# JCHVT: A Kind of Software Visualization Method for Java Code

Guoqiang Wang[1], Changzhen Hu[1], Zhigang Wang[1,3], Jun Zhen[2], Zhongyang Kan[1]
1. Lab of Computer Network Defense Technology, Beijing Institute of Technology
2. Beijing Laboratory of Intelligent Information Technology, Beijing Institute of Technology
3. School of Mechatronical Engineering, Beijing Institute of Technology
Beijing, China
Email: wgq_121@163.com

***Abstract*—Java Code Hierarchical Visualization Tool (JCHVT) is a software visualization system oriented to Java source code, which has been designed on the concept of service and with reference to Class diagram and Polymetric view. JCHVT is able to automatically create a hierarchical view of the software system, which can reflect the relationships among classes, structure of the system and even the roles of classes played in the system. Meanwhile, JCHVT provides various methods of human-computer interactions which can help user to understand the graph and identify the useful information in the view.**

***Keywords-software visualization; hierarchical view of system structure; class diagram; Java code***

## I. INTRODUCTION

Software visualization [1] [2] concerns about the static or animated 2-D or 3-D visual representation of information about software systems based on their structure, history, or behavior. Software visualization is helpful for maintenance, reengineering, and evolution of software systems, especially on reverse engineering based on source code. Moreover, it also can identify the system's components and their relations by graphical view, especially for Object-Oriented program, and can create representations of the system at a higher level of abstraction.

There are lots of tools for software visualization based on Polymetric views [3] [4], which permit to enrich visualization with up to five different software metrics, two for the size, two for the position, and one for the color of each node [3]. Typical tools based on this theory like CodeCrawler [5], Softwarenaut [6], X-Ray [7], CodeCity [8] [9], etc, of which Softwarenaut's feature is providing views for the dependencies between modules. X-Ray is based on CodeCrawler, Softwarenaut and Creole[10], but works as a Plug-in of Eclipse. While CodyCity implements a 3D view to realize Polymetric views. System views based on Polymetric views can be divided into two categories: views based on inheritance and non-inheritance relationship. The system views based on inheritance relationship cannot express the relation between different inheritance trees, how they cooperate with each other in the system, and the intuitionist system structure. While the system views based on non-inheritance have limitations to express classes' relations and can only reflect one side of system structure, e.g. the views created by CodeCity can only express the information of classes' files storage and the information of classes, but cannot express the relations among classes. Class diagram of UML is often used to reveal the information of the structure of software. However it's not suitable for such situation which requires the view or diagram to be created directly from source code. Firstly, the relationships in class diagram UML obtained through reversing are not accurate, for the relations defined in UML are semantically related and can only be obtained by specific code environment and semantic. Secondly the graph structure of class diagram have too many intersection points and circle paths which keep the views away from expressing the structure of software system clearly.

For the reasons mentioned above, a novel software visualization is declared: Hierarchical System Structure View, which is based on the concept of service and combined with Polymetric views. It is built for the principle of maximizing to reflect the system structure. HSSV can not only express the different relations between classes, but also reflect the structure of software system and the role of each class played in the system clearly, which is helpful for developer to quickly understand the structure of system and identify the important parts from source code. Based on HSSV, we designed a kind of software visualization tool by Java source reverse engineering, which is called Java Code Hierarchical Visualization Tool (JCHVT).

## II. JCHVT SYSTEM STRUCTURE

JCHVT is a kind of Java visualization program which can automatically generate HSSV from Java source code. The whole system consists of five modules, as shown in Fig. 1:

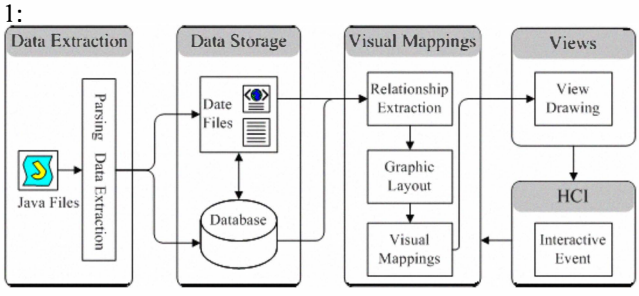


Figure 1. The system structure of JCHVT

*Data Extraction module* is used to analyze and extract keywords from Java files. *Data Storage module* implements the management of the original data, including the interfaces for database and data files (e.g. xml, csv). *Visual Mappings*

978-1-4244-7237-6/10/$26.00 ©2010 IEEE

*module* transforms the original data tables into visual data tables (visual abstraction), which firstly extracts the relations between classes from database or data files, then calculates the location of each class node in HSSV by graphic layout algorithm, finally take the data structure to subject to visual mappings to create a visual abstraction, a data model that includes visual features such as spatial layout, color, size, and shape. The visual abstraction is responsible for containing all the information needed to draw a visual representation of the data. Then the *Views module* will draw the contents of the visual abstraction into any number of interactive views. *Human-Computer Interaction (HCI) module* is designed for dealing events. Interactive events generated by actions of mouse or keyboard can be feed back into this system, causing changes or updates at any stage of the visualization pipeline.

