Visualizing Arrays in the Eclipse Java IDE

Bilal Alsallakh, Peter Bodesinsky[1], Silvia Miksch, Dorna Nasseri[2]
Centre of Visual Analytics Science and Technology (CVAST)
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
Email: {alsallakh, miksch}@cvast.tuwien.ac.at, {[1]e0304343, [2]e0227552}@student.tuwien.ac.at

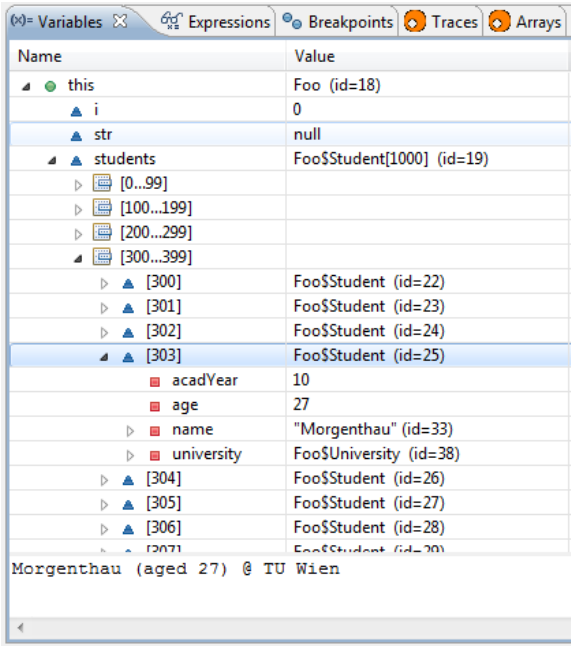
***Abstract*—The Eclipse Java debugger uses an indented list to view arrays at runtime. This visualization provides limited insight into the array. Also, it is cumbersome and time-consuming to search for certain values at an unknown index. We present a new Eclipse plugin for visualizing large arrays and collections while debugging Java programs. The plugin provides three views to visualize the data. These views are designed to support different tasks more efficiently. A tabular view gives detailed information about the elements in the array, such as the value of their field variables. A line chart aims to depict the values of a numerical field over the array. Lastly, bar charts and histograms show how the values of a field are distributed. We show how these views can be used to explore linear data structures and hashes from the Collections Framework. The plugin features tight integration with the Eclipse IDE, and is freely available as an open-source project. Developers' feedback confirmed the utility of the plugin to explore large arrays in real-world scenarios.**

I. Introduction

The debugger is one of the most important components of modern IDEs. It comprises a set of tools that aim to test and debug programs while they are being executed. One important feature of the debugger is to watch the values of variables while stepping through the program. Watching variables when the execution is suspended is essential for understanding the current state of the program, and hence localizing bugs. Eclipse has emerged as one of the most advanced IDEs for the Java programming language [1]. The debugger offered by Eclipse Java development tools (JDT) allows the navigation in the frames of the stack-trace of all suspended threads. At each frame the developer can watch and modify variables accessible at this frame, or evaluate valid expressions at this frame. Also, the developer can advance the execution at the location in the source-code that corresponds to the selected frame. Eclipse offers a dedicated view to watch and modify variables, and a dedicated view to evaluate expressions.

When the value being watched is of type `ReferenceType` (i.e. sub-class of `Object`), Eclipse presents it in an indented list (Fig. 1). The root of this list represents the value itself. Sub-elements of the root represent the values of the field variables of this object. If a sub-element is of type `ReferenceType` it can be further expanded and visualized in the same manner. Elements of `PrimitiveType` (such as `int` or `char`) cannot be further expanded, but their values can be changed in place. Additionally, a textual representation for the selected element is displayed (using the `toString()` method for elements of type `ReferenceType`).

