

The Overture AST and Plug-in Development

Kenneth Lausdahl

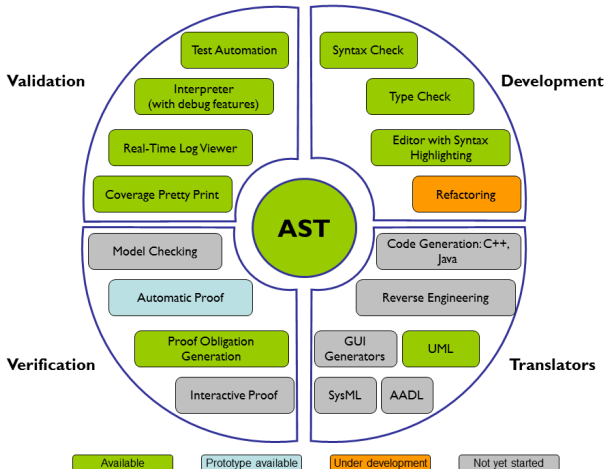
Department of Engineering, Aarhus University, Denmark

28 August 2012 / 10th International Workshop
Overture/VDM

Outline

- 1 Introduction
- 2 The Overture AST
- 3 Creating Plug-ins For Overture

Features



Tool Overview

Core Tools

- AST
- Parser
- Type Checker
- Interpreter
- Proof Obligation Generator

IDE

- Core Plug-in
 - Resource Management
 - Parser & TC Framework
- UI Plug-in
 - Editor
 - Outline
 - Wizards
- Language Specific Plug-ins

Tool Overview

Core Tools

- **AST**
- Parser
- Type Checker
- Interpreter
- Proof Obligation Generator

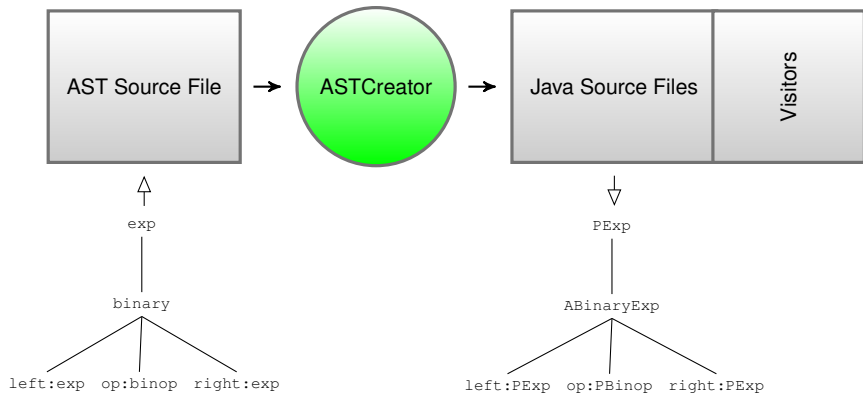
IDE

- Core Plug-in
 - Resource Management
 - Parser & TC Framework
- UI Plug-in
 - Editor
 - Outline
 - Wizards
- Language Specific Plug-ins

Outline

- 1 Introduction
- 2 The Overture AST
- 3 Creating Plug-ins For Overture

AST Generation



AST Generation

AST Source File

Abstract Syntax Tree

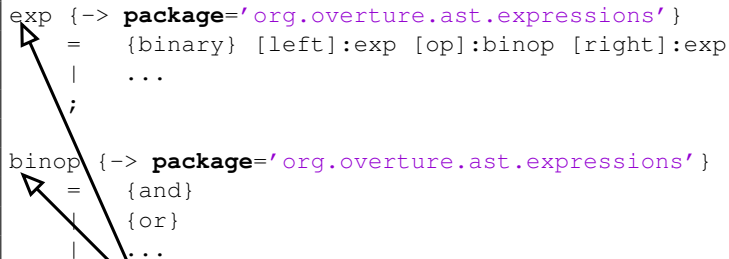
```
exp {-> package='org.overture.ast.expressions' }  
    = {binary} [left]:exp [op]:binop [right]:exp  
    |   ...  
    ;  
  
binop {-> package='org.overture.ast.expressions' }  
      = {and}  
      | {or}  
      |   ...  
      ;
```


AST Generation

AST Source File

Abstract Syntax Tree

```
exp {-> package='org.overture.ast.expressions' }  
= {binary} [left]:exp [op]:binop [right]:exp  
| ...  
;  
  
binop {-> package='org.overture.ast.expressions' }  
= {and}  
| {or}  
| ...
```

A diagram showing two arrows pointing from a box labeled 'Root nodes' to the 'exp' and 'binop' definitions in the code above. The arrow to 'exp' starts from the top of the box and points to the first 'exp' definition. The arrow to 'binop' starts from the bottom of the box and points to the first 'binop' definition.