## III. JCHVT IMPLEMEMTATION

### A. Element Representation

#### 1) Representation of Class

In this paper, we classify Java classes into four types: Abstract, Interface, General and Undefined, as shown in Table 1. Abstract type is the class defined by the keyword "abstract" in the class files, and the Interface type defined by the keyword "interface". Other classes defined in the class files are General type. However, there are some classes which are not defined in the source files, such as the classes coming from external packages or libs, and whose type cannot be inferred from the source files. These classes are defined as Undefined type. The representation of each kind of class including label title format, filled color and node style is shown in Table 1.

TABLE I. THE CLASSIFICATION AND REPRESENTATION OF CLASS

| Classification | Label Title Format | Color | Example |
|---|---|---|---|
| Abstract | Italics | Pink | *A* |
| Interface | Include in"<<>>" | Green | <<A>> |
| General | General | Yellow | A |
| Undefined | Include in"[]" | Orange | [A] |

#### 2) Representation of Relation

Based on the concept of service, the system divides the relations between classes into five types: Nested Relation (NR), Extends Inheritance (ER), Implements Inheritance (IR), Constructor contain (CR), Method include (MR), as shown in Table 2. If the class A is embedded in class B, the relation between class A and B is B NR A; if the class B is extended or Implemented from class A, the relation defines as B ER A or B IR A; else if A's object is B's attribute or a parameter of B's constructor function, that is B CRA; else if A is the value of B's function returned or the parameter of B's function (not constructor function), that is B MR A. Each relation types are represented as a direct line with different styles, shown in Table 2, column 5.

TABLE II. THE REPRESENTATION OF RELATION

| Relation | Representation | Explanation | P |
|---|---|---|---|
| Nested relation | A ◆— B | A is embedded in B | 1 |
| Extends inheritance | A —▶ B | B extends from A | 2 |
| Implements inheritance | A —▷ B | B implements from A | 2 |
| Constructor contain | A —▶ B | A is B's attribute or is a parameter of B's constructor function | 3 |
| Method include | A — B | A is the value of B's function returned or the parameter of B's function | 4 |

### B. Visualization Implementation

#### 1) Priority for Relation

According to coupling extent, relations are divided into four grades, just as the last column (P) of Table 2 shown. The smaller value means higher priority. ER and IR have the same priority value, because they are mutually exclusive. A is a part of B, if the relation of A and B is B NR A. that means they have highest coupling. B ER/IR A implies that B has the information of its parent class A, namely that B is A's specific, so ER/IR have higher coupling. NR, ER and IR are the main relationships between classes of software system. Compared with NE, ER or IR, CR's and MR's coupling extent is weaker. However B CR A implies that A's object is B's attribute or can convert to, which means that CR have higher coupling than MR.

#### 2) Hierarchical System Structure View (HSSV)

HSSV's implementation is based on the concept of Service Providers and Service Beneficiaries, which defined as follows:

**Service Provider**: as shown in Table 2, column 2, define the left class of the relation is Service Provider. Compared with the right class the left class is basic and provides data structure, interface or data management for the right one.

**Service Beneficiary**: compared to Service Provider, the beneficiary will implement certain function based on the Service Provider. Service Beneficiary and Service Provider present as a pair.

HSSV actually is a Directed Acyclic Graph (DAG) G<V, E>. V is the node set. Elements of V are classes in the software system, which styles are shown in Table 1. E is the edge set, which styles are shown in Table 2. E can be got from R {NR, ER, IR, CR, MR}, using the priority to remove the relations, which priority are lowest and which can cause multiple edges, from R, then using the Service Provider and Beneficiary to remove the circuits from R. The nodes which in-degree is zero are used for the first layer. Then JCHVT

subject to the relation of Service Provider and Beneficiary to do deep search and set the nodes with the same depth in the same layer. As soon as the search is complete, the HSSV is created.

