What's new in MPS 2.0 M3

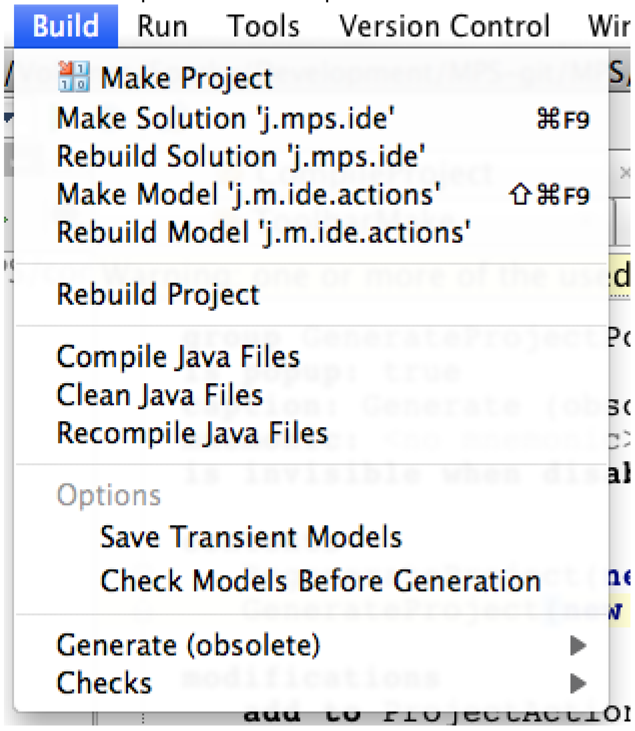
2.0 M3

Make and Rebuild

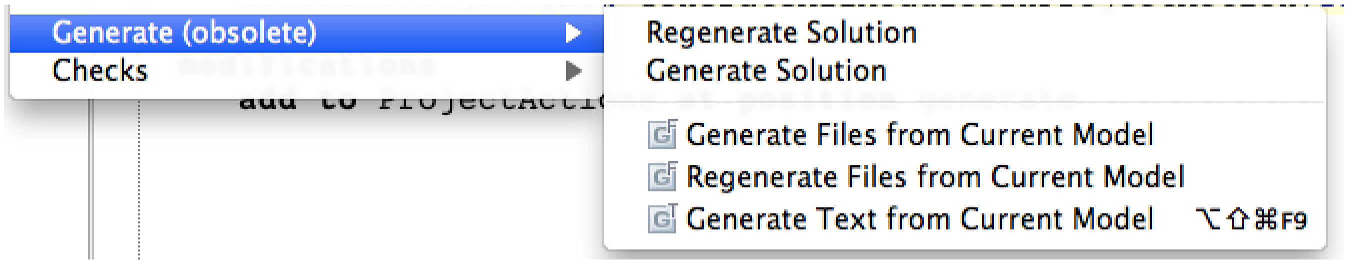
Important change
Since this release there'll be no more "Generate" and "Regenerate" actions in the UI. They have been replaced by "Make" and "Rebuild". See below for details.

The main "Build" menu has been re-worked completely. Here's what changed:

- New actions "Make Project" and "Rebuild Project"
- "Generate" is now called "Make"
- "Regenerate" is now called "Rebuild"
- What was previously called "Make" and "Rebuild" are now "Compile" and "Recompile"

