SoQueT (Sorts Query Tool)

This is an alpha version of SoQueT - the tool support for consistently describing and documenting crosscutting concerns as sort instances.

The tool provides two main views:
- The "Search Sorts" view allows querying the code for describing and documenting crosscutting concerns as sort instances. The user can run new queries, display the results and navigate the source code from these results.
- The "Concern Model" view provides support for modeling relations in tree structures. The user can build a concern model with elements of two types: (1) sort instances are leaf-elements described by a query aimed at emphasizing an atomic crosscutting relation; The templates for the queries to describe sort instances are available through the "Search Sorts" view. (2) Composite concerns are complex relations that group related sort instances or other composite concerns to show a more complex relation.
**The alpha version**

This version misses several GUI features, like evolution bars for some of the queries or other long-running operations like loading an existing concern model. Several queries are only partially implemented, like Design enforcement and Dynamic behavior enforcement. Yet, the tool can be used as it is, and documentation of concerns in JHotDraw is available for download on the tool’s web-site.

---

**Example - How to document an Observer pattern instance**

This section shows how to build a concern model for an Observer pattern instance in JHotDraw. We use the version 54b1 of the JHotDraw project, and import this project in Eclipse as:

Eclipse project name: JHotDraw54b1
Java Build path: JHotDraw54b1/src (the package structure (CH.ifa.draw.*) starts from this src folder)
libs: jdo (jdo.jar)
    batik(batik-awt-util.jar, batik-dom.jar, batik-svggen.jar, batik-util.jar)
    junit
Step 1: Select the “Add a New Concern” option in the “Concern Model” view.

Step 2: We start with a top concern for the JHotDraw project – this is a composite concern, with the same name as the project.

Step 3: Repeat the previous operation to build composite concerns for “DesignPatterns” and Observer instances in JHotDraw. Note that for adding a child-concern, the parent concern should be selected in the “ConcernModel” view.
Next, we are going to document an instance of the Observer pattern in JHotDraw, which implements a notification mechanism for changes in the Figure elements. We create a new Composite concern for this pattern instance: FigureChangedListener.

Step 4: Add Sort instances elements to the model to document crosscutting concerns. An Observer pattern instance exhibits a number of crosscutting concerns, like the two roles super-imposed to the participants in the pattern’s implementation (Subject and Observer), the consistent notification of the changes in the Subject’s state, etc. The Observers for Figure changes implement the FigureChangeListener interface. We first document this Observer (Listener) role.
Step 5: To describe and document crosscutting concerns consistently, each sort instance can be associated a sort-specific query. To access the templates for the sort-queries, select the menu button shown in figure.

Step 6: To document super-imposed roles we use the template for the “Role superimposition” sort-query. This allows querying for all the elements that implement an interface, in this case, FigureChangeListener. The query takes as input the interface that defines the role, and the context into which this role is relevant; that is, we are interested in the classes that implement the interface defining the role and that also belong to the selected context. In our case, all the implementers in the JHotDraw project are relevant.
Step 7: The results of the query are shown in the “Search Sorts” view. You can change the layout in the view for displaying the results of the query. For a number of options, like code inspection, you can right-click a result-element in the view.

To associate the query with a sort-instance in the Concern model, select the sort instance in the “Concern Model” view (here, FigureChangeListenerRole), and select the Add button in the Search Sorts view where the results of the query are described.

Step 8: The definition of the Subject role for this implementation of the pattern is tangled with the Figure interface. The role is defined by a number of method-members, which permit to attach and remove listeners for changes, and to notify listeners of changes, before and after the modifications take place.

To document the role, we first create a new sort instance in the “Concern model” view, as child of the FigureChangedListener concern: FigureChangeSubjectRole.

Next, we need to define a _virtual_ interface for this role; that is, we specify what elements in the Figure interface define the role we want to document. Open the query template interface for Role superimposition, just as for the previous query.

We are interested in implementers of the Figure interface (select Figure in the field for the name of the role element).

Select the “Virtual interface” check-box, and then the (method) elements that define the role: addFigureChangeListener(FigureChangeListener), removeFigureChangeListener(..), changed(), willChange(), etc.

