Onion Graphs for Focus+Context Views of UML Class Diagrams

Huzefa Kagdi, Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent Ohio 44242
{hkagdi, jmaletic}@cs.kent.edu

Abstract

The paper introduces the use of onion graphs as a focus+context technique for visualizing large UML class models. The focus area, which can be manually or automatically derived, is visualized using the standard UML notations. The remainder of the model is abstracted (context) and presented at varying levels of detail in onion notation. A selective aggregation technique for achieving the abstractions is presented. Finally, the technique is demonstrated by examples on two subsystems of an open source project.

1. Introduction

Developers find UML class models[1] useful for designing systems along with understanding and maintaining existing systems. During forward engineering, the specification of class models is often done via a large number of small UML class diagrams. Eventually each class diagram of interest may need to be comprehended in the context of the entire model (i.e., the union of all the class diagrams). However, in the case of reverse-engineered class models from existing source, the situation is quite a bit more problematic. That is, there are no such small class-diagrams specific to design features or domain concerns. UML reverse-engineering environments (including commercial tools) are restricted to automatic generation of an entire class model from the given source code. At best they support the automatic generation of a class diagram from manually selected classes.

Large systems typically consists of 1000s of classes and many more relationships. Clearly, viewing the entire model in a form of a single class diagram is typically of little use due to cognitive overload [24]. Graph layout algorithms based on aesthetics criteria such as minimizing edge-crossings and maximizing symmetry have proven useful for comprehension of small class diagrams (typically 10-15 classes), however they are inadequate in dealing with a large information space particularly in the UML domain [16-20]. Preservation of aesthetic criteria such as minimizing edge-crossing is a computationally difficult problem for complex graphs [11]. A general solution to effectively visualize a large UML class diagram with layout alone is unlikely. Therefore, it is desirable and meaningful to investigate visual abstractions for representing UML class models. This approach is akin to the work by Munzner on visualizing large graphs such as the world wide web [14] and the work on fisheye views by Furnas [8].

Here, we propose a focus+context technique [4] for UML class models. The focus area is presented in detail and visualized with standard UML notation. The remainder of the model (the context) is abstracted at various levels of detail and presented in onion notation [22]. The formation of an onion graph from UML notations and onion notations provides a single view with a combined focus and context. For example, the onion graph shown in Figure 6 is one possible view of the complete UML class diagram shown in Figure 3. Such combined views may help reduce cognitive overhead by reducing cluttering due to edges [3]. A series of onion graphs supports semantic zooming and incremental exploration [11]. This provides a UML class diagram browser/explorer that can be considered analogues to modern file browsers. Furthermore, our investigation is aimed at designing abstractions that reduce edges. The motivation behind this is rooted in an observation from previous unfavorable results for edges and their layouts in supporting comprehension of large UML class models [16-20].

The remainder of the paper is organized as follows. In Section 2, our semantic focus+context technique including onion notations and a selective aggregation technique is presented. In Section 3, we present examples of using this method on two *Hippodraw* subsystems. Finally, we present our conclusions in Section 4.

[1] In the rest of paper, model or class-model refers to a UML class-model or a UML class-diagram unless explicitly specified.

1-4244-0600-5/07/$20.00 ©2007 IEEE

2. A Semantic Focus+Context Technique

UML graphical notations used to denote classes and various relationships (e.g., generalization and associations) are well established. Therefore, designing new graphical notations (i.e., extensions) for representing visual abstractions[2] of the corresponding elements or relationships is a major challenge (actually a road block). It is imperative to have visual elements that clearly preserve the structure and semantics of the considered UML model elements. Any attempt to deviate from these "standard" notations imposes a risk of total rejection or utter confusion. The same applies to a transformation mechanism employed to reduce the UML class diagrams to abstract views. The resultant view must not violate the structure and semantic of the considered UML model of a system. Therefore, we feel that the following constrains need to be enforced to preserve a developer's mental model, reduce cognitive load, and preserve the information domain,

- **Similarity with UML notations (C1):** The graphical notations for abstracted information must bear a close resemblance to the UML notations.
- **Preservation of UML syntax and semantics (C2)**: The abstracted views must preserve the information about the structure and semantics of a represented UML class model (or class diagram).
- **Level of detail (C3):** The transformed views must provide information varying from the greatest detail of UML class diagram notations to the abstracted overview without losing relevant context.

From a bird's eye view, the proposed technique consists of three main components: focus area, onion notations, and selective-aggregation technique. We can utilize, for example, UML model slicing [12] to automatically extract a context-free model slice that forms the focus area. Alternatively, a user can also manually determine a part of the model to focus. In order to represent the rest of the model (the context), we introduce onion notations for the UML class model. Additionally, we present a selective-aggregation technique for UML class models that represents the classes in the context at a varying level of detail. The overall focus+context view is represented as an onion graph. Next, our onion notations are presented.

2.1. Onion Notations for UML Class Models

Traditional notations visually represent (any) graph as nodes connected by edges. Onion notations represent a graph with nested nodes, i.e., nodes involved in a relationship [22]. A graph represented with a mixture of both traditional and onion notations is said to be an onion graph. In our approach, onion notations are used to present the context information i.e., classes and relations not included in a focus area. The onion notations presented here are based on the *Hicon* approach [22]. The Hicon approach allows for onion notations for multiple types of relationships. The notations consider the structural properties of the UML class model and preserve the semantics defined by those properties. Here, we use the term structural property to refer to a grouping of classes or relationships between classes. We divide the notations in two categories: *pure-onion* and *mix-onion* notations. The pure-onion notations represent abstractions in which a set of structural properties holds for all the members in the group (e.g., all the grouped classes have a generalization relationship). A representative subset of the pure-onion notations is shown in Figure 1.

The symbol in Figure 1(a) is used to represent a class involved in maximum of one relation while the one in Figure 1(b) is used to represent a class involved in two relations (may be of same or different type). The 2-relation class symbol can be similarly generalized to represent n-relation class. The symbols in the Figure 1(c) to Figure 1(g) are used to denote abstractions of classes and the relationships among them. The symbol in Figure 1(c) represents classes that are related via a generalization relation. Similarly, we define symbols for other types of UML class diagram relations. Note that this new notation is directly derived from UML notation.

The mix-onion notations represent abstractions that are partitioned into disjoint groups such that each group holds a set of structural properties. Notice that it is valid for a class to be a member of more than a single partition as long as it satisfies the structural properties of the corresponding partitions. Our goal is to preserve both the structural and semantic information that is present in the original UML class model. In the case of UML class diagram, we have to deal with additional information of layout. That is, ordering and relative positions of elements are important to preserve the mental-model of a developer. The mix-onion notations realize this goal. A subset of mix-onion notations is given in Figure 2.

The symbol in Figure 2(a) is an abstraction of immediate classes (i.e., leaves) on the left side followed by generalizations of classes on the right side. Similarly, we can explain for the notation in Figure 2(b). Figure 2(c) shows a notation representing classes involved in associations followed by generalizations of classes. The Figure 2(d) representing empty partitions are particularly interesting. They satisfy special needs in graphs such as UML class model. For example, in the case of multiple-inheritance only a partial reduction may be possible (we elaborate on this more in Section 3.2). Note that the choice of a notation for a class representation, such as ones given in Figure 1(a) and Figure 1(b), is dependent

[2] Abstraction refers to visual abstractions

on the level of detail needed. Therefore, we may have a partition in a notation represent a single class (more partitions) or a group of classes (less partitions).

Our notation appears to have a deficiency at first glance. The UML modeling language is a combination of text and symbols. The name of the UML elements and other information regarding the structural complexity (e.g., how deep is the hierarchy?) appears to be lost. However, our intention is not to represent the entire UML class model or class diagram in onion notations. The focus obtained by a context-free slice is represented in the UML notation and only the context information is represented in the onion notations. Furthermore, the purpose of having context-information in the same view is to facilitate the further exploration and explanation of the elements in the model.

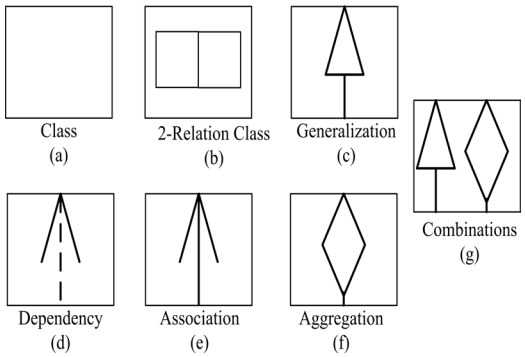

Figure 1. A Subset of Pure-Onion Notations for the UML Class Model

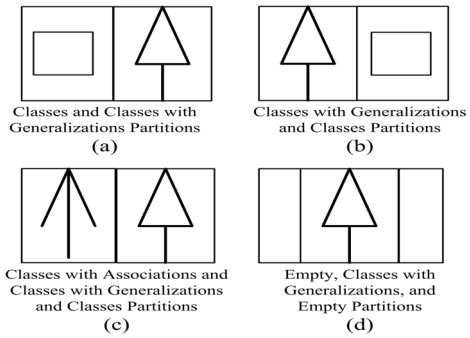

Figure 2. A Subset of Mix-Onion Notations for the UML Class Model

We can now introduce secondary notations [9] of node-metrics as used in [6]. The width of a symbol is used to represent the number of classes abstracted and the height of the symbol is used to represent the number of levels abstracted along a relation. This technique provides more context information than is available in a typical class diagram like representation. In this paper we map only the structural information to the node-metrics and not the information external to the domain of UML class model (e.g., OO metric information indicating the quality of a design, design patterns, etc). However, one could easily map such semantic information to the secondary notations.

In summary, the onion notations satisfy the conditions C1 (preservation of UML syntax and semantics) and C2 (close similarity with UML symbols). However, it is further required that the classes and their relationships in a given UML class model are mapped to the appropriate onion notations. To address this issue, we present a selective aggregation technique.

2.2. A Selective-Aggregation Technique

Our focus+context technique takes as input the UML class model (or diagram) and produces views in the form of onion graph as determined by the selective-aggregation technique. We apply reduction only to the context part of the view and the focus is preserved as it is (unless the focus is explicitly changed by a user). The whole technique is defined formally as follows,

Let $M = (E, R, \Gamma)$ be a UML class model. The focus F is the computed context-free model slice or user selected subset. Thus, we have,

$F = S_{cf}(M, C_{cf}) = M' = (E', R', \Gamma')$.

Therefore, the context in the model CM_F with respect to a given focus F is the remainder of the classes and relations in the model that are not in focus F. Thus, we have, $CM_F = M/M' = (E/E', R/R', \Gamma/\Gamma')$.

Once we have the context CM_F, the selective-aggregation technique is basically a view-transformation technique where the UML class diagram is transformed to an onion graph representation. Therefore, we need a mechanism, i.e., criteria and operations to select classes that can be reduced to an abstraction and map the obtained reduction/abstraction to appropriate onion notations. Moreover, the selective-aggregation technique must satisfy the additional constrain of never producing a view that violates the underlying semantics given to classes by the relations and the assumptions of the developer (e.g., layout in case of class diagram).

The operations of the selective-aggregation technique are based on the traversal of the graph. A UML class model is a directed graph (more precisely a multi-graph). However, it is interesting to examine the directionality of the relations (e.g., bidirectional in case of binary-associations and derived-to-base directionality in case of generalizations). The most important point here is the directionality implied by a particular UML relation is semantic (e.g., order in which the classes collaborate or provide reuse) and not navigational. Therefore, we are at liberty to base our navigation technique on the semantics or the underlying undirected graph.

In our approach, the navigation is based on the breadth-first traversal of the underlying graph starting at

the leaves. By doing so, we induce the *parent* and *sibling* relations between classes based on their traversal order, and classify each class as an *internal* or a *leaf* node. Furthermore, on account of the multi-graph nature of UML class model, we distinguish between *siblings* as *pure-siblings* and *mix-siblings*. The pure-siblings are the classes that have all the identical parents while the mix-siblings are the classes that have at least one but not all parents identical. We now define two orthogonal reduction-operations.

The sibling-order compaction operates on pure-siblings and reduces them to pure-onion and/or mix-onion notations. The number of nodes in the graph is reduced and at the same time the abstraction of nodes increases as a result of a successful operation. In other words, the goal of this operation is to decrease the overall width of the context part of the onion graph.

The level-order compaction operates on both the pure-siblings and mix-siblings at a given level and reduces them to pure-onion and/or mix-onion notations. The number of levels in the graph reduces and the abstraction level increases as a result of a successful operation. In other words, the goal of this operation is to decrease the overall height of the context part of the onion-graph. However, not all the leaf-nodes are reduced. If a leaf-node has either pure-siblings or mix-siblings that are internal nodes, it is ignored by the level-order compaction until the internal-nodes become leaf-nodes by successive operations.

The reduction operators reduce siblings and abstract them in the onion notations. The onion notation used to represent a reduction or aggregation is chosen such that it satisfies the underlying semantics of the classes or the relations. The mapping function needs only the information about the type of node (e.g., pure-onion and mix-onion) and the relations (e.g., generalization and association) involved in the reduction. Therefore, the semantics of the relations as implied by the directionality is always preserved in the onion notations and are not sacrificed for the ease of navigation and reduction.

The breadth-first traversal augments each node with information about its parents and its *leaf* or *internal* status. The reduction technique proceeds in a bottom-up fashion staring with leaves and moving towards the root(s). The sibling-order compaction consists of many atomic operations. A single such operation considers as operands only the leaf-nodes. Thus, the sibling-order compaction is also recursive and consists of multiple reduction operations. Similar explanation follows for the level-order compaction but with one additional function. The level-order compaction is also responsible for updating the status of the onion-notation node used to represent the abstraction. As the level-order compaction reduces the levels in the graph, some of the nodes that are internal may turn into leaf nodes. The arities of a single sibling-order and level-order operators are specified by the control variables *scdegree* and *lcdegree* respectively.

A number of intermediate focus+context views can be generated after each atomic operator or the completion of the entire sibling-order or level-order compaction. This decision depends on the trade-off between the level of detail in each view and the number of views.

3. Illustrations

We conducted a primary investigation on an open source data analysis tool, *Hippodraw* (*version 1.12.9*). *Hippdraw* is an application written in C++ and consists of about 238 classes. The documentation for *Hippodraw* does not contain UML class diagrams. We reverse-engineered the UML class model from the C++ code-base with the help of a visual modeling tool, *Visual Paradigm (version 4.1)*. This tool does reverse engineer generalizations well but other relationships are not well supported. Here, we demonstrate our semantic focus+context technique on two subsystems: *transforms* and *projectors*. The directory *transforms* exhibits instances of single-inheritance while the directory *projectors* exhibits instances of multiple-inheritance.

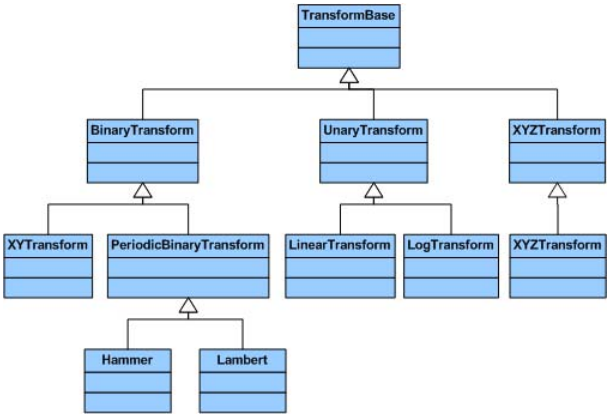

Figure 3. A Partial UML Class Diagram of *Transforms* – Single Inheritance

Reverse-engineering with *Visual Paradigm* results in a UML class model. It also supports automatic UML class diagram generation given manually selected classes. Furthermore, it supports a few automatic layout algorithms. Different UML class diagrams consisting of all the classes and generalizations were constructed for both the *transforms* and *projectors* cases. In both the cases, a hierarchical layout algorithm was applied. Therefore, in the examples we impose an additional constrain of preserving the original layout i.e., the relative positions of nodes and edges. The onion notations and the figures here were generated using with *Microsoft Visio* as this produced cleaner diagrams.

3.1. Onion Graphs of *Transforms* – Single Inheritance

A partial UML class diagram for the *transforms* modules is shown in Figure 3. Let us consider that a developer is considering a task of extending the functionality of the *XYZTransfrom* class. Therefore, the first simple, but helpful, activity is the comprehension of the entire inheritance hierarchy of the *XYZTransfrom* class. Only the base classes and generalizations of the *XYZTransfrom* class are required to be in detail (focus) and all the other elements are considered as context.

Using a straightforward formulation we can automatically compute a context-free slice of the inheritance hierarchy. Alternatively, this selection could be done manually. This becomes the focus F and the rest of the elements in the class diagram form the context CM_F with respect to the focus F. Now, we present a series of views obtained by the application of the selective-aggregation technique.

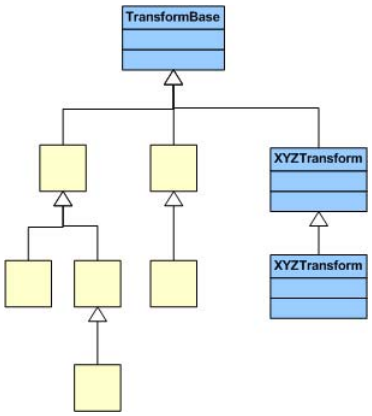

Figure 4. A View after the First Sibling-Order Compaction for *Transforms*.

The default focus+context view of the *transforms* module shows no selective-aggregation. The elements in focus are displayed in the UML notations and the elements in the context are displayed in the onion notations. We present only those views that are obtained after the entire sibling-order or level-order reduction due to space-constraint. The first view obtained after sibling-order reduction is shown in Figure 4. All the leaf-nodes are reduced to pure-onion notations. The width of the onion notations indicates the number of siblings that are abstracted. In this case two siblings were merged in each case and are represented by pure-onion notations. The scale for node-metrics representation is linear and varies across views. This is due to continuous reduction of the visualization space needed for the context information. However, the scale must suffice to make a perceivable distinction in any given view.

The next step in the reduction process produces a level-order compaction. The pure-onion notations are used to represent this compaction. This reduction occurred along the generalization dimension. The semantics of the reduction as well as the UML semantics of generalizations are preserved by the chosen notations. Notice that some *internal* nodes become *leaves* on account of this operation. The height of the nodes reflects the number of levels reduced. Also, the first leaf node on the level two is not abstracted. It waits for the left internal-node to become a leaf. The view obtained after this compaction is shown in Figure 5.

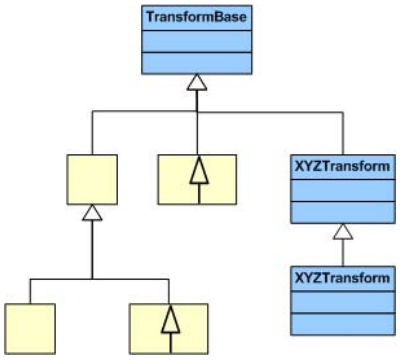

Figure 5. A View after the First Level-Order Compaction for *Transforms*.

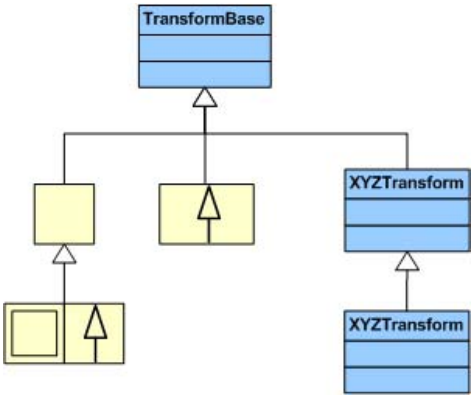