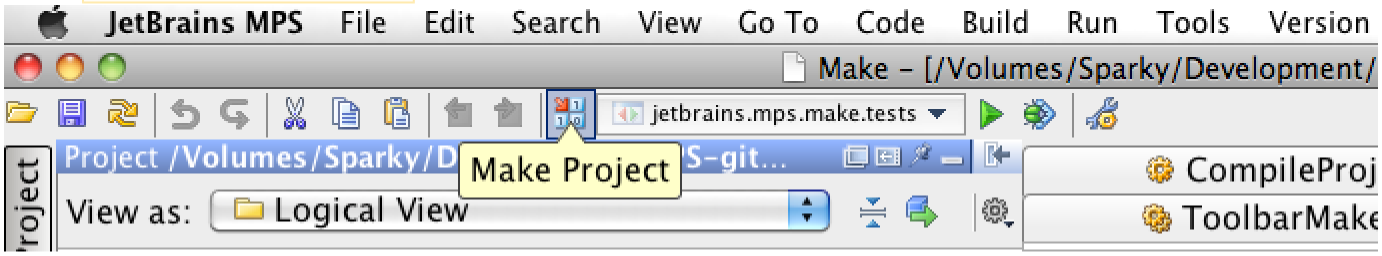


- The old "Generate" options are still available

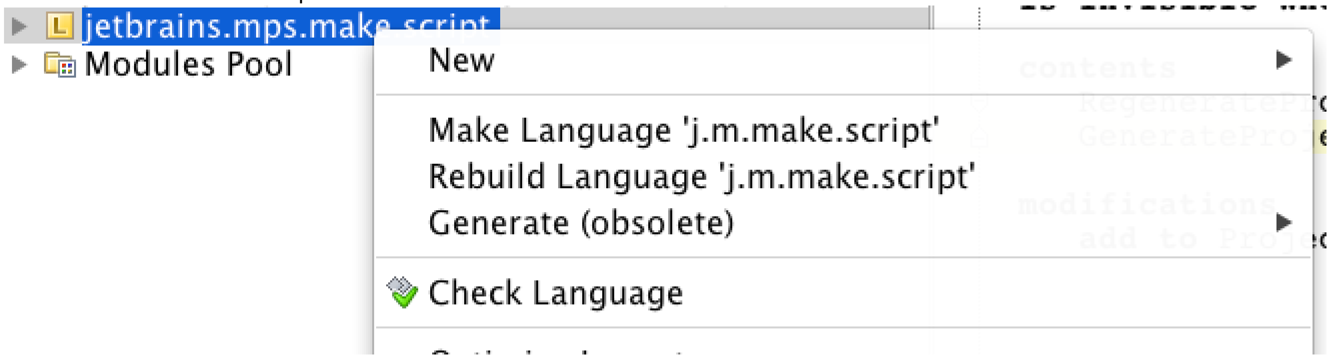

Generate (obsolete)
Checks
Regenerate Solution
Generate Solution
Generate Files from Current Model
Regenerate Files from Current Model
Generate Text from Current Model ⌥⇧⌘F9

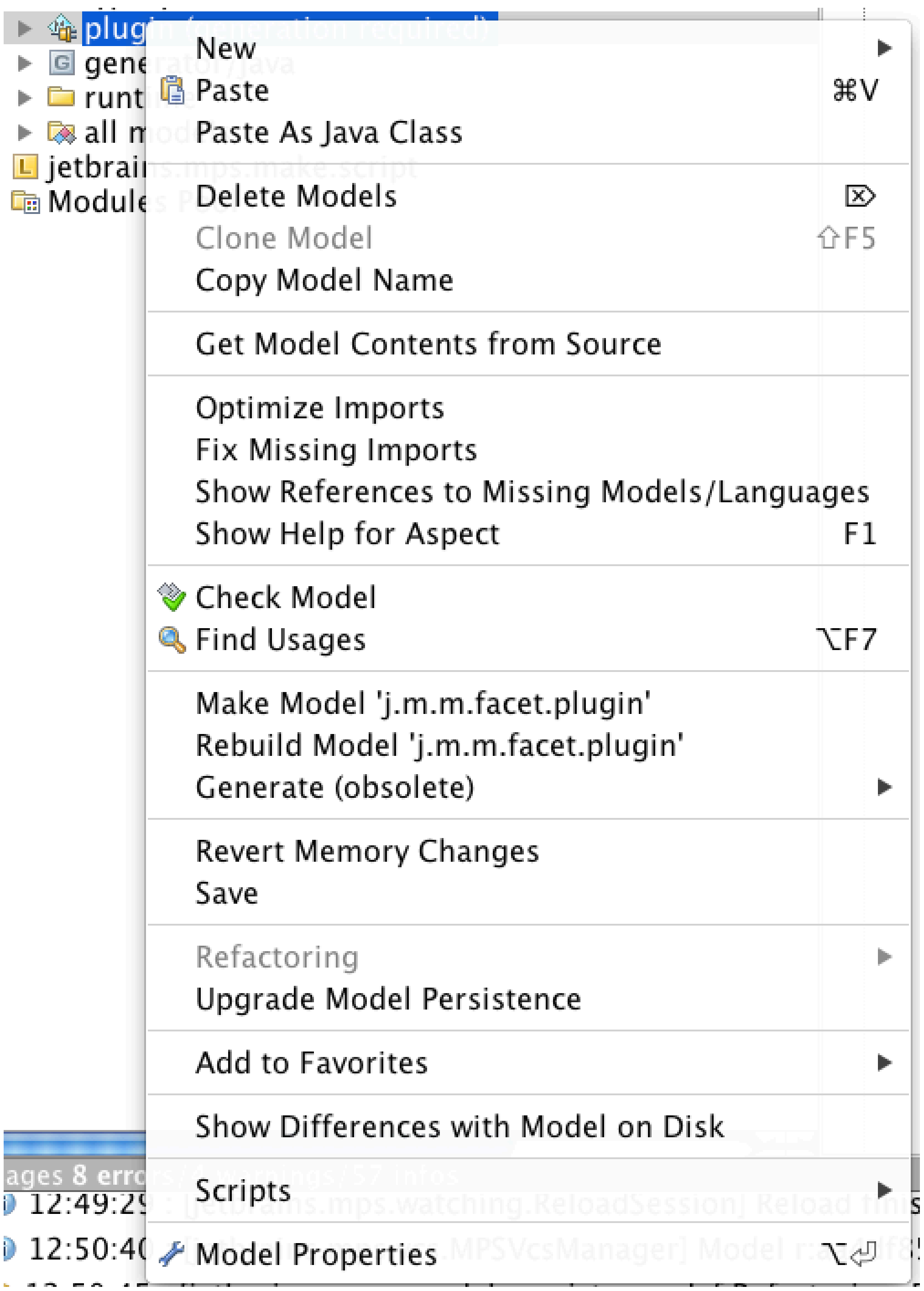
- There is a new button on the main toolbar, replacing three obsolete ones

