MARPLE SAR Suite

A Software Architecture Reconstruction approach through the adoption of software metrics and the use of code microstructures

---

Project team:
Francesca Arcelli, Stefano Maggioni, Christian Tosi, Marco Zanoni, Fabio Andreoli, Alberto Gatti

DISCo – Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
20126 – Milan – Italy

MARPLE SAR Suite is a submodule of the MARPLE project devoted to the reconstruction of Java systems architectures. At now, MARPLE SAR Suite basically provides two sets of views and architectural abstractions on the analyzed systems:

- Metrics- and source-code-based views, which are basically obtained by the calculation of common object oriented and system metrics, resulting in the production of views that aim at getting information about the complexity of the analyzed system; one of these views isn't directly derived by the calculation of metrics, but by the analysis of the relationships and links that exist among the classes composing the analyzed system.
- Microstructures-based views, produced starting from the output of the Basic Elements Detector module of MARPLE, which focus more on the strict architecture of the system, letting the users concentrate on the architecture of classes, packages and their relationships.

METRICS- AND SOURCE-CODE-BASED VIEWS

At now, three views belong to this set:

- The Type Blueprint view;
- The System Complexity view;
- The Type Graph view.

The Type Blueprint view

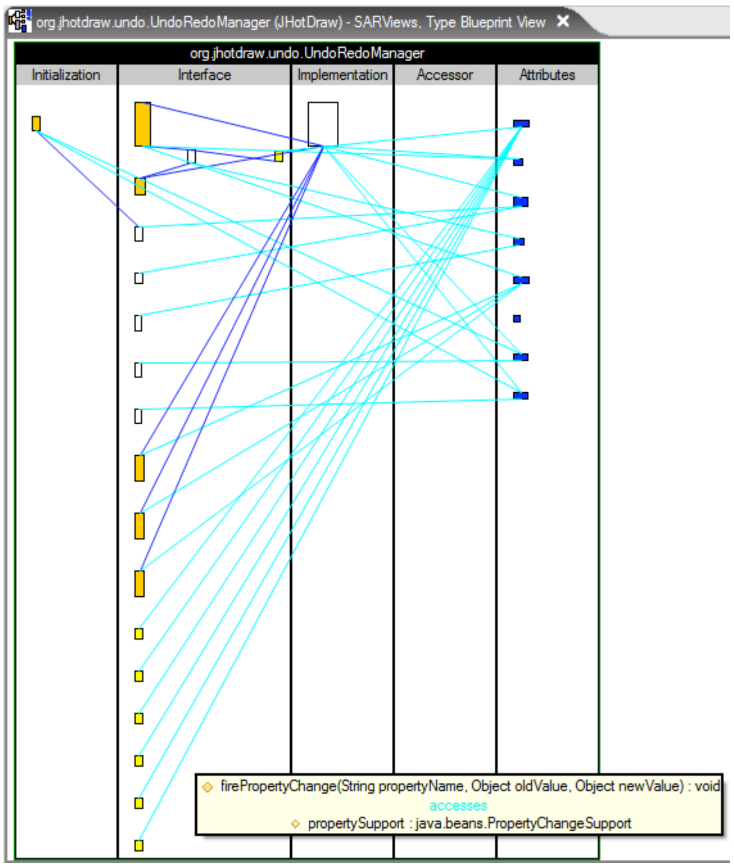
The Type Blueprint view is used to focus on the details of a single class, minding at its attributes and methods. Therefore, each node of the view can represent both an attribute or a method, according to the following specifications:

NODES COLOR			
Attribute	*Blue*	*Constant* method (returning a constant value)	*Grey*
Abstract method	*Sky blue*	Method belonging to the initialization layer	*Orange*
Extending method (invoking the method overridden by it)	*Orange*	Getter method	*Red*
Overriding method	*Brown*	Setter method	*Orange*
Delegating method	*Yellow*	Other kinds of methods	*White*

Moreover, both attribute and method nodes are characterized by metrics values, leading to the graphical representation of the results as a so-called *polymetric view* [LD03]. The nodes are reported according to these specifications:

Method nodes		
Width	NI	*Number of Invocations*
Height	LOC	*Lines of Code*
Attribute nodes		
Width	NLA	*Number of Local Accesses*
Height	NGA	*Number of Global Accesses*

This view is directly mutuated from the CodeCrawler tool [CC]. As we think this view is really useful for the analysis of each single class, as it gives an overview about its complexity and the interactions among its elements, we decided to include this view in the set of views generated by MARPLE SAR Suite.
Figure 1 represents a Type Blueprint view on JHotDraw's UndoRedoManager class.


org.jhotdraw.undo.UndoRedoManager (JHotDraw) - SARViews, Type Blueprint View
org.jhotdraw.undo.UndoRedoManager
Initialization
Interface
Implementation
Accessor
Attributes
fireProertyChange(String propertyName, Object oldValue, Object newValue) : void
accesses
propertySupport : java.beans.PropertyChangeSupport
Fig. 1 - The Type Blueprint view on JHotDraw's UndoRedoManager class

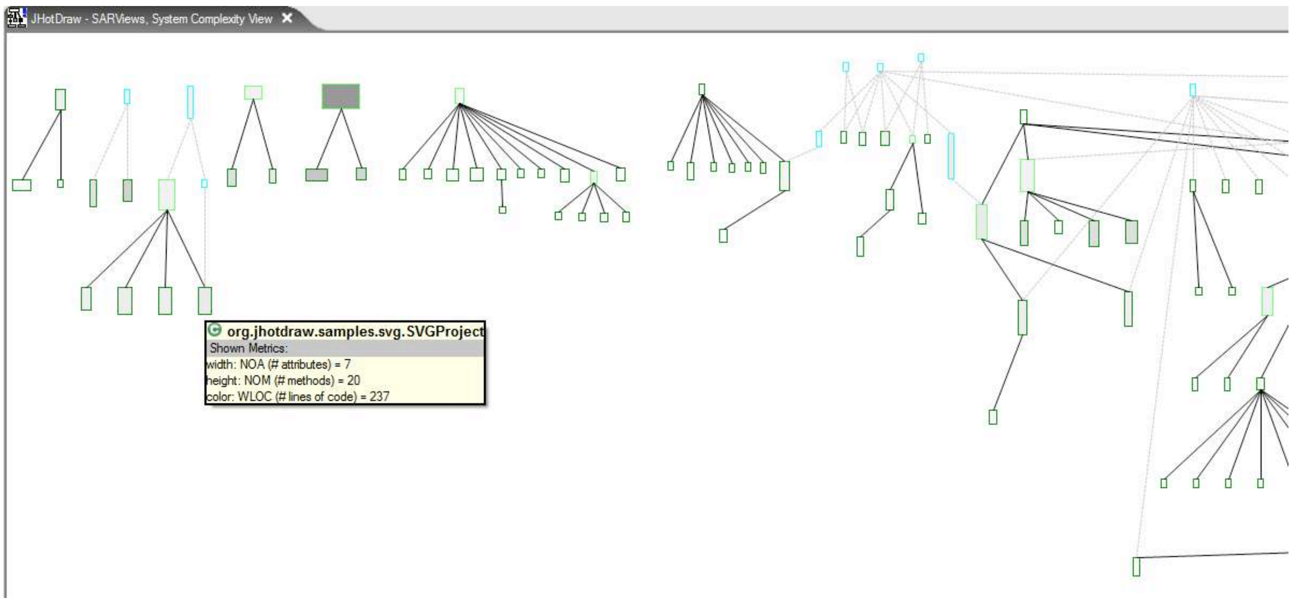
The System Complexity view

The System Complexity view is used to obtain a general knowledge about the complexity of the analyzed system. It is another polymetric view already used in CodeCrawler [CC]. The view is generated by the calculation of these object oriented metrics on each single class composing the system:

- Number of attributes (NOA): the number of attributes declared by the classes;
- Number of methods (NOM): the number of methods declared by the classes;

- Number of lines of code (WLOC): the lines of code that compose each single class.