Root nodes

AST Generation

AST Source File

Abstract Syntax Tree

```
exp {-> package='org.overture.ast.expressions' }  
  =  {binary} [left]:exp [op]:binop [right]:exp  
  |  ...  
  ;  
  
binop {-> package='org.overture.ast.expressions' }  
  =  {and}  
  |  {or}  
  |  ...  
  ;
```

Sub nodes

AST Generation

AST Source File

Abstract Syntax Tree

```
exp {-> package='org.overture.ast.expressions' }  
  = {binary} [left]:exp [op]:binop [right]:exp  
  | ...  
  ;  
  
binop {-> package='org.overture.ast.expressions' }  
  = {and}  
  | {or}  
  | ...  
  ;
```

Node fields

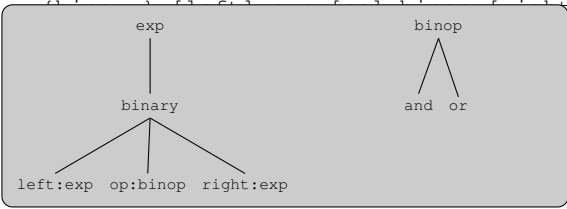
- I Tree fields - only one parent [name]:Type
- II Graph fields - multiple parents (name):Type

AST Generation

AST Source File

Abstract Syntax Tree

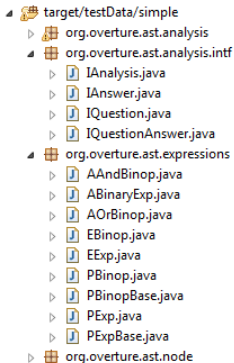
```
exp {-> package='org.overture.ast.expressions' }  
= (binop [left] [right]):exp  
|  
;  
  
binop  
= (left:exp op:binop right:exp  
  ...  
  ;  
  ;
```



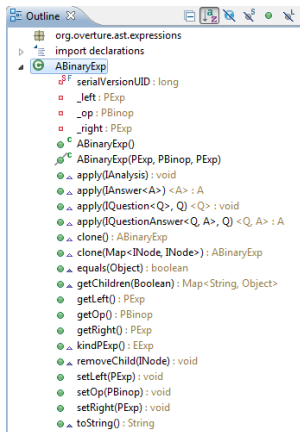
AST Generation

Java Output

Package View



Outline of ABinaryExp



AST Generation

AST Source File Hierarchy

Abstract Syntax Tree

```
exp {-> package='org.overture.ast.expressions' }  
  =   #Binary  
  |   ...  
  ;  
  
#Binary {-> package='org.overture.ast.expressions' }  
  =   {plus}  
  |   {and}  
  |   ...  
  ;
```

Aspect Declaration

```
%exp->#Binary = [left]:exp [op]:LexToken [right]:exp
```

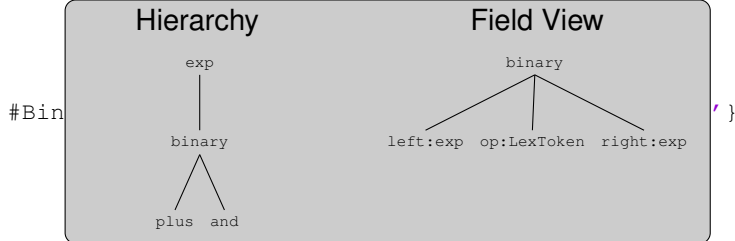
AST Generation

AST Source File Hierarchy

Abstract Syntax Tree

```
exp {-> package='org.overture.ast.expressions' }
```

```
= #Binary
```



Aspect Declaration

```
%exp->#Binary = [left]:exp [op]:LexToken [right]:exp
```

AST Generation

Java Output Features

Node

- `INode parent()`
- `INode getAncestor(Class<INode> classType)`
- **Enumerations**
 - `NodeEnum kindNode()` - return `NodeEnum.EXP`
 - `...EExp kindPExp()` - return `EExp.BINARY`