## IV. SYSTEM APPLICATION

In addition, the HSSV created by the JCHVT also provides a number of human-computer interactions, such as dragging into a view, zooming into a view, searching a special node, changing the layout and hiding a certain node. All of the human-computer interactions are helpful for understanding the information shown by the view.

Fig. 2 shows a certain HSSV, created by JCHVT from a minesweeper game developed by Java, which is downloaded from internet. The figure shows following information:

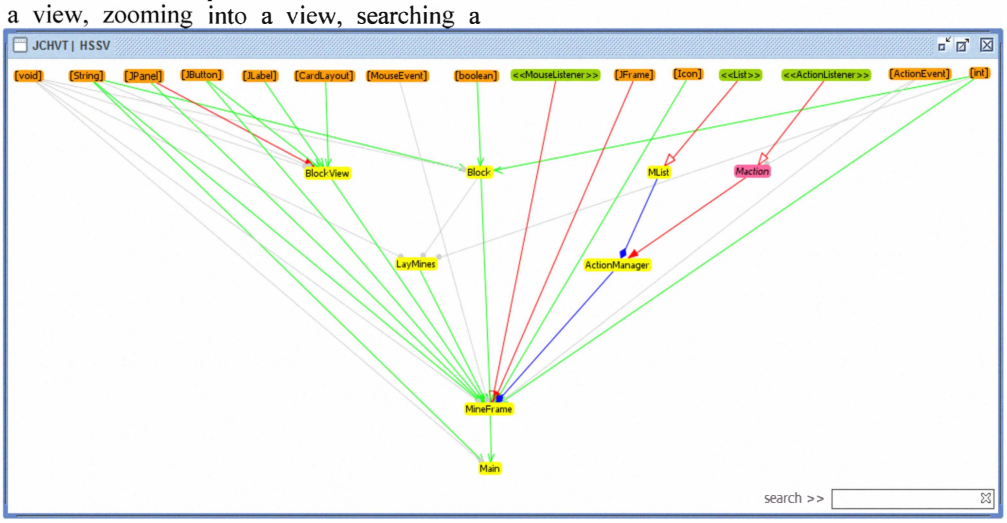


Figure 2. HSSV for a minesweeper game (including all classes).

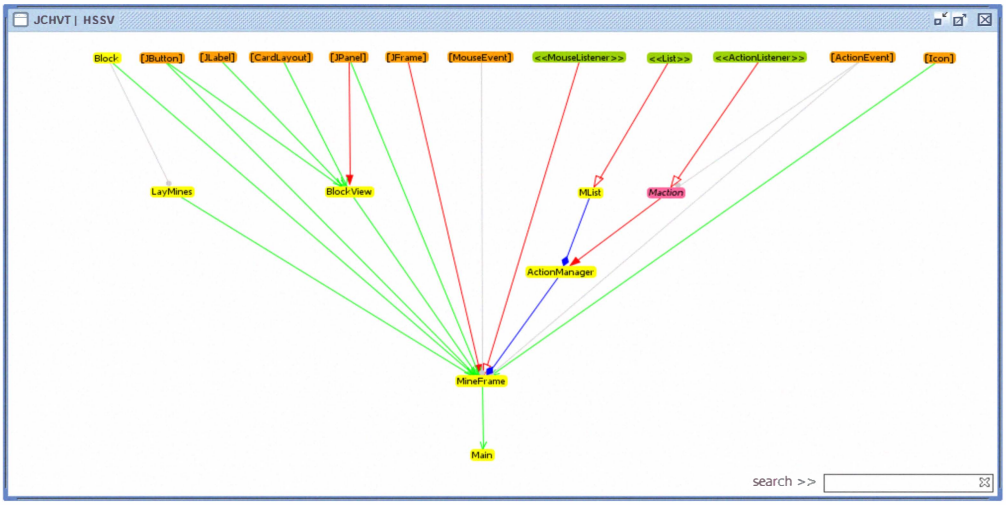


Figure 3. HSSV for a minesweeper game (removed some basic classes)

### A. *The Relations among Classes*

The inheritance relations between classes can be easily identified by the red line with triangular arrow. Even the details of inheritance can be determined by the style of the arrow and the node. Take Maction for instance, with an unfilled triangular arrow, it is a class which implements interface from class ActionListener, and node ActionManager, while it is extended from the abstract class Maction with a filled red triangular arrow. At the same time, ActionManager also has an arrow with diamond-shaped, which means ActionManager contains the body of class MList. Apart from those relations mentioned above, CR or MR are shown clearly in the figure too with green lines ended by arrow or with gray lines ended by dot.

