UnLock() ;
                  return fs
              )
          )
        pre  card dom unit_m = 1 ;
public
        AddSetOfFiles : FileName-set ^o→ ()
        AddSetOfFiles (files) △
          (   dcl new_file_s : FileName-set := {};
              Lock() ;
              new_file_s := files \ dom file_m;
              if new_file_s ≠ {}
              then for all file ∈ new_file_s
                   do def fstat : FileStatus = new FileStatus () in
                      (   dcl fileid : ℕ1 := minFileId;
                          let fileid_s = {id | mk_(-,-,id,-) ∈ rng file_m} in
                          while fileid ∈ fileid_s
                          do fileid := fileid + 1;
                          file_m := file_m ⩊ {file ↦ mk_({},fstat,fileid,nil )}
                      ) ;
              UnLock() ;
              if new_file_s ≠ {}
              then mediator.CallBack(mk_AddFiles (new_file_s))
          );
public
        RemoveSetOfFiles : FileName-set ^o→ 𝔹
        RemoveSetOfFiles (files) △
          (   dcl rfiles : FileName-set := {},
                  rmds : ModuleName-set := {};
              Lock() ;
              for all file ∈ files
```

```
        do if file ∈ dom file_m
           then let mk_(unitnm_s, -, -, -) = file_m(file) in
                (  file_m := {file} ⩤ file_m;
                   let remmod_s = {u | u ∈ unitnm_s ·
                                        (card {f | f ∈ dom file_m·let mk_(m_s, -, -, -) = file_m
                                                                  u ∈ m_s}) = 0} in
                   (  unit_m := remmod_s ⩤ unit_m;
                      rmds := rmds ∪ remmod_s;
                      rfiles := rfiles ∪ {file}
                   )
                ) ;
           let parsedfiles_s = {f | f ∈ dom file_m·let mk_(mod_s, -, -, -) = file_m(f) in
                                                 card mod_s > 0},
               ok = parsedfiles_s = {} in
           (  UnLock() ;
              mediator.CallBack(mk_RemoveFiles(rfiles)) ;
              mediator.CallBack(mk_RemoveModules(rmds)) ;
              return ok
           )
        );
```

RemoveSetOfFiles will only remove the corresponding modules to the file if no other files refer to these modules. This is basicly for flat specifications.

public

```
        SetFileTimestamp : FileName × ℕ →o ()
        SetFileTimestamp(f, t) ≙
          (  dcl change : 𝔹 := false;
             Lock() ;
             if f ∈ dom file_m
             then let mk_(-, filestat, -, -) = file_m(f) in
                     change := filestat.SetTimestamp(t);
             UnLock() ;
             if change
             then mediator.CallBack(mk_RemoveFileChangedMark(f))
          );
```

The *SetFileTimestamp* is called when a file is parsed and removes possibly a file-changed-mark in the interface.

public

```
        GetFileTimestamp : FileName →o ℕ
        GetFileTimestamp(f) ≙
          (  dcl ts : ℕ := 0;
             Lock() ;
             if f ∈ dom file_m
             then let mk_(-, filestat, -, -) = file_m(f) in
                     ts := filestat.GetTimestamp();
             UnLock() ;
             return ts
          );
```

public

```
        FileModified : FileName →o B
        FileModified (f) △
          (   dcl modified : B := false;
              Lock() ;
              if f ∈ dom file_m
              then let mk_(-, filestat, -, -) = file_m (f) in
                     modified := filestat.IsModified ();
              UnLock() ;
              return modified
          );
public
        SetFileModified : FileName × UpdateSes →o ()
        SetFileModified (f, -) △
          (   dcl changed : B := false;
              Lock() ;
              if f ∈ dom file_m
              then let mk_(-, filestat, -, -) = file_m (f) in
                     (   filestat.SetModified() ;
                         changed := true
                     ) ;
              UnLock() ;
              if changed
              then mediator.CallBack(mk_AddFileChangedMark (f))
          )
```

If a file in the project is modified on the file system, *SetFileModified* is called (from Repository). Which action to be taken is not yet decided, but in any case is the file set as modified by the call to *filestat.SetModified*. Then a call back to add a file-changed mark is generated. Another possibility is to set syntax check status to *NONE* and disable corresponding units. This is commented out in this version.

```
functions
        isJava : ModuleName→̃B
        isJava (moduleName) △
          moduleName.nm (1, ..., 5) = ".java"
operations
public
        JavaModules : () →o ModuleName-set
        JavaModules () △
          (   Lock() ;
              let mod_s = {m | m ∈ dom unit_m · isJava (m)} in
              (   UnLock() ;
                  return mod_s
              )
          );
public
```

```
        VDMModules : () ^o→ ModuleName-set
        VDMModules () ≜
          (   Lock() ;
              let mod_s = {m | m ∈ dom unit_m · ¬ isJava (m)} in
              (   UnLock() ;
                  return mod_s
              )
          );
public
        AllModules : () ^o→ ModuleName-set
        AllModules () ≜
          (   Lock() ;
              let mod_s = dom unit_m in
              (   UnLock() ;
                  return mod_s
              )
          );
public
        Files : () ^o→ FileName-set
        Files () ≜
          (   Lock() ;
              let file_s = dom file_m in
              (   UnLock() ;
                  return file_s
              )
          );
public
        ModulesInFile : FileName ^o→ ModuleName-set
        ModulesInFile (file) ≜
          (   dcl mn_s : ModuleName-set := {};
              Lock() ;
              if file ∈ dom file_m
              then let mk_(mod_s, -, -, -) = file_m (file) in
                   mn_s := mod_s;
              UnLock() ;
              return mn_s
          );
public
        SetModulesInFile : FileName × ModuleName-set ^o→ ()
        SetModulesInFile (nm, nm_s) ≜
          (   Lock() ;
              if nm ∈ dom file_m
              then let mk_(-, stat, file_id, tmpfile) = file_m (nm) in
                   file_m := file_m † {nm ↦ mk_(nm_s, stat, file_id, tmpfile)};
              UnLock()
          );
public
        FileOfModule : ModuleName ^o→ FileName-set
        FileOfModule (mnm) ≜
          (   dcl file_s : FileName-set := {};
```

```
        Lock() ;
        for all fnm ∈ dom file_m
        do let mk_(mod_s,-,-,-) = file_m(fnm) in
           if mnm ∈ mod_s
           then file_s := file_s ∪ {fnm};
        UnLock() ;
        return file_s
    );
```

FileOfModule of module updated to handle modules distributed over a number of files (DefaultMod in case of flat specifications)

```
public
    ParsedFiles : () ->o FileName-set
    ParsedFiles () ≜
      (   Lock() ;
          let file_s = {f | f ∈ dom file_m · let mk_(mod_s,-,-,-) = file_m(f) in
                                            card mod_s > 0} in
          (   UnLock() ;
              return file_s
          )
      );
public
    StoreState () ≜
      (   Lock() ;
          storeState := mk_(unit_m, file_m);
          UnLock()
      );
```

StoreState is called to save the current state of file and unit maps. This is done in order to detect changes in presence of files and modules. This information is used to the call-back to the interface.

```
public
    GenCallBack : ModuleName-set ->o ()
    GenCallBack (chModSt) ≜
      (   dcl domOldUnit : ModuleName-set := {},
              domOldFile : FileName-set := {},
              domUnit : ModuleName-set := {},
              domFile : FileName-set := {};
          Lock() ;
          def mk_(oldUnit_m, oldFile_m) = storeState in
          (   domOldUnit := dom oldUnit_m;
              domOldFile := dom oldFile_m
          ) ;
          domUnit := dom unit_m;
          domFile := dom file_m;
          UnLock() ;
          let eqUnit = domOldUnit = domUnit,
```

```
                eqFile = domOldFile = domFile in
            if ¬(eqUnit ∧ eqFile)
            then if eqFile
                 then def newMod = domUnit \ domOldUnit;
                          delMod = domOldUnit \ domUnit in
                      (   if newMod ≠ {}
                          then mediator.CallBack(mk_AddModules(newMod)) ;
                          if delMod ≠ {}
                          then mediator.CallBack(mk_RemoveModules(delMod))
                      )
                 else if eqUnit
                      then def newFile = domFile \ domOldFile;
                               delFile = domOldFile \ domFile in
                           (   if newFile ≠ {}
                               then mediator.CallBack(mk_AddFiles(newFile)) ;
                               if delFile ≠ {}
                               then mediator.CallBack(mk_RemoveFiles(delFile))
                           ) ;
            if chModSt ≠ {}
            then mediator.CallBack(mk_ChangedModuleStatus(chModSt))
        );
```

GenCallBack detects the change in the repositorys state concerning present files and modules and generates corresponding call-back's to the interface. It is called with a set of module names indicating that these module have changed their status. After file and module call-backs are generated, the call-back for module status is issued (this has to be done after the call- back for modules).

```
public
        UnitDomResBy : ModuleName-set →o ()
        UnitDomResBy(els) ≙
          (   Lock() ;
              unit_m := (els) ⩤ unit_m;
              UnLock()
          );
public
        GetFileStat : FileName →o FileStat
        GetFileStat(nm) ≙
          (   Lock() ;
              def mk_(-, stat, -, -) = file_m(nm) in
              let fs = stat.GetStatus() in
              (   UnLock() ;
                  return fs
              )
          );
public
        GetFileStatSet : ModuleName →o FileStat-set
        GetFileStatSet(nm) ≙
          (   dcl file_s : FileName-set := {};
              Lock() ;
              for all fnm ∈ dom file_m
```

```
            do let mk_(mod_s, -, -, -) = file_m(fnm) in
               if nm ∈ mod_s
               then file_s := file_s ∪ {fnm};
            let st_s = {let mk_(-, ref, -, -) = file_m(fl) in
                        ref.GetStatus() | fl ∈ file_s} in
            (   UnLock();
                return st_s
            )
        );
public
    SetFileStatus : FileName × FileAction × Status →o B × B
    SetFileStatus(nm, action, stt) △
        (   dcl stats : B × B := mk_(false, false);
            Lock();
            if nm ∈ dom file_m
            then def mk_(-, stat, -, -) = file_m(nm) in
                 def filest : FileStatus = stat in
                 stats := filest.SetStatus(action, stt);
            UnLock();
            return stats
        );
public
    FileStatusAllowed : FileName × Action →o B
    FileStatusAllowed(nm, kind) △
        (   Lock();
            let mk_(-, stat, -, -) = file_m(nm) in
            let allowed = stat.Allowed(kind) in
            (   UnLock();
                return allowed
            )
        );
public
    GetUnitStat : ModuleName →o UnitStat
    GetUnitStat(nm) △
        (   Lock();
            def unit : VDMUnitElem = unit_m(nm) in
            let us = unit.GetStatus() in
            (   UnLock();
                return us
            )
        );
public
    SetUnitStatus : ModuleName × UnitAction × Status →o B
    SetUnitStatus(nm, action, stt) △
        (   dcl stat : B := false;
```

```
            Lock() ;
            if nm ∈ dom unit_m
            then def unit : VDMUnitElem = unit_m (nm) in
                 stat := unit.SetStatus (action, stt);
            UnLock() ;
            return stat
        );
public
    UnitStatusAllowed : ModuleName × Action →o B
    UnitStatusAllowed (nm, kind) △
        (   Lock() ;
            def unit : VDMUnitElem = unit_m (nm) in
            let allowed = unit.Allowed (kind) in
            (   UnLock() ;
                return allowed
            )
        );
public
    GetMediator : () →o ToolMediator
    GetMediator () △
        return mediator;
public
    DisableUnit : ModuleName →o B
    DisableUnit (nm) △
        (   Lock() ;
            def unit : VDMUnitElem = unit_m (nm) in
            let stat = unit.Disable () in
            (   UnLock() ;
                return stat
            )
        );
public
    EnableUnit : ModuleName × AstVal × Status →o B
    EnableUnit (nm, ast, sc) △
        (   Lock() ;
            if nm ∉ dom unit_m
            then unit_m := unit_m ⩊ {nm ↦ new VDMUnitElem ()};
            (   dcl unit : VDMUnitElem := unit_m (nm);
                def changeSet = unit.Set (ast);
                    changeEnab = if sc = OK
                                 then unit.Enable ()
                                 else false in
                (   UnLock() ;
                    return changeSet ∨ changeEnab
                )
            )
        );
```

For the RTF parser we need a temporary file that will exist in the current project. When project is changed or the toolbox quit-ed, the temporary file must be removed from the file system.

The RTF parser will call AddTempFile to get the temporary file name. This operation creates a new temporary file if one is not already in the file info map. In the GUI, the file opener must check if a temporary file exists and in that case, open this file instead of the original rtf file. This is achieved by calling GetTempFileName.

```
public
        AddTempFile : FileName ->o TmpFileName
        AddTempFile (filenm) ≜
          (   self.AddSetOfFiles({filenm}) ;
              Lock() ;
              def mk_(mod_s, fs, fid, tfile) = file_m (filenm) in
              let tmpfile = if tfile = nil
                            then CreateTempFile ()
                            else tfile in
              (   file_m := file_m † {filenm ↦ mk_(mod_s, fs, fid, tmpfile)};
                  UnLock() ;
                  return tmpfile
              )
          );
public
        GetTempFileName : FileName ->o TmpFileName
        GetTempFileName (filenm) ≜
          (   dcl tnm : TmpFileName := mk_FileName ("");
              Lock() ;
              if filenm ∈ dom file_m
              then let mk_(-, -, -, tmpfile) = file_m (filenm) in
                   if tmpfile ≠ nil
                   then tnm := tmpfile;
              UnLock() ;
              return tnm
          );
public
        GetRealFileName : TmpFileName ->o FileName
        GetRealFileName (tfilenm) ≜
          (   dcl rnm : FileName := mk_FileName ("");
              Lock() ;
              if tfilenm ∈ dom file_m
              then rnm := tfilenm
              else let realf_s = {rf | rf ∈ dom file_m · let mk_(-, -, -, tf) = file_m (rf) in
                                                        tf = tfilenm} in
                   if card realf_s = 1
                   then let {real_name} = realf_s in
                        rnm := real_name;
              UnLock() ;
              return rnm
          );
public
        RemTempFiles : () ->o ()
        RemTempFiles () ≜
          (   Lock() ;
              for all fn ∈ dom file_m
```

```
                do let mk_(-,-,-,tmpfile) = file_m(fn) in
                   if tmpfile ≠ nil
                   then self.RemoveTempFile(tmpfile) ;
                UnLock()
             );
public
          CreateTempFile : () ->o TmpFileName
          CreateTempFile () △
            is not yet specified;
public
          RemoveTempFile : TmpFileName ->o ()
          RemoveTempFile (tmpnm) △
            is not yet specified;
public
          SaveFiles : StateStore ->o B
          SaveFiles (store) △
            (   Lock() ;
                let ok = store.WriteNames(dom file_m) in
                (   UnLock() ;
                    return ok
                )
            )
functions
          IsJavaFile : FileName->B
          IsJavaFile (fn) △
            fn.nm(len fn.nm - 4, ..., len fn.nm) = ".java"
operations
public
          LoadFiles : StateStore ->o B
          LoadFiles (store) △
            def file_s = store.ReadNames() in
            if file_s ≠ nil
            then (   dcl vdm_file_l : FileName* := [],
                         java_file_l : FileName* := [];
                     for all f ∈ file_s
                     do if IsJavaFile(f)
                        then java_file_l := java_file_l ^ [f]
                        else vdm_file_l := vdm_file_l ^ [f];
                     self.AddSetOfFiles(file_s) ;
                     def - = mediator.GetBaseTools().SyntaxCheck(vdm_file_l);
                         - = mediator.GetBaseTools().JavaSyntaxCheck(java_file_l) in
                     return true
                 )
            else return false ;
public
          Init : () ->o ()
          Init () △
            is not yet specified;
public
```

```
        Finish : () ->o ()
        Finish () ≜
          is not yet specified;
public
        Lock : () ->o ()
        Lock () ≜
          is not yet specified;
public
        UnLock : () ->o ()
        UnLock () ≜
          is not yet specified;
public
        nameIsDefined : ModuleName | FileName ->o B
        nameIsDefined (nm) ≜
          (   Lock() ;
              let ok = cases true :
                          (is_FileName (nm)) -> nm ∈ dom file_m,
                          (is_ModuleName (nm)) -> nm ∈ dom unit_m,
                          others -> false
                      end in
              (   UnLock() ;
                  return ok
              )
          )
end
RepDatabase
```

3.5.2 Class VDMUnitElem

```
class
VDMUnitElem is subclass of ProjectTypes
instance variables
        ast : AST := new AST ();
        status : UnitStatus := new UnitStatus ();
        enabled : B := true;

operations
public
        Enable : () ->o B
        Enable () ≜
          def changedEnable = ¬ enabled in
          (   enabled := true;
              if changedEnable
              then def stat = status.GetStatus () in
                     return stat ≠ mk_UnitStat (NONE, NONE, NONE)
              else return false
          );
public
```

```
        Disable : () →o 𝔹
        Disable () ≜
          def changedEnable = enabled in
          (   enabled := false;
              if changedEnable
              then def stat = status.GetStatus () in
                     return stat ≠ mk_UnitStat (NONE, NONE, NONE)
              else return false
          );
```

Enable and *Disable* return true if the enabling changes and the status is different from the disabled status (disabled status is *mk-UnitStat*(< *NONE* >, < *NONE* >, < *NONE* >)).

```
public
        IsEnabled : () →o 𝔹
        IsEnabled () ≜
          return enabled;
public
        GetStatus : () →o UnitStat
        GetStatus () ≜
          if enabled
          then status.GetStatus()
          else return mk_UnitStat (NONE, NONE, NONE) ;
public
        SetStatus : UnitAction × Status →o 𝔹
        SetStatus (action, tp) ≜
          def changed = status.SetStatus (action, tp) in
          return enabled ∧ changed;
public
        Allowed : UnitAction →o 𝔹
        Allowed (kind) ≜
          if enabled
          then status.Allowed(kind)
          else return false ;
public
        Get : () →o AstVal
        Get () ≜ ast.
          Get()
        pre enabled ;
public
        Set : AstVal →o 𝔹
        Set (ast′) ≜
          def changed = self.SetStatus (TYPECHECK, NONE) in
          (   ast.Set(ast′) ;
              return changed
          );
public
        Delete : () →o ()
        Delete () ≜ ast.
          DeleteAST() ;
public
```

```
        Save : StateStore →o B
        Save (store) △
          def ast_ok = ast.Save (store);
              status_ok = status.Save (store);
              ena_ok = store.WriteVal (enabled) in
          return ast_ok ∧ status_ok ∧ ena_ok;
public
        Load : StateStore →o B
        Load (store) △
          def ast_ok = ast.Load (store);
              status_ok = status.Load (store);
              ena = store.ReadVal () in
          if ast_ok ∧ status_ok ∧ is_B (ena)
          then (   enabled := ena;
                   return true
                )
          else (   ast := new AST ();
                   status := new UnitStatus ();
                   enabled := false;
                   return true
                )
end
VDMUnitElem
```

3.5.3 Class UnitStatus

```
class
UnitStatus is subclass of ProjectTypes
instance variables
        status : UnitStat := mk_UnitStat (NONE, NONE, NONE);
```

status in the objectstate is status for typechecking and code generation respectively.

```
operations
public
        GetStatus : () →o UnitStat
        GetStatus () △
          return status;
public
        SetStatus : UnitAction × (Status | POS | DEF) →o B
        SetStatus (action, tp) △
          def oldstat = status in
           cases action:
             TYPECHECK → (   status := mk_UnitStat (tp, NONE, NONE);
                             return oldstat ≠ status
                         ),
```

```
    CG → def mk_UnitStat(curTC, -, curJCG) = status in
           let newCG = if ¬ tcOk(curTC)
                       then NONE
                       else tp in
           (   status := mk_UnitStat(curTC, newCG, curJCG);
               return oldstat ≠ status
           ),
    JCG → def mk_UnitStat(curTC, curCG, -) = status in
            let newJCG = if ¬ tcOk(curTC)
                         then NONE
                         else tp in
            (   status := mk_UnitStat(curTC, curCG, newJCG);
                return oldstat ≠ status
            ),
    others → return false
  end;
```

SetStatus updates the status of its associated *VDMUnitElem*. The return value indicates whether the status of the module has changed:

true: the status has changed,

false: the status is unchanged.