The classes of the systems are organized in hierarchies according to the extension and implementation relationships that are found among them. Therefore, each link connects two classes that are placed on two different layers, so that classes on the lower layer extend classes placed on the layer immediately above.
Figure 2 represents the System Complexity view of JHotDraw.



org.jhotdraw.samples.svg.SVGProject
Shown Metrics:
width: NOA (# attributes) = 7
height: NOM (# methods) = 20
color: WLOC (# lines of code) = 237

Fig.2 - The System Complexity view of JHotDraw

The Type Graph view

The Type Graph view can be centered on each single class of the system. Starting from this class, all the classes related to it are shown, specifying which kind of relationships connect the target class to the others. Four kinds or relationships can exist among classes, each one represented by a particular arrow, according to the following specification:

↑	Inheritance: the source class extends the destination class
⇡	Implementation: the source class implements the destination interface
⇡	Association: the source class has a reference to the destination class
⇡	Containment: the source class contains the destination class in its body

The Type Graph view is mutuated from Doxygen's class diagrams [Doxy], adapting them to show various kinds of relationships according to the previous table. This view gives an immediate glance to the concrete classes composing the analyzed system, and helps in categorizing the interactions among them.
Figure 3 represents the Type Graph related to the AbstractTool class of JHotDraw.

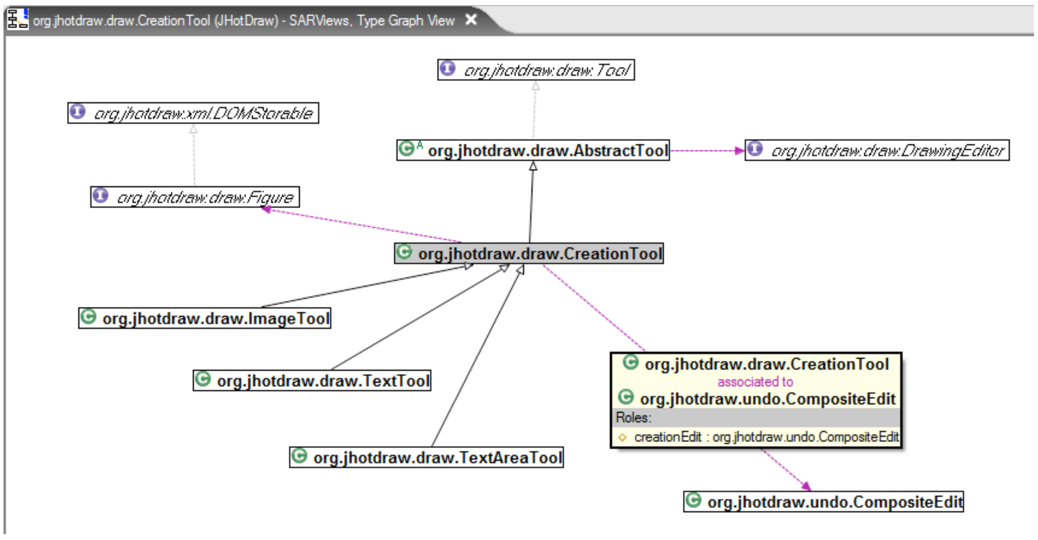

org.jhotdraw.draw.CreationTool (JHotDraw) - SARViews, Type Graph View
org.jhotdraw.draw.Tool
org.jhotdraw.xml.DOMStorable
org.jhotdraw.draw.AbstractTool
org.jhotdraw.draw.DrawingEditor
org.jhotdraw.draw.Figure
org.jhotdraw.draw.CreationTool
org.jhotdraw.draw.ImageTool
org.jhotdraw.draw.TextTool
org.jhotdraw.draw.TextAreaTool
org.jhotdraw.draw.CreationTool
associated to
org.jhotdraw.undo.CompositeEdit
Roles:
creationEdit : org.jhotdraw.undo.CompositeEdit
org.jhotdraw.undo.CompositeEdit

Fig. 3 – The Type Graph related to the AbstractTool class of JHotDraw

MICROSTRUCTURES-BASED VIEWS

These views are generated by analyzing the output of the Basic Elements Detector module. Three views belong to this category:

- The Class Compact view;
- The Class Extended view;
- The Package view.