When the value being visualized is an array, the sub-elements represent the values in this array. Here also, the sub-elements can be individually expanded if they are of type `ReferenceType`. The indented list provides detailed information about an individual element, but it is not suitable for gaining an overview of the whole array. Moreover, navigation in the list becomes more difficult when the array is large. Eclipse copes with this problem by grouping the elements of a large array into buckets. Each bucket contains up to one hundred elements. This solution helps when the developer wants to inspect an element at a known index. However, the indented list is inefficient when the developer wants to find certain values at unknown indexes.

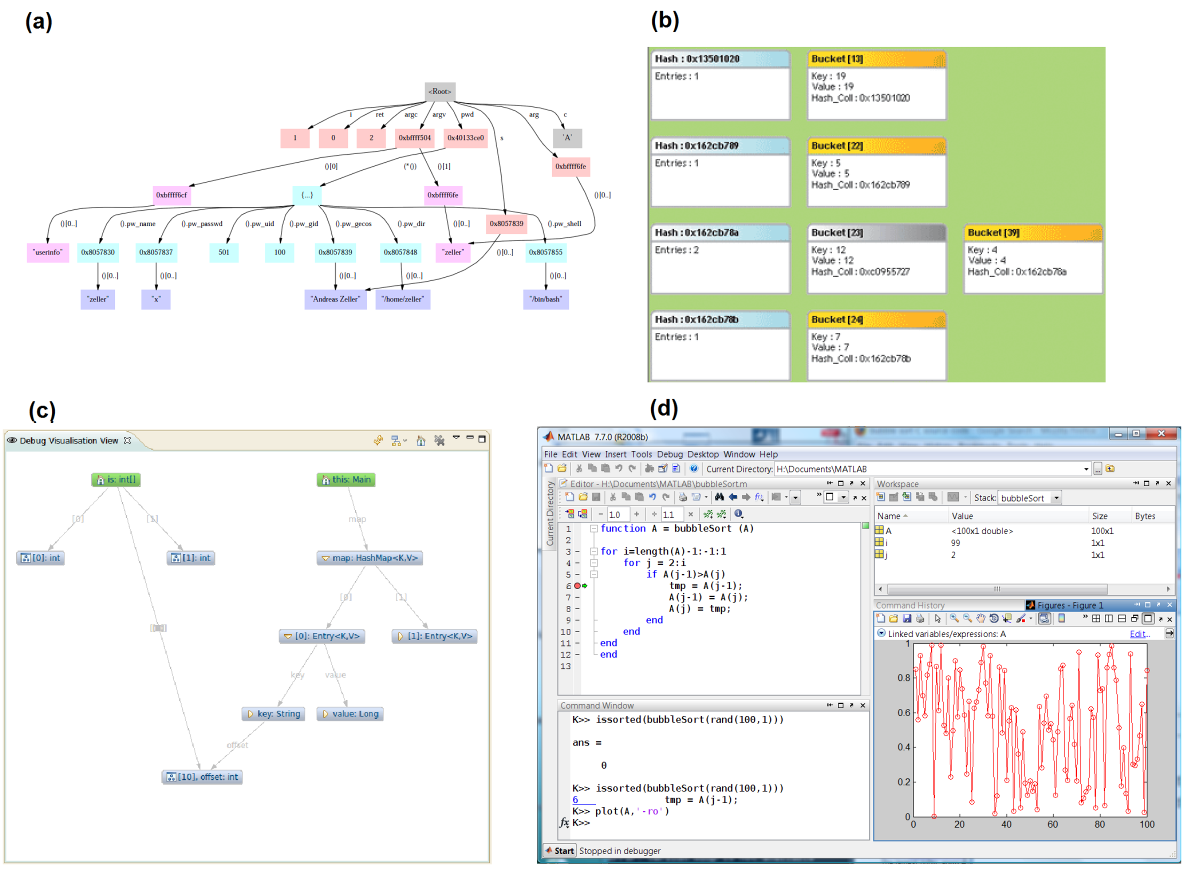


Name	Value
this	Foo (id=18)
i	0
str	null
students	Foo$Student[1000] (id=19)
[0...99]	
[100...199]	
[200...299]	
[300...399]	
[300]	Foo$Student (id=22)
[301]	Foo$Student (id=23)
[302]	Foo$Student (id=24)
[303]	Foo$Student (id=25)
acadYear	10
age	27
name	"Morgenthau" (id=33)
university	Foo$University (id=38)
[304]	Foo$Student (id=26)
[305]	Foo$Student (id=27)
[306]	Foo$Student (id=28)
[307]	Foo$Student (id=29)

```
Morgenthau (aged 27) @ TU Wien
```