```
public
        Allowed : UnitAction →o B
        Allowed(kind) △
          return cases kind :
                   TYPECHECK → true,
                   CG → status.type = POS,
                   JCG → status.type = POS,
                   others → false
                 end;
public
        Save : StateStore →o B
        Save(store) △
          def ok1 = store.WriteStatus(status.type);
              ok2 = store.WriteStatus(status.cg);
              ok3 = store.WriteStatus(status.javaCg) in
          return ok1 ∧ ok2 ∧ ok3;
public
        Load : StateStore →o B
        Load(store) △
          def type = store.ReadStatus();
              cg = store.ReadStatus();
              jcg = store.ReadStatus() in
          if type ≠ nil ∧ cg ≠ nil
          then (   status := mk_UnitStat(type, cg, jcg);
                   return true
                )
```

```
                else (    status := mk_UnitStat (NONE, NONE, NONE);
                          return false
                      )
functions
                tcOk : Status | POS | DEF→ℬ
                tcOk (t) ≜
                  t = OK ∨ t = POS ∨ t = DEF
end
UnitStatus
```

3.5.4 Class AST

```
class
AST is subclass of ProjectTypes
types
                 LocationType = MEMORY | NONE
instance variables
                location : LocationType := NONE;
                ast : [AstVal] := nil ;
                inv location = MEMORY  ⇒
                    ast ≠ nil  ∧
                    location = NONE  ⇒
                    ast = nil

operations
public
                Get : () →o AstVal
                Get () ≜
                  return ast
                pre  location ≠ NONE ;
public
                DeleteAST : () →o ()
                DeleteAST () ≜
                  (    location := NONE;
                       ast := nil
                  );
public
                Set : AstVal →o ()
                Set (ast′) ≜
                   cases  location:
                     MEMORY → ast := ast′,
                     NONE → (    ast := ast′;
                                 location := MEMORY
                             )
                  end;
public
                Same : AstVal →o ℬ
                Same (ast′) ≜
                  return false;
```

Same is used to compare two AST's. Two AST's are considered to be the same if they are structural equal. That is, if the two ASTs represent the same specification where only the position and type information are different. *Same* should be used to decide the status of a class after syntax checking.

```
public
        Save : StateStore ->o B
        Save (store) ≜
          if location = NONE
          then store.WriteASTVal(location)
          else store.WriteASTVal(ast) ;
public
        Load : StateStore ->o B
        Load (store) ≜
          def val = store.ReadASTVal () in
          (   if is_AstVal (val)
              then (   ast := val;
                       location := MEMORY
                   )
              else (   ast := nil ;
                       location := NONE
                   ) ;
              return val ≠ nil
          )
end
AST
```

3.5.5 Class FileStatus

```
class
FileStatus is subclass of ProjectTypes
instance variables
        status : FileStat := mk_FileStat (NONE, NONE);
        timestamp : N := 0;
        modified : B := false;
```

The *modified* flag signals the correspondance between the parsed file and the file on the file system, if FALSE the file on the file system is unmodified compared to the time of parsing, if TRUE the file on the file system is newer then the parsed file.

```
operations
public
        GetStatus : () ->o FileStat
        GetStatus () ≜
          return status;
public
```

$$
\begin{array}{l}
SetStatus : FileAction \times Status \xrightarrow{o} \mathbb{B} \times \mathbb{B} \\
SetStatus\,(action, tp) \triangleq \\
\quad \mathsf{def}\ oldstat = status\ \mathsf{in} \\
\quad (\quad \mathsf{cases}\ action: \\
\qquad \mathrm{EDIT} \to (\quad status := \mathsf{mk\_}FileStat\,(\mathrm{NONE}, \mathrm{NONE}); \\
\qquad\qquad\qquad\quad \mathsf{let}\ changedStatus = oldstat \neq status\ \mathsf{in} \\
\qquad\qquad\qquad\quad \mathsf{return}\ \mathsf{mk\_}(\mathsf{true}, changedStatus) \\
\qquad\qquad\quad), \\
\qquad \mathrm{SYNTAXCHECK} \to (\quad status := \mathsf{mk\_}FileStat\,(tp, \mathrm{NONE}); \\
\qquad\qquad\qquad\qquad\quad \mathsf{let}\ changedStatus = oldstat \neq status\ \mathsf{in} \\
\qquad\qquad\qquad\qquad\quad \mathsf{return}\ \mathsf{mk\_}((tp \neq \mathrm{OK}), changedStatus) \\
\qquad\qquad\qquad\quad), \\
\qquad \mathrm{PP} \to (\quad status := \mathsf{mk\_}FileStat\,(\mathrm{OK}, tp); \\
\qquad\qquad\qquad\quad \mathsf{let}\ changedStatus = oldstat \neq status\ \mathsf{in} \\
\qquad\qquad\qquad\quad \mathsf{return}\ \mathsf{mk\_}(\mathsf{false}, changedStatus) \\
\qquad\qquad\quad) \\
\qquad \mathsf{end}\ ; \\
\qquad \mathsf{return}\ \mathsf{mk\_}(\mathsf{false}, \mathsf{false}) \\
\quad);
\end{array}
$$

SetStatus updates the status of its associated file. The first field of the return value indicates whether the file must update its associated units:

- *mk-*(*false*, -) :enable units in file, delete units not longer in file
- *mk-*(*true*, -): disable all units

whereas the second field indicates that the status of the file has changed:

- *mk-*(-, *true*): the status has changed,
- *mk-*(-, *false*): the status is unchanged.

public

$$
\begin{array}{l}
SetTimestamp : \mathbb{N} \xrightarrow{o} \mathbb{B} \\
SetTimestamp\,(t) \triangleq \\
\quad \mathsf{let}\ oldtime = timestamp\ \mathsf{in} \\
\quad (\quad timestamp := t; \\
\qquad modified := \mathsf{false}; \\
\qquad \mathsf{return}\ timestamp > oldtime \wedge oldtime > 0 \\
\quad);
\end{array}
$$

SetTimestamp sets the timestamp for the associated file. If the file was modified on the filesystem, *modified* is set to FALSE. The return value indicates whether the *modified* flag has changed:

- TRUE: The file modification status is changed and should be updated in the interface,
- FALSE: The file modification status is not changed, and update in the interface is not necessary.

SetTimestamp is called from class *RepDatabase*.

```
public
        GetTimestamp : () ->o ℕ
        GetTimestamp () ≜
          return timestamp;
public
        IsModified : () ->o 𝔹
        IsModified () ≜
          return modified;
public
        SetModified : () ->o ()
        SetModified () ≜
          modified := true;
public
        Allowed : Action ->o 𝔹
        Allowed (kind) ≜
          (     cases kind:
                  EDIT,
                  SYNTAXCHECK → return true,
                  PP → return status.syntax = OK,
                  others → return false
                end
          );
public
        Save : StateStore ->o 𝔹
        Save (store) ≜
          def ok1 = store.WriteStatus (status.syntax);
              ok2 = store.WriteStatus (status.pp) in
          return ok1 ∧ ok2;
public
        Load : StateStore ->o 𝔹
        Load (store) ≜
          def syntax = store.ReadStatus ();
              pp = store.ReadStatus () in
          if syntax ≠ nil ∧ pp ≠ nil
          then (    status := mk_FileStat (syntax, pp);
                    return true
                )
          else (    status := mk_FileStat (NONE, NONE);
                    return false
                )
end
FileStatus
```

3.6 Status Info for Files and Modules/Classes

The StatusInfo class extracts status and allowed actions for files and modules/classes.

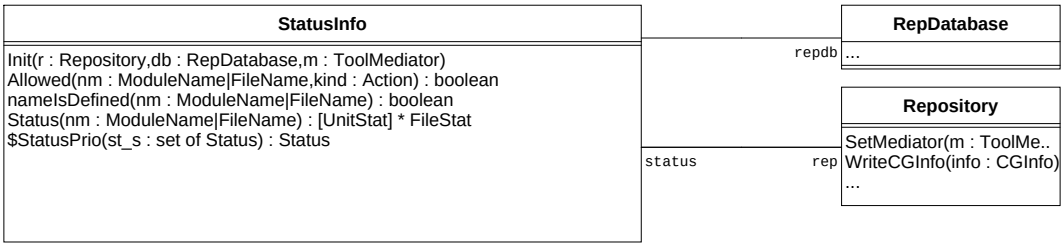

Figure 8: Interface to StatusInfo class

3.6.1 Class StatusInfo

class
$StatusInfo$ is subclass of $ToolColleague$
instance variables

$$rep : Repository;$$
$$repdb : RepDatabase;$$

Note: rep is not used in this version
operations
public

$$Init : Repository \times RepDatabase \times ToolMediator \xrightarrow{o} ()$$
$$Init\,(r, db, m) \triangleq$$
$$(\quad rep := r;$$
$$\quad repdb := db;$$
$$\quad mediator := m$$
$$);$$

Methods handling status and values (asts)
public

$$Allowed : (ModuleName \mid FileName) \times Action \xrightarrow{o} \mathbb{B} \times \mathsf{char}^*$$
$$Allowed\,(nm, kind) \triangleq$$
$$\mathsf{def}\ ok = repdb.nameIsDefined\,(nm)\ \mathsf{in}$$
$$\mathsf{cases}\ \ \mathsf{true}:$$
$$(\mathsf{is\_}FileName\,(nm)) \rightarrow \mathsf{if}\ ok$$
$$\mathsf{then\ def}\ stallow = repdb.FileStatusAllowed\,(nm, kind)$$
in
$$\mathsf{return\ mk\_}(stallow, \texttt{""})$$
$$\mathsf{elseif}\ kind = \mathrm{SYNTAXCHECK}$$
$$\mathsf{then\ return\ mk\_}(\mathsf{true}, \texttt{""})$$
$$\mathsf{else\ return\ mk\_}(\mathsf{false}, \texttt{"}File\ \texttt{"}\curvearrowright nm.nm\curvearrowright\texttt{"}\ not\ defined\texttt{"})$$
,

```
                (is_ModuleName (nm)) → if ok
                                       then def unallow = repdb.UnitStatusAllowed (nm, kind)
in
                                            return mk_ (unallow, "")
                                       else return mk_ (false, "Name "⌢nm.nm⌢" not defined")
,
                others → return mk_ (false, "Name not FileName or ModuleName")
            end;
```

Allowed returns with a tuple consisting of a boolean value and a message in a sequence of char. If the boolean value is *true*, the action is allowed. If the value is *false*, the returned sequence of char can be empty or not empty. If empty this means that the action is not allowed. If not empty this indicates that an error has occured and the string contains a message about this. This will happen if *Allowed* is called with an unknown name. However, syntax check is always allowed, also for filenames not in repository. The returned sequence of char is generated by the call to *NameIsDefined*.

```
public
        Status : ModuleName | FileName →o [UnitStat] × FileStat
        Status (nm) △
          def ok = repdb.nameIsDefined (nm) in
           cases true:
            (is_ModuleName (nm) ∧ ok) → def us : UnitStat = repdb.GetUnitStat (nm);
                                              st_s : FileStat-set = repdb.GetFileStatSet (nm)
in
                                        (   dcl sc_s : Status-set := {},
                                                pp_s : Status-set := {};
                                            for all mk_FileStat (sc, pp) ∈ st_s
                                            do (   sc_s := sc_s ∪ {sc};
                                                   pp_s := pp_s ∪ {pp}
                                               ) ;
                                            let sum_sc = StatusPrio (sc_s) in
                                            let sum_pp = if sum_sc = OK
                                                         then StatusPrio (pp_s)
                                                         else NONE in
                                            return mk_ (us, mk_FileStat (sum_sc, sum_pp))
                                        ),
            (is_FileName (nm) ∧ ok) → def fs : FileStat = repdb.GetFileStat (nm) in
                                          return mk_ (nil , fs),
            others → return mk_ (nil , mk_FileStat (ERROR, NONE))
          end
```

Status returns a tuple consisting of status for the module and status for the corresponding file. If called with a filename the status for module is nil. Run-time error if called with an unknown name.

If called with a module name a set of file names is created consisting of the names of the files the module is defined in. This is a generel solution - in case of a structured specification the set will only hold one element and in case of a flat specification there can be more than one element. The file status of a module is then computed as the most fundamental status of all involved files.

functions
$StatusPrio : Status\text{-set} \tilde{\rightarrow} Status$
$StatusPrio\,(st\_s) \triangleq$
if $\mathrm{NONE} \in st\_s$
then NONE
elseif $\mathrm{ERROR} \in st\_s$
then ERROR
else OK
operations
public
$IsAllTypeCorrect : () \xrightarrow{o} \mathbb{B}$
$IsAllTypeCorrect\,() \triangleq$
(def $module\_s = repdb.AllModules\,()$ in
return $\forall\, m \in module\_s \cdot$ let mk_$UnitStat\,(tc, \text{-}, \text{-}) = repdb.GetUnitStat\,(m)$ in
$(tc = \mathrm{OK} \vee tc = \mathrm{POS} \vee tc = \mathrm{DEF})$
);
public
$IsSyntaxCorrect : FileName \xrightarrow{o} \mathbb{B}$
$IsSyntaxCorrect\,(nm) \triangleq$
if $repdb.nameIsDefined\,(nm)$
then let mk_$FileStat\,(syntax, \text{-}) = repdb.GetFileStat\,(nm)$ in
return $syntax = \mathrm{OK}$
else return false
end
StatusInfo

3.7 Project Handling

Project handling concerns creation of a new project and opening an existing project. Furthermore must it be possible to load and save the state of a project.

3.7.1 Class UpdateProject

class
UpdateProject is subclass of *ToolColleague*
instance variables
$projectname : [FileName] := \mathrm{nil}$;
$rep : Repository$;
$repdb : RepDatabase$;
$dep : Dependency$;

projectname is the name of the current repository (*nil* if no repository is open).
operations
public

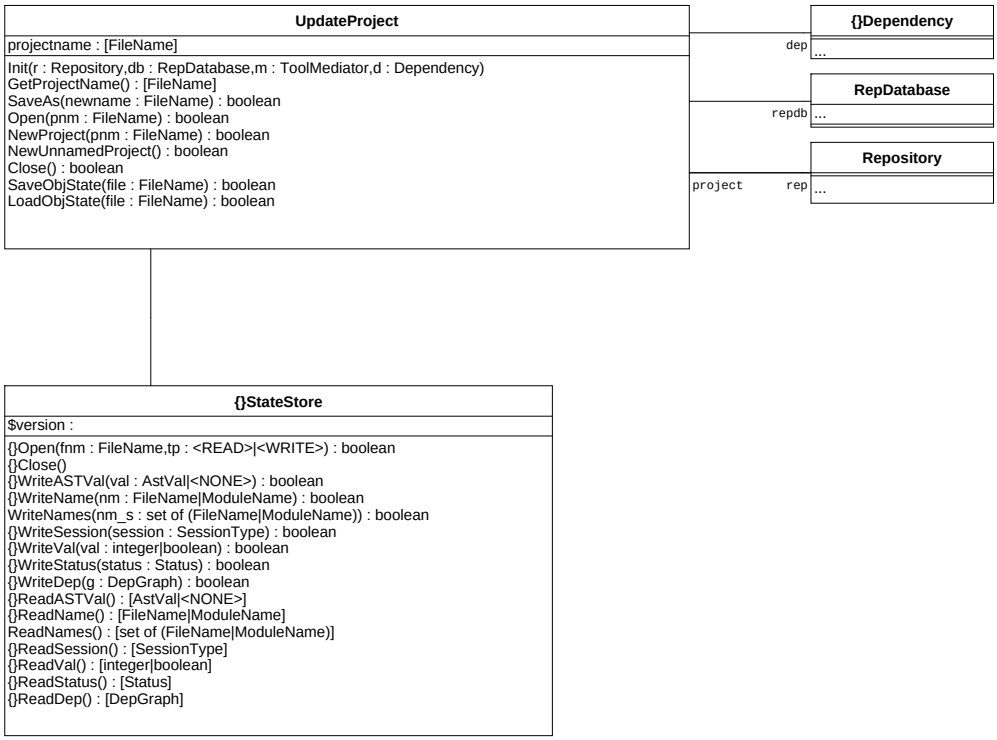

Figure 9: Interface to UpdateProject and StateStore classes.

```
Init : Repository × RepDatabase × ToolMediator × Dependency ^o→ ()
Init (r, db, m, d) ≜
  (   rep := r;
      repdb := db;
      mediator := m;
      dep := d
  );
```

Open, Close, Save and New Project

public

```
GetProjectName : () ^o→ [FileName]
GetProjectName () ≜
  return projectname;
```

public

```
SaveAs : FileName ^o→ 𝔹
SaveAs (newname) ≜
  def ok = self.SaveObjState (newname) in
  (   if ok
      then self.UpdateProjName(newname) ;
      return ok
  );
```

public

```
Open : FileName ^o→ 𝔹
Open (pnm) ≜
  (   rep.ClearAll() ;
      def open_ok = self.LoadObjState (pnm) in
      (   if open_ok
          then (   self.UpdateProjName(pnm) ;
                   mediator.GetBaseTools ( ) .UpdateToolbox()
               )
          else (   rep.ClearAll() ;
                   self.UpdateProjName(nil )
               ) ;
          return open_ok
      )
  );
```

public

```
NewUnnamedProject : () ^o→ 𝔹
NewUnnamedProject () ≜
  (   rep.ClearAll() ;
      self.UpdateProjName(nil ) ;
      mediator.GetBaseTools ( ) .UpdateToolbox() ;
      return true
  );
```

public

$UpdateProjName : [FileName] \xrightarrow{o} ()$
$UpdateProjName\,(newname) \triangleq$
(if $projectname \neq newname$
then $mediator.CallBack(\mathsf{mk\_}ChangedProjName\,(projectname, newname))$;
$projectname := newname$
);

Save and Load of State

public
$SaveObjState : FileName \xrightarrow{o} \mathbb{B}$
$SaveObjState\,(file) \triangleq$
(dcl $store : StateStore := \mathsf{new}\ StateStore\,()$,
$ok : \mathbb{B}$;
$ok := store.Open\,(file, \mathrm{WRITE})$;
if $\neg\, ok$
then return false ;
def $ok' = repdb.SaveFiles\,(store)$ in
$ok := ok \wedge ok'$;
$store.Close()$;
$rep.SetSavedFileState(ok)$;
return ok
);
public
$LoadObjState : FileName \xrightarrow{o} \mathbb{B}$
$LoadObjState\,(file) \triangleq$
(dcl $store : StateStore := \mathsf{new}\ StateStore\,()$,
$ok : \mathbb{B}$;
$ok := store.Open\,(file, \mathrm{READ})$;
if $\neg\, ok$
then return false ;
def $ok' = repdb.LoadFiles\,(store)$ in
$ok := ok \wedge ok'$;
$store.Close()$;
$rep.SetSavedFileState(ok)$;
return ok
)
end
UpdateProject

3.7.2 Project File Syntax, version 2

```
projectFile = prj_id, version, modules, files, session, dep_graph, tc_env;
prj_id      = 'ProjectFile' | 'ProjectFilePP';
version     = '2';
modules     = N, module{0..N}
module      = name, ast, mod_status;
mod_status  = tc_status, cg_status;

files       = N, file{0..N};
file        = name, mod_of_file, file_status, fileid;
mod_of_file = N, name{0..N};
file_status = sc_status, pp_status;
```

3.7.3 Project File Syntax, version 3