The Class Compact view

The Class Compact view represents alle the classes and interfaces belonging to a single package, and the relationships among them, basically aggregation, assotiation, extension and implementation. These relationships are deducted and represented according to the Elemental Design Patterns (EDPs) that connect each couple of related classes.
This view is useful especially with small-sized systems, with packages composed by a little numer of classes, in order to avoid too much overwhelmed views. Figure 4 represents the Class Compact view of the CH.ifa.draw.figure package.

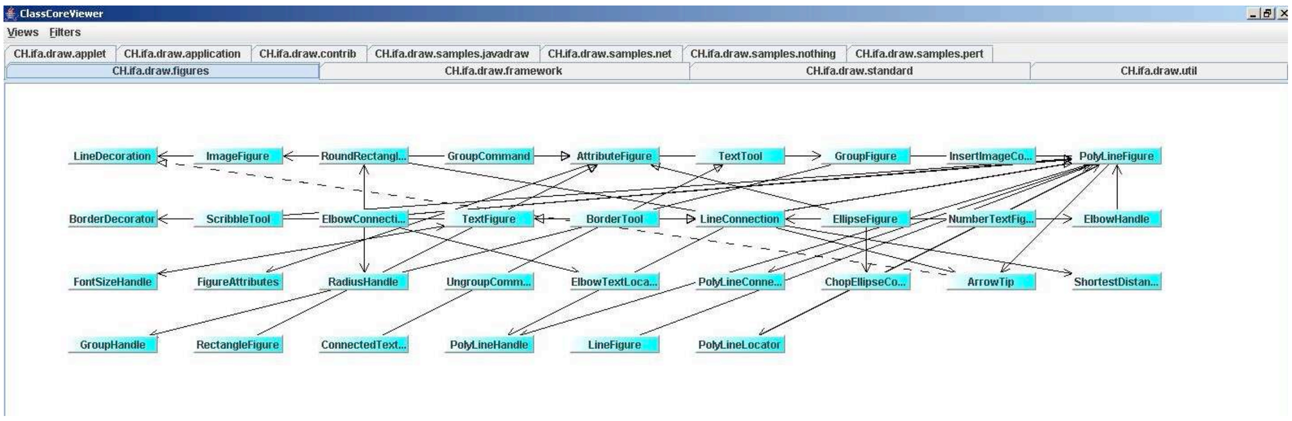


Fig.4 – The Class Compact view of the CH.ifa.draw.figure package

The Class Extended view

The Class Extended view represents alle the classes and interfaces belonging to a single package, and the relationships among them, basically aggregation, assotiation, extension and implementation. Differently from the previous view, the Class Extended view is composed by many different graphs, each one deriving from a single class. In this way, each graph represents the relationships that each single class has with its immediate neighbours, according to the same relationships that are used also in the previous view. Figure 5 represents the Class Extended view of the CH.ifa.draw.samples.javadraw package.