Fig. 1: Watching variables in Eclipse. Arrays can be also explored in the indented list. Array elements are grouped into buckets. An element must be expanded to explore its content.

In this work we present an alternative method to visualize arrays and collections in Eclipse. Our method intends to cope with the above-mentioned limitations by providing a quick overview of the array elements and their sub-field, and an efficient mechanism to search for certain values in the array.



Hash : 0x13501020 — Entries : 1 — Bucket [13]: Key : 19, Value : 19, Hash_Coll : 0x13501020

Hash : 0x162cb789 — Entries : 1 — Bucket [22]: Key : 5, Value : 5, Hash_Coll : 0x162cb789

Hash : 0x162cb78a — Entries : 2 — Bucket [23]: Key : 12, Value : 12, Hash_Coll : 0xc0955727 — Bucket [39]: Key : 4, Value : 4, Hash_Coll : 0x162cb78a

Hash : 0x162cb78b — Entries : 1 — Bucket [24]: Key : 7, Value : 7, Hash_Coll : 0x162cb78b

```
function A = bubbleSort (A)

for i=length(A)-1:-1:1
    for j = 2:i
        if A(j-1)>A(j)
            tmp = A(j-1);
            A(j-1) = A(j);
            A(j) = tmp;
        end
    end
end
end
```

Name	Value	Size	Bytes
A	<100x1 double>	100x1	
i	99	1x1	
j	2	1x1	

```
K>> issorted(bubbleSort(rand(100,1)))

ans =

     0

K>> issorted(bubbleSort(rand(100,1)))
6               tmp = A(j-1);
K>> plot(A,'-ro')
K>> 
```

Fig. 2: Methods for visualizing data structures while debugging: (a) Memory Graphs [2]. (b) Type Visualizers [3] in Visual Studio. (c) Debug Visualization Plugin for Eclipse [4]. (d) Linked plots [5] in MATLAB.

II. Related Work

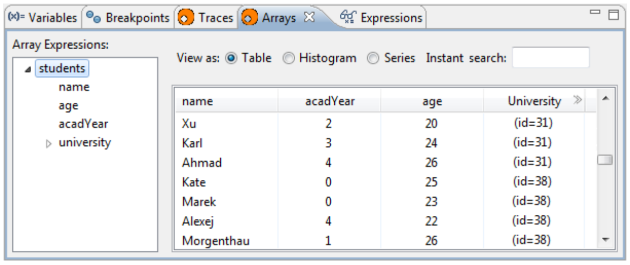
A variety of approaches and tools have been devised to visualize data structures and memory state for debugging purposes. Zimmermann and Zeller [2] presented a method for visualizing memory graphs (Fig. 2-a). Their method enables querying the graphs and comparing memory states. Korn and Appel [6] proposed a traversal-based approach for visualizing data structures. Their method enables user-defined visual mappings to be used to produce abstract displays of the data and find specific patterns. Methods to visualize data structure have also been developed for popular IDEs. Type Visualizers [3] are a general mechanism for data structure visualization in Visual Studio. It allows the developer to write custom classes to specify how to display a certain data type. Fig. 2-(b) shows a visualization of the key-value pairs of a hashmap. The Debug Visualization Plugin for Eclipse [4] applies graph-based techniques to display and navigate in data structures. It uses directed graphs to display Java objects as nodes and relations between them as edges (fig. 2-c). Variables can be added to the graph from the "Variables" view in Eclipse. Array variables are also displayed as a directed graph. The root node represents the array itself, and the child nodes represent the array elements. Due to the linear arrangement of elements in an array, a graph-based representation might not be suitable for gaining insight into large arrays. In MATLAB, variables can be visualized in linked plots while debugging programs [5]. Fig. 2-(d) shows a linked plot of a one-dimensional matrix. The Vcc program animation tool [7] has a dedicated view for visualizing numerical arrays as bar charts or line charts. Kranzlmüller et al. ([8], [9]) proposed a method for visualizing multi-dimensional numerical arrays for debugging parallel programs. The above presented methods and tools either do not focus on arrays or are restricted to numerical arrays. We address these limitations by providing a method for visualizing generic arrays in the Eclipse Java IDE.