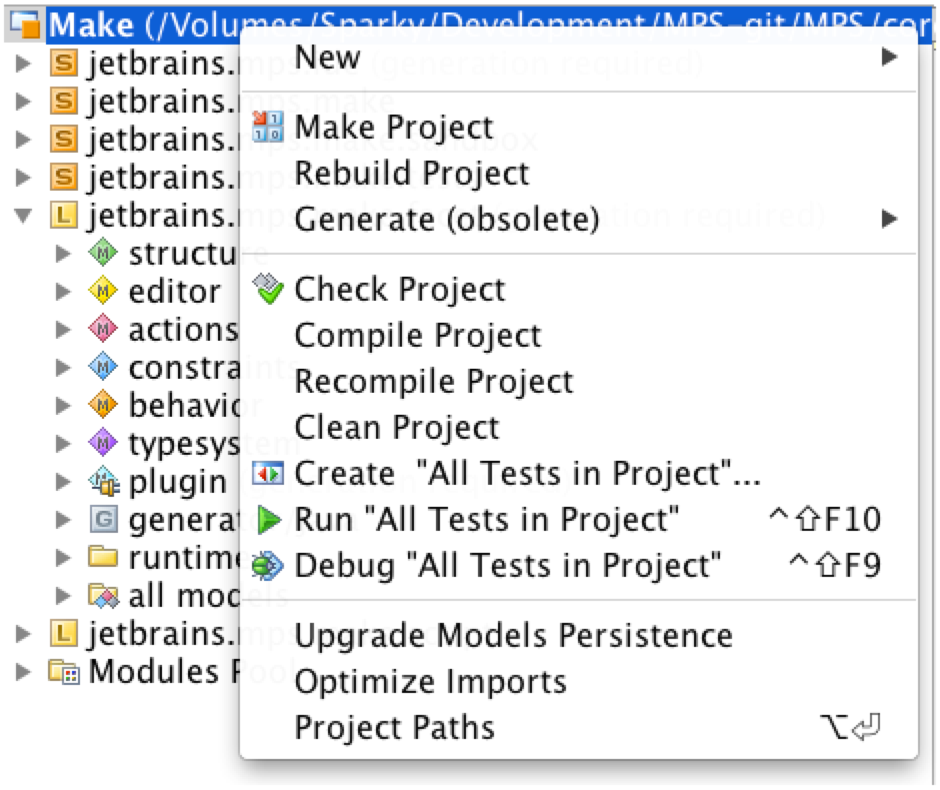
⚠ Removed feature
The obsolete feature "generate used languages that require generation" is removed. It is superseded with the new make action.



- Menus for solutions and models have been updated too


jetbrains.mps.make.script
Modules Pool
New
Make Language 'j.m.make.script'
Rebuild Language 'j.m.make.script'
Generate (obsolete)
Check Language

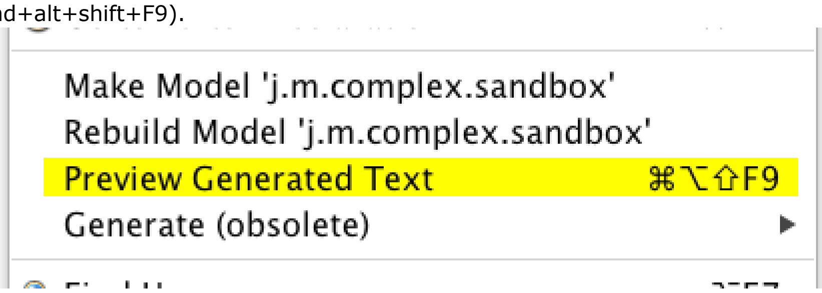

New
Paste ⌘V
Paste As Java Class
Delete Models ⌦
Clone Model ⇧F5
Copy Model Name
Get Model Contents from Source
Optimize Imports
Fix Missing Imports
Show References to Missing Models/Languages
Show Help for Aspect F1
Check Model
Find Usages ⌥F7
Make Model 'j.m.m.facet.plugin'
Rebuild Model 'j.m.m.facet.plugin'
Generate (obsolete)
Revert Memory Changes
Save
Refactoring
Upgrade Model Persistence
Add to Favorites
Show Differences with Model on Disk
Scripts
Model Properties ⌥⏎