AST Generation

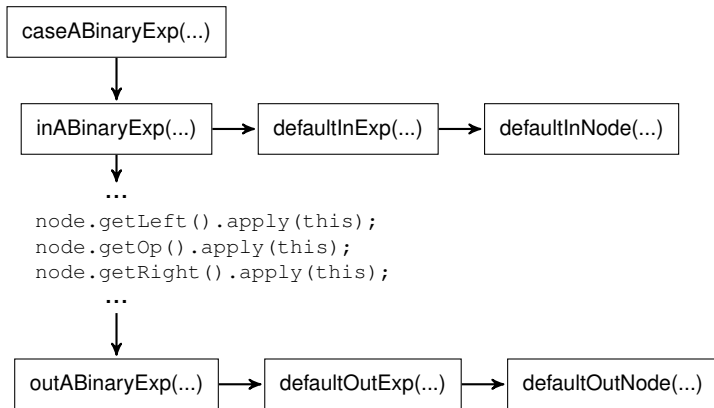
Analysis

Analysis

- **Analysis - Default visitor**
 - `void apply(IAnalysis analysis)`
- **Question**
 - `<Q> void apply(IQuestion<Q> caller, Q question)`
- **Answer**
 - `<A> A apply(IAnswer<A> caller)`
- **Question Answer**
 - `<Q, A> A apply(IQuestionAnswer<Q, A> caller,
Q question)`

AST Generation

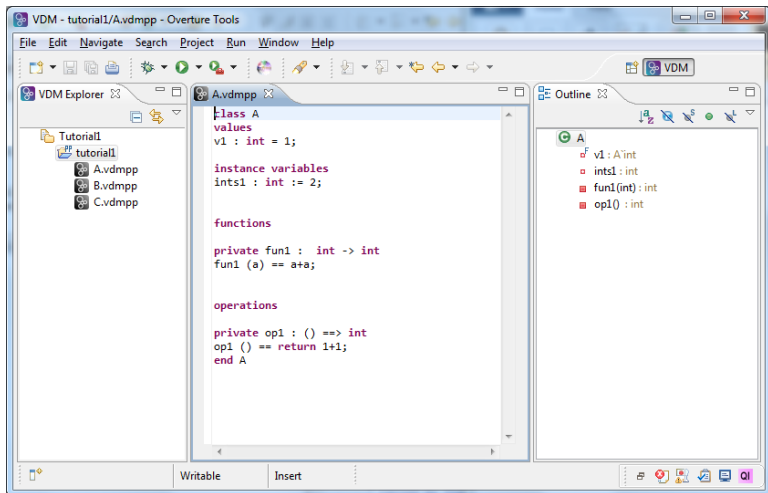
Analysis Depth First Search



Outline

- 1 Introduction
- 2 The Overture AST
- 3 Creating Plug-ins For Overture**

Creating a plug-in



Creating a plug-in

Development Environment

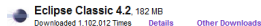
Requirements:

- Java SDK
- Eclipse Classic
 - Extended with the Overture core feature

Creating a plug-in

Development Environment - Setup

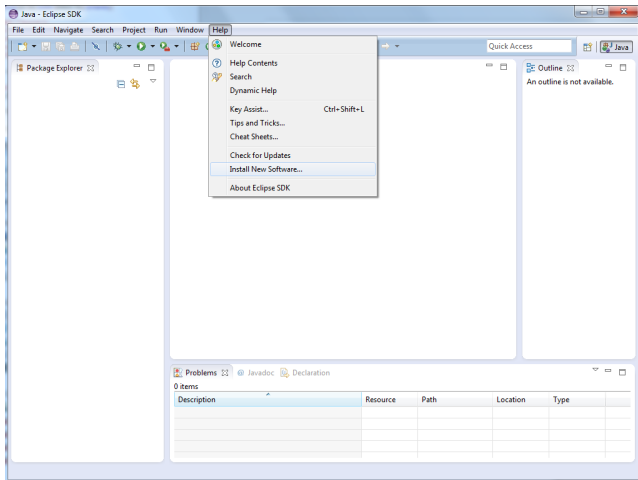
Download Eclipse Classic - www.eclipse.org