```
projectFile = prj_id, version, files;
prj_id      = 'ProjectFile' | 'ProjectFilePP';
version     = '3';

files       = N, file{0..N};
file        = name
```

3.7.4 Class StateStore

class
StateStore is subclass of *ProjectTypes*
values

$$version = 3$$

version identifies the current version of the project file. This number is used to convert different repository versions (done in open project).
instance variables

$$content : prjFileType \times vers \times files;$$
$$format : prjFileFormat;$$
$$rw : \text{READ} \mid \text{WRITE};$$

types

$$prjFileType = \text{PROJECTFILE} \mid \text{PROJECTFILEPP};$$
$$prjFileFormat = (prjFileType \mid vers \mid \mathbb{Z} \mid FileName)^{*};$$
$$vers = \mathbb{Z};$$
$$files = FileName\text{-set}$$

operations

public

$Open : FileName \times (\text{READ} \mid \text{WRITE}) \xrightarrow{o} \mathbb{B}$
$Open\,(fnm, tp) \triangleq$
  is not yet specified;

public

$Close : () \xrightarrow{o} ()$
$Close\,() \triangleq$
  is not yet specified;

public

$WriteASTVal : AstVal \mid \text{NONE} \xrightarrow{o} \mathbb{B}$
$WriteASTVal\,(val) \triangleq$
  is not yet specified;

public

$WriteName : FileName \mid ModuleName \xrightarrow{o} \mathbb{B}$
$WriteName\,(nm) \triangleq$
  is not yet specified;

public

$WriteNames : (FileName \mid ModuleName)\text{-set} \xrightarrow{o} \mathbb{B}$
$WriteNames\,(nm\_s) \triangleq$
  (dcl $ok : \mathbb{B} := \text{self}.WriteVal\,(\text{card } nm\_s)$;
    for all $nm \in nm\_s$
    do def $ok' = \text{self}.WriteName\,(nm)$ in
      $ok := ok \wedge ok'$;
    return ok
  );

public

$WriteSession : SessionType \xrightarrow{o} \mathbb{B}$
$WriteSession\,(session) \triangleq$
  is not yet specified;

public

$WriteVal : \mathbb{Z} \mid \mathbb{B} \xrightarrow{o} \mathbb{B}$
$WriteVal\,(val) \triangleq$
  is not yet specified;

public

$WriteStatus : Status \xrightarrow{o} \mathbb{B}$
$WriteStatus\,(status) \triangleq$
  is not yet specified;

public

$WriteDep : DepGraph \xrightarrow{o} \mathbb{B}$
$WriteDep\,(g) \triangleq$
  is not yet specified;

public

$WriteTCEnv : TCEnv \xrightarrow{o} \mathbb{B}$
$WriteTCEnv\,(e) \triangleq$
  is not yet specified;

public

$ReadASTVal : () \xrightarrow{o} [AstVal \mid \text{NONE}]$
$ReadASTVal\,() \triangleq$
  is not yet specified;

public

```
                    ReadName : () →o [FileName | ModuleName]
                    ReadName () △
                      is not yet specified;
public
                    ReadNames : () →o [(FileName | ModuleName)-set]
                    ReadNames () △
                      (   dcl nm_s : (FileName | ModuleName)-set := {};
                          def size = self.ReadVal () in
                          if is_ℕ (size)
                          then for i = 1 to size
                                 do def nm = self.ReadName () in
                                    if is_FileName (nm) ∨ is_ModuleName (nm)
                                    then nm_s := nm_s ∪ {nm}
                                    else return nil
                          else return nil  ;
                          return nm_s
                      );
public
                    ReadSession : () →o [SessionType]
                    ReadSession () △
                      is not yet specified;
public
                    ReadVal : () →o [ℤ | 𝔹]
                    ReadVal () △
                      is not yet specified;
public
                    ReadStatus : () →o [Status]
                    ReadStatus () △
                      is not yet specified;
public
                    ReadDep : () →o [DepGraph]
                    ReadDep () △
                      is not yet specified;
public
                    ReadTCEnv : () →o [TCEnv]
                    ReadTCEnv () △
                      is not yet specified
end
StateStore
```

3.8 Session Handling

The session handling classes are a *state* pattern ([EJ95]) with the UpdateSes as super class. This class contains methods common to all subclasses. Only the behaviour of the syntax checker depends on the type of session, i.e. whether it is a flat or structured specification.

Figure 10: Interface and inheritance for UpdateSes classes.

3.8.1 Class UpdateSes

class
$UpdateSes$ is subclass of $ProjectTypes$
operations
public

$$Type : () \xrightarrow{o} ProjectTypes`SessionType$$
$$Type\,() \triangleq$$
is subclass responsibility;

public

$$UpdateSC : Repository \times RepDatabase \times FileName \times ((Module^*) \mid FlatSpec) \xrightarrow{o} ()$$
$$UpdateSC\,(rep, repdb, nm, spec) \triangleq$$
is subclass responsibility;

public

$$UpdateTC : RepDatabase \times ModuleName \times [\mathrm{POS} \mid \mathrm{DEF}] \xrightarrow{o} ()$$
$$UpdateTC\,(repdb, nm, tcType) \triangleq$$
let $stt =$ if $tcType \neq$ nil
then $tcType$
else ERROR,

```
                    changed = repdb.SetUnitStatus (nm, TYPECHECK, stt) in
                  if changed
                  then repdb.GetMediator ( ) .CallBack(mk_ChangedModuleStatus ({nm}));
public
                  UpdateCG : RepDatabase × ModuleName × (CPP | JAVA) × Succes ^o→
()
                  UpdateCG (rep, nm, kind, suc) ≜
                    let stt = if suc
                              then OK
                              else ERROR,
                        changed = if kind = CPP
                                  then rep.SetUnitStatus (nm, CG, stt)
                                  else rep.SetUnitStatus (nm, JCG, stt) in
                    if changed
                    then rep.GetMediator ( ) .CallBack(mk_ChangedModuleStatus ({nm}));
public
                  UpdatePP : RepDatabase × FileName × Succes ^o→ ()
                  UpdatePP (rep, nm, suc) ≜
                    let stt = if suc
                              then OK
                              else ERROR,
                        mk_(-, changedStatus) = rep.SetFileStatus (nm, PP, stt) in
                    if changedStatus
                    then rep.GetMediator ( ) .CallBack(mk_ChangedFileStatus (nm));
public
                  DisableUnits : RepDatabase × ModuleName-set ^o→ ModuleName-set
                  DisableUnits (rep, un_s) ≜
                    (   dcl changedMod_s : ModuleName-set := {};
                        for all un ∈ un_s
                        do def changed = rep.DisableUnit (un) in
                           if changed
                           then changedMod_s := changedMod_s ∪ {un};
                        return changedMod_s
                    );
```

DisableUnits is called both from flat and structured specifications. It disables all units, and returns with a set of module names that have changed their status.

```
public
                  Disable : () ^o→ ()
                  Disable () ≜
                    skip;
public
                  Enable : () ^o→ ()
                  Enable () ≜
                    skip
end
UpdateSes
```

3.8.2 Class NoneSes

class
NoneSes is subclass of *UpdateSes*
instance variables
disabled : $\mathbb{B}$:= false;

The *disabled* is dealing with change of session type in current project. Is set by *Disable* and *Enable* methods, called from the *Repository*.

```
operations
public
                    Type : () ==> ProjectTypes`SessionType
                    Type () ==
                      return if disabled
                             then DISABLED
                             else NONE;
public
                    Disable : () ==> ()
                    Disable () ==
                      disabled := true;
public
                    Enable : () ==> ()
                    Enable () ==
                      disabled := false;
public
                    UpdateSC : Repository * RepDatabase * FileName * ((Module*) |
FlatSpec) ==> ()
                    UpdateSC (rep, rdb, nm, spec) ==
                      if spec <> []
                      then (    dcl nextses : FlatSes | StructSes := if is_FlatSpec (spec)
                                                                      then new FlatSes ()
                                                                      else new StructSes ();
                                rep.SetSession(nextses) ;
                                rep.UpdateSC(nm, spec)
                           )
                      else (    rdb.AddSetOfFiles({nm}) ;
                                def mod_s = rdb.ModulesInFile (nm) in
                                def mk_(-, changedFileStat) = rdb.SetFileStatus (nm, SYNTAXCHECK, E
in
                                def changedModStat = self.DisableUnits (rdb, mod_s) in
                                (    if changedFileStat
                                     then rdb.GetMediator ( ) .CallBack(mk_ChangedFileStatus (nm));
                                     if changedModStat <> {}
                                     then rdb.GenCallBack(changedModStat)
                                )
                           )
end
NoneSes
```

3.8.3 Class StructSes

```
class
StructSes is subclass of UpdateSes
operations
public
                    Type : () ^o→ ProjectTypes`SessionType
                    Type () ≜
                      return STRUCTURED;
public
                    UpdateSC : Repository × RepDatabase × FileName × ((Module*) |
FlatSpec) ^o→ ()
                    UpdateSC (rep, repdb, nm, spec) ≜
                      (   dcl changedModStat : ModuleName-set := {};
                          let suc = if spec = []
                                    then ERROR
                                    else OK in
                          (   repdb.AddSetOfFiles({nm}) ;
                              repdb.StoreState() ;
                              def old_s = repdb.ModulesInFile (nm) in
                              def mk_(disable, changedStatus) = repdb.SetFileStatus (nm, SYNTAXCHE
in
                              (   dcl ast_m : ModuleName ^m→ AstVal := {↦};
                                  for mk_Module (mnm, ast) in reversespec
                                  do if mnm ∈ dom ast_m
                                     then repdb.GetMediator    ( )  .Log(mnm.nm ↷
" multiple defined in file " ↷ nm.nm)
                                     else ast_m := ast_m † {mnm ↦ ast};
                                  let new_s = dom ast_m in
                                  (   if disable
                                      then changedModStat := self.DisableUnits (repdb, old_s)
                                      else (   def modules = repdb.AllModules () in
                                               let multdefs = (new_s \ old_s) ∩ modules in
                                               if multdefs ≠ {}
                                               then self.MultDefMod(repdb, multdefs) ;
                                               changedModStat := self.EnableUnits (repdb, ast_m);
                                               repdb.UnitDomResBy(old_s \ new_s) ;
                                               repdb.SetModulesInFile(nm, new_s)
                                           ) ;
                                      let toremove = old_s \ new_s in
                                      if toremove ≠ {}
                                      then rep.GetDependency    ( ) .Remove(toremove);
                                      if (nm.nm (len nm.nm−4, . . . , len nm.nm) ≠ ".java")
                                      then (   rep.GetDependency    ( ) .UpdateGraph(spec);
                                               def subc = rep.GetDependency ().GetAllSubClasses (new
```

```
                                                usedby = rep.GetDependency().GetAllUsedBy(new_
in
                                                def changedDepUnits = self.DependantUnits(repdb, (sub
usedby) \ new_s) in
                                                changedModStat := changedModStat∪changedDepUnits
                                            )
                                        ) ;
                                        repdb.GenCallBack(changedModStat) ;
                                        if changedStatus
                                        then repdb.GetMediator ( ) .CallBack(mk_ChangedFileStatus(nm))
                                    )
                                )
                            );
public
                    EnableUnits : RepDatabase×ModuleName -m-> AstVal -o-> ModuleName-set
                    EnableUnits(repdb, ast_m) ≜
                      ( dcl changedMod_s : ModuleName-set := {};
                        for all un ∈ dom ast_m
                        do def change = repdb.EnableUnit(un, ast_m(un), OK) in
                           if change
                           then changedMod_s := changedMod_s ∪ {un};
                        return changedMod_s
                      );
```

EnableUnits is only called by structured specifications. It enables all units and sets the corresponding ast's. It returns with a set of module names that have changed their status.

```
public
                    MultDefMod : RepDatabase × ModuleName-set -o-> ()
                    MultDefMod(repdb, md_s) ≜
                      for all md ∈ md_s
                      do def fnm_s = repdb.FileOfModule(md) in
                         ( let fnm =
                                     cases fnm_s :
                                       {nm} → "in file " ↷ nm.nm,
                                       others → "(no file information)"
                                     end in
                           repdb.GetMediator ( ) .Log(md.nm ↷ " " ↷ fnm ↷
" overwritten") ;
                           for all nm ∈ fnm_s
                           do let mod_s = repdb.ModulesInFile(nm) in
                              repdb.SetModulesInFile(nm, mod_s \ {md})
                         ) ;
public
                    DependantUnits : RepDatabase×ModuleName-set -o-> ModuleName-set
                    DependantUnits(repdb, cl_s) ≜
                      ( dcl changed : ModuleName-set := {};
                        for all cl ∈ cl_s
```

```
                    do def statusChanged = repdb.SetUnitStatus (cl, TYPECHECK, NONE)
in
                        if statusChanged
                        then changed := changed ∪ {cl};
                    return changed
                )
end
StructSes
```

3.8.4 Class FlatSes