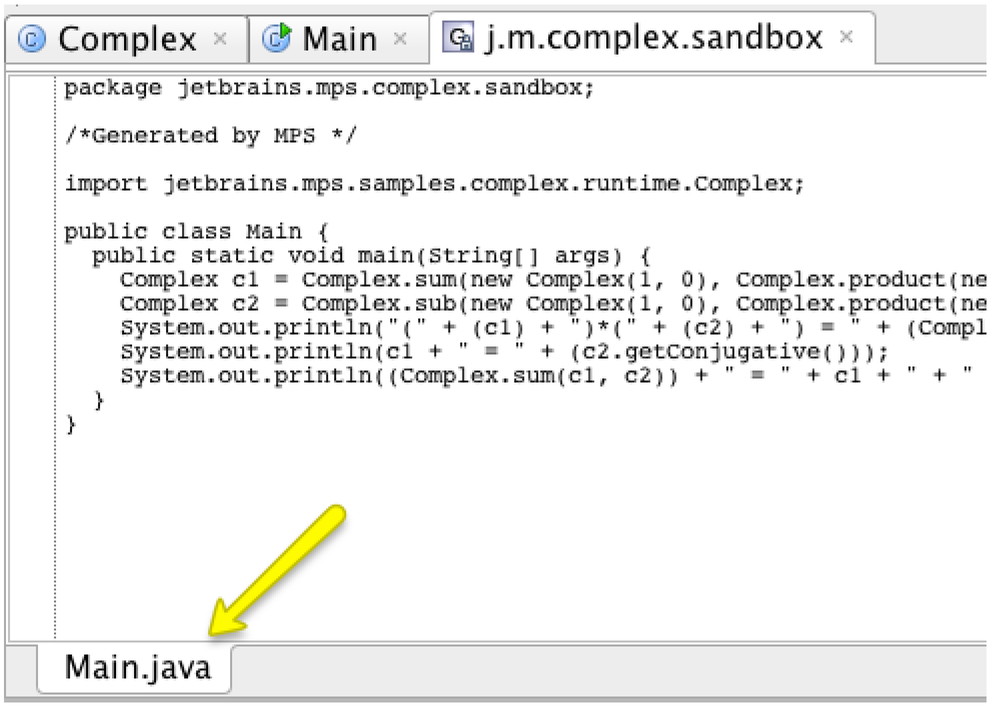


Generated Text Preview

A new action "Preview Generated Text" opens preview in an multi-tab editor. Each tab corresponds to a single file.

The action is available on a model and is also bound to a shortcut ctrl+alt+shift+F9 (MacOS cmd+alt+shift+F9).


Make Model 'j.m.complex.sandbox'
Rebuild Model 'j.m.complex.sandbox'
Preview Generated Text ⌘⌥⇧F9
Generate (obsolete)

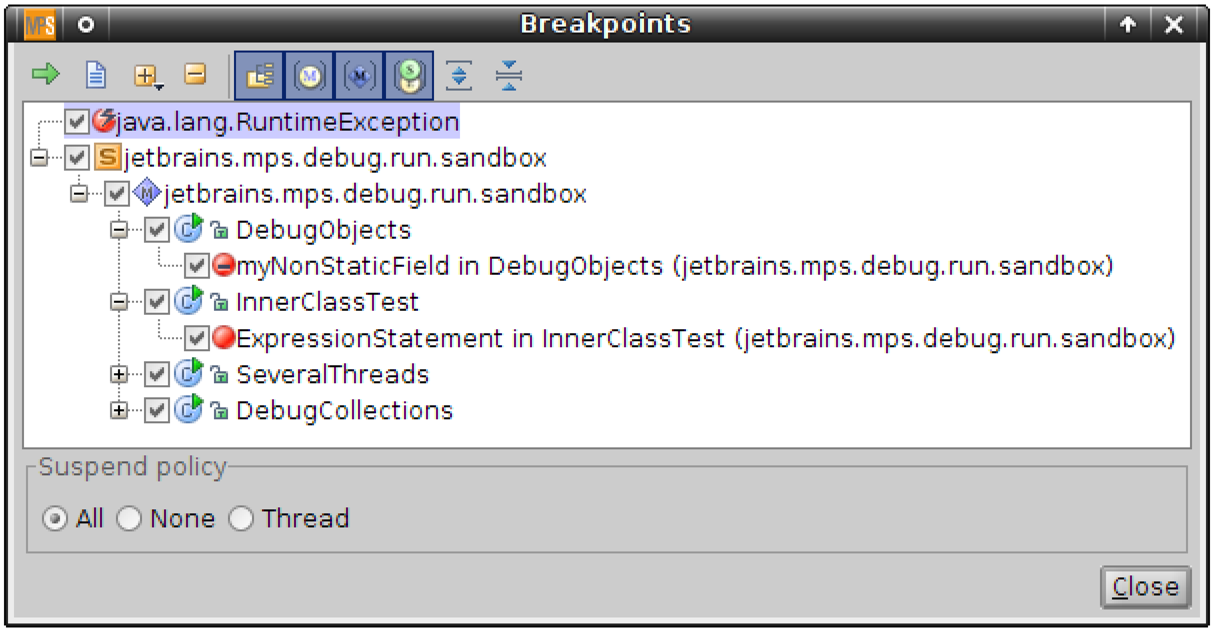


```
package jetbrains.mps.complex.sandbox;

/*Generated by MPS */

import jetbrains.mps.samples.complex.runtime.Complex;

public class Main {
  public static void main(String[] args) {
    Complex c1 = Complex.sum(new Complex(1, 0), Complex.product(ne
    Complex c2 = Complex.sub(new Complex(1, 0), Complex.product(ne
    System.out.println("(" + (c1) + ")*(" + (c2) + ") = " + (Compl
    System.out.println(c1 + " = " + (c2.getConjugative()));
    System.out.println((Complex.sum(c1, c2)) + " = " + c1 + " + "
  }
}
```

New Breakpoints and Watches in Java Debugger

New Breakpoints Features

- Debugger API improvements:
  - support for different kinds of breakpoints;
  - breakpoint properties panel;
  - create breakpoint button in Breakpoints View.
- Java breakpoints:
  - field watchpoints;
  - exception breakpoints;
  - suspend policy for java breakpoints;
  - relevant breakpoint data (like thrown exception or changed field value) is displayed in variables tree.