Creating a plug-in

Development Environment - Setup

Install New Software

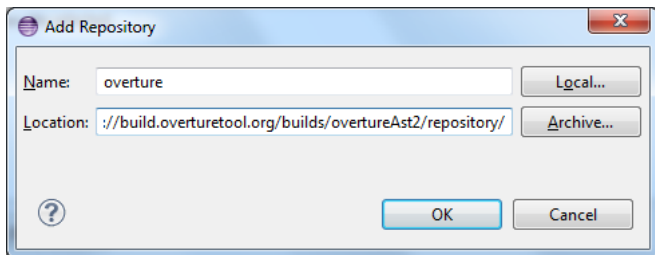


Creating a plug-in

Development Environment - Setup

Add the Overture repository -

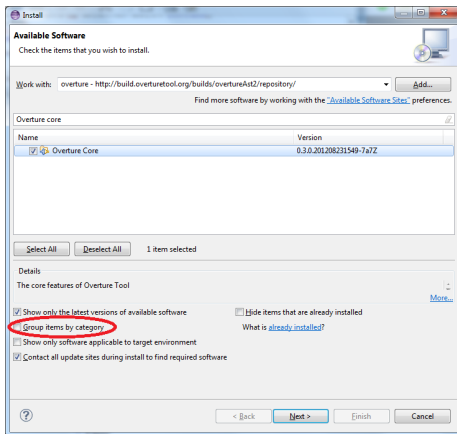
`http://build.overturetool.org/builds/overtureAst2/repository/`



Creating a plug-in

Development Environment - Setup

- 1 Uncheck *Group items by Category*
- 2 Select **Overture Core** and install



Tutorial

Simple Class Analysis Plug-in

- For each Class count:
 - Values
 - Instance Variables
 - Functions
 - Operations
- For each Operation and Function count the total number of expressions and statements

Tutorial

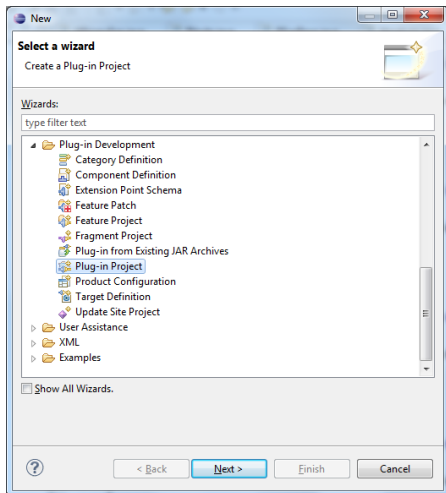
Overview

What do we need to do?

- 1 Create a plug-in project
- 2 Add dependencies to Eclipse and Overture
- 3 Add a command to invoke the functionality
- 4 Add a menu to show the command in the UI
- 5 Add a Handler to invoke the implementation when the command is called
- 6 Add a view to show the result
- 7 Add the Overture-specific VDM analysis code

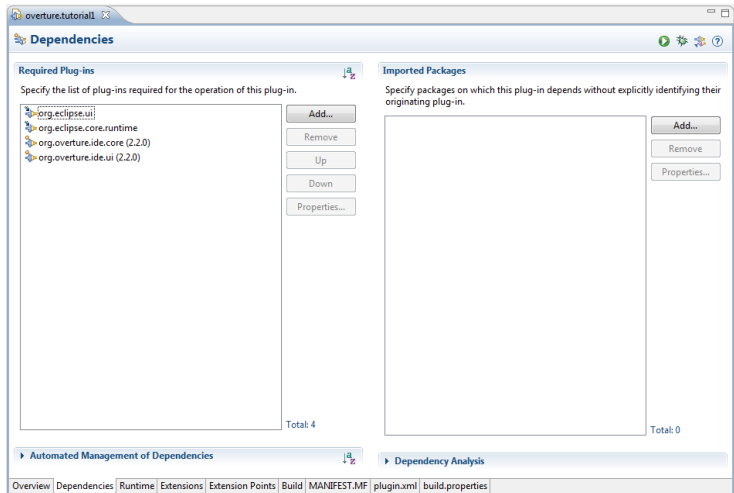
Tutorial

Create plug-in project



Tutorial

Add dependencies



Tutorial





Add dependencies

Dependencies

Required Plug-ins



Specify the list of plug-ins required for the operation of this plug-in.

-  org.eclipse.ui
-  org.eclipse.core.runtime
-  org.overture.ide.core (2.2.0)
-  org.overture.ide.ui (2.2.0)

Add...