```
class
FlatSes is subclass of UpdateSes
operations
public
                Type : () →o ProjectTypes`SessionType
                Type () △
                  return FLAT;
public
                UpdateSC : Repository × RepDatabase × FileName × ((Module*) |
FlatSpec) →o ()
                UpdateSC (rep, repdb, nm, spec) △
                  (   dcl changedModStat : ModuleName-set := {};
                      let suc = if spec = []
                                then ERROR
                                else OK in
                      (   repdb.AddSetOfFiles({nm}) ;
                          repdb.StoreState() ;
                          def old_s = repdb.ModulesInFile (nm) in
                          def mk_(disable, changedStatus) = repdb.SetFileStatus (nm, SYNTAXCHI
in
                          (   if is_FlatSpec (spec)
                              then repdb.SetModulesInFile(nm, {spec.nm}) ;
                              if disable
                              then changedModStat := self.DisableUnits (repdb, old_s)
                              else changedModStat := self.EnableFlat (rep, repdb, spec);
                              repdb.GenCallBack(changedModStat) ;
                              if changedStatus
                              then repdb.GetMediator ( ) .CallBack(mk_ChangedFileStatus (nm))
                          )
                      )
                  );
public
                EnableFlat : Repository×RepDatabase×FlatSpec →o ModuleName-set
                EnableFlat (rep, repdb, sp) △
                  let mk_FlatSpec (un, ast) = sp,
                      mk_(-, mk_FileStat (sc, -)) = rep.GetStatus ().Status (un),
```

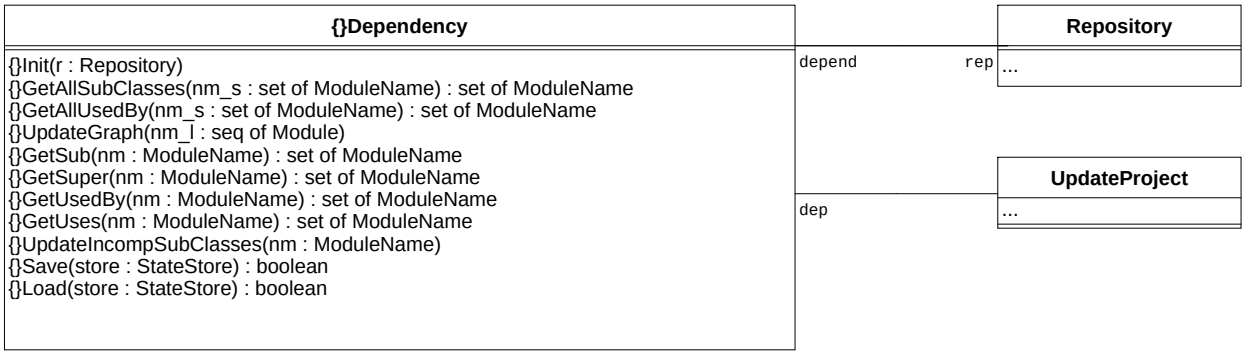

Figure 11: Interface for Dependency class

$$changed = repdb.EnableUnit\,(un, ast, sc)\ \text{in}$$
$$\text{return if } changed$$
$$\text{then } \{un\}$$
$$\text{else } \{\}$$

end
FlatSes

EnableFlat is only called by flat specifications. The module can only be enabled if all corresponding files are syntax checked with succes. If the status of the module changes the name inserted in a set is returned.

3.9 Dependency Information

The dependency information is defined in the VDM-SL module *DEP*, the "Dependency Abstract Syntax". The *Dependency* is an abstract class interface to this module.

3.9.1 Class Dependency

class
Dependency is subclass of *ProjectTypes*
instance variables
$$rep : Repository;$$

Init must be called in order to init the *DEP* module state, i.e when the program is launched and when the project is changed.
operations
public

$Init : Repository \xrightarrow{o} ()$
$Init\,(r) \triangleq$
is not yet specified;

public

$Depend : () \xrightarrow{o} ()$
$Depend\,() \triangleq$
is not yet specified;

public

$GetAllSubClasses : ModuleName\text{-set} \xrightarrow{o} ModuleName\text{-set}$
$GetAllSubClasses\,(nm\_s) \triangleq$
is not yet specified;

public

$GetAllUsedBy : ModuleName\text{-set} \xrightarrow{o} ModuleName\text{-set}$
$GetAllUsedBy\,(nm\_s) \triangleq$
is not yet specified;

GetAllSubClasses and *GetAllUsedBy* are called from *UpdateSC* with a set of *ModuleNames* that are parsed and return with a set of *ModuleName*. Note that all levels of sub/used classes are returned and not only the first level. These methods are used to update the status info in the interface.

public

$UpdateGraph : Module^* \xrightarrow{o} ()$
$UpdateGraph\,(nm\_l) \triangleq$
is not yet specified;

UpdateGraph is called from *UpdateSC* with a sequence of parsed classes.

public

$Remove : ModuleName\text{-set} \xrightarrow{o} ()$
$Remove\,(cl\_s) \triangleq$
is not yet specified;

Remove is called from *UpdateSC* with a set of class names not longer in the parsed file.

The next four methods are called with a *ModuleName* and return with a set of *ModuleName* and only in one level. This information can be used to update the corresponding list boxes in the interface.

public

$GetSub : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetSub\,(nm) \triangleq$
is not yet specified;

public

$GetSuper : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetSuper\,(nm) \triangleq$
is not yet specified;

public

$GetUsedBy : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetUsedBy\,(nm) \triangleq$
is not yet specified;

public

$GetUses : ModuleName \xrightarrow{o} ModuleName\text{-set}$
$GetUses\,(nm) \triangleq$
is not yet specified;

Methods for saving and loading of dependency graph.
public
$Save : StateStore \xrightarrow{o} \mathbb{B}$
$Save\,(store) \triangleq$
is not yet specified;
public
$Load : StateStore \xrightarrow{o} \mathbb{B}$
$Load\,(store) \triangleq$
is not yet specified;

Methods for process information, i.e. order of type checking classes.
public
$IsCyclic : ModuleName \xrightarrow{o} \mathbb{B}$
$IsCyclic\,(nm) \triangleq$
is not yet specified;
public
$OrderOfProcess : ModuleName \xrightarrow{o} ModuleName\text{-set}^*$
$OrderOfProcess\,(nm) \triangleq$
is not yet specified
end
Dependency

3.10 BaseTools

The BaseTools and subclasses are the Specification Manager interface to the internals of the VDM-SL and VDM^{++} toolboxes. The BaseTools class defines the fundamental functionalites needed by the ascii and gui interface to the toolboxes. Enhanced functionalities for the Specification Animator is defined in the SABaseTools class.

3.10.1 Class BaseTools

class
BaseTools is subclass of *ToolColleague*
types
public $EvalState = \text{BREAKPOINT} \mid \text{INTERRUPT} \mid \text{SUCCESS} \mid$
ERROR
operations
public
$SetMediator : ToolMediator \xrightarrow{o} ()$
$SetMediator\,(m) \triangleq$
is not yet specified;
public

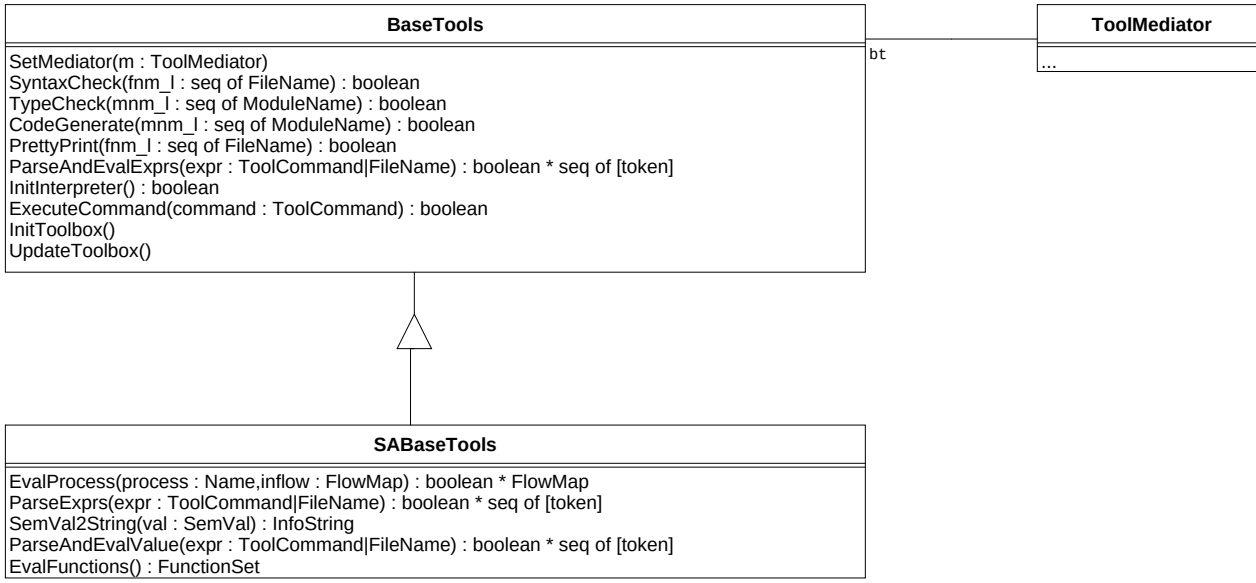

Figure 12: Interface and inheritance for BaseTools classes.

$$SyntaxCheck : FileName^+ \xrightarrow{o} \mathbb{B}$$
$$SyntaxCheck\,(fnm\_l) \triangleq$$
is not yet specified;

SyntaxCheck parses a seqence of *FileName* and if successfully inserts the AST's in the Repository. Special care has been taken concerning changing from flat to structured specifications and vice verca. If the user selects at least all parsed files in the current project for syntax checking (i.e. either by selecting all modules in the GUI or specify all corresponding files as arguments to the "read" command), this is turned into a temporary disabling of the current "session" type. If the "session" type does not change, the type is just enabled again, otherwise a *Repository*!*ClearAll* command is issued. This allows you to go from one session type to another if you just parse all files in one action (and of course edit your files properly). The type of the first file in the list will decide which session type to be used.

public

$$TypeCheck : ModuleName^+ \xrightarrow{o} \mathbb{B}$$
$$TypeCheck\,(mnm\_l) \triangleq$$
is not yet specified;

public

$$CodeGenerate : ModuleName^+ \times (\mathrm{JAVA} \mid \mathrm{CPP}) \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times [\mathsf{char}^*] \times \mathbb{B} \times \mathbb{B} \xrightarrow{o} \mathbb{B}$$
$$CodeGenerate\,(mnm\_l, m, s, p, t, skt, package\_name, cop, testcond) \triangleq$$
is not yet specified;

public

$PrettyPrint : FileName^+ \xrightarrow{o} \mathbb{B}$
$PrettyPrint\,(fnm\_l) \triangleq$
is not yet specified;

public

$ClassDepend : ModuleName \xrightarrow{o} \mathbb{B}$
$ClassDepend\,(nm) \triangleq$
is not yet specified;

public

$InhTree : () \xrightarrow{o} \mathbb{B}$
$InhTree\,() \triangleq$
is not yet specified;

public

$Depend : () \xrightarrow{o} ()$
$Depend\,() \triangleq mediator.GetRepository$
$().GetDependency\quad (\,)\ .Depend();$

public

$ParseAndEvalExprs : ToolCommand \mid FileName \xrightarrow{o} \mathbb{B} \times [\mathsf{token}]^*$
$ParseAndEvalExprs\,(expr) \triangleq$
is not yet specified;

ParseAndEvalExpr evaluates the expression(s) expr. The expression(s) can either be contained in a file or a string (corresponding to a *ToolCommand*). It returns bool and a sequence of the following elements:

- semantic value (SEM‘VAL) if ok,
- nil if a run-time error occur.

The first field in the resulting tuple is true if no syntax or run-time errors occur

public

$ParseAndDebugExprs : ToolCommand \mid FileName \xrightarrow{o} EvalState \times$
$[\mathsf{token}]^*$
$ParseAndDebugExprs\,(expr) \triangleq$
is not yet specified;

ParseAndDebugExprs is equivalent to *ParseAndEvalExprs* the only difference being that the expression is debugged while being evaluated. It returns the state of the evaluation instead of a boolean value.

public

$InitInterpreter : () \xrightarrow{o} \mathbb{B}$
$InitInterpreter\,() \triangleq$
is not yet specified;

InitInterpreter initialises the interpreter with the AST's currently in the repository

public

$ExecuteCommand : ToolCommand \xrightarrow{o} \mathbb{B}$
$ExecuteCommand\,(command) \triangleq$
is not yet specified;

ExecuteCommand executes the command line command contained in the paramater command.

e.g. $ExecuteCommand(mk_ToolCommand(\text{"print 10"}))$. *ExecuteCommand* returns

- true, if the main loop shall continue at the current call back level (commands like print, read, modules etc.).
- false, if the main loop must leave the current call back level (commands like cont, step, finish etc).

The operation *SetBreakOnName* is used to set a break point for a function or operation. The formal parameters correspond to the module (or class) in which the function or operation is located, and the name of the function or operation respectively.
public

$$SetBreakOnName : ModuleName \times Name \xrightarrow{o} \mathbb{N}$$
$$SetBreakOnName\,(modnm, nm) \triangleq$$
is not yet specified;

The operation *SetBreakOnPos* is used to set a break point for a function or operation. The formal parameters correspond to the file in which the function or operation is located, and the line number and col number indicating the position in the file.
public

$$SetBreakOnPos : FileName \times \mathbb{N} \times \mathbb{N} \xrightarrow{o} \mathbb{N}$$
$$SetBreakOnPos\,(filenm, line, col) \triangleq$$
is not yet specified;

The operation *DeleteBreakPoint* is used to delete a break point. The formal parameter corresponds to the number of the breakpoint that was returned by SetBreakOnPos or by SetBreakOnName
public

$$DeleteBreakPoint : \mathbb{N} \xrightarrow{o} ()$$
$$DeleteBreakPoint\,(num) \triangleq$$
is not yet specified;

The operations *DebugStep*, *DebugStepIn*, *DebugSingleStep* and *DebugContinue* are used to perform a debugging step. They call STKM‘EvalStep, EvalStepIn, EvalSingleStep and EvalContinue and convert the results to a seq of tokens. The bool value indicates if stepping was possible.
public

$$DebugStep : () \xrightarrow{o} EvalState \times [\text{token}^*]$$
$$DebugStep\,() \triangleq$$
is not yet specified;

public

$$DebugStepIn : () \xrightarrow{o} EvalState \times [\text{token}^*]$$
$$DebugStepIn\,() \triangleq$$
is not yet specified;

public

$$DebugSingleStep : () \xrightarrow{o} EvalState \times [\text{token}^*]$$
$$DebugSingleStep\,() \triangleq$$
is not yet specified;

public

$DebugContinue : () \xrightarrow{o} EvalState \times [\text{token}^*]$
$DebugContinue\,() \triangleq$
  is not yet specified;

public

$InitToolbox : () \xrightarrow{o} ()$
$InitToolbox\,() \triangleq$
  is not yet specified;

InitToolbox is called from the repository when the Open or New commands are issued. Cleans up the toolbox.

public

$SaveTypeCheckEnv : StateStore \xrightarrow{o} \mathbb{B}$
$SaveTypeCheckEnv\,(s) \triangleq$
  is not yet specified;

public

$LoadTypeCheckEnv : StateStore \xrightarrow{o} \mathbb{B}$
$LoadTypeCheckEnv\,(s) \triangleq$
  is not yet specified;

public

$UpdateToolbox : () \xrightarrow{o} ()$
$UpdateToolbox\,() \triangleq$
  is not yet specified;

UpdateToolbox is called from the repository when the Open or New commands are completed. UpdateToolbox are called after the the Repository is updated with respect to the new project.

public

$CheckForModifiedFiles : () \xrightarrow{o} ()$
$CheckForModifiedFiles\,() \triangleq$
  is not yet specified;

public

$SetPriorityFile : FileName \xrightarrow{o} ()$
$SetPriorityFile\,(fn) \triangleq$
  is not yet specified;

public

$SetPrimarySchedulingAlgorithm : Name \xrightarrow{o} ()$
$SetPrimarySchedulingAlgorithm\,(nm) \triangleq$
  is not yet specified;

public

$SetTimeFile : FileName \xrightarrow{o} ()$
$SetTimeFile\,(fn) \triangleq$
  is not yet specified;

GetCurrentModule, *PopModule* and *PushModule* are used to modify the Module-stack of the VDM-SL Toolbox.

public

$GetCurrentModule : () \xrightarrow{o} \mathbb{B} \times [ModuleName]$
$GetCurrentModule\,() \triangleq$
  is not yet specified;

public

$PopModule : () \xrightarrow{o} \mathbb{B}$
$PopModule\ () \triangleq$
is not yet specified;

public

$PushModule : ModuleName \xrightarrow{o} \mathbb{B}$
$PushModule\ (nm) \triangleq$
is not yet specified;

public

$GetPossibleInterfaces : () \xrightarrow{o} ModuleName\text{-set}$
$GetPossibleInterfaces\ () \triangleq$
is not yet specified;

public

$ResetInterfaces : () \xrightarrow{o} ()$
$ResetInterfaces\ () \triangleq$
is not yet specified;

public

$JavaSyntaxCheck : FileName^{+} \xrightarrow{o} \mathbb{B}$
$JavaSyntaxCheck\ (\text{-}) \triangleq$
is not yet specified;

public

$JavaTypeCheck : ModuleName^{+} \xrightarrow{o} \mathbb{B}$
$JavaTypeCheck\ (\text{-}) \triangleq$
is not yet specified;

public

$JavaGenerateVDM : ModuleName^{+} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \xrightarrow{o} \mathbb{B}$
$JavaGenerateVDM\ (\text{-},\text{-},\text{-},\text{-}) \triangleq$
is not yet specified;

public

$PogGenerate : ModuleName^{+} \xrightarrow{o} \mathbb{B}$
$PogGenerate\ (\text{-}) \triangleq$
is not yet specified;

public

$NewUnnamedProject : () \xrightarrow{o} ()$
$NewUnnamedProject\ () \triangleq$
is not yet specified;

public

$LoadProject : FileName \xrightarrow{o} ()$
$LoadProject\ (\text{-}) \triangleq$
is not yet specified;

public

$AddFiles : FileName\text{-set} \xrightarrow{o} ()$
$AddFiles\ (\text{-}) \triangleq$
is not yet specified;

public

$RemoveFiles : FileName\text{-set} \xrightarrow{o} ()$
$RemoveFiles\ (\text{-}) \triangleq$
is not yet specified;

public

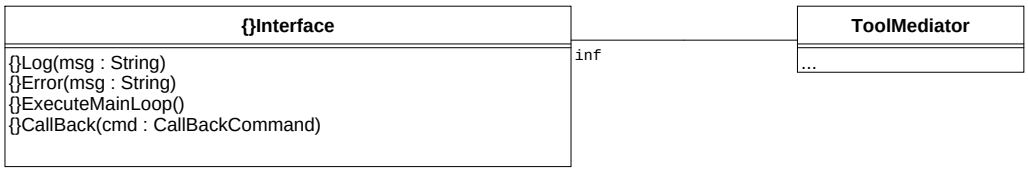

Figure 13: Interface for the Interface class.

$SaveProjectAs : FileName \xrightarrow{o} \mathbb{B}$
$SaveProjectAs\,(\text{-}) \triangleq$
is not yet specified;

public

$SaveProject : FileName \xrightarrow{o} \mathbb{B}$
$SaveProject\,(\text{-}) \triangleq$
is not yet specified;

public

$GetProjectName : () \xrightarrow{o} [FileName]$
$GetProjectName\,() \triangleq$
is not yet specified

end
BaseTools

3.11 Interface

The Interface class is an abstract class. It defines how to handle log and error messages and what to do when a call back is received.

3.11.1 Class Interface

class
Interface is subclass of *ProjectTypes*
operations
public

$Log : String \xrightarrow{o} ()$
$Log\,(msg) \triangleq$
is subclass responsibility;

public

$Error : String \xrightarrow{o} ()$
$Error\,(msg) \triangleq$
is subclass responsibility;

public

$InterpreterLog : String \xrightarrow{o} ()$
$InterpreterLog\,(msg) \triangleq$
is subclass responsibility;

public

$$ExecuteMainLoop : () \xrightarrow{o} ()$$
$$ExecuteMainLoop\,() \triangleq$$
is subclass responsibility;

ExecuteMainLoop is used by the debugger in BaseTools to call back to the main loop when the interpretation stops at a break point.
public

$$GetBtSeq : () \xrightarrow{o} \mathsf{token} \mid \mathbb{Z}$$
$$GetBtSeq\,() \triangleq$$
is not yet specified;

public

$$SetBtSeq : \mathsf{token} \mid \mathbb{Z} \xrightarrow{o} ()$$
$$SetBtSeq\,(dummy) \triangleq$$
is not yet specified;

public

$$CallBack : CallBackCommand \xrightarrow{o} ()$$
$$CallBack\,(cmd) \triangleq$$
is subclass responsibility;

The following five functions handle the display of a progress meter.

Call *InitMeter* to create and initialize a new meter; the default total number of steps is 100, corresponding to a percentage.,
public

$$InitMeter : String \times String \xrightarrow{o} ()$$
$$InitMeter\,(title, label) \triangleq$$
skip;

To set a total other than the default, use *SetMeterTotal*:
public

$$SetMeterTotal : \mathbb{R} \xrightarrow{o} ()$$
$$SetMeterTotal\,(pct) \triangleq$$
skip;

IncrementMeter updates the progress meter by one unit, without affecting the title or label.
public

$$IncrementMeter : [String] \xrightarrow{o} ()$$
$$IncrementMeter\,(\text{-}) \triangleq$$
skip;

Alternatively, *UpdateMeter* updates the progress to be value *pct* and also modifies the label.
public

$$UpdateMeter : \mathbb{R} \times String \xrightarrow{o} ()$$
$$UpdateMeter\,(pct, label) \triangleq$$
skip;

When finished with, the meter may be cleaned up using *DestroyMeter*.
public

$DestroyMeter : () \xrightarrow{o} ()$
$DestroyMeter\ () \triangleq$
  skip;

The following operations has to do with the Stop button in the gui interpreter.
public

$EnableUserInput : () \xrightarrow{o} ()$
$EnableUserInput\ () \triangleq$
  skip;

public

$DisableUserInput : () \xrightarrow{o} ()$
$DisableUserInput\ () \triangleq$
  skip;

public

$RefreshInterface : () \xrightarrow{o} ()$
$RefreshInterface\ () \triangleq$
  skip

end
Interface

CallBack is used by BaseTools when when parts of the interface must be changed or updated. E.g. when a break-point is set or deleted in the command window.

3.12 Errors

The Errors class is an abstract class for errors arising from the internal tools in the toolbox (i.e. syntax checker, type checker etc.). It has an association to an ErrorState class, which is a *state* pattern. The ErrorState subclasses defines the different ways of showing errors.

3.12.1 Class Errors

class
Errors is subclass of *ToolColleague*
instance variables

protected $msgs : Message^* := []$;
protected $msgPtr : \mathbb{Z} := 0$;
$msgStatus : \mathbb{Z} \times \mathbb{Z} := \mathsf{mk\_}(0,0)$;
$state : ErrorState$;

The *state* instance variable contains the current state of the errors class. The current state is a subclass of ErrorState and defines the state specific methods. Currently valid states are:

- ScriptErr
- PromptErr
- BatchErr

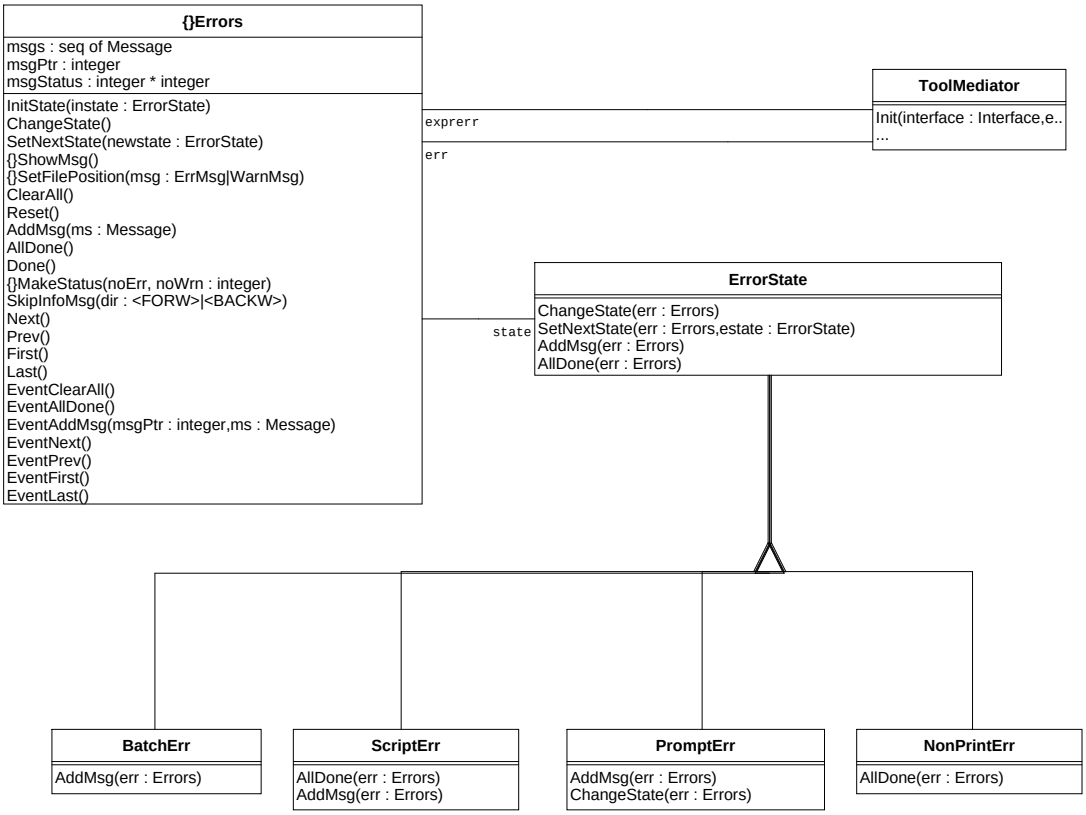

Figure 14: Interface and inheritance for Errors and ErrorState classes.

- NonPrintErr

Change of state can either be on external demand or be under responsibility of the current state.

When a user command is issued, *ClearAll* should be called resetting the old sequence of errors. A user command is one or more tool commands. The tools can use the commands *Reset*, *AddMsg* and *Done*. When the user command terminates, *AllDone* should be called. It may show up that *Reset* is unnecessary.

operations

public

$$InitState : ErrorState \xrightarrow{o} ()$$
$$InitState\,(instate) \triangleq$$
$$\quad state := instate;$$

InitState should be called before any operations on the state as soon it is known which state to start up in.

public

$$ChangeState : () \xrightarrow{o} ()$$
$$ChangeState\,() \triangleq state.$$
$$\quad ChangeState(\mathsf{self})\ ;$$

ChangeState changes possibly the state depending on the current state and should be called when the "script" command is issued.

public

$$SetNextState : ErrorState \xrightarrow{o} ()$$
$$SetNextState\,(newstate) \triangleq$$
$$\quad state := newstate;$$

SetNextState should only be called by the subclasses of ErrorState

public

$$ShowMsg : () \xrightarrow{o} ()$$
$$ShowMsg\,() \triangleq$$
$$\quad \text{is subclass responsibility};$$

ShowMsg prints out the current message, i.e. $msgs(msgPtr)$.

public

$$SetFilePosition : ErrMsg \mid WarnMsg \xrightarrow{o}$$
$$()$$
$$SetFilePosition\,(msg) \triangleq$$
$$\quad \text{is subclass responsibility};$$

SetFilePosition indicates the position of error message msg

public