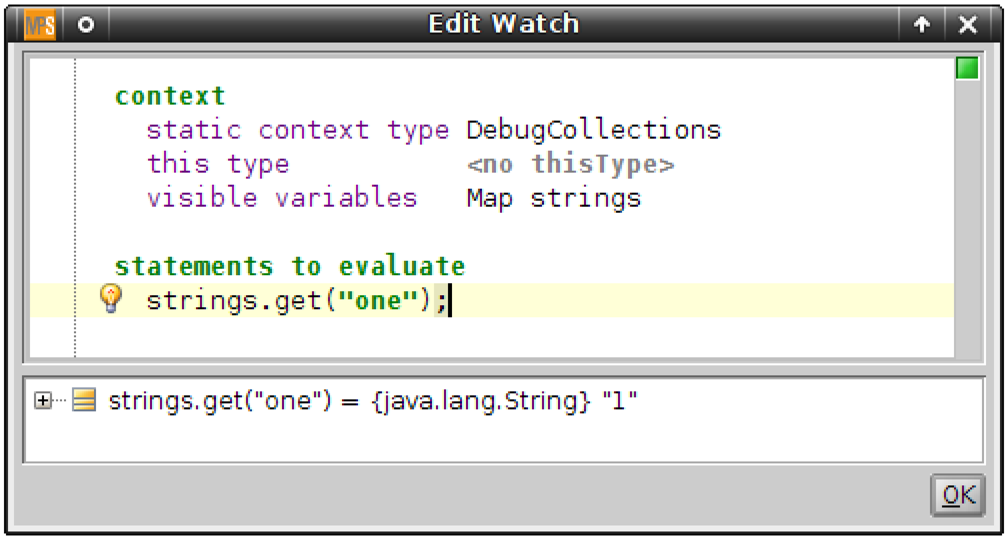


```
☑ java.lang.RuntimeException
☑ jetbrains.mps.debug.run.sandbox
  ☑ jetbrains.mps.debug.run.sandbox
    ☑ DebugObjects
      ☑ myNonStaticField in DebugObjects (jetbrains.mps.debug.run.sandbox)
    ☑ InnerClassTest
      ☑ ExpressionStatement in InnerClassTest (jetbrains.mps.debug.run.sandbox)
    ☑ SeveralThreads
    ☑ DebugCollections
```

Suspend policy: [✓ All] None Thread

Low-Level Watches

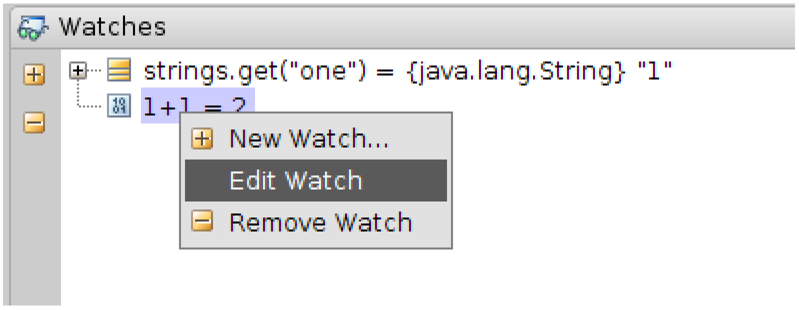
Watches API and low-level watches for java debugger are implemented. "Low-level" means that user can write expressions using variables, available on the stack. To edit a watch, a so-called "context"(used variables, static context type and this type) must be specified. If the stack frame is available at the moment, context is filled automatically.



```
context
  static context type DebugCollections
  this type           <no thisType>
  visible variables   Map strings

statements to evaluate
  strings.get("one");
```

```
strings.get("one") = {java.lang.String} "1"
```

Watches can be viewed in "Watches" tree in "Debug" tool window. Watches could be created, edited and removed via context menu or toolbar buttons.



```
strings.get("one") = {java.lang.String} "1"
1+1 = 2
```

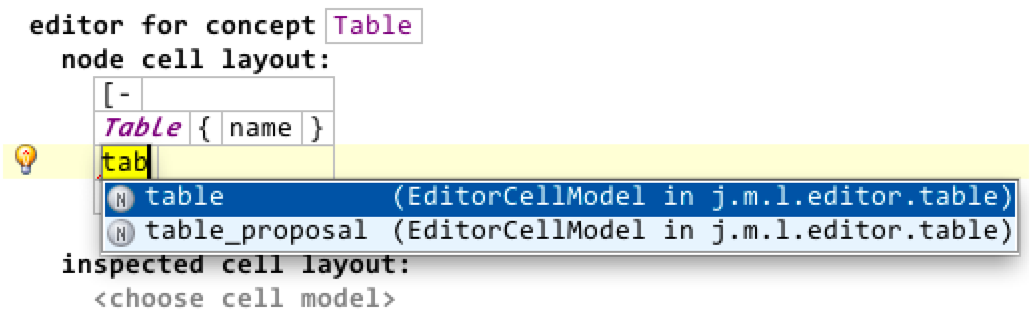
Trace information generation is done via textGen language

Trace information generation is done now via textGen language. Concepts that require trace information generation should implement one of the three interface concepts: `TraceableConcept`, `ScopeConcept` and `UnitConcept`. Use script "Upgrade Trace Info Generation" from plugin language to upgrade.