Remove

Up

Down

Properties...

Imported Packages

Specify packages on which this plug-in depends on the originating plug-in.

Tutorial

Add a command

```
<extension
  point="org.eclipse.ui.commands">
  <command
    id="overture.tutorial1.commandAnalysis"
    name="Analysis">
  </command>
</extension>
```

Tutorial

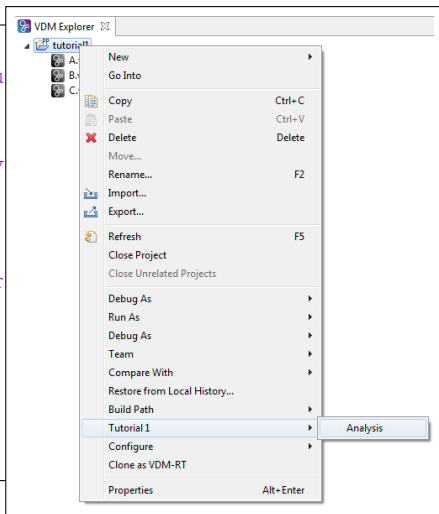
Add a menu

```
<extension
    point="org.eclipse.ui.menu">
<menuContribution
    allPopups="false"
    locationURI="popup:org.overture.ide.ui.VdmExplorer">
<menu
    label="Tutorial 1">
    <command
        commandId="overture.tutorial1.commandAnalysis"
        label="Analysis"
        style="push">
    </command>
</menu>
</menuContribution>
</extension>
```


Tutorial

Add a menu

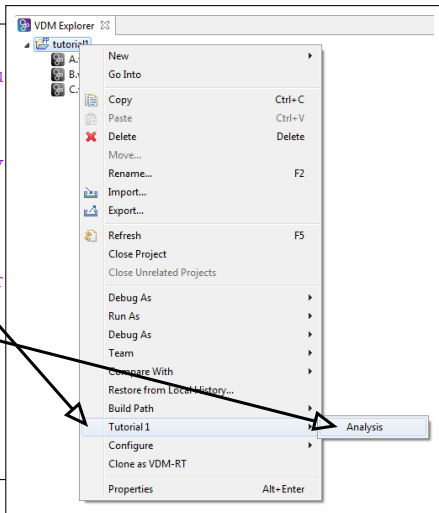
```
<extension
    point="org.eclipse.ui"
    <menuContribution
        allPopups="false"
        locationURI="popup:org.ov
    <menu
        label="Tutorial 1">
        <command
            commandId="overtur
            label="Analysis"
            style="push">
        </command>
    </menu>
</menuContribution>
</extension>
```



Tutorial

Add a menu

```
<extension
  point="org.eclipse.ui"
  <menuContribution
    allPopups="false"
    locationURI="popup:org.ov
  <menu
    label="Tutorial 1">
    <command
      commandId="overtur
      label="Analysis"
      style="push">
    </command>
  </menu>
</menuContribution>
</extension>
```



Tutorial

Add a Handler

```
<extension
  point="org.eclipse.ui.handlers">
  <handler
    class="org.overture.ide.analysis.AnalysisHandler"
    commandId="overture.tutorial1.commandAnalysis">
  </handler>
</extension>
```

Tutorial

Handler Implementation

```
public class AnalysisHandler extends AbstractHandler
{
    public Object execute(ExecutionEvent event) {
        ISelection selection = HandlerUtil.getCurrentSelection(event);
        IStructuredSelection structuredSelection = (IStructuredSelection)
            selection;

        Object firstElement = structuredSelection.getFirstElement();

        IProject p = ((IResource)firstElement).getProject();

        IVdmProject vdmProject =
            (IVdmProject) p.getAdapter(IVdmProject.class);

        a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),
            HandlerUtil.getActiveShell(event));
        a.runAnalysis();

        ....
    }
}
```

Tutorial

Handler Implementation

Get Selection through
HandlerUtil

```
Handler extends AbstractHandler

public Object execute(ExecutionEvent event) {
    ISelection selection = HandlerUtil.getCurrentSelection(event);
    IStructuredSelection structuredSelection = (IStructuredSelection)
        selection;

    Object firstElement = structuredSelection.getFirstElement();

    IProject p = ((IResource)firstElement).getProject();

    IVdmProject vdmProject =
        (IVdmProject) p.getAdapter(IVdmProject.class);

    a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),
        HandlerUtil.getActiveShell(event));
    a.runAnalysis();

    ....
}
```