```
                                        ClearAll : () ->o ()
                                        ClearAll () ≜
                                          (   msgs := [];
                                              msgPtr := 0;
                                              msgStatus := mk_(0, 0);
                                              self.EventClearAll()
                                          );
public
                                        Reset : () ->o ()
                                        Reset () ≜
                                          skip;
public
                                        AddMsg : Message ->o ()
                                        AddMsg (ms) ≜
                                          def mk_(noErr, noWrn) = msgStatus
in
                                          (   msgs := msgs ^~ [ms];
                                              msgPtr := msgPtr + 1;
                                              if is_ErrMsg (ms)
                                              then msgStatus := mk_(noErr+
1, noWrn)
                                              elseif is_WarnMsg (ms)
                                              then msgStatus := mk_(noErr, noWrn+
1);
                                              state.AddMsg(self) ;
                                              self.EventAddMsg(msgPtr, ms)
                                          );
public
                                        AllDone : () ->o ()
                                        AllDone () ≜
                                          (   if msgPtr > 0
                                              then msgPtr := 1;
                                              state.AllDone(self) ;
                                              self.EventAllDone()
                                          );
public
                                        Done : () ->o ()
                                        Done () ≜
                                          def mk_(noErr, noWrn) = msgStatus
in
                                          (   self.MakeStatus(noErr, noWrn);
                                              msgStatus := mk_(0, 0)
                                          );
public
                                        MakeStatus : ℤ × ℤ ->o ()
                                        MakeStatus (noErr, noWrn) ≜
                                          is subclass responsibility;
```

MakeStatus creates the message that appears in the errors window after each file/module, i.e. counting errors and warnings (or ok).

It should be decided whether the status message should be inserted or not in the sequence of messages "msgs". This could also be subclass responsibility.

public

$$
\begin{array}{l}
SkipInfoMsg : \mathrm{FORW} \mid \mathrm{BACKW} \xrightarrow{o} () \\
SkipInfoMsg\,(dir) \triangleq \\
\quad \mathsf{let}\ step = \mathsf{if}\ dir = \mathrm{FORW} \\
\qquad\qquad\qquad \mathsf{then}\ 1 \\
\qquad\qquad\qquad \mathsf{else}\ -1, \\
\qquad lenMsgs = \mathsf{len}\ msgs\ \mathsf{in} \\
\quad \mathsf{while\ def}\ \ \mathsf{in} \\
\qquad\quad \mathsf{is\_}InfoMsg\,(ms) \\
\quad \mathsf{do\ def}\ nextMsgPtr = msgPtr{+}step\ \mathsf{in} \\
\qquad \mathsf{if}\ nextMsgPtr \leq lenMsgs \vee nextMsgPtr \geq 1 \\
\qquad\quad \mathsf{then}\ msgPtr := nextMsgPtr;
\end{array}
$$

public

$$
\begin{array}{l}
Next : () \xrightarrow{o} () \\
Next\,() \triangleq \\
\quad \mathsf{if}\ msgPtr < \mathsf{len}\ msgs \\
\quad \mathsf{then}\ (\quad msgPtr := msgPtr + 1; \\
\qquad\qquad \mathsf{self}.SkipInfoMsg(\mathrm{FORW})\,; \\
\qquad\qquad \mathsf{self}.ShowMsg()\,; \\
\qquad\qquad \mathsf{self}.EventNext() \\
\qquad\quad) \\
\quad \mathsf{else\ skip};
\end{array}
$$

public

$$
\begin{array}{l}
Prev : () \xrightarrow{o} () \\
Prev\,() \triangleq \\
\quad \mathsf{if}\ msgPtr > 1 \\
\quad \mathsf{then}\ (\quad msgPtr := msgPtr - 1; \\
\qquad\qquad \mathsf{self}.SkipInfoMsg(\mathrm{BACKW}); \\
\qquad\qquad \mathsf{self}.ShowMsg()\,; \\
\qquad\qquad \mathsf{self}.EventPrev() \\
\qquad\quad) \\
\quad \mathsf{else\ skip};
\end{array}
$$

public

$$
\begin{array}{l}
First : () \xrightarrow{o} () \\
First\,() \triangleq \\
\quad \mathsf{if\ len}\ msgs > 0 \\
\quad \mathsf{then}\ (\quad msgPtr := 1; \\
\qquad\qquad \mathsf{self}.SkipInfoMsg(\mathrm{FORW})\,; \\
\qquad\qquad \mathsf{self}.ShowMsg()\,; \\
\qquad\qquad \mathsf{self}.EventFirst() \\
\qquad\quad)\,;
\end{array}
$$

public

```
Last : () →o ()
Last () ≜
  if len msgs > 0
  then (    msgPtr := len msgs;
            self.SkipInfoMsg(BACKW);
            self.ShowMsg() ;
            self.EventLast()
       ) ;
```

The following three methods allows direct access to the errors held by the *Errors* class.

public

```
GetStatus : () →o ℤ × ℤ
GetStatus () ≜
  return msgStatus;
```

public

```
GetErrors : () →o Message*
GetErrors () ≜
  return [msgs (i) | i ∈ inds msgs·is_ErrMsg (msgs (i))];
```

public

```
GetWarnings : () →o Message*
GetWarnings () ≜
  return [msgs (i) | i ∈ inds msgs·is_WarnMsg (msgs (i))]
```

The following methods should notify the interface if necessary and should be implemented in the subclass.

public

```
EventClearAll : () →o ()
EventClearAll () ≜
  skip;
```

public

```
EventAllDone : () →o ()
EventAllDone () ≜
  skip;
```

public

```
EventAddMsg : ℤ × Message →o ()
EventAddMsg (-,-) ≜
  skip;
```

public

```
EventNext : () →o ()
EventNext () ≜
  skip;
```

public

```
EventPrev : () →o ()
EventPrev () ≜
  skip;
```

public

```
EventFirst : () →o ()
EventFirst () ≜
  skip;
```

public

$EventLast : () \xrightarrow{o} ()$
$EventLast\ () \triangleq$
skip

end
Errors

3.12.2 Class ErrorState

The *ErrorState* class is the behavioral pattern "state" as described in "Design Patterns". This class specifies the default behaviour for the state specific methods.

class
ErrorState is subclass of *ProjectTypes*
operations
public

$ChangeState : Errors \xrightarrow{o} ()$
$ChangeState\ (\text{-}) \triangleq$
skip;

public

$SetNextState : Errors \times ErrorState \xrightarrow{o} ()$
$SetNextState\ (err, estate) \triangleq err.$
$SetNextState(estate)$;

public

$AddMsg : Errors \xrightarrow{o} ()$
$AddMsg\ (\text{-}) \triangleq$
skip;

public

$AllDone : Errors \xrightarrow{o} ()$
$AllDone\ (\text{-}) \triangleq$
skip

end
ErrorState

3.12.3 Class BatchErr

In the *BatchErr* state all errors and warnings are printed out as soon as they occur. This state never changes.

class
BatchErr is subclass of *ErrorState*
operations
public

$AddMsg : Errors \xrightarrow{o} ()$
$AddMsg\ (err) \triangleq err.$
$ShowMsg()$

end
BatchErr

3.12.4 Class PromptErr

In the *PromptErr* state only the first error is printed out and the user is prompted to ask for the next errors.

class
PromptErr is subclass of *ErrorState*
operations
public

$$
\begin{array}{l}
AddMsg : Errors \xrightarrow{o} () \\
AddMsg\,(err) \triangleq \\
\quad \mathsf{def}\ newstate = \mathsf{new}\ NonPrintErr\,() \ \mathsf{in} \\
\quad (\quad err.ShowMsg()\,; \\
\qquad \mathsf{self}.SetNextState(err, newstate) \\
\quad);
\end{array}
$$

AddMsg prints out the first error when it occurs and changes state to *NonPrintErr*.

public

$$
\begin{array}{l}
ChangeState : Errors \xrightarrow{o} () \\
ChangeState\,(err) \triangleq \\
\quad \mathsf{def}\ newstate = \mathsf{new}\ ScriptErr\,() \ \mathsf{in} \\
\quad \mathsf{self}.SetNextState(err, newstate)
\end{array}
$$

end
PromptErr

ChangeState is called when the "script" command is issued. State changes to *ScriptErr*.

3.12.5 Class ScriptErr

Prestate: *PromptErr*

Poststate: *PromptErr*

If a script command is issued in the *PromptErr* state, state changes to *ScriptErr* allowing errors to printed out forthcoming. When the script command terminates, state changes back to *PromptErr*.

class
ScriptErr is subclass of *ErrorState*
operations
public

$$
\begin{array}{l}
AllDone : Errors \xrightarrow{o} () \\
AllDone\,(err) \triangleq \\
\quad \mathsf{def}\ newstate = \mathsf{new}\ PromptErr\,() \ \mathsf{in} \\
\quad \mathsf{self}.SetNextState(err, newstate)\,;
\end{array}
$$

public

$$
\begin{array}{l}
AddMsg : Errors \xrightarrow{o} () \\
AddMsg\,(err) \triangleq err. \\
\quad ShowMsg()
\end{array}
$$

end

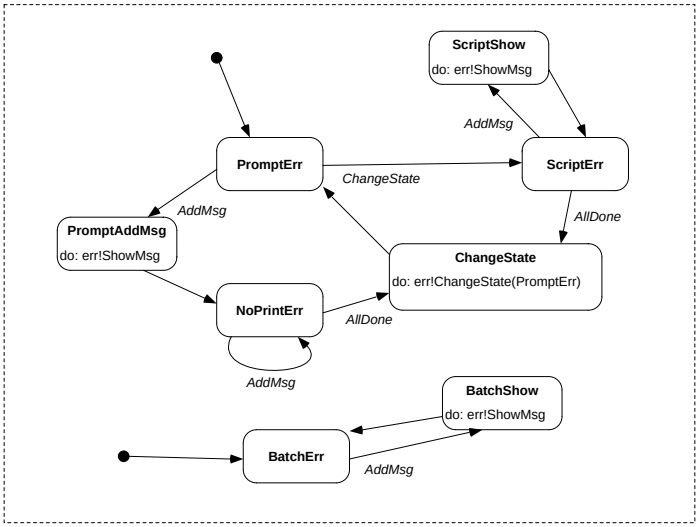

Figure 15: States for Errors class

ScriptErr

3.12.6 Class NonPrintErr

Prestate: *PromptErr*

Poststate: *PromptErr*

When the first error occurs in the *PromptErr* state, it is printed out and state changes to *NonPrintErr*. When the user command terminates, state changes back to *PromptErr*.

class
NonPrintErr is subclass of *ErrorState*
operations
public

$$AllDone : Errors \xrightarrow{o} ()$$
$$AllDone\,(err) \triangleq$$
$$\quad \text{def } newstate = \text{new } PromptErr\,() \text{ in}$$
$$\quad \text{self}.SetNextState(err, newstate)$$

end
NonPrintErr

3.13 Options

This class is not yet used.

3.13.1 Class Options

class
Options is subclass of *ToolColleague*
operations
public

$$Read : FileName \xrightarrow{o} ()$$
$$Read\,(project) \triangleq$$
is not yet specified;

public

$$Save : FileName \xrightarrow{o} ()$$
$$Save\,(project) \triangleq$$
is not yet specified

end
Options

3.14 Global Types

In order to share types between classes a global superclass ProjectTypes is defined. Classes that need types from this class must be subclasses of ProjectTypes. In practise this means that all classes in the Specification Manager are subclasses of ProjectTypes.

Some types are implemented different from the specification. This is types that are outside the specification, e.g. the AST. The `AstVal` is specified as a record containing a token but in the implementation the token is exchanged with the real AST record type. This can be done because the code generator only generates the first level of the type, i.e. a Record.

3.14.1 Class ProjectTypes

class
ProjectTypes
types
public

$$ModuleName :: nm : String;$$

public

$$FileName :: nm : String;$$

public

$$Name :: nm : String;$$

public

$$InfoString :: nm : String;$$
public $String = \mathsf{char}^*$;
public $FileId = \mathbb{N}$;

public

$$AstVal :: val : \mathsf{token};$$

public

$$DepGraph :: g : \mathsf{token};$$

public

$TCEnv :: e : \mathsf{token};$

AstVal and *DepGraph* are implemented as Records. *TCEnv* is implemented as Tuple.

public $Message = ErrMsg \mid WarnMsg \mid InfoMsg;$

public

$ErrMsg :: fid : \mathbb{Z}$
$line : \mathbb{Z}$
$col : \mathbb{Z}$
$msg : \mathsf{char}^{**};$

public

$WarnMsg :: fid : \mathbb{Z}$
$line : \mathbb{Z}$
$col : \mathbb{Z}$
$msg : \mathsf{char}^{**};$

public

$InfoMsg :: msg : \mathsf{char}^{**};$
public $Action = FileAction \mid UnitAction;$
public $FileAction = \text{EDIT} \mid \text{SYNTAXCHECK} \mid \text{PP};$

public $UnitAction = \text{TYPECHECK} \mid \text{CG} \mid \text{JCG};$

public $Status = \text{NONE} \mid \text{ERROR} \mid \text{OK} \mid \text{DISABLED};$

public $Succes = \mathbb{B};$

public

$UnitStat :: type : Status \mid \text{POS} \mid \text{DEF}$
$cg : Status$
$javaCg : Status;$

public

$FileStat :: syntax : Status$
$pp : Status;$
public $SessionType = \text{NONE} \mid \text{STRUCTURED} \mid \text{FLAT} \mid \text{DISABLED};$

public

$Module :: nm : ModuleName$
$ast : AstVal;$

public

$FlatSpec :: nm : ModuleName$
$ast : AstVal;$

public

$CGInfo :: repos : \mathsf{token};$

CGInfo represents the CGRepository.

public

$ToolCommand :: command : \mathsf{token};$

ToolCommand represents a command in the command window. E.g. $print 10 + 10$. Commands are parsed to the command interpreter as *String* inclosed in a *Token*.

public *CallBackCommand* = *BreakNameSet* | *BreakPosSet* | *BreakRemove* | *BreakEnable* | *BreakDisable* | *BackTrace* | *BtGotoLevel* | *AddFiles* | *RemoveFiles* | *AddModules* | *RemoveModules* | *ClearAll* | *ChangedFileStatus* | *ChangedModuleStatus* | *DependInfo* | *DependUpdate* | *RemoveFileChangedMark* | *AddFileChangedMark* | *DrawInhTree* | *UpdateSelections* | *ClearInhTree* | *ChangedProjName* | *ClearDebugWindow* | *PogCleanUp* | *PogAdd* | *PogUpdateFilter*;

CallBackCommand is used to call back from BaseTools to the Interface class, when parts of the interface must be changed or updated Also called from repository when change in present modules and files

public
BreakNameSet :: *bp* : token
num : $\mathbb{N}$;

public
BreakPosSet :: *bp* : token
line : $\mathbb{N}$
col : $\mathbb{N}$
num : $\mathbb{N}$;

public
BreakRemove :: *bp* : $\mathbb{N}$;

public
BreakEnable :: *bp* : $\mathbb{N}$;

public
BreakDisable :: *bp* : $\mathbb{N}$;

public
BackTrace :: *info* : (token $\times$ token*)*;

public
BtGotoLevel :: *level* : $\mathbb{N}$;

public
AddFiles :: *nms* : *FileName*-set;

public
RemoveFiles :: *nms* : *FileName*-set;

public
AddModules :: *nms* : *ModuleName*-set;

public

$RemoveModules :: nms : ModuleName\text{-set};$

public

$ClearAll :: \ ;$

public

$ChangedFileStatus :: nm : FileName;$

public

$ChangedModuleStatus :: nms : ModuleName\text{-set};$

public

$$\begin{aligned} DependInfo :: & \ name : ModuleName \\ & \ supers : ModuleName\text{-set} \\ & \ subs : ModuleName\text{-set} \\ & \ uses : ModuleName\text{-set} \\ & \ used : ModuleName\text{-set} \\ & \ compl : \mathbb{B}; \end{aligned}$$

public

$DependUpdate :: names : ModuleName\text{-set};$

public

$$\begin{aligned} ChangedProjName :: & \ oldn : [FileName] \\ & \ newn : [FileName]; \end{aligned}$$

public

$RemoveFileChangedMark :: nm : FileName;$

public

$AddFileChangedMark :: nm : FileName;$

public

$PogCleanUp :: nms : ModuleName^*;$

public

$$\begin{aligned} PogAdd :: & \ checked : \mathsf{char}^* \\ & \ modnm : \mathsf{char}^* \\ & \ memnm : \mathsf{char}^* \\ & \ locclass : \mathsf{char}^* \\ & \ kind : \mathsf{char}^* \\ & \ no : \mathbb{N} \\ & \ file : \mathsf{char}^* \\ & \ tmpfn : \mathsf{char}^* \\ & \ line : \mathbb{N} \\ & \ column : \mathbb{N} \\ & \ length : \mathbb{N} \\ & \ po : \mathsf{char}^*; \end{aligned}$$

public

$PogUpdateFilter :: \ ;$

public

$FlowMap :: flows : \mathsf{token} \xrightarrow{m} \mathsf{token};$

The data type *DrawInhTree* represents the graphical representation of the inheritance tree to be drawn.

public

$DrawInhTree :: graph : (Node \mid Line)^*;$

public

$Node :: x : \mathbb{N}$
$y : \mathbb{N}$
$name : \mathsf{char}^*;$

public

$Line :: pos1 : \mathbb{N} \times \mathbb{N}$
$pos2 : \mathbb{N} \times \mathbb{N};$

The *UpdateSelections* data type is used in making a call back to set the selection of classes/modules and files.

public

$UpdateSelections ::$;

The *ClearInhTree* data type is used in making a call back to clear the inheritance tree.

public

$ClearInhTree ::$;

The *ClearDebugWindow* data type is used in making a call back to clear the debug window (the display window) in the interprete tool.

public

$ClearDebugWindow ::$;

FlowMap is used by the SDA/Toolbox integration. It is used to map the name of a flow to a semantic value

public

$SemVal :: v : \mathsf{token};$

public

$FunctionSet :: nms : \mathsf{token\text{-}set};$

FunctionSet is used by the SDA/Toolbox integration. Used for the set of function and operation names in the current module

Types for Repository and related classes

public $UnitState = ModuleName \xrightarrow{m} VDMUnitElem;$
public $FileState = FileName \xrightarrow{m} FileInfo;$
public $FileInfo = ModuleNames \times FileStatus \times FileId \times [TmpFileName];$
public $TmpFileName = FileName;$
public $ModuleNames = ModuleName\text{-}\mathsf{set};$
public $StateType = UnitState \times FileState;$
public $IrregFileIds =$ CMDLINEFILEID | CGFILEID | TCFILEID | TMPFILEID | STDINFILEID

end
ProjectTypes

4 Using the Specification Manager in the Toolbox development

4.1 Setting up the Specification Manager

In order to make an instance of the Specification Manager, you must:

1. Define the abstract classes Errors and Interface.
2. Instantiate a ToolKit class.
3. Create an object reference to an Interface class.
4. Instanciate two Errors classes and create two object references to them.
5. Create proper object reference to the initial state for each Errors instance and call the Errors `InitState` method.
6. Initialize the ToolKit instance with the Interface and Errors object references.

In the main program for the ascii interface, `vdmde.cc`, the instantiation part looks like:

```
// Set up the ToolKit
//  (we use the default settings of vdm_log and vdm_err)
toolkit = new vdm_ToolKit ();
ObjectRef TK (toolkit);
ObjectRef inf (new AsciiInterface ());

AsciiErrors * errs = new AsciiErrors();
ObjectRef ERRS (errs);

AsciiErrors * exprerrs = new AsciiErrors();
ObjectRef ExprERRS (exprerrs);

//  The Errors class must be initiated with a state.
//  This is PromptErr in case of Ascii interface,
//  and BatchErr in case of GUI or batch version.