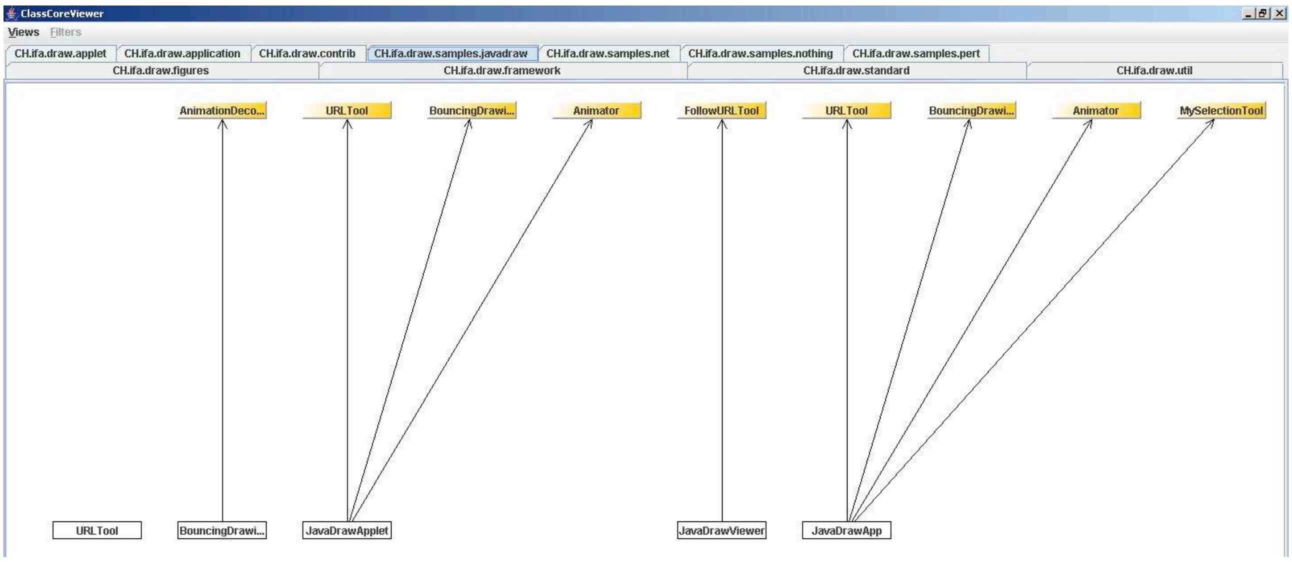


Fig. 5 – The Class Extended view of the CH.ifa.draw.samples.javadraw package

The Package view

The Package view represents all the packages composing the system and the relationships among them. A certain kind of relationship is drawn between two packages if the same kind of relationship has been found at least between a class belonging to the source package and a class belonging to the destination package. Figure 6 represents the packages belonging to JHotDraw and all the relationships among them.

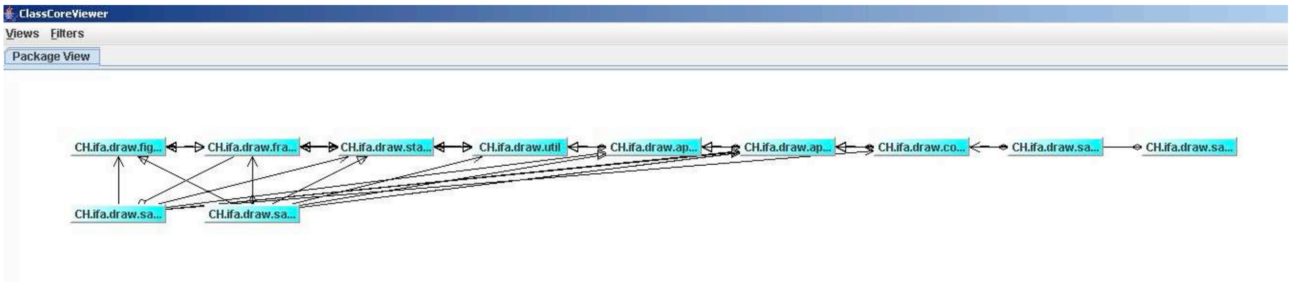


Fig. 6 – The Package view of JHotDraw

REFERENCES

[CC] CodeCrawler, www.inf.unisi.ch/faculty/lanza/codecrawler.html

[LD03] M. Lanza, S. Ducasse, Polymetric Views—A Lightweight Visual Approach to Reverse Engineering, *IEEE Transactions on Software Engineering*, Vol. 29 No. 9, September 2003.

[SA4J] IBM Structural Analysis for Java, www.alphaworks.ibm.com/tech/sa4j

[Doxy] Doxygen, www.doxygen.org/