III. Array Visualization

Our approach visualizes arrays in a dedicated Eclipse view. A list in the left panel of this view shows the variables or the expressions of array type that are selected for visualization. If the array is of type `ReferenceType`, its node can be expanded to explore the fields of this type. When a list item is double-clicked, a visualization of the corresponding array is added to the right panel. The user can select between three types of visualizations, depending on the information she needs: a tabular listing, a line chart, or a bar chart (or histogram). The next sub-sections present these visualizations in details.

A. *Tabular Listing*

This view provides a textual listing of array elements in a table (Fig. 3). When the array is of type `ReferenceType`, the table columns represent the fields of this type. A column can be further expanded if the field it represents is of type `ReferenceType`. This substitutes it with a group of columns that represent the fields of this type.



View as: [✓ Table] Histogram Series

name	acadYear	age	University
Xu	2	20	(id=31)
Karl	3	24	(id=31)
Ahmad	4	26	(id=31)
Kate	0	25	(id=38)
Marek	0	23	(id=38)
Alexej	4	22	(id=38)
Morgenthau	1	26	(id=38)

Fig. 3: Tabular view of an array.

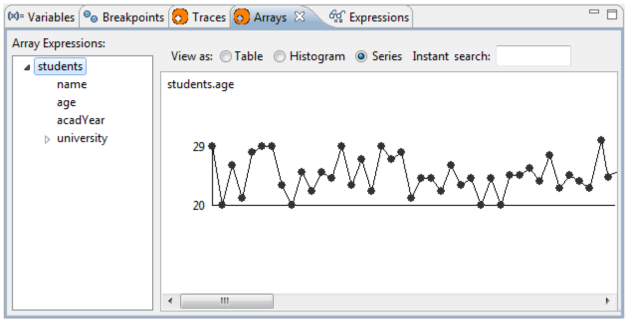
Compared with the built-in view for exploring arrays in Eclipse (Fig. 1) the tabular view provides a more natural way for visualizing generic arrays. It shows more details about the array at once, and is more space-efficient. Also, it makes it easier to compare values at arbitrary depth (i.e. inner object) from different elements. Additionally, it offers richer interactivity that enables more insight into the array. For example, by sorting the table by a column, or by filtering it, searching for specific field values at an unknown index becomes easier. Also, using the instant search functionality, values that contain a specific text can be quickly highlighted. Color can also be used to highlight certain values such as `null`.

However, the tabular view can only show information from the compile-time type of the array. Information from the actual runtime type of an individual element needs to be inspected in the Eclipse popup dedicated for this purpose. This popup can be opened by right-clicking the element. Color can be used to reveal the different runtime types of the elements.

B. *Line Charts*

The values of a numerical array or of a numerical field in an array can be plotted in a two-dimensional line chart (Fig 4). The x-axis represents the index of the array element, while the y-axis represents its value. The line chart offers an alternative to the text-based visualization used by the tabular view. It provides a compact overview of how the values vary along the array. Also, the developer can identify where the high and low values are concentrated. More importantly, line charts of different arrays can be compared by plotting them below each other. An array element or the sub-field value being visualized can take the `null` value. This is represented by a discontinuity in the line chart.