ObjectRef batch_state (new vdm_BatchErr());
errs->vdm_InitState(batch_state);

exprerrs->vdm_InitState(batch_state);
toolkit->vdm_Init (inf,ERRS, ExprERRS);
```

4.2 Abstract classes

The **Interface** and **Errors** classes are abstract classes in the specification and it is the users responsibility to define the concrete subclasses as needed in the implementation. This way the Specification Manager can be adapted to the different needs from e.g. an ascii interface or a graphical interface. Only the *delegated* methods needs to be defined.

4.2.1 Call back

The Specification Manager generates call back calls to the Interface when parts of the Interface must be changed or updated. E.g. when a file is added or removed. The user must define which actions the Interface will take.

4.3 Calling the Specification Manager

The Specification Manager is called either from the Toolbox or from the Interface. Calls from the Interface to the Toolbox should go through the Specification Manager. In order to access the type definitions in class ProjectTypes auxiliary functions is defined in `projectval.h`. For each type `<type>` in class ProjectTypes there should be a `mk_<type>`-function and a `Extract<type>`-function. If necessary there is also a `is_<type>`-function. From `projectval.h`:

```
Record mk_ToolCommand (String command);
bool is_ToolCommand(Record toolcom);
String ExtractToolCommand (Record ToolCommand);
```

4.4 Adding new calls to Toolbox

Calls from the Interface to the Toolbox go via the BaseTools classes. These methods are defined as userimplinary methods and must be hand implemented. The convention is that the BaseTools methods just pass the call to a corresponding method in `tools.cc`. If needed, a new subclass to BaseTools can be defined. It must then be assured that the Toolkit class is initialised with this class (see also section 3.3).

A Implementation of userimplinary methods

A.1 Class BaseTools

```
/***
*  * WHAT
```

```
*  *    Implementation of preliminary methods in class BaseTools
*  * FILE
*  *    $Source: /home/vdmtools/cvsroot/toolbox/code/specman/code/BaseTools_userimpl.cc,v
*  * VERSION
*  *    $Revision: 1.38 $
*  * DATE
*  *    $Date: 2006/06/12 08:36:17 $
*  * STATUS
*  *    Under development
*  * REFERENCES
*  *    IFAD-VDM28
*  * PROJECT
*  *    Toolbox
*  * AUTHOR
*  *    Henrik Voss + $Author: vdmtools $
*  * COPYRIGHT
*  *    (C) 2006 CSK, Japan
***/

#include <stdio.h>
#include <sys/stat.h>

#ifndef _MSC_VER
#include <unistd.h>
#endif // _MSC_VER

#include "BaseTools.h"
#include "tools.h"
#include "codegen_tools.h"
#include "javagen_tools.h"
#include "projectval.h"
#include "StateStore.h"
#include "NoneSes.h"
#include "astaux.h"
#include "Interface.h"
#include "tbutils.h"
#include "tbdebug.h"
#include "pog-interface.h"
#include "settings.h"
#include "tb_exceptions.h"

//
// Implementation of preliminary methods in class BaseTools
//

// SetMediator
// vdm_m : ToolMediator
// ==> ()
void vdm_BaseTools::vdm_SetMediator(const type_ref_ToolMediator & vdm_m)
{
```

```
  vdm_mediator = vdm_m;
  vdm_ToolMediator* mediator = ObjGet_vdm_ToolMediator (vdm_mediator);
  ToolMediator::SetMediator (mediator);
}

// SyntaxCheck
// filenmae_l : seq1 of FileName
// ==> bool
Bool vdm_BaseTools::vdm_SyntaxCheck (const type_21ProjectTypes_FileNameCL & filename_l)
{
  bool succ = true;

  // Check to see if at least all parsed files in current project
  // is going to be parsed. If this is the case,
  // we disable the current session by setting it to NoneSes.
  // The session type will be restored  according to the type
  // of the first successfully parsed file in filename_l
  // (in UpdateRepository in tools.cc)
  if (!filename_l.IsEmpty())
  {
    SET<TYPE_ProjectTypes_FileName> parsedfiles_s (ToolMediator::ParsedFiles ());
    if (!parsedfiles_s.IsEmpty() && parsedfiles_s.SubSet (filename_l.Elems ())) {
      ToolMediator::DisableSession ();
    }

// 20120525 -->
    SET<TYPE_ProjectTypes_FileName> files (ToolMediator::Files());
    files.ImpIntersect(filename_l.Elems());
    files.ImpIntersect(ToolMediator::ParsedFiles ());
    if (!files.IsEmpty())
    {
      ToolMediator::RemoveFiles ( files );
#ifdef VDMPP
      TOOLS::CheckSSParseEnv();
#endif // VDMPP
    }
// <--20120525

    ToolMediator::Errs()->vdm_ClearAll();

    ToolMediator::Interf()->vdm_InitMeter(SEQ<Char>(L"Syntax Checking"), type_cL());
    ToolMediator::Interf()->vdm_SetMeterTotal(filename_l.Length());

    bool errlimit = false;
    int len_filename_l = filename_l.Length();
    for (int idx = 1; (idx <= len_filename_l) && !errlimit; idx++)
    {
      const TYPE_ProjectTypes_FileName & filenm (filename_l[idx]);
      // Update meter
      SEQ<Char> label (L"Syntax checking ");
```

```
      label.ImpConc(SEQ<Char>(PTAUX::ExtractFileName(filenm)));
      ToolMediator::Interf()->vdm_IncrementMeter(label);
      //
      succ = TOOLS::EvalRead (filenm) && succ;

      errlimit = (ToolMediator::Errs()->vdm_GetErrors().Length() > 100);
    }
#ifdef VDMPP
    ToolMediator::EvalInhTree();
#endif //VDMPP

    ToolMediator::UpdateSelections();
    PTAUX::SomethingIsTypeIncorrect();
    ToolMediator::Interf()->vdm_DestroyMeter();
    ToolMediator::Errs()->vdm_AllDone();

    if (errlimit)
    {
      vdm_log << L"Syntax error limit exceeded." << endl;
      vdm_log << L"Syntax checking aborted." << endl;
    }
  }
  return Bool (succ);
}

// TypeCheck
// mnm_l : seq1 of ModuleName
// ==> bool
Bool vdm_BaseTools::vdm_TypeCheck (const type_23ProjectTypes_ModuleNameCL & mnm_l)
{
  bool succ = true;

  if (!mnm_l.IsEmpty())
  {
    ToolMediator::Errs()->vdm_ClearAll();

    ToolMediator::Interf()->vdm_InitMeter(SEQ<Char>(L"Type Checking"), type_cL());
    ToolMediator::Interf()->vdm_SetMeterTotal(mnm_l.Length());

    TOOLS::ResetSSNames();

    bool errlimit = false;
    int len_mnm_l = mnm_l.Length();
    for (int idx = 1; (idx <= len_mnm_l) && !errlimit; idx++)
    {
      const TYPE_ProjectTypes_ModuleName & mnm (mnm_l[idx]);
      SEQ<Char> label(L"Type checking ");
      label.ImpConc(SEQ<Char>(PTAUX::ExtractModuleName(mnm)));
      ToolMediator::Interf()->vdm_IncrementMeter(label);
      //
```

```
      succ = TOOLS::EvalTypeCheck (mnm, 0) && succ;

      errlimit = (ToolMediator::Errs()->vdm_GetErrors().Length() > 100);
    }
#ifdef VDMPP
    ToolMediator::EvalInhTree();
#endif //VDMPP

    ToolMediator::UpdateSelections();
    PTAUX::UpdateIsEverythingTypeCorrect();

    ToolMediator::Interf()->vdm_DestroyMeter();

    ToolMediator::Errs()->vdm_AllDone();

    if (errlimit)
    {
      vdm_log << L"Type error limit exceeded." << endl;
      vdm_log << L"Type checking aborted." << endl;
    }
  }
  return Bool (succ);
}

// CodeGenerate
// mnm_l : seq1 of ModuleName
// m : (<JAVA>|<CPP>)
// s : bool
// p : bool
// t : bool
// st : bool
// package_name : [seq of char]
// cop : bool
// testcond : bool
// ==> bool
Bool vdm_BaseTools::vdm_CodeGenerate (const type_23ProjectTypes_ModuleNameCL & mnm_l,
                                      const Generic & m,
                                      const Bool & s,
                                      const Bool & p,
                                      const Bool & t,
                                      const Bool & st,
                                      const Generic & package_name,
                                      const Bool & cop,
                                      const Bool & testcond)
{
  bool succ = true;

  if (!mnm_l.IsEmpty())
  {
    ToolMediator::Errs()->vdm_ClearAll();
```

```
    ToolMediator::Interf()->vdm_InitMeter(SEQ<Char>(L"Generating C++"), type_cL());
    ToolMediator::Interf()->vdm_SetMeterTotal(mnm_l.Length());

//  ResetCG(); // 20070921
    TOOLS::ResetSSNames();

    size_t len_mnm_l = mnm_l.Length();
    for (size_t idx = 1; idx <= len_mnm_l; idx++)
    {
      SEQ<Char> label;
      if (m == Quote(L"CPP"))
        label = SEQ<Char>(L"Generating C++ for ");
      else
        label = SEQ<Char>(L"Generating Java for ");

      label.ImpConc(SEQ<Char>(PTAUX::ExtractModuleName(mnm_l[idx])));
      ToolMediator::Interf()->vdm_IncrementMeter(label);
      //
      if (GetCGTools().ResetCG(m)) // 20070921
      {
        succ = GetCGTools().EvalCodeGen (mnm_l[idx],m,s,p,t,st, package_name,cop, testcond
      }
      else
        succ = false;
    }
#ifdef VDMPP
    ToolMediator::EvalInhTree();
#endif //VDMPP
    ToolMediator::UpdateSelections();
    ToolMediator::Interf()->vdm_DestroyMeter();
    ToolMediator::Errs()->vdm_AllDone();
  }
  return Bool (succ);
}

// PrettyPrint
// fnm_l : seq1 of FileName
// ==> bool
Bool vdm_BaseTools::vdm_PrettyPrint (const type_21ProjectTypes_FileNameCL & fnm_l)
{
  bool succ = true;

  if (!fnm_l.IsEmpty())
  {
    ToolMediator::Errs()->vdm_ClearAll();

    ToolMediator::Interf()->vdm_InitMeter(SEQ<Char>(L"Pretty printing"), type_cL());
    ToolMediator::Interf()->vdm_SetMeterTotal(fnm_l.Length());
```

```
    size_t len_fnm_l = fnm_l.Length();
    for (size_t idx = 1; idx <= len_fnm_l; idx++)
    {
      SEQ<Char> label(L"Pretty printing ");
      label.ImpConc(SEQ<Char>(PTAUX::ExtractFileName(fnm_l[idx])));
      ToolMediator::Interf()->vdm_IncrementMeter(label);
      //
      succ = TOOLS::EvalLatex (fnm_l[idx]) && succ;
    }

    // The Tcl call to modules`selectFunctions has been removed from
    // modules`changeState in order to avoid the update of the VDM++
    // dependencies after each module/class being pretty printed.
    // UpdateSelections performs a callback to modules`selectFunctions.
    ToolMediator::UpdateSelections();

    ToolMediator::Interf()->vdm_DestroyMeter();

    ToolMediator::Errs()->vdm_AllDone();
  }
  return Bool (succ);
}

// ClassDepend
// nm : ModuleName
// ==> bool
Bool vdm_BaseTools::vdm_ClassDepend(const TYPE_ProjectTypes_ModuleName & nm)
{
#ifdef VDMSL
  return Bool(true);
#endif // VDMSL
#ifdef VDMPP
  return ToolMediator::EvalDependInfo(nm);
#endif // VDMPP
}

// InhTree
// ==>  bool
Bool vdm_BaseTools::vdm_InhTree()
{
#ifdef VDMSL
  return Bool(true);
#endif // VDMSL
#ifdef VDMPP
  return ToolMediator::EvalInhTree();
#endif // VDMPP
}

// Depend
```

```
// ParseAndEvalExprs
// expr : ToolCommand | FileName
// ==> bool * seq of [token]
type_bUL2P vdm_BaseTools::vdm_ParseAndEvalExprs(const Record & expr)
{
  return TBDEBUG::ParseAndFullyEvalExprs (expr, vdm_log, SEQ<Char>(L"---------"));
}

// ParseAndDebugExprs
// expr : ToolCommand | FileName
// ==>  EvalState * seq of [token]
type_UUL2P vdm_BaseTools::vdm_ParseAndDebugExprs(const Record & expr)
{
  return TBDEBUG::ParseAndStartEvalExprs (expr, vdm_log, SEQ<Char>(L"---------"));
}

// InitInterpreter
// ==> bool
Bool vdm_BaseTools::vdm_InitInterpreter()
{
  return TBDEBUG::InitCurrentDefinition(false, vdm_iplog);
}

// ExecuteCommand
// command : ToolCommand
// ==> bool
Bool vdm_BaseTools::vdm_ExecuteCommand (const TYPE_ProjectTypes_ToolCommand & command)
{
  return TOOLS::ExecuteCommand (command);
}

/**
 * sets a breakpoint and returns the number of the breakpoint
 * @param mod name of module
 * @param nm name of expression
 * @returns -1 on failure
 */
// SetBreakOnName
// mod : ModuleName
// nm : Name
// ==> nat
Int vdm_BaseTools::vdm_SetBreakOnName(const TYPE_ProjectTypes_ModuleName & mod, const TYPE
{
  return Int (TBDEBUG::EvalBreakOnName(PTAUX::ExtractModuleName(mod) + L"`" + PTAUX::Extra
}

// sets a breakpoint and returns the number of the breakpoint
// @param file name of the file that contains the specified line
// @param line line number
// @param col column number
```

```
// @returns -1 on failure
// SetBreakOnPos
// file : FileName
// line : nat
// col : nat
// ==> nat
Int vdm_BaseTools::vdm_SetBreakOnPos(const TYPE_ProjectTypes_FileName & file, const Int &
{
  std::wstring lineStr (line.ascii());
  std::wstring colStr (col.ascii());

  std::wstring fileName;
  TYPE_ProjectTypes_String seq (file.get_nm());
  if (!seq.GetString (fileName))
    return -1;

  return Int (TBDEBUG::EvalBreakOnPos(fileName, lineStr, colStr, vdm_iplog));
}

// deletes a breakpoint
// @param num Number of breakpoint, returned by SetBreakPointBy[Pos|Name]
// DeleteBreakPoint
// num : nat
// ==> ()
void vdm_BaseTools::vdm_DeleteBreakPoint (const Int & num)
{
  TBDEBUG::EvalDeleteBreak (num.ascii(), vdm_iplog);
}

// DebugStep
// ==> EvalState * [seq of token]
type_UU2P vdm_BaseTools::vdm_DebugStep ()
{
  return TBDEBUG::EvalDebugStep(vdm_iplog);
}

// DebugStepIn
// ==> EvalState * [seq of token]
type_UU2P vdm_BaseTools::vdm_DebugStepIn ()
{
  return TBDEBUG::EvalDebugStepIn(vdm_iplog);
}

// DebugSingleStep
// ==> EvalState * [seq of token]
type_UU2P vdm_BaseTools::vdm_DebugSingleStep ()
{
  return TBDEBUG::EvalDebugSingleStep(vdm_iplog);
}
```

```
// DebugContinue
// ==> EvalState * [seq of token]
type_UU2P vdm_BaseTools::vdm_DebugContinue ()
{
  return TBDEBUG::EvalDebugContinue(vdm_iplog);
}

// InitToolbox
// ==> ()
void vdm_BaseTools::vdm_InitToolbox()
{
// 20081027
//  InitToolbox(true);
  TOOLS::InitToolbox(TOOLS::isBatchMode());
}

// SaveTypeCheckEnv
// stor : StateStore
// ==> bool
Bool vdm_BaseTools::vdm_SaveTypeCheckEnv (const type_ref_StateStore & stor)
{
#ifdef VDMPP
  Tuple env (TOOLS::SaveTypeCheckEnv());
  TYPE_ProjectTypes_TCEnv tcenv (PTAUX::mk_TCEnv (env));
  Bool ok (ObjGet_vdm_StateStore (stor)->vdm_WriteTCEnv (tcenv));
  return ok;
#endif // VDMPP
#ifdef VDMSL
  return Bool (true);
#endif // VDMSL
}

// LoadTypeCheckEnv
// stor : StateStore
// ==> bool
Bool vdm_BaseTools::vdm_LoadTypeCheckEnv (const type_ref_StateStore & stor)
{
#ifdef VDMPP
  Generic tcenv (ObjGet_vdm_StateStore (stor)->vdm_ReadTCEnv ());
  if (tcenv.IsRecord ()) {
    Tuple env (PTAUX::ExtractTCEnv (tcenv));
    return TOOLS::LoadTypeCheckEnv (env);
  }
  else
    return Bool (false);
#endif // VDMPP
#ifdef VDMSL
  return Bool (true);
#endif // VDMSL
}
```

```
// UpdateToolbox
// ==> ()
void vdm_BaseTools::vdm_UpdateToolbox()
{
  TOOLS::UpdateToolbox();
}

// CheckForModifiedFiles
// () ==> ()
void vdm_BaseTools::vdm_CheckForModifiedFiles ()
{
  ToolMediator::CheckForModifiedFiles ();
}

// SetPriorityFile
// fn : FileName
// ==> ()
void vdm_BaseTools::vdm_SetPriorityFile(const TYPE_ProjectTypes_FileName& fn)
{
#ifdef VDMPP
  TBDEBUG::EvalPrioorityfile(PTAUX::ExtractFileName(fn));
#endif //VDMPP
}

// SetPrimarySchedulingAlgorithm
// nm : Name
// ==> ()
void vdm_BaseTools::vdm_SetPrimarySchedulingAlgorithm(const TYPE_ProjectTypes_Name &nm)
{
#ifdef VDMPP
  TOOLS::SetPrimarySchedulingAlgorithm(PTAUX::ExtractName(nm));
#endif //VDMPP
}

// SetTimeFile
// fn : FileName ==> ()
void vdm_BaseTools::vdm_SetTimeFile(const TYPE_ProjectTypes_FileName & fn)
{
#ifdef VICE
  TOOLS::EvalTimefile(PTAUX::ExtractFileName(fn));
#endif //VICE
}

// returns a tuple, holding a boolean value that
// indicates if the operation was successful and
// the name of the current module

// GetCurrentModule
// ==> bool * [ModuleName]
```

```
type_bU2P vdm_BaseTools::vdm_GetCurrentModule ()
{
  return TBDEBUG::GetCurrentModule();
}

// pops a module from the stack
// returns true if successful
// PopModule
// ==> bool
Bool vdm_BaseTools::vdm_PopModule ()
{
  return TBDEBUG::EvalPopModule (vdm_iplog);
}

// Pushes a module onto the stack
// returns true if successful
// PushModule
// name : ModuleName
// ==> bool
Bool vdm_BaseTools::vdm_PushModule (const TYPE_ProjectTypes_ModuleName & name)
{
  return TBDEBUG::EvalPushModule (name, vdm_iplog);
}

// GetPossibleInterfaces
// ==> set of ModuleName
type_23ProjectTypes_ModuleNameCS vdm_BaseTools::vdm_GetPossibleInterfaces()
{
#ifdef VDMPP
// 20110603 -->
  SET<TYPE_ProjectTypes_ModuleName> nms (PTAUX::ASNameSet2ModuleNameSet(ToolMediator::GetA
  SEQ<TYPE_ProjectTypes_ModuleName> vdmModules;
  Generic mn;
  for (bool bb = nms.First(mn); bb; bb = nms.Next(mn))
  {
    status_type st = PTAUX::ExtractStatus(ToolMediator::Status(mn));
    switch (st.type) {
      case status_type::status_ok:
      case status_type::status_pos:
      case status_type::status_def: {
        break;
      }
      case status_type::status_error:
      case status_type::status_none:
      default: {
        vdmModules.ImpAppend(mn);
        break;
      }
    }
  }
```

```
  // save POS/DEF setting
  bool the_Setting_DEF = Settings.IsDEF();
  Settings.DefOff();

  bool succ = true;
  if (!vdmModules.IsEmpty())
    succ = ToolMediator::BTools()->vdm_TypeCheck(vdmModules);
  else if (!nms.IsEmpty()) // 20071025
    succ = true;

  // restore POS/DEF setting
  if(the_Setting_DEF)
    Settings.DefOn();
  else
    Settings.DefOff();

// <-- 20110603
  GetCGTools().ResetCG(Quote(L"JAVA"));
  TOOLS::ResetSSNames();
  SET<TYPE_AS_Name> possInterfaces (GetCGTools().EvalPossibleInterfaces());
  SET<TYPE_ProjectTypes_ModuleName> returnset (PTAUX::ASNameSet2ModuleNameSet(possInterfac
  return returnset;
#endif // VDMPP
#ifdef VDMSL
  return SET<TYPE_ProjectTypes_ModuleName>();
#endif // VDSLM
}

// ResetInterfaces
// ==> ()
void vdm_BaseTools::vdm_ResetInterfaces()
{
#ifdef VDMPP
  GetCGTools().ResetJCGInterfaces();
#endif //VDMPP
}

// JavaSyntaxCheck
// filename_l : seq1 of FileName
// ==> bool
Bool vdm_BaseTools::vdm_JavaSyntaxCheck(const type_21ProjectTypes_FileNameCL & filename_l)
{
  bool succ = true;
#ifdef VDMPP
  if (!filename_l.IsEmpty())
  {
    ToolMediator::Errs()->vdm_ClearAll();

    ToolMediator::Interf()->vdm_InitMeter(SEQ<Char>(L"Syntax Checking"), type_cL());
```