Define the context (the JHotDraw54b1 project), execute the query, and associate the query and the results to the Sort instance for the pattern’s Subject role (FigureChangeSubjectRole).
Step 9: The Sort-instance element for the listener role is shown next. To re-run the sort instance’s query, select the instance in the Concern model view, right-click, and select “Expand”.
Step 10: The notification of changes is an instance of the Consistent behavior sort and consists of calls to the Figure.changed() method.
Persistence of the concern model

A concern model can be stored to and read from a file. A model documenting a number of concerns in JHotDraw is available for download on the tool’s web-site. This model can be loaded into SoQueT by selecting the “Load concern model” button in the “Concern model” view.

The operation for loading the concern model could take around 1 minute and Eclipse will not respond to user-actions ( - this is a long-running operation for which a progress bar is going to be implemented).
To run the sort-query describing a crosscutting concern, select the “Expand” option from the context-menu of the sort instance documenting the concern.
Other queries

This section shows a number of other queries in SoQueT. Note that not all the queries are documented here.

Redirection layer

Input:
- The class performing the redirection: DecoratorFigure
- (The type of) the reference to redirect to – indicate (the type of) the field storing the reference (myDecoratedFigure) or (the type of) the method returning the reference (getDecoratedFigure);
  (In the second case, the “Method returning reference” check-box has to be selected before selecting the Choose button.)
- (Select the whole project as context)
The query returns the methoda in the redirection layer that consistently forward their calls to dedicated methods of the “target” type.
Exception propagation

This query returns (recursively) the callers of a given method that re-throw the specified exception.
Input:
- Specify a method in the call-chain that throws the exception of interest: StorableInput.readStorable().
- Specify the thrown exception of interest by selecting the “Choose” button (one of the exceptions thrown by the previously selected method): IOException
- Select the scope/context element – the JHotDraw54b1 project

The query returns the callers of readStorable that re-throw IOException. The user can expand any of the reported callers in the results-view, and navigate through the callers of the selected method that re-throw the same exception.
Support classes for role superimposition

The query returns the set of support classes (nested classes) in a given context that implement a specified role (interface). Similar to role superimposition, the crosscuttingness documented through this query consists of secondary role(s), but implemented through nested classes and not directly.

Input:
- The role implemented by the support classes – select the interface defining this role. For example, select the “Choose” button and type Undoable for the interface defining the role.
- Define the context crosscut by support classes, for instance, the Command hierarchy.

That is, we are looking for support classes in the Command hierarchy that implement the Undoable role.

The query returns all the nested classes implementing Undoable in the hierarchy of Command.
 undone! - 11 implementors in hierarchy of 'Command'

4 LindaActivity - CHifa.draw.figures.GroupCommand
1 LindaActivity - CHifa.draw.figures.InsertImageCommand
2 LindaActivity - CHifa.draw.figures.UngroupCommand
1 LindaActivity - CHifa.draw.standard.AlignCommand
2 LindaActivity - CHifa.draw.standard.BringToFrontCommand
3 LindaActivity - CHifa.draw.standard.ChangeAttributeCommand
2 LindaActivity - CHifa.draw.standard.CutCommand
1 LindaActivity - CHifa.draw.standard.DeleteCommand
1 LindaActivity - CHifa.draw.standard.PasteCommand
2 LindaActivity - CHifa.draw.standard.SelectAllCommand
2 LindaActivity - CHifa.draw.standard.SendToBackCommand
Expose context

Query for the callees of a given method that are passed the specified argument, also received by the given caller-method.

Input:
- The method initiating the context passing chain: StorableOutput.writeStorable
- The name of the argument used to pass context; this is received by the previously specified method and further passed to other callee(s): storable

The query returns the callees of “writeStorable” that are passed the same argument as the one received by the method, i.e., “storable”.
WriteStorable - 3 references in project CHI5eDrawUtil

- heapStorable() - CHI5eDrawUtil.StorableOutput
- mappedStorable() - CHI5eDrawUtil.StorableOutput
- writeRefStorable() - CHI5eDrawUtil.StorableOutput