When the values are not numeric, the y-axis cannot be used to show different values. Hence, line charts are not suitable to show useful information about non-numeric arrays.

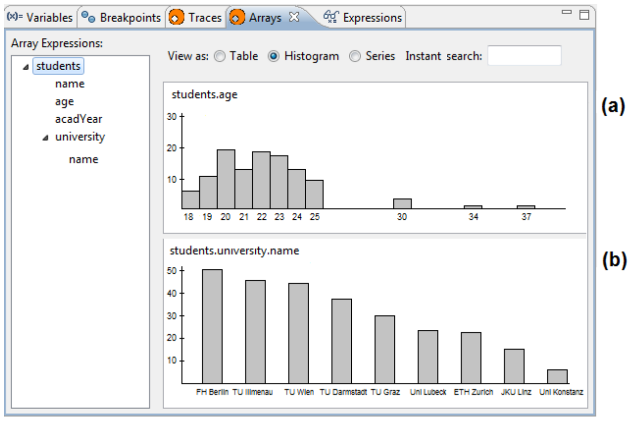


View as: Table Histogram [✓ Series]

Fig. 4: Line chart of the values of a numerical field.

C. *Histograms and Bar Charts*

For a numerical array or a numerical field in an array, a histogram can be plotted to show the distribution of the values in the array (Fig. 5-a). It helps in quickly finding the maximum and minimum of these values. For a non-numerical array, a bar chart of all unique values can be plotted to show the frequency of each value in the array (Fig. 5-b).



View as: Table [✓ Histogram] Series

Fig. 5: Frequency-based visualizations: (a) a histogram of a numerical array and (b) a bar chart of a reference array.

Both histograms and bar charts have the advantage of being familiar visualizations. They allow to quickly find the most frequent elements in the array. Moreover, frequency-based visualizations have the advantage of higher scalability for numerical arrays: the number of bins is highly independent of the number of elements. Likewise, large non-numerical arrays can be visualized effectively when the number of distinct values is reasonably small.

IV. Implementation Details

We implemented our methods as a plugin for Eclipse. The plugin provides an Eclipse view named ”Arrays”. It also extends Eclipse menus that deal with watching variables and expressions with a menu item for visualizing arrays. This item is enabled when the selected variable or expression evaluates to a `Value` of type

`ArrayReference`. The context menu of the Java editor was extended by defining a `viewerContribution` for `traget#CompilationUnitEditorContext`. The context menus in the ”Variables” and ”Expressions” views where extended by defining an `objectContribution` for items of the following classes:

- `org.eclipse.debug.core.model.IWatchExpression`
- `org.eclipse.jdt.debug.core.IJavaVariable`
- `com.sun.jdi.ObjectReference`

When the menu item is clicked, the ”Arrays” view is activated. The corresponding variable or expression is added to the item list and is evaluated to array values using API of Java Debug Interface (JDI) and Eclipse JDT. The values are visualized in a new part in the visualization area. To refresh the views upon stepping, we extend the command `org.eclipse.debug.ui.commands.Resume` provided by Eclipse, with a contribution. In this contribution, the expressions of the open array views are re-evaluated and the views are updated. The user can choose to disable the automatic updating of some arrays.

V. Discussion

We performed an informal evaluation of our tool with practitioners that have been using Eclipse to develop Java applications for several years. No tasks were required to be solved. We were mainly interested in finding out how intuitive the proposed views are, and whether developers can make use of them in real-world scenarios. The visualizations were perceived as intuitive and familiar. The tabular listing was found the most useful for many practical scenarios. This is probably due to the fact that it views the values of all array fields for all items in a simple and familiar way. The limitation of showing only the fields of the declared array type was considered as a major shortcoming. This happens when the declared type is an interface or a generic abstract class that does not contain the fields needed for debugging.

The utility of histograms and bar charts was appreciated by some developers. They found it a useful means of getting an impression of what different values a specific field takes in the whole array. They mentioned scenarios in which this can help them find errors. Though line charts of numerical arrays were perceived as intuitive, however, most developers preferred the tabular listing over it. Also, line charts do not scale for large arrays (with tens of thousands of element). One developer who works with object management and O/R mapping mentioned that the tool can make his debugging tasks easier. Another developer who works with scientific computing and computer graphics appreciated the utility of the tool in quicking debugging his algorithms. Finally, it is worth mentioning that the plugin imposes no noticeable performance overhead on the Eclipse Java debugger.

Applicability to Collections

The above-mentioned visualizations are not limited to arrays. They can be as well used to explore a generic `Collection` (such as `List` and `Set`). In this case the fields are inferred from the type parameter <`T`> of these generic data structures. Likewise, a `Map` can be treated as a collection whose items are the <key , value> pairs. The tabular view of these pairs is by far more convenient than the built-in indented list which can only show the underlying data structure of the map (a sparse array in the case of `HashMap`).

VI. Conclusion

We presented a novel open-source Eclipse plugin for visualizing arrays while debugging Java programs at runtime. The plugin offers a tabular listing of the array showing the values of the fields of its declared type. Columns can further be expanded if they have sub-fields. Also, a line chart or a histogram (or bar chart) can be created for a specific field. Compared with the built-in array explorer in Eclipse, our plugin provides richer interaction and more insights. It helps in finding desired values at unknown indexes. It also helps in understanding the distribution of the values of a field in the array. Both tasks are cumbersome in the built-in explorer. The proposed views can also be used to explore other linear data structures and hashes from the Java Collections Framework.

Future work will address the limitation of using declared types for determining the table columns. One solution would be to use the most specific type shared among all objects in the array. Another solution is to use getter-methods or methods specified by the user as computed columns. Also, more work is needed for revealing changes and comparing different arrays. Furthermore, we aim to design more suitable visualizations for specific data structures such as trees and graphs. Finally, more evaluation of user experience need to be collected.

Acknowledgement The authors would like to thank the developers from AMOS Austria GmbH who contributed their time for evaluating the work. This work was supported by CVAST Centre for Visual Analytics Science and Technology (funded by the Austrian Federal Ministry of Economy, Family and Youth in the Laura Bassi Centres of Excellence initiative #822746).

References

[1] G. Goth, “Beware the march of this IDE: Eclipse is overshadowing other tool technologies,” *Software, IEEE*, vol. 22, no. 4, pp. 108 – 111, 2005.

[2] T. Zimmermann and A. Zeller, “Visualizing memory graphs,” in *Revised lectures on Software Visualization*. Springer-Verlag, 2001, pp. 191–204.

[3] L. J. H. Anson Horton, Michael Montwil, “Visualizer system and methods for debug environment,” February 2010. [Online]. Available: http://www.freepatentsonline.com/7657873.html

[4] “Debug Visualization Plugin,” http://code.google.com/p/debugvisualisation/, accessed in January 2012.

[5] MATLAB Documentation, “Making graphs responsive with data linking,” http://blogs.mathworks.com/desktop/2008/12/15/visual-debugging-with-linked-plots/, accessed in January 2012.

[6] J. Korn and A. W. Appel, “Traversal-based visualization of data structures,” in *IEEE Symposium on Information Visualization*, 1998, pp. 11–18.

[7] R. A. Baeza-yates, G. Quezada, and G. Valmadre, “Visual debugging and automatic animation of C programs,” in *Software Visualization*, 1996, vol. 7, pp. 46–58.

[8] D. Kranzlmüller and J. Volkert, “Why debugging parallel programs needs visualization,” in *Proceedings of the IEEE VL2000 Workshop on Visual Methods for Parallel and Distributed Programming*. IEEE, 2000.

[9] D. Kranzlmüller and A. Rimnac, “Parallel program debugging with MAD: a practical approach,” in *Proceedings of the 2003 international conference on Computational science*, ser. ICCS’03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 201–210.