```
    ToolMediator::Interf()->vdm_SetMeterTotal(filename_l.Length());

    SET<TYPE_ProjectTypes_FileName> parsedfiles_s (ToolMediator::ParsedFiles ());
    if (!parsedfiles_s.IsEmpty() && parsedfiles_s.SubSet (filename_l.Elems ())) {
      ToolMediator::DisableSession ();
    }

    size_t len_filename_l = filename_l.Length();
    for (size_t idx = 1; idx <= len_filename_l; idx++)
    {
      const TYPE_ProjectTypes_FileName & filenm (filename_l[idx]);
      // Update meter
      SEQ<Char> label(L"Syntax checking ");
      label.ImpConc(SEQ<Char>(PTAUX::ExtractFileName(filenm)));
      ToolMediator::Interf()->vdm_IncrementMeter(label);
      //
      try {
        succ = JavaGenTools::EvalJavaParse (filenm) && succ;
      }
      catch(TB_Exception & e)
      {
        vdm_log << L"Runtime Error: " << filenm.get_nm ().GetString() << endl << flush;
      }
    }
    ToolMediator::UpdateSelections();

    ToolMediator::Interf()->vdm_DestroyMeter();

    ToolMediator::Errs()->vdm_AllDone();
  }
#endif //VDMPP
  return Bool (succ);
}

// JavaTypeCheck
// mnm_l : seq1 of ModuleName
// ==> bool
Bool vdm_BaseTools::vdm_JavaTypeCheck(const type_23ProjectTypes_ModuleNameCL& mnm_l)
{
  bool succ = true;
#ifdef VDMPP
  if (!mnm_l.IsEmpty())
  {
    ToolMediator::Errs()->vdm_ClearAll();

    ToolMediator::Interf()->vdm_InitMeter(SEQ<Char>(L"Java type Checking"), type_cL());
    ToolMediator::Interf()->vdm_SetMeterTotal(mnm_l.Length());

    size_t len_mnm_l = mnm_l.Length();
    for (size_t idx =1; idx <= len_mnm_l; idx++)
```

```
    {
      const TYPE_ProjectTypes_ModuleName & mnm (mnm_l[idx]);
      SEQ<Char> label(L"Java type checking ");
      label.ImpConc(SEQ<Char>(PTAUX::ExtractModuleName(mnm)));
      ToolMediator::Interf()->vdm_IncrementMeter(label);
      //
      succ = JavaGenTools::EvalJavaTypeCheck (mnm) && succ;
    }
    ToolMediator::UpdateSelections();
    JavaGenTools::AfterJavaTypeCheck();

    ToolMediator::Interf()->vdm_DestroyMeter();

    ToolMediator::Errs()->vdm_AllDone();
  }
#endif //VDMPP
  return Bool (succ);
}

// JavaGenerateVDM
// mnm_l : seq1 of ModuleName
// p_stubs : bool
// p_rename : bool
// strans : set of nat
// etrans : set of nat
// ==> bool
Bool vdm_BaseTools::vdm_JavaGenerateVDM(const type_23ProjectTypes_ModuleNameCL& mnm_l,
                                        const Bool & p_stubs,
                                        const Bool & p_rename,
                                        const Bool & p_trans)
{
  bool succ = true;
#ifdef VDMPP
  if (!mnm_l.IsEmpty())
  {
    ToolMediator::Errs()->vdm_ClearAll();

    ToolMediator::Interf()->vdm_InitMeter(SEQ<Char>(L"Generating VDM++"), type_cL());
    ToolMediator::Interf()->vdm_SetMeterTotal(mnm_l.Length());

    size_t len_mnm_l = mnm_l.Length();
    for (size_t idx = 1; idx <= len_mnm_l; idx++)
    {
      const TYPE_ProjectTypes_ModuleName & mnm (mnm_l[idx]);
      SEQ<Char> label(L"Generating VDM++ for ");
      label.ImpConc(SEQ<Char>(PTAUX::ExtractModuleName(mnm)));
      ToolMediator::Interf()->vdm_IncrementMeter(label);
      //
      succ = JavaGenTools::EvalJavaGenerateVDM (mnm, p_stubs, p_rename, p_trans) && succ;
    }
```

```
    ToolMediator::UpdateSelections();

    ToolMediator::Interf()->vdm_DestroyMeter();

    ToolMediator::Errs()->vdm_AllDone();
  }
#endif //VDMPP
  return Bool (succ);
}

// vdm_NewUnnamedProject
void vdm_BaseTools::vdm_NewUnnamedProject ()
{
#ifdef VDMPP
  TOOLS::EvalDestroyAllObjects();
#endif // VDMPP
  TOOLS::InitToolbox( false ); // 20051110
  ToolMediator::ClearAll(); // 20051110
  GetCI().clear(); // 20060123

  ToolMediator::NewUnnamedProject();

#ifdef VDMPP
  ToolMediator::UMLT()->vdm_ResetMapper();
#endif // VDMPP

  init_POG_uMEDIATOR();
}

// vdm_LoadProject
void vdm_BaseTools::vdm_LoadProject (const TYPE_ProjectTypes_FileName & pnm)
{
#ifdef VDMPP
  TOOLS::EvalDestroyAllObjects();
#endif // VDMPP
  TOOLS::InitToolbox( false ); // 20051110
  ToolMediator::ClearAll(); // 20051110
  GetCI().clear(); // 20060123

  try {
    ToolMediator::Open(pnm);
  }
  catch(TB_Exception & e)
  {
    vdm_log << L"Runtime Error: " << pnm.get_nm ().GetString() << endl << flush;
  }

  init_POG_uMEDIATOR();

  TOOLS::set_spec_init( false ); // 20051222
```

```
  wstring projectFile(PTAUX::ExtractFileName (pnm));
  wstring basedir (TBUTILS::tb_getbasedir(projectFile));
  TBUTILS::SetDefaultPath(basedir);
  TBUTILS::tb_chdir(basedir);

#ifdef VDMPP
  ToolMediator::UMLT()->vdm_ResetMapper();
#endif // VDMPP
}

// vdm_AddFiles
void vdm_BaseTools::vdm_AddFiles (const type_21ProjectTypes_FileNameCS & fnm_s)
{
  ToolMediator::AddFiles ( fnm_s );

  TOOLS::set_spec_init( false ); // 20051222

// 20121121 -->
  if (!fnm_s.IsEmpty() && ToolMediator::BTools()->vdm_GetProjectName().IsNil())
  {
    wstring file (PTAUX::ExtractFileName (fnm_s.GetElem()));
    wstring basedir (TBUTILS::tb_getbasedir(file));
    TBUTILS::SetDefaultPath(basedir);
    TBUTILS::tb_chdir(basedir);
  }
// <-- 20121121
}

// vdm_RemoveFiles
void vdm_BaseTools::vdm_RemoveFiles (const type_21ProjectTypes_FileNameCS & fnm_s)
{
  ToolMediator::RemoveFiles ( fnm_s );

// 20120521 -->
#ifdef VDMPP
  TOOLS::CheckSSParseEnv();
#endif // VDMPP
  SET<TYPE_ProjectTypes_ModuleName> nms (PTAUX::ASNameSet2ModuleNameSet(ToolMediator::GetA
  Generic mnm;
  for (bool cc = nms.First(mnm); cc; cc = nms.Next(mnm))
  {
    ToolMediator::UpdateTC (mnm, Quote (L"NONE"));
  }
// <-- 20120521

  // breakpoint
  // source
  TOOLS::set_spec_init( false ); // 20051222
}
```

```
// vdm_SaveProjectAs
Bool vdm_BaseTools::vdm_SaveProjectAs (const TYPE_ProjectTypes_FileName & pnm)
{
  Bool result(ToolMediator::SaveAs(pnm));
  if (result.GetValue())
  {
    wstring projectFile(PTAUX::ExtractFileName (pnm));
    wstring basedir (TBUTILS::tb_getbasedir(projectFile));
    TBUTILS::SetDefaultPath(basedir);
    TBUTILS::tb_chdir(basedir);
  }
  return result;
}

// vdm_SaveProject
Bool vdm_BaseTools::vdm_SaveProject (const TYPE_ProjectTypes_FileName & pnm)
{
  Bool result(ToolMediator::SaveAs(pnm));
  if (result.GetValue())
  {
    wstring projectFile(PTAUX::ExtractFileName (pnm));
    wstring basedir (TBUTILS::tb_getbasedir(projectFile));
    //TBUTILS::SetDefaultPath(basedir);
    TBUTILS::tb_chdir(basedir);
  }
  return result;
}

Generic vdm_BaseTools::vdm_GetProjectName()
{
  return ToolMediator::GetProjectName();
}

// PogGenerate
// mnm_l : seq1 of ModuleName
// ==> bool
Bool vdm_BaseTools::vdm_PogGenerate (const type_23ProjectTypes_ModuleNameCL & module_l)
{
  bool succ = true;

  if (!module_l.IsEmpty())
  {
    SEQ<TYPE_ProjectTypes_ModuleName> vdmModules;

    size_t len_module_l = module_l.Length();
    for (size_t index = 1; index<= len_module_l; index++)
    {
      const TYPE_ProjectTypes_ModuleName & mn (module_l[index]);
      status_type st = PTAUX::ExtractStatus(ToolMediator::Status(mn));
```

```
  switch (st.type) {
    case status_type::status_ok:
    case status_type::status_pos:
    case status_type::status_def: {
      break;
    }
    case status_type::status_error:
    case status_type::status_none:
    default: {
      vdmModules.ImpAppend(mn);
      break;
    }
  }
}

// save POS/DEF setting
bool the_Setting_DEF = Settings.IsDEF();
Settings.DefOff();

if (!vdmModules.IsEmpty())
  succ = ToolMediator::BTools()->vdm_TypeCheck(vdmModules);
else if (!module_l.IsEmpty()) // 20071025
  succ = true;

// restore POS/DEF setting
if(the_Setting_DEF)
  Settings.DefOn();
else
  Settings.DefOff();

if (succ)
{
  //postGUIEvent(new PogCleanUpEvent(modnmList));
  ToolMediator::Interf()->vdm_CallBack(TYPE_ProjectTypes_PogCleanUp().Init(module_l));

  ToolMediator::Interf()->vdm_InitMeter((SEQ<Char>) Sequence(wstring(L"Generating inte
  ToolMediator::Interf()->vdm_SetMeterTotal(len_module_l);

  PogInterface & pog = GetPogInterface();
  for (size_t index = 1; index<= len_module_l; index++)
  {
    TYPE_ProjectTypes_String moduleName (module_l[index].get_nm());
    Sequence label(L"Generating integrity property for ");
    label.ImpConc(moduleName);
    ToolMediator::Interf()->vdm_IncrementMeter(label);

    vdm_log << L"Generating integrity property for " << moduleName.GetString() << L" .

    try
    {
```

```
  pog.setup();
  pog.genPO(mk_sequence(ASTAUX::MkNameFromId(moduleName, NilContextId)));
  //PogInterface::setup();
  //PogInterface::genPO(mk_sequence(ASTAUX::MkNameFromId(moduleName, NilContextId)
}
catch(TB_Exception & e)
{
  vdm_log << L"Runtime Error: " << moduleName.GetString() << endl << flush;
  continue;
}
Sequence pogseq (pog.getTextPOs ());
//Sequence pogseq (PogInterface::getTextPOs ());
if (!pogseq.IsEmpty())
{
  size_t len_pogseq = pogseq.Length();
  if (len_pogseq == 1)
    vdm_log << L"1 property" << endl;
  else
    vdm_log << len_pogseq << L" properties" << endl;

  for (size_t idx = 1; idx <= len_pogseq; idx++)
  {
    Tuple t (pogseq[idx]);
    //postGUIEvent(new PogAddEvent(Qt2TB::wstring2qstring(t.GetSequence(1).GetStri
    //                             Qt2TB::wstring2qstring(t.GetSequence(2).GetStri
    //                             Qt2TB::wstring2qstring(t.GetSequence(3).GetStri
    //                             Qt2TB::wstring2qstring(t.GetSequence(4).GetStri
    //                             Qt2TB::wstring2qstring(t.GetSequence(5).GetStri
    //                             t.GetIntValue(6),                    // no
    //                             Qt2TB::wstring2qstring(t.GetSequence(7).GetStri
    //                             Qt2TB::wstring2qstring(t.GetSequence(12).GetStr
    //                             t.GetIntValue(8),                    // line
    //                             t.GetIntValue(9),                    // column
    //                             t.GetIntValue(10),                   // length
    //                             Qt2TB::wstring2qstring(t.GetSequence(11).GetStr
    //                            )
    //            );
    ToolMediator::Interf()->vdm_CallBack(TYPE_ProjectTypes_PogAdd().Init(t.GetSequ
                                                                         t.GetSequ
                                                                         t.GetSequ
                                                                         t.GetSequ
                                                                         t.GetInt(
                                                                         t.GetSequ
                                                                         t.GetSequ
                                                                         t.GetInt(
                                                                         t.GetInt(
                                                                         t.GetInt(
                                                                         t.GetSequ
  }
```

```
        }
      }
      //postGUIEvent(new PogUpdateFilterEvent());
      ToolMediator::Interf()->vdm_CallBack(TYPE_ProjectTypes_PogUpdateFilter());

      ToolMediator::Interf()->vdm_DestroyMeter();
      vdm_log << L"done" << flush;
    }
  }
  return Bool(succ);
}
```

A.2 Class Dependency

```
/***
*  * WHAT
*  *    Implementation of preliminary methods in class Dependency
*  * FILE
*  *    $Source: /home/vdmtools/cvsroot/toolbox/code/specman/code/Dependency_userimpl.cc,v
*  * VERSION
*  *    $Revision:
*  * DATE
*  *    $Date: 2001/06/12 15:04:56 $
*  * STATUS
*  *    Under development
*  * REFERENCES
*  *    IFAD-VDM28
*  * PROJECT
*  *    Toolbox
*  * AUTHOR
*  *    $Author: paulm $
*  * COPYRIGHT
*  *    (C) 2007 CSK, Japan
***/

#include "intconvquotes.h"

#include "AS.h"
#ifdef VDMPP
#include "DEP.h"
#endif //VDMPP
#include "StateStore.h"

#include "Dependency.h"
#include "projectval.h"
#include "tbutils.h"
```

```
void vdm_Dependency::vdm_Init (const type_ref_Repository & r)
{
#ifdef VDMPP
  init_DEP();
#endif //VDMPP
  vdm_rep = r;
}

void vdm_Dependency::vdm_Depend()
{
#ifdef VDMPP
  vdm_DEP_Update();
#endif // VDMPP
}

type_23ProjectTypes_ModuleNameCS vdm_Dependency::vdm_GetAllSubClasses(const type_23Project
{
#ifdef VDMPP
  SET<TYPE_ProjectTypes_ModuleName> nm_s_q (nm_s);
  SET<TYPE_AS_Name> subclass_s;
  Generic g;
  for (bool bb = nm_s_q.First (g); bb; bb = nm_s_q.Next(g)) {
    subclass_s.ImpUnion (vdm_DEP_AllSubClasses (PTAUX::ModuleName2ASName (g)));
  }
  SET<TYPE_ProjectTypes_ModuleName> returnset_s (PTAUX::ASNameSet2ModuleNameSet (subclass_
  return returnset_s;
#else
  return SET<TYPE_ProjectTypes_ModuleName>();
#endif //VDMPP
}

type_23ProjectTypes_ModuleNameCS  vdm_Dependency::vdm_GetAllUsedBy (const type_23ProjectTy
{
#ifdef VDMPP
  SET<TYPE_AS_Name> usedby_s;
  SET<TYPE_ProjectTypes_ModuleName> nm_s_q (nm_s);
  Generic g;
  for (bool bb = nm_s_q.First (g); bb; bb = nm_s_q.Next(g)) {
    usedby_s.ImpUnion (vdm_DEP_AllClients (PTAUX::ModuleName2ASName (g)));
  }
  SET<TYPE_ProjectTypes_ModuleName> returnset_s (PTAUX::ASNameSet2ModuleNameSet (usedby_s)
  return returnset_s;
#else
  return SET<TYPE_ProjectTypes_ModuleName> ();
#endif //VDMPP
}

void vdm_Dependency::vdm_UpdateGraph (const type_19ProjectTypes_ModuleCL & nm_l)
{
#ifdef VDMPP
```

```
  // Sequence of Modules
  SEQ<TYPE_AS_Class> depupd;
  size_t len_nm_l = nm_l.Length();
  for (size_t idx = 1; idx <= len_nm_l; idx++)
  {
    const TYPE_ProjectTypes_Module & mod (nm_l[idx]);
    Generic ast (PTAUX::ExtractModuleAst (mod));
    depupd.ImpAppend (INT2Q::h2gAS (ast));
  }
  vdm_DEP_UpdateSC(depupd);
#endif //VDMPP
}

void vdm_Dependency::vdm_Remove (const type_23ProjectTypes_ModuleNameCS & nm_s)
{
#ifdef VDMPP
  SET<TYPE_ProjectTypes_ModuleName> nm_s_q (nm_s);
  SET<TYPE_AS_Name> asnm_s;
  Generic g;
  for (bool bb = nm_s_q.First (g); bb; bb = nm_s_q.Next (g))
    asnm_s.Insert (PTAUX::ModuleName2ASName (g));
  vdm_DEP_Remove (asnm_s);
#endif //VDMPP
}

type_23ProjectTypes_ModuleNameCS vdm_Dependency::vdm_GetSub (const TYPE_ProjectTypes_Modul
{
#ifdef VDMPP
  SET<TYPE_AS_Name> res (vdm_DEP_GetSub (PTAUX::ModuleName2ASName (nm)));
  return PTAUX::ASNameSet2ModuleNameSet (res);
#else
  return SET<TYPE_ProjectTypes_ModuleName>();
#endif //VDMPP
}

type_23ProjectTypes_ModuleNameCS vdm_Dependency::vdm_GetSuper (const TYPE_ProjectTypes_Mod
{
#ifdef VDMPP
  SET<TYPE_AS_Name> res (vdm_DEP_GetSuper (PTAUX::ModuleName2ASName (nm)));
  return PTAUX::ASNameSet2ModuleNameSet(res);
#else
  return SET<TYPE_ProjectTypes_ModuleName>();
#endif //VDMPP
}

type_23ProjectTypes_ModuleNameCS vdm_Dependency::vdm_GetUsedBy (const TYPE_ProjectTypes_Mo
{
#ifdef VDMPP
  SET<TYPE_AS_Name> res (vdm_DEP_GetClients (PTAUX::ModuleName2ASName (nm)));
  return PTAUX::ASNameSet2ModuleNameSet(res);
```

```
#else
  return SET<TYPE_ProjectTypes_ModuleName>();
#endif //VDMPP
}

type_23ProjectTypes_ModuleNameCS vdm_Dependency::vdm_GetUses (const TYPE_ProjectTypes_Modu
{
#ifdef VDMPP
  SET<TYPE_AS_Name> res (vdm_DEP_GetServers (PTAUX::ModuleName2ASName (nm)));
  return PTAUX::ASNameSet2ModuleNameSet(res);
#else
  return SET<TYPE_ProjectTypes_ModuleName>();
#endif //VDMPP
}

Bool vdm_Dependency::vdm_Save (const type_ref_StateStore & stor)
{
#ifdef VDMPP
  TYPE_DEP_DependencyGraph gr (vdm_DEP_ExportDepGraph());
  TYPE_ProjectTypes_DepGraph dep (PTAUX::mk_DepGraph (gr));
  Bool ok (ObjGet_vdm_StateStore (stor)->vdm_WriteDep (dep));//cast LTO ???
  return ok;
#else
  return Bool (true);
#endif //VDMPP
}

Bool vdm_Dependency::vdm_Load (const type_ref_StateStore & store)
{
#ifdef VDMPP
  Generic dep (ObjGet_vdm_StateStore (store)->vdm_ReadDep ());
  if (dep.IsRecord ()) {
    TYPE_DEP_DependencyGraph gr (PTAUX::ExtractDepGraph (dep));
    vdm_DEP_ImportDepGraph (gr);
    return Bool (true);
  }
  return Bool (false);
#else
  return Bool (true);
#endif //VDMPP
}

Bool vdm_Dependency::vdm_IsCyclic (const TYPE_ProjectTypes_ModuleName & nm)
{
#ifdef VDMPP
  Bool cyc (vdm_DEP_IsCyclic (PTAUX::ModuleName2ASName (nm)));
  return cyc;
#else
  return Bool (false);
#endif // VDMPP
```