See more infomation in Debugger documentation page

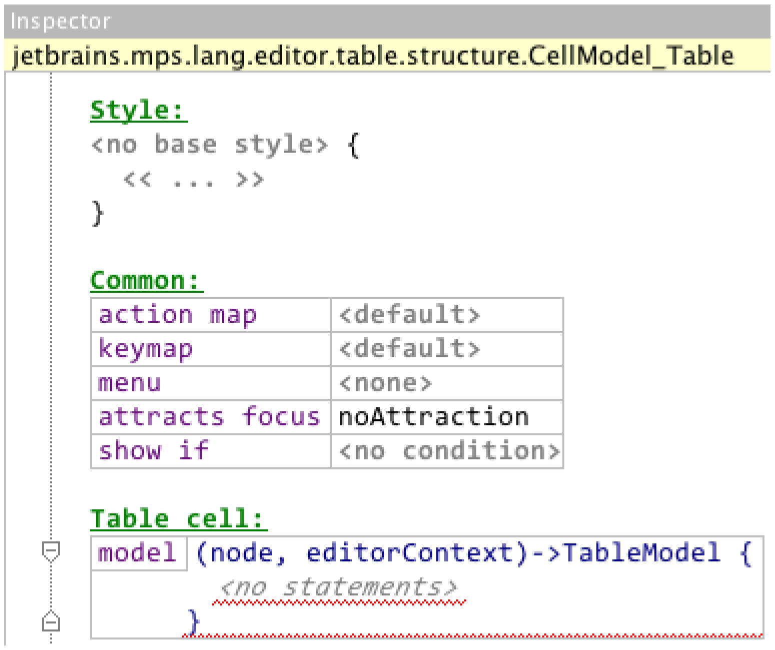
Tables support in editor

- New language jetbrains.mps.lang.editor.table was added to MPS distribution. This language should be imported into the editor model to enable table support. For now this language supports two different kinds of table: table and hierachical table:



```
editor for concept Table
  node cell layout:
    [-
    Table { name }
    tab
      table           (EditorCellModel in j.m.l.editor.table)
      table_proposal  (EditorCellModel in j.m.l.editor.table)
  inspected cell layout:
    <choose cell model>
```

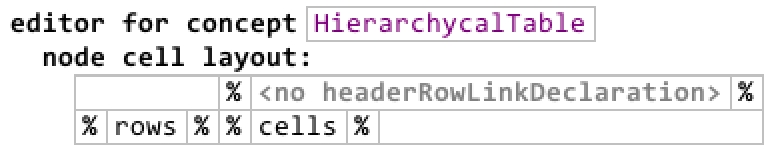
- table cell model was intended for generic table support in editor. An instance of TableModel interface must be returned from model function in Table cell section within inspector of this cell model:



```
Inspector
jetbrains.mps.lang.editor.table.structure.CellModel_Table

Style:
<no base style> {
  << ... >>
}

Common:
action map       <default>
keymap           <default>
menu             <none>
attracts focus   noAttraction
show if          <no condition>

Table cell:
model (node, editorContext)->TableModel {
        <no statements>
      }
```

Specified TableModel instance used to create and edit table grid. Each table child cell contains common MPS editor specified for given child node. TableModel interface as well as a number of it's reusable implementations are defined in jetbrains.mps.lang.editor.table.runtime solution.

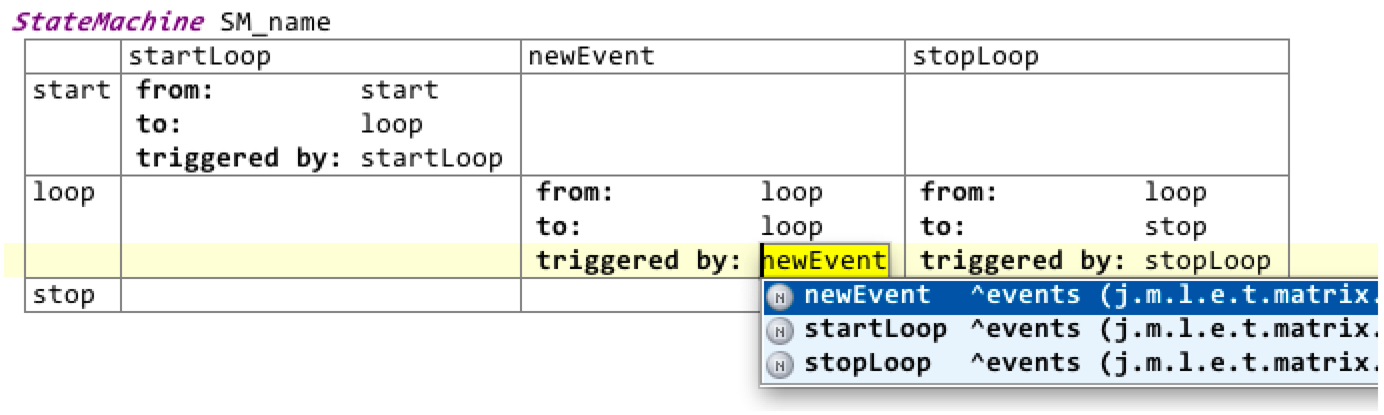
- hierarchical table cell model was created to cover rather frequent use case of table-based child nodes editing then a concept, represented by table, has multiple child reference to another concept Row keeping table cells as it's children:



```
editor for concept HierarchycalTable
  node cell layout:
                 % <no headerRowLinkDeclaration> %
    % rows %     % cells %
```

On this screenshot rows is a name of child role keeping Row concept instances and cells is a name of child role defined within Row concept keeping DataCell concept which will be displayed within table grid. In addition optional headerRowlinkDeclaration - multiple child reference keeping nodes representing table header row - can be specified there.