### B. *The Relation between different Inheritance Trees*

In Fig. 2, the relations among different inheritance trees can also be observed. For example, there is a line, which comes from BlockView to MainFrame highlighted with green color. This line connects the tree which consists of

JPanel and BlockView with the tree whose subclass is MainFrame.

### C. Structure of the System

The most important point is that the HSSV can reflect the structure of software system. The following information can be got, as shown in Fig. 2:

- **How the system composed of and what role the class plays in the system.** Because the view is drew based on the concept of service, Classes on top layer of HSSV mostly are the basic, such as: void, String, int, boolean, which are all the basic variable types in Java language; JPanel, JButton, JLabel, JFrame, which are all basic classes for UI, coming from the swing package; MouseEvent, MouseListener, ActionListener, ActionEvent which are all basic classes for events. The deeper the class node location is, the more specific the class is. For example, the class ActionManager implemented from class ActionListener, which is the basic interface of action, may deal with the event for this application.
- **How the system implements and which part is important for users.** Main class is the entrance of the system and then a MainFrame object is instanced, which plays as the center of the system, which manages other classes, such as ActionManager and BlockView, etc, to complete the function of the system. So, for a learner who wants to learn or change this system should pay more attention to this center class and can save a lot of time.
- **What the system will do.** Pay attention to MainFrame shown in Fig. 3, which is clearer than Fig. 2. Classes connected with MainFram can be divided into three parts. One is the listeners which imply the system will implement human-computer interaction; one is BlockView whose Service Providers have reference with UI, which means this part will complete the designing and drawing UI. The other part is LayMines whose Service Providers are Block and int, which can be inferred to implement the layout algorithm.

Look up Fig. 2, there are lots of intersections, mostly starting with the node on the top layer, such as: void, int, String, boolean. However, most of them are the Java language basic classes, which can be removed without bringing much negative impacts. Using the human-computer interaction to hide those nodes, the new view is shown in Fig. 3, which looks clearer and possesses more readability.

## V. CONCLUSION

Experiment shows that HSSV, which is created by JCHVT, has more advantages such as clear-cut layers, less intersection points, definite relations, and better interaction. HSSV can reflect the structure of reversed software system well, relations among classes and inheritance trees, as well as information about classes' roles which play in the system. The future work will try to combine information about packages that contain Java physical files and information about classes together to accurately represent the physical file structure while describing software system structure.

## ACKNOWLEDGMENT

This work is supported by a grant from the National High Technology Research and Development Program of China (863 Program) (No. 2009AA01Z433)

## REFERENCES

[1] S. Diehl, Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Berlin, German: Springer, 2007, pp. 3-4.

[2] "Software visualization," Internet: http://en.wikipedia.org/wiki/Software_visualization, Mar.1, 2010 [Apr.15, 2010].

[3] M. Lanza, "Object-Oriented Reverse Engineering — Coarse-grained, Fine-grained, and Evolutionary Software Visualization," Ph.D. thesis, University of Bern, May 2003.

[4] M. Lanza and S. Ducasse, "Polymetric Views - A Lightweight Visual Approach to Reverse Engineering," IEEE Transactions on Software Engineering, 29(9):782–795, September 2003.

[5] M. Lanza, S. Ducasse, H. Gall, and M. Pinzger, "Codecrawler: an information visualization tool for program comprehension," in ICSE 2005: Proceedings of the 27th international conference on Software engineering, New York, NY, USA: ACM, 2005, pp. 672–673.

[6] LUNGU M, LANZA M. Softwarenaut: Exploring hierarchical system decompositions[C]. In Proceedings of CSMR 2006 (10th European Conference on Software Maintenance and Reengineering).Los Alamitos CA :IEEE Computer Society Press, 2006, pp. 351-354.

[7] J. MALNATI, "X-Ray — A Software Visualization Plug-in for Eclipse," Internet:http://atelier.inf.unisi.ch/~malnatij/xray.html, Jul.5 2008 [Oct.13, 2009].

[8] R. Wettel and M.Lanza, "Program Comprehension through Software Habitability," Proceedings of the IEEE Int. Conf.e on Program Comprehension (ICPC), NW Washington, DC USA: IEEE Computer Society, 2007, pp. 231-240.

[9] R. Wettel and M. Lanza. "Codecity: CodeCity: 3D visualization of large-scale software," In ICSE Companion 2008: Companion of the 30th International Conference on Software Engineering, New York ,USA : ACM Press, 2008, pp. 921–922.

[10] R. Lintern, J. Michaud, M. Story, and X.Wu, "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse," In Proceedings of ACM Symposium on Software Visualization, June 2003.