Tutorial

Handler Implementation

Get Selection through
HandlerUtil

Handler **extends** AbstractHandler

```
public Object execute(ExecutionEvent event) {  
    ISelection selection = HandlerUtil.getCurrentSelection(event);  
    IStructuredSelection structuredSelection = (IStructuredSelection)
```

Get Project

```
    Object firstElement = structuredSelection.getFirstElement();
```

```
    IProject p = ((IResource)firstElement).getProject();
```

```
    IVdmProject vdmProject =  
        (IVdmProject) p.getAdapter(IVdmProject.class);
```

```
    a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),  
                       HandlerUtil.getActiveShell(event));
```

```
    a.runAnalysis();
```

```
    ....
```

Tutorial

Handler Implementation

Get Selection through
HandlerUtil

Handler **extends** AbstractHandler

```
public Object execute(ExecutionEvent event) {  
    ISelection selection = HandlerUtil.getCurrentSelection(event);  
    IStructuredSelection structuredSelection = (IStructuredSelection)
```

Get Project

```
    Object firstElement = structuredSelection.getFirstElement();
```

```
    IProject p = ((IResource)firstElement).getProject();
```

```
    IVdmProject vdmProject =  
        (IVdmProject) p.getAdapter(IVdmProject.class);
```

```
    a = new VdmAnalysis(vdmProject, HandlerUtil.getActivePart(event),  
        HandlerUtil.getActiveShell(event));  
    a.runAnalysis();
```

.... Adapt selection to a VDM Project

Tutorial

View

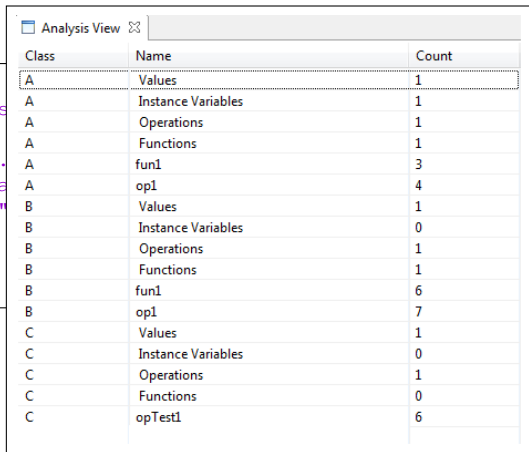
```
<extension
  point="org.eclipse.ui.views">
  <view
    class="org.overture.ide.analysis.AnalysisViewPart1"
    id="overture.tutorial1.view1"
    name="Analysis View"
    restorable="true">
  </view>
</extension>
```

The implementation takes a data object and displays it in a table view. (Not shown here)

Tutorial

View

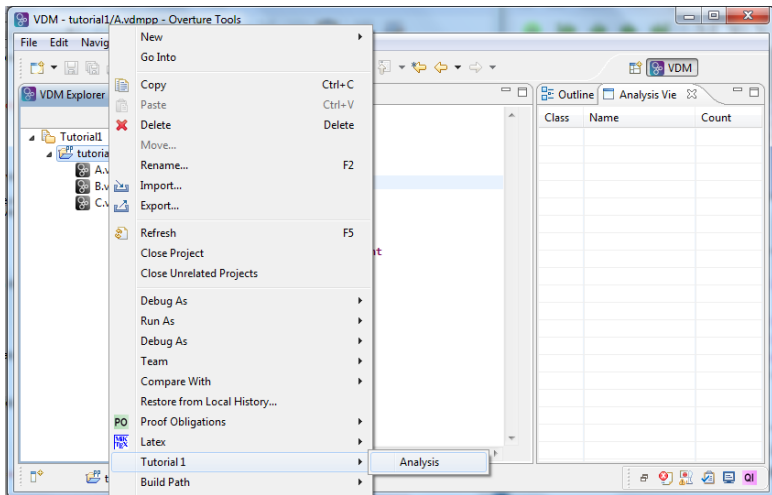
```
<extension
  point="org.eclipse
  <view
    class="org.overture.
    id="overture.tutoria
    name="Analysis View"
    restorable="true">
  </view>
</extension>
```



The screenshot shows a window titled "Analysis View" containing a table with three columns: "Class", "Name", and "Count". The table lists various elements categorized by class (A, B, C) and their respective counts.