- jetbrains.mps.editor.tableTest language with exemplary table editors were added to MPS distribution. This language contains possible table-based concept editor definitions. Some table editing tricks are:
    - new table row can be added by pressing Enter on a cell located outside table grid either before- or after- current row
    - new table column can be added by pressing Enter in the beggining/end of any existing cell in a column
    - common MPS editor with completion can be used to edit child nodes



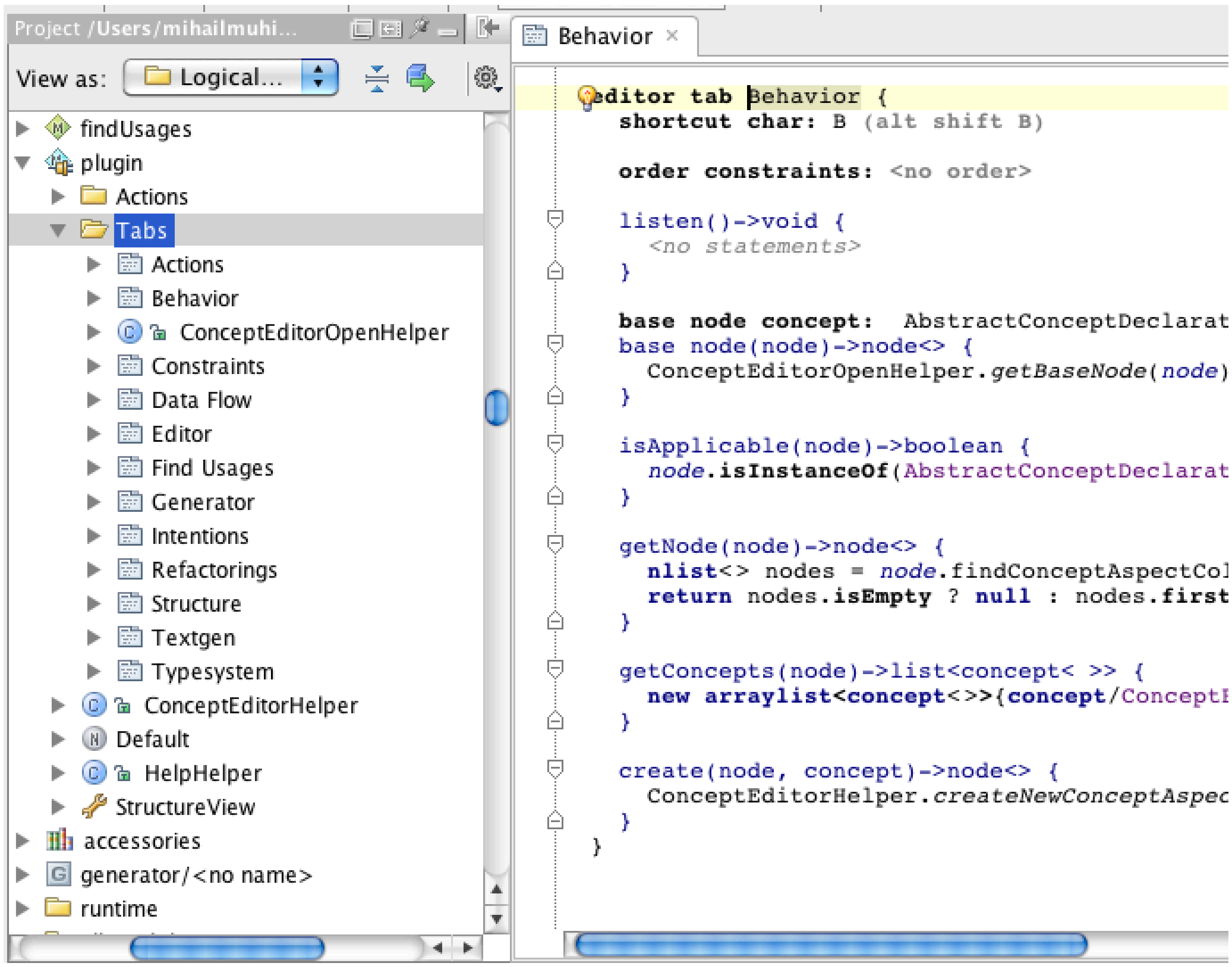
StateMachine SM_name

	startLoop	newEvent	stopLoop
start	**from:** start **to:** loop **triggered by:** startLoop		
loop		**from:** loop **to:** loop **triggered by:** newEvent	**from:** loop **to:** stop **triggered by:** stopLoop
stop			

newEvent ^events (j.m.l.e.t.matrix.
startLoop ^events (j.m.l.e.t.matrix.
stopLoop ^events (j.m.l.e.t.matrix.