```
}

type_23ProjectTypes_ModuleNameCSL vdm_Dependency::vdm_OrderOfProcess(const TYPE_ProjectTyp
{
#ifdef VDMPP
  type_7AS_NameCSL asnm_l (vdm_DEP_OrderOfProcess(PTAUX::ModuleName2ASName (nm)));
  type_23ProjectTypes_ModuleNameCSL modnm_l;
  size_t len_asnm_l = asnm_l.Length();
  for (size_t idx = 1; idx <= len_asnm_l; idx++)
    modnm_l.ImpAppend (PTAUX::ASNameSet2ModuleNameSet (asnm_l[idx]));
  return modnm_l;  // seq of set of ProjectTypes`ModuleName
#else
  return type_23ProjectTypes_ModuleNameCSL();
#endif //VDMPP
}
```

A.3 Class StateStore

```
/***
*  * WHAT
*  *    Implementation of preliminary methods in class StateStore
*  * FILE
*  *    $Source: /home/vdmtools/cvsroot/toolbox/code/specman/code/StateStore_userimpl.cc,v
*  * VERSION
*  *    $Revision: 1.11 $
*  * DATE
*  *    $Date: 2006/02/20 01:50:44 $
*  * STATUS
*  *    Under development
*  * REFERENCES
*  *    IFAD-VDM28
*  * PROJECT
*  *    Toolbox
*  * AUTHOR
*  *    Henrik Voss + $Author: vdmtools $
*  * COPYRIGHT
*  *    (C) 2007 CSK, Japan
***/

#include "StateStore.h"
#include "projectval.h"
#include "tbutils.h"
#include "tb_wstring.h"

#include <fstream>
#include <string>
```

```
#ifdef VDMSL
static Quote project_id = Quote (L"ProjectFile");
#endif // VDMSL
#ifdef VDMPP
static Quote project_id = Quote (L"ProjectFilePP");
#endif // VDMPP

int StateStoreAux::current_ast_version = 3;
std::wstring StateStoreAux::projectfile;
StateStoreAux::stasto_type StateStoreAux::stype;
Sequence StateStoreAux::ProjectStore;

void StateStoreAux::AppendItem(const Generic & e)
{
  ProjectStore.ImpAppend(e);
}

Generic StateStoreAux::GetHeadItem()
{
  if (!ProjectStore.IsEmpty ()) {
    Generic e = ProjectStore.Hd();
    ProjectStore.ImpTl();
    return e;
  }
  else
    return Nil();
}

void StateStoreAux::ClearItems()
{
  ProjectStore.Clear();
}

void StateStoreAux::SetItems(const Sequence & s)
{
  ProjectStore = s;
}

Sequence StateStoreAux::GetItems()
{
  return ProjectStore;
}

void StateStoreAux::SetProjectFilename(const std::wstring & fnm)
{
  projectfile = fnm;
}

std::wstring StateStoreAux::GetProjectFilename()
{
```

```
  return projectfile;
}

void StateStoreAux::SetReadMode()
{
  stype = stasto_read;
}

void StateStoreAux::SetWriteMode()
{
  stype = stasto_write;
}

bool StateStoreAux::IsWriteMode()
{
  return (stype == stasto_write);
}

int StateStoreAux::ExtractVersion(const Generic & project, int current_version)
{
  if (project.IsSequence() && Sequence (project).Length() > 1) {
    Sequence l = project;
    Generic id = l[1];
    Generic num = l[2];
    if (id.IsQuote() && (id == project_id) && num.IsInt()) {
      int v = Int(num).GetValue();
      if (v > 1 && v <= current_version)
        return v;
    }
  }
  else if (project.IsTuple () && Tuple (project).Length () == 2) {
    Generic version = Tuple (project).GetField (1);
    Generic ast = Tuple (project).GetField (2);
    if (version == Int (2) && ast.IsSequence ())
      return 1; // project version 1 identified
  }

  return 0; // invalid project
}

Sequence StateStoreAux::ConvertProject (int version, Generic project, int cur_ver)
{
  switch(version) {
    case 1: {
      Sequence old_project = Tuple(project).GetField(2);
      Sequence new_project;

      new_project.ImpAppend(Quote(L"ProjectFile"));
      new_project.ImpAppend(Int(2));
```

```
    if (old_project.Length() > 0)
      old_project.ImpTl(); // Remove version number

    // Remove the module environment map
    size_t len_old_project = old_project.Length();
    for (size_t idx = 1; idx <= len_old_project; idx++)
    {
      const Generic & e (old_project[idx]);
      if (e.IsMap())
        continue; // don't include the module environment map
      else if (e.IsRecord())
        // potential problem with ast version
        new_project.ImpAppend(StateStoreAux::WriteAST(e));
      else
        new_project.ImpAppend(e);
    }
    return new_project;
  }
  case 2: {
    // Version before AST restructuring.
    // Will remove AST, keep list of files in project.
    // Find files in version 2 project
    Sequence ConvProj;
    Sequence proj = project;
    size_t len_proj = proj.Length();
    for (size_t idx = 1; idx <= len_proj; idx++)
    {
      const Generic & e (proj[idx]);
      if (e.IsTuple() && Tuple(e).Length () == 2)
      {
        Tuple val (e);
        if (val.GetField (1).IsToken()) {
          Token tp = val.GetField (1);
          if (tp == Token(L"file"))
            ConvProj.ImpAppend (val);
        }
      }
    }
    // Prepend with number of files
    int numFiles = ConvProj.Length ();
    ConvProj.ImpPrepend (Int (numFiles));
    // Prepend with new version and Project file type
    ConvProj.ImpPrepend (Int (cur_ver));
    ConvProj.ImpPrepend (Sequence (project)[1]);
    return ConvProj;
  }
  default: {
    return Sequence(); // unknown version
  }
}
```

```
}

Sequence StateStoreAux::AbsToRelPath(const Sequence & store, const std::wstring & pname)
{
  Sequence ret;
  ret.ImpAppend( store[1] ); // id
  ret.ImpAppend( store[2] ); // version
  ret.ImpAppend( store[3] ); // number of files
  int n = (Int)store[3];
  for( int i = 0; i < n; i++ )
  {
    Tuple t (store[ i + 4 ]);
    if (Token(L"file") == t.GetField(1))
    {
      Token s (t.GetField(2));
      std::wstring afile (s.GetString()); // 20100616
      std::wstring rfile (TBUTILS::GetRelativePath( afile, pname ));
//      Tuple nt(2);
//      nt.SetField( 1, t.GetField(1) );
//      nt.SetField( 2, Token( rfile ) );
      t.SetField( 2, Token( rfile ) );
    }
//    ret.ImpAppend( nt );
    ret.ImpAppend( t );
  }
  return ret;
}

// RelToAbsPath
// store :
// pname :
// ==>
Sequence StateStoreAux::RelToAbsPath(const Sequence & store, const std::wstring & pname)
{
  Sequence ret;
  ret.ImpAppend( store[1] ); // id
  ret.ImpAppend( store[2] ); // version
  ret.ImpAppend( store[3] ); // number of files
  int n = (Int)store[3];
  for( int i = 0; i < n; i++ )
  {
    Tuple t (store[ i + 4 ]);
    if (Token(L"file") == t.GetField(1))
    {
      Token s (t.GetField(2));
      std::wstring rfile (s.GetString()); // 20100616
      std::wstring afile (TBUTILS::GetAbsolutePath( rfile, pname ));
//      Tuple nt(2);
//      nt.SetField( 1, t.GetField(1) );
//      nt.SetField( 2, Token( afile ) );
```

```
      t.SetField( 2, Token( afile ) );
    }
//    ret.ImpAppend( nt );
    ret.ImpAppend( t );
  }
  return ret;
}

/*
// WriteAST writes AST into the file name given by fnm result:
//   1 => success
//   2 => something went wrong. The AST has not written
bool TBUTILS::WriteAST (const Sequence& AST, const std::wstring& fnm)
{
  ofstream ostr;
  ostr.open(wstring2fsstr(fnm).c_str());
  if (!ostr.good ()) {
    vdm_log << L"Couldn't open file '" << fnm << L"'" << endl << flush;
    return false;
  }
  Tuple astrepos (2);
  astrepos.SetField (1, Int (current_ast_version));
  astrepos.SetField (2, AST);
  astrepos.WriteVal (ostr);
  ostr.close ();
  return true;
}
*/

Generic StateStoreAux::WriteAST (const Generic & AST)
{
  return mk_(Int (current_ast_version), AST);
}

/*
// ReadAST read an AST from the file name given by fnm
// result:
//   1 => success
//        Parameter AST will contain the resultion AST
//   2 => something went wrong. No AST has been read.
// side effects:
//   the global variable TestSuiteFile is updated with the name of the
//   file from which the AST was read.
bool TBUTILS::ReadAST (Sequence& AST, const std::wstring& fnm)
{
  ifstream istr (wstring2fsstr(fnm).c_str());
  if (istr) {
    Generic astrep = ReadVal (istr);
    if (!astrep.IsTuple () || Tuple (astrep).Length () != 2) {
      vdm_log << L"Content of '" << fnm << L"' is not a valid format"
```

```
              << endl << flush;
      return false;
    }
    Generic version = Tuple (astrep).GetField (1);
    Generic ast = Tuple (astrep).GetField (2);
    if (! (version == Int (current_ast_version)) || !ast.IsSequence ()) {
      vdm_log << L"Content of '" << fnm << L"' is outdated"
              << endl << flush;
      return false;
    }
    // We have a valid AST!
    AST = ast;
    //    TestSuiteFile = fnm;
    return true;
  }
  else {
    vdm_log << L"Couldn't open file '" << fnm << L"'" << endl << flush;
    return false;
  }
}
*/

Generic StateStoreAux::ReadAST (const Generic & store)
{
  if (!store.IsTuple () || Tuple (store).Length () != 2)
    return Nil();

  Generic version = Tuple (store).GetField (1);
  Generic ast = Tuple (store).GetField (2);
  if (! (version == Int (current_ast_version)) || !ast.IsRecord ())
    return Nil();

  // We (possible) have a valid AST!
  return ast;
}

// Open
// fnm : ProjectTypes`FileName
// tp : (<READ> | <WRITE>)
// ==> bool
Bool vdm_StateStore::vdm_Open(const TYPE_ProjectTypes_FileName &fnm, const Generic & tp)
{
  std::wstring file = PTAUX::ExtractFileName (fnm);

  if (tp == Quote (L"WRITE")) {
// FIX : 20050928
// if the projectfile exists and is read only
    std::ifstream istr (TBWSTR::wstring2fsstr(file).c_str());
    if( istr )
    {
```

```
      istr.close();
      std::ofstream ostr (TBWSTR::wstring2fsstr(file).c_str(), ios_base::app);
      if( !ostr ) return Bool (false);
    }
    StateStoreAux::ClearItems();
    StateStoreAux::SetWriteMode();
    StateStoreAux::SetProjectFilename(file);
    StateStoreAux::AppendItem(project_id);
    StateStoreAux::AppendItem(vdm_version);
    return Bool (true);
  }
  else {
    // (tp == Quote (L"READ"))
    std::ifstream istr (TBWSTR::wstring2fsstr(file).c_str());
    if (istr) {
      Generic astrep = ReadVal (istr);
      int version = StateStoreAux::ExtractVersion (astrep, vdm_version);
      if (version == 0) {
        vdm_log << L"Invalid project file " << file << std::endl;
        return Bool(false);
      }

      Sequence ss;
      if (version != vdm_version) {
        vdm_log << L"Project file " << file << L" is outdated (version " << version << L")
        vdm_log << L"Updating project to version " << vdm_version << std::endl;
        ss = StateStoreAux::ConvertProject(version, astrep, vdm_version);
      }
      else
        ss = astrep;

      // hack for rerative path 2005/11/16
      Sequence s (StateStoreAux::RelToAbsPath( ss, file ));
      Sequence s2 (s.ImpTl()); // remove id
      Sequence s3 (s2.ImpTl()); // remove version

      StateStoreAux::ClearItems();
      StateStoreAux::SetReadMode();
      StateStoreAux::SetItems(s3);

      return Bool (true);
    }
    else
      return Bool(false);
  }
}

// Close
// ==> ()
void vdm_StateStore::vdm_Close()
```

```
{
  if (StateStoreAux::IsWriteMode())
  {
    std::wstring projectfile (StateStoreAux::GetProjectFilename());
    ofstream ostr;
    ostr.open(TBWSTR::wstring2fsstr(projectfile).c_str());
    if (!ostr.good ())
      vdm_log << L"Couldn't open file '" << projectfile << std::endl;
    else {

      Sequence ss (StateStoreAux::GetItems());
      // hack for rerative path 2005/11/16
      Sequence s (StateStoreAux::AbsToRelPath( ss, projectfile ));
      s.WriteVal (ostr);
      ostr.close ();
    }
  }
  StateStoreAux::ClearItems();
}

// WriteASTVal
// val : ProjectTypes`AstVal | <NONE>
// ==> bool
Bool vdm_StateStore::vdm_WriteASTVal(const Generic & val)
{
  if (val.IsRecord())
    StateStoreAux::AppendItem(StateStoreAux::WriteAST(PTAUX::ExtractAstVal(val)));
  else
    StateStoreAux::AppendItem(val);
  return Bool (true);
}

// WriteName
// nm : ProjectTypes`FileName | ProjectTypes`ModuleName
// ==> bool
Bool vdm_StateStore::vdm_WriteName(const Record & nm)
{
  switch (nm.GetTag()) {
    case TAG_TYPE_ProjectTypes_ModuleName: {
      Tuple val (mk_(Token(L"module"), Token(PTAUX::ExtractModuleName(nm))));
      StateStoreAux::AppendItem(val);
      return Bool (true);
    }
    case TAG_TYPE_ProjectTypes_FileName: {
      Tuple val (mk_(Token(L"file"), Token(PTAUX::ExtractFileName(nm))));
      StateStoreAux::AppendItem(val);
      return Bool (true);
    }
    default: {
      return Bool (false);
```

```
    }
  }
}

// WriteSession
// session : ProjectTypes`SessionType
// ==> bool
Bool vdm_StateStore::vdm_WriteSession(const TYPE_ProjectTypes_SessionType & session)
{
  StateStoreAux::AppendItem(session);
  return Bool (true);
}

// WriteVal
// val : int | bool
// ==> bool
Bool vdm_StateStore::vdm_WriteVal(const Generic &val)
{
  StateStoreAux::AppendItem(val);
  return Bool (true);
}

// WriteStatus
// status : ProjectTypes`Status
// ==> bool
Bool vdm_StateStore::vdm_WriteStatus(const TYPE_ProjectTypes_Status & status)
{
  StateStoreAux::AppendItem(status);
  return Bool (true);
}

// WriteDep
// graph : ProjectTypes`DepGraph ==> bool
Bool vdm_StateStore::vdm_WriteDep (const TYPE_ProjectTypes_DepGraph & graph)
{
  StateStoreAux::AppendItem(graph);
  return Bool (true);
}

// WriteTCEnv
// e : ProjectTypes`TCEnv
//  ==> bool
Bool vdm_StateStore::vdm_WriteTCEnv (const TYPE_ProjectTypes_TCEnv & e)
{
  StateStoreAux::AppendItem(e);
  return Bool (true);
}

// ReadTCEnv
// ==> [ProjectTypes`TCEnv]
```

```
Generic vdm_StateStore::vdm_ReadTCEnv ()
{
  Generic e (StateStoreAux::GetHeadItem());
  if (e.Is(TAG_TYPE_ProjectTypes_TCEnv))
    return e;
  else
    return Nil();
}

// ReadDep
// ==> [ProjectTypes`DepGraph]
Generic vdm_StateStore::vdm_ReadDep ()
{
  Generic d (StateStoreAux::GetHeadItem());
  if (d.Is(TAG_TYPE_ProjectTypes_DepGraph))
    return d;
  else
    return Nil();
}

// ReadASTVal
// ==> [AstVal | <NONE>]
Generic vdm_StateStore::vdm_ReadASTVal()
{
  Generic val (StateStoreAux::GetHeadItem());
  if (!val.IsNil())
  {
    if (val == Quote(L"NONE"))
      return val;
    else {
      Generic ast (StateStoreAux::ReadAST(val));
      if (ast.IsRecord())
        return PTAUX::mk_AstVal(ast);
    }
  }
  return Nil();
}

// ReadName
// ==> [FileName | ModuleName]
Generic vdm_StateStore::vdm_ReadName()
{
  Generic g (StateStoreAux::GetHeadItem());
  if (g.IsTuple() && Tuple(g).Length () == 2)
  {
    Tuple val (g);
    Token tp (val.GetField (1));
    Token nm (val.GetField (2));
    if (tp == Token(L"module"))
      return PTAUX::mk_ModuleName (nm.GetString()); // 20100616
```

```
    else if (tp == Token(L"file"))
      return PTAUX::mk_FileName (nm.GetString()); // 20100616
  }
  return Nil();
}

// ReadSession
// ==> [SessionType]
Generic vdm_StateStore::vdm_ReadSession()
{
  Generic g (StateStoreAux::GetHeadItem());
  if (g == none_session || g == flat_session || g == struct_session)
    return g;
  else
    return Nil ();
}

// ReadVal
// ==> [int | bool]
Generic vdm_StateStore::vdm_ReadVal()
{
  Generic g (StateStoreAux::GetHeadItem());
  if (g.IsInt() || g.IsBool ())
    return g;
  else
    return Nil ();
}

// ReadStatus
// ==>  [Status]
Generic vdm_StateStore::vdm_ReadStatus()
{
  Generic g (StateStoreAux::GetHeadItem());
  if (g.IsQuote())
    return g;
  else
    return Nil ();
}
```

A.4 Class CGRepository

```
/***
*  * WHAT
*  *    Implementation of preliminary methods in class CGRepository
*  * FILE
*  *    $Source: /home/vdmtools/cvsroot/toolbox/code/specman/code/CGRepository_userimpl.co
*  * VERSION
*  *    $Revision: 1.4 $
```

```
*  * DATE
*  *    $Date: 2005/01/21 03:16:06 $
*  * STATUS
*  *    Under development
*  * REFERENCES
*  *    IFAD-VDM28
*  * PROJECT
*  *    Toolbox
*  * AUTHOR
*  *    Henrik Voss + $Author: vdmtools $
*  * COPYRIGHT
*  *    (C) 2007 CSK, Japan
***/

#include "CGRepository.h"
#include "projectval.h"
#include "tbutils.h"
#include "rt_errmsg.h"
#include "tb_exceptions.h"
#include "tb_wstring.h"
#include <fstream>
#include <string>

static std::string CGReposFile ("m4tag_rep");

void vdm_CGRepository::vdm_WriteCGInfo (const TYPE_ProjectTypes_CGInfo &cg_info)
{
  // Write new TagRepository to m4tag_rep
  std::ofstream ostr;
  ostr.open(CGReposFile.c_str(), ios::out);
  if (!ostr) {
    vdm_err << L"Couldn't open file '" << TBWSTR::string2wstring(CGReposFile) << L"'\n";
    return;
  }
  Tuple info (PTAUX::ExtractCGInfo (cg_info));
  info.WriteVal(ostr);
  ostr.close();
}

TYPE_ProjectTypes_CGInfo vdm_CGRepository::vdm_GetCGInfo ()
{
  Tuple info(3);
  std::ifstream istr (CGReposFile.c_str());

  if (istr) {
    // Read existing TagRepository.
    Generic rep;
    rep = ReadVal (istr);
    if (rep.IsTuple ()) {
      info = rep;
```

```
      istr.close();
    }
    else {
      vdm_log << TBWSTR::string2wstring(CGReposFile) << L" is not a tag repository\n";
      istr.close();
      throw TB_Exception(ERR_CG);
      return (Generic) Record (); // to avoid warnings
    }
  }
  else  {
    // Construct an empty tag repository.
    info.SetField(1, Map());
    info.SetField(2, Set());
    info.SetField(3, Int(0));
  }
  return (Generic) PTAUX::mk_CGInfo (info);
}
```

A.5 Class Options

userimplinary methods is not yet implemented.

A.6 Class ToolMediator

userimplinary methods is not yet implemented.

References

[EJ95] R.Johnson E.Gamma, R.Helm and J.Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, 1995.

Index