Tutorial AspectMaps

The goal of this tutorial is to introduce the features of *AspectMaps* to the participants of our experiment. This tutorial consists of a live demo of the tool, applied to two sample systems: `AJHotDraw` and `AMTest`. Each of the participants of the experiment is given a demo by one of the authors of the paper prior to that the participants get any hands-on experience with AspectMaps.

Top-level overview of the system

Case study: `AJHotDraw`

1. The demo starts with an overview of the top-level view offered by the tool. In the top pane of the tool, this view shows all the packages in the system that is being analyzed. Each package is represented by a box with a label on top of it showing the package's name. The color of the package corresponds to the aspects that affect source code within that package. If there are multiple aspects applying, the package is by default colored black. In the bottom pane of the tool the list of all aspects in the system is shown. Each aspect is given its own color, that is used throughout the visualization to identify that aspect. By default, all aspects are enabled; aspects can be toggled on and off from within the aspect list.
2. Note that the visual representation of AspectMaps is customizable. To this end, we have 4 input controls in the bottom pane of the tool. For the namespaces view (the packages) that we see now, we can switch between the *Plain* view (the default), and for example the *Advice* view. This view shows a distribution map of which advices apply within the package. Using this view, it is possible to see which aspects apply to the package, even if there are multiple.
3. We start the demo by disabling all aspects and then manually enabling `PersistentAttributeFigure` and `PersistentCompositeFigure`. Note that this changes the color of the affected packages.
4. By clicking on the 'Max out' button, a user can always return to this view.
5. By clicking the 'Toggle intervened Pkgs' button, we can select whether we want to see all packages, or only those in which aspects apply.

Zoomable visualization

Case study: `AJHotDraw`

1. Aspect Maps is a zoomable visualization: by double-clicking on an entity, this entity is expanded and shown in more detail. To show this level of detail for all the packages, click the 'Pkg contents' button. To illustrate this concept, double-click on the package `org.jhotdraw.figures`. We now see the visualization at the class level: each class is represented by a rectangle. Aspect definitions are also shown at this level: similar to classes, they are shown as a rectangle (with as border color the color of the aspect). Inheritance relationships within the same package are shown. Similar to packages, we can change the visualization of classes. Not only can we replace the default visualization with a distribution map, but we can also vary the dimensions of the rectangle representing the class based on various metrics.
2. By double-clicking the class `AttributeFigure` (the one colored class), we zoom further in. We can show this level of detail for all classes in the system by clicking the 'Class contents' button. At this level, the attributes and methods of the class are shown along with the intertype declarations. So in this case, the `PersistentAttributeFigure` aspect introduced two methods (`read` and `write`) in this class.
3. When hovering over an entity, the next level of detail is shown in a pop-up window.
4. By single-clicking on an entity, its source code is shown (if available) in a panel in the lower half of the tool.

Method level visualization

Case study: `AJHotDraw`

1. The finest level of detail shown by the Aspect Maps visualization is the method level. We illustrate how methods are visualized by means of the `CommandContracts` aspect. First, we disable all other aspects except this one. Next, we click on the ‘Enabled Asp’ button to zoom in to the finest level of detail where all enabled aspects apply (in this case on `CommandContracts`).

2. Each method is visualized by means of a rectangle. At the top of the rectangle, the method’s name is shown. The rest of the rectangle is divided into three sections. From top to bottom, these sections visualize the before, around and after advice of execute join points. In this example, we see that the `CommandContracts` aspect always intervenes both before and after the execution of methods named `execute`.

3. The section in the middle represents the execution of the method’s body. It is in this section that call join points are visualized. To illustrate this, let’s zoom out to the max level and enable the `SelectionChangedNotification` aspect. As I know that this aspect intervenes in a method named `removeFromSelection()` I am going to find this method in the visualization. I do this by using the ‘By Query’ button and using **removeFromSelection** as a query. If we go to this method implemented by class `StandardDrawingView`, we see that the aspect intervenes after a call to `invalidate`. We get this information by hovering over the visualized join point.

4. Now, by using the context-menu we can click on the ‘Reveal Aspect’ option. This option shows us the implementation of the aspect (in this case in the package `org.jhotdraw.ccconcerns.figures.figureselectionbrowser`). As for classes, AspectMaps provided a more detailed visualization of aspects. For this example, we see two poincuts (shown as ovals), and two pieces of advice (shown as rectangles). When zooming in on advice, the same visualization as for methods applies.

Aspect precedence

Case study: `AMTest`

1. As a final feature, we show how Aspect Maps visualizes the precedence relationships between aspects. These relationships are either declared explicitly by the aspect developer (`declare precedence` in AspectJ), or are implied (from the order of the pointcuts/advice in the specification file). We illustrate this using a second case study `AMTest`, which is actually just a small toy example used while developing AspectMaps. For this case, we make sure that all aspects are enabled and zoom in to the most detailed level by clicking on ‘Zoom in’.

2. Here we see that, if multiple aspects apply to a single join point, our visualization will indicate, by means of an arrow, that one has precedence over the other. Note that, if the arrow is black, this precedence is explicitly declared; if it is gray, it is implicitly deduced from the rules of the used aspect language. If there are multiple aspects applying to a single join point, and no arrows are shown between them, this simply means that no precedence relation exists, and that hence at run-time the order of the execution of the aspects is undeterministic.