Class	Name	Count
A	Values	1
A	Instance Variables	1
A	Operations	1
A	Functions	1
A	fun1	3
A	op1	4
B	Values	1
B	Instance Variables	0
B	Operations	1
B	Functions	1
B	fun1	6
B	op1	7
C	Values	1
C	Instance Variables	0
C	Operations	1
C	Functions	0
C	opTest1	6

Tutorial Status



Tutorial

Overture Analysis Implementation

Check the model and run the Type Checker:

```
final IVdmModel model = project.getModel();  
if (model != null && model.isParseCorrect())  
{  
    if (!model.isTypeCorrect())  
    {  
        VdmTypeCheckerUi.typeCheck(shell, project);  
    }  
}
```

Run the analysis:

```
if(model.isTypeCorrect())  
{  
    AnalysisData data = analyse(model.getRootElementList());  
}
```

Tutorial

Overture Analysis Implementation 2

```
private AnalysisData analyse(List<INode> rootElementList){  
    AnalysisVisitor visitor = new AnalysisVisitor();  
  
    for (INode node : rootElementList)  
    {  
        try  
        {  
            node.apply(visitor);  
        } catch (AnalysisException e){  
        }  
    }  
  
    return visitor.data;  
}
```

Tutorial

Overture Analysis Implementation 3 Visitor

```
class AnalysisVisitor extends DepthFirstAnalysisAdaptor
{

    void inAClassClassDefinition (AClassClassDefinition node) {
        data.initClass (node.getName ().name);
    }

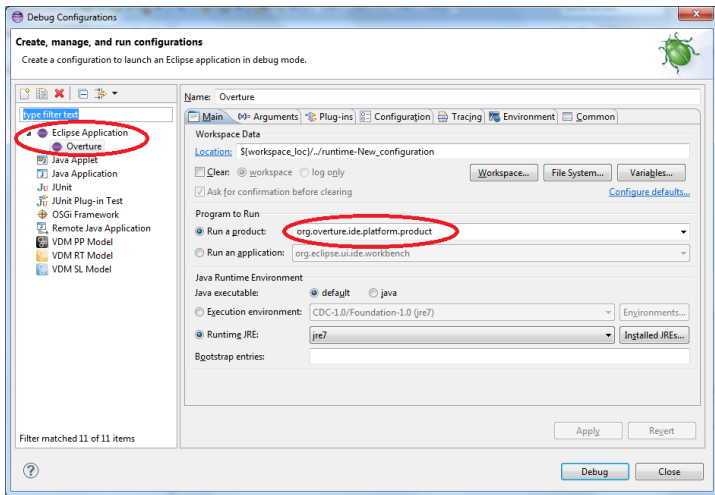
    void inAExplicitFunctionDefinition (...) {
        expCount = 0;
    }

    void outAExplicitFunctionDefinition (... node) {
        data.fun.get (data.activeClass).put (node.getName ().name, expCount);
    }

    void defaultInPExp (PExp node) {
        expCount++;
    }
}
```

Creating a plug-in

Launching the Debugger



Creating a plug-in

Final plug-in

The screenshot shows the VDM IDE interface. The main editor displays the source code for class A:

```
class A
values
v1 : int = 1;

instance variables
ints1 : int := 2;

functions

private fun1 : int -> int
fun1 (a) == a+a;

operations

private op1 : () ==> int
op1 () == return 1+1;
end A
```

The VDM Explorer on the left shows the project structure:

- Tutorial1
 - tutorial1
 - A.vdmpp
 - B.vdmpp
 - C.vdmpp

The Analysis View on the right displays the following table:

Class	Name	Count
A	Values	1
A	Instance Variables	1
A	Operations	1
A	Functions	1
A	fun1	3
A	op1	4
B	Values	1
B	Instance Variables	0
B	Operations	1
B	Functions	1
B	fun1	6
B	op1	7
C	Values	1
C	Instance Variables	0
C	Operations	1
C	Functions	0
C	opTest1	6

Creating a plug-in

Tutorial 2: Extension

Try to extend the example given in tutorial 1 by:

- Count expressions in initialization of:
 - Values
 - Instance Variables
- Enable the analysis for Modules
- ... or what you have in mind

Thanks

You can download the tutorial at:

`http://tinyurl.com/overture10tutorial`

Wiki Overture Workshop 10