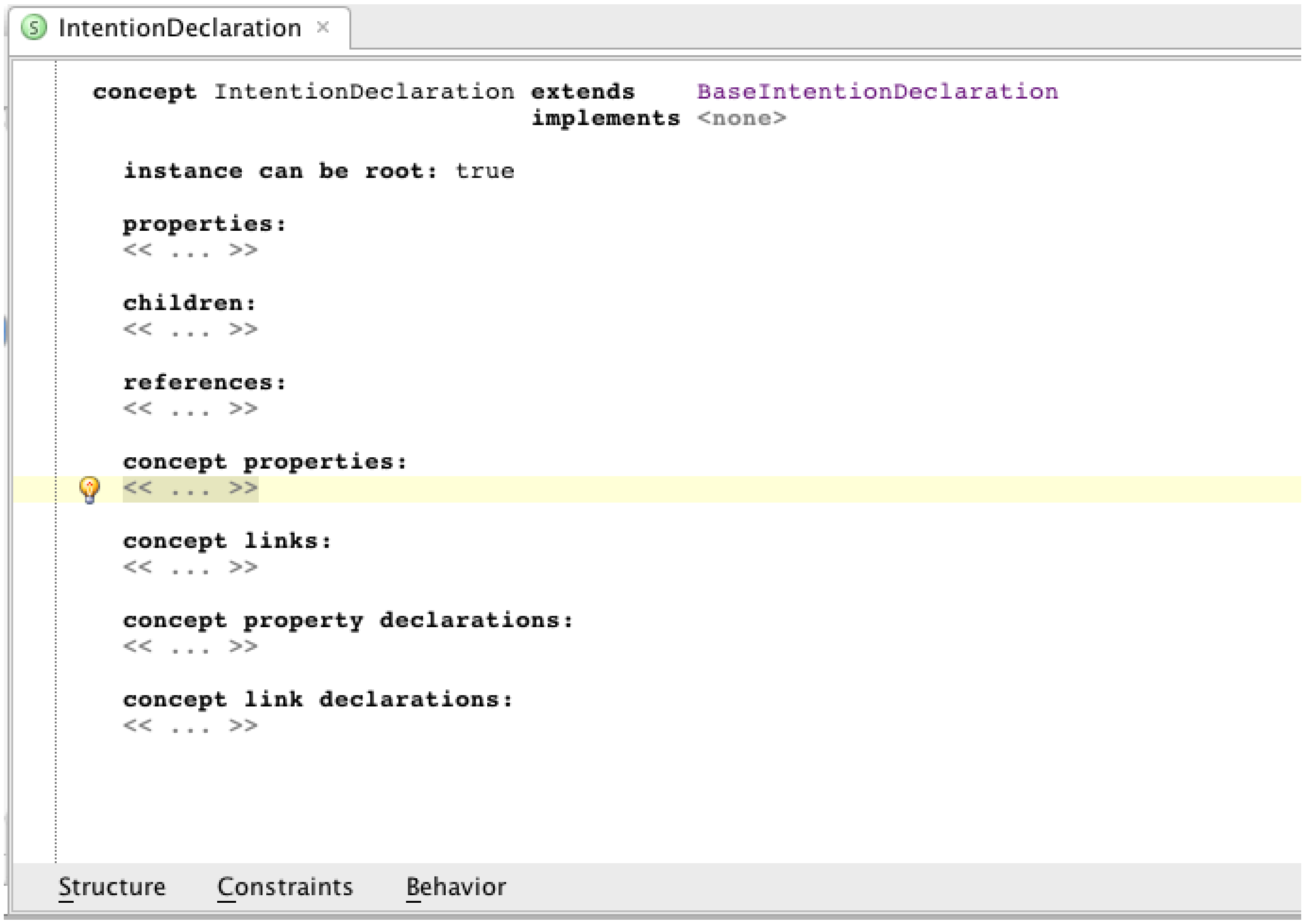
New Tabbed Editors

The tabbed editors part of plugin language was slightly improved. Now new aspects can be added for a concept from any language, which will allow to "extend" tabbed editors, in contrast with old TabbedEditor allowing to define the set of aspects only once.



```
editor tab Behavior {
  shortcut char: B (alt shift B)

  order constraints: <no order>

  listen()->void {
    <no statements>
  }

  base node concept:  AbstractConceptDeclarat
  base node(node)->node<> {
    ConceptEditorOpenHelper.getBaseNode(node)
  }

  isApplicable(node)->boolean {
    node.isInstanceOf(AbstractConceptDeclarat
  }

  getNode(node)->node<> {
    nlist<> nodes = node.findConceptAspectCol
    return nodes.isEmpty ? null : nodes.first
  }

  getConcepts(node)->list<concept< >> {
    new arraylist<concept<>>{concept/ConceptE
  }

  create(node, concept)->node<> {
    ConceptEditorHelper.createNewConceptAspec
  }
}
```

The editors themselves changed their appearance - now there is no 3rd level of tabs. Instead, an editor has a toolbar showing all aspects available for the main node and allowing to create a new ones.m



```
concept IntentionDeclaration extends    BaseIntentionDeclaration
                             implements <none>

  instance can be root: true

  properties:
  << ... >>

  children:
  << ... >>

  references:
  << ... >>

  concept properties:
  << ... >>

  concept links:
  << ... >>

  concept property declarations:
  << ... >>

  concept link declarations:
  << ... >>
```

Plugin framework improvements

Now it's possible to create a non-language plugin for MPS. This will allow to create "standalone" plugins, which will not require the presence of plugin's module for the plugin to work (e.g. a VCS plugin).
Note: Icons in actions don't work properly in non-language plugins for now.

In addition, plugin components are no more created via reflection. This will improve reloading performance of large plugins.