Identifying cycle causes with CycleTable

Jannik Laval, Simon Denier, Stéphane Ducasse

Identifying cycle causes with CycleTable

Jannik Laval, Simon Denier, Stéphane Ducasse
RMoD Team, INRIA, Lille, France
firstname.lastname@inria.fr

Abstract

Identifying and understanding cycles in large applications is a challenging task. In this paper we present CycleTable which displays both direct and indirect references. It shows how minimal cycles are intertwined through shared edges and allows the reengineer to devise simple strategies to remove them.

1 Introduction

Understanding the structure of large applications is a challenging but important task. Several approaches provide information on packages and their relationships, by visualizing software artifacts, metrics, their structure and their evolution. Software metrics can be somehow difficult to understand since they are dependent on projects. Distribution Map [1] alleviates this problem by showing how properties are spread over an application. Lanza et al. [2] propose to recover high level views by visualizing relationships. Package Surface Blueprint [3] reveals the package internal structure and relationships among other packages.

In previous work, we enhanced Dependency Structure Matrix (eDSM) [4] to visualize dependencies between packages, showing in each matrix cell information about the dependencies at the fine grain of classes (inheritance, class references, invocations, referencing and referenced classes/methods). eDSM proved useful to provide an overview of dependencies, detect direct cycles (a cycle between two packages) and provide information to remove them. However, removing all direct cycles does not necessarily remove all cyclic dependencies because there could be indirect cycles (between more than two packages). Although indirect cycles are also displayed by eDSM, they are hard to read in the eDSM layout, making the task inefficient.

In this paper, we present a new visualization, called CycleTable, entirely dedicated to cyclic dependencies assessment. CycleTable layout displays both direct and indirect cycles and shows how minimal cycles are intertwined through shared edges. CycleTable combines this layout with the enriched cells of eDSM to provide the details of dependencies at class level.

Next section introduces the challenges of cycle analysis with graph layout and eDSM. Section 3 explains CycleTable layout and enriched cells. Section 4 presents a sample of cyclic dependencies displayed with CycleTable and discusses the criteria to break cycles as highlighted by the visualization.

2 Cycle Visualization

Figure 2 shows a sample graph with five nodes and three minimal cycles. Each edge is weighted. Notice that cycle A-B-C and A-B-E share a common edge (in yellow) from A to B. This shared edge is interesting to spot since it joins two cycles and by removing it we would break those cycles.

Graph layouts offer an intuitive representation of graphs, and some handle cyclic graph better than others. On large graphs, complex layouts may reduce the clutter but this is often not simple to achieve. In addition, in the context of DSM we enhanced the DSM cells to convey much more information about the strength, the nature and consequences of a cycle. Now it is difficult to combine a graph layout with enriched cells (see Figure 4 for a cell sample), as an enriched cell represents an edge in a graph. Enriched cell is an important asset of our work with eDSM as it provides fine-grained details of a dependency between two packages, and enables *small multiples* as well as *micro-macro reading* effects [5].

In eDSM we used a matrix, the traditional support of DSM. It provides a regular structure which is the same at any scale: it handles the repetitive arrangement of enriched cells, enabling the above effects. It is a fundamental element of eDSM design. In Figure 1, four packages belonging to Pharo kernel (an open-source Smalltalk) are presented in a eDSM. It shows edges involved in direct cycles in red and pink cells, indirect cycles in yellow. More explanations about the design and usage of eDSM are available in [4].

While eDSM allows us to analyze direct cycles comfortably, we could not address the problem of indirect cycles left over after removal of direct cycles. The main reason

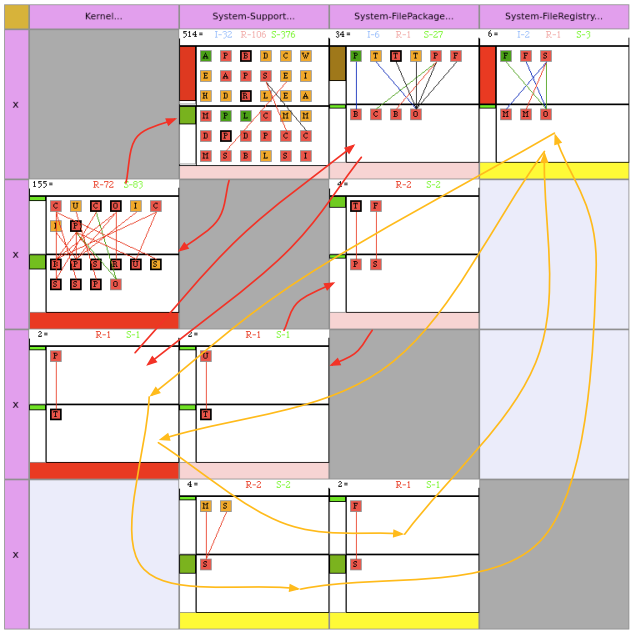

Figure 1. Some Pharo core packages in a eDSM (with cycles overlayed)

for this problem is that it is difficult to read an indirect cycle in the matrix, *i.e.,* to follow the sequence of cells representing the sequence of edges in the cycle. The order can appear quite arbitrary as one has to jump between different columns and rows (this problem does not exist with direct cycles as there is only two cells, mirroring each other along the diagonal). Cycles have been overlayed in Figure 1 to show the complexity of reading indirect cycles, intertwined with direct cycles.

3 CycleTable

We have built the CycleTable design with the single purpose of visualizing cycles. As a consequence, CycleTable does not show a complete overview of dependencies between packages as eDSM does. It complements eDSM. CycleTable is a rectangular matrix where nodes are placed in rows and cycles in columns. CycleTable (i) shows each minimal cycle clearly and independently, (ii) highlights shared edges between minimal cycles, and (iii) uses enriched cells [4] to provide information about edge internals, enabling *small multiples and micro-macro reading* [5] *i.e.,* variations of the same structure to reveal information. We detail each of these points now.

3.1 Minimal Cycle

A minimal cycle is a cycle with no repeated nodes other than the starting and ending nodes. For example, in Figure 2, A-B-E and A-B-C are two different minimal cycles, but A-B-C-D-C is not because C is present twice. With minimal cycles, the visualization provides all edges contains in cycles. Therefore it is not necessary to show complex (intertwined) cycles. In the CycleTable layout, each minimal cycle is represented directly and independently in its own column.

3.2 CycleTable Layout

The CycleTable layout is presented in Figure 3. This figure shows a sample CycleTable layout for the graph in Figure 2. The first column (header) contains the name of packages involved in cycles, then all minimal cycles are represented column by column. A box at the intersection of a row and a column indicates that the package is involved in the cycle.

One package per row. Each row contains dependencies (as boxes) from the package in the header. Number in each box represents the weight of the edge. In Figure 3, first row represents package A, which depends on package B with a weight of 10. Second row represents package B, which depends on E (weight 9) and C (weight 4).

One cycle per column. In the right part of the table, each column represents a cycle. In Figure 3, the first column involves packages A, B and E in a cycle. Each minimal cycle is represented independently.

Shared edges. Cells with the same background color represent the same edge between two packages, shared by multiple cycles. In Figure 3, first row contains two boxes with a yellow background color. It represents the same edge from A to B, involved in the two distinct cycles A-B-E and A-B-C. It is a valuable information for reengineering cycles. Indeed, removing or reversing A-B would solve two cycles.

3.3 CycleTable Cell

Box content is customized to display enriched cells as in eDSM [4]. An enriched cell displays all dependencies at class level from one package to the other. A CycleTable cell is structured in three parts: (i) on top, position in the cycle, (ii) in center, an enriched cell as in DSM, and (iii) on right, a colored frame if the edge is shared by multiple cycles.

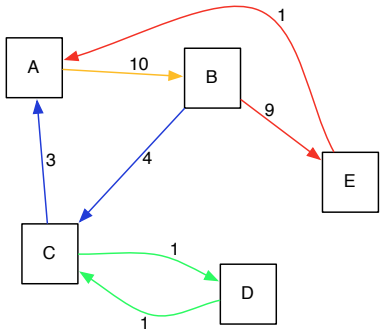

Figure 2. Sample graph with three cycles.

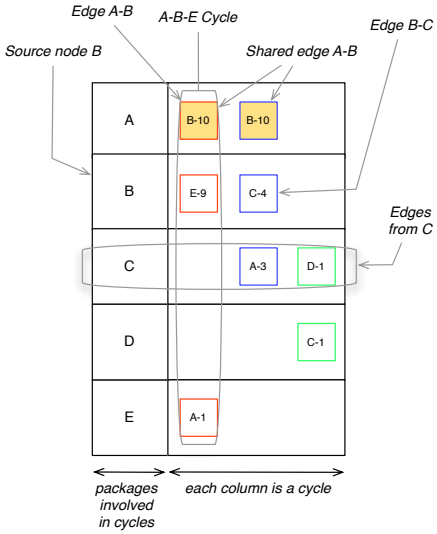

Figure 3. CycleTable for sample graph.

Position in the cycle. The position in the cycle represents a relative order between edges. This number is sometimes necessary to retrieve the exact order of edges in indirect cycles. In Figure 5, numbers allow one to read indirect cycles in third and fifth columns.

Enriched cell. Cell contents gives a detailed overview of dependencies from the package in the header of the row (*source package*) to the next package in the cycle (*target package*). Each cell represents a small context, which enforces comparison with others. The objective is to create *small multiples* [5].

An enriched cell is composed of three parts. The top row gives an overview of the strength and nature of the dependencies. The two large boxes detail dependencies going from the top box to the bottom box *i.e.,* from the *source package* to the *target package*. Each box contains squares that represent involved classes: referencing classes in the source package and referenced classes in the target package. Edges between squares links each source class (in top box) to its target classes (in bottom box). As this structure is the same as in eDSM [4], we give no more explanation in this paper.

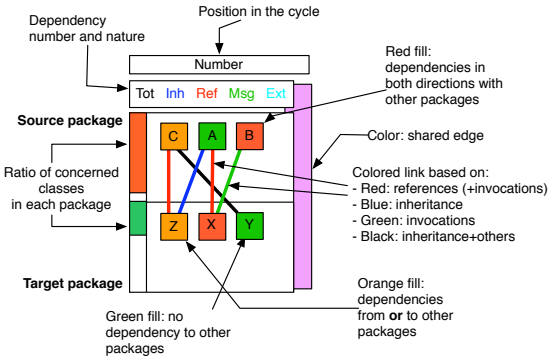

Figure 4. Information in CycleTable cell.

4 Breaking Cycles with CycleTable

Figure 5 shows a CycleTable with the sample four packages of Pharo core presented in Figure 1. Five minimal cycles are immediately made visible. It also appears that three edges are each involved in multiple cycles (with the red, blue, and orange frames).

An important asset of CycleTable is that it does not focus on a single solution to break cyclic dependencies. It rather highlights different options as there are many ways to resolve such cyclic dependencies. Only the reengineer can select what he thinks is the best solution based on a matter of criteria. We now discuss how CycleTable allows one to consider solutions for solving cycles in Figure 5.

First things to consider in CycleTable are the shared edges, the number of cycles they are involved in, and their weight. For example, the blue cell linking `System-FileRegistry` to `Kernel` is in the two indirect cycles. Yet it has a weight of six dependencies and involves six classes as well, which can require some work to remove. Finally, from a semantic point of view, `Kernel` is at the root of many functions in the system so it seems difficult to remove such dependencies from `System-FileRegistry`.

Instead, we turn our focus to the red cells, linking `Kernel` to `System-FilePackage`. It has a very low weight and involves only two classes. This is the minimal dependency we can get between two packages and seems as a prime candidate for removal. Removing this dependency (by moving a class, changing the method, or making a class

Figure 5. Pharo core from Figure 1 in CycleTable.

extension) would break the first two cycles from the left.

The third cycle links directly `System-FilePackage` and `System-Support` and is not intertwined with others. There is a weak link from the second package to the first which seems easy enough to remove.

Finally, the orange edge links the last two cycles. However, it is obvious that it is too complex to be removed. The fourth cycle seems solvable by removing its last edge (from `System-Support` to `System-FileRegistry`). The last cycle, linking directly `Kernel` and `System-Support` is too complex so that a simple removal strategy can be devised with CycleTable. Both packages are at the very core of Pharo system and are highly coupled together.

5 Conclusion

In this paper we presented CycleTable. This visualization shows cycles between packages in a system. Each cycle is presented in a separated column. A colored frame show which edge is shared by several cycles. To complete the visualization, enriched cells (proposed in [4]) have been adapted and integrated to represent each edge.

This visualization is for now a good complement of a DSM visualization. This visualization allows us to solve cycles separately or conjointly.

Future work will focus on computing heuristics for highlighting interesting edges in cycles (either because of their low weight or because they are shared by many cycles), prone to removal by the reengineer.

References

[1] S. Ducasse, T. Gîrba, and A. Kuhn. Distribution map. In *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pages 203–212, Los Alamitos CA, 2006. IEEE Computer Society.

[2] S. Ducasse and M. Lanza. The class blueprint: Visually supporting the understanding of classes. *Transactions on Software Engineering (TSE)*, 31(1):75–90, Jan. 2005.

[3] S. Ducasse, D. Pollet, M. Suen, H. Abdeen, and I. Alloui. Package surface blueprints: Visually supporting the understanding of package relationships. In *ICSM '07: Proceedings of the IEEE International Conference on Software Maintenance*, pages 94–103, 2007.

[4] J. Laval, S. Denier, S. Ducasse, and A. Bergel. Identifying cycle causes with enriched dependency structural matrix. In *WCRE '09: Proceedings of the 2009 16th Working Conference on Reverse Engineering*, 2009.

[5] E. R. Tufte. *Visual Explanations*. Graphics Press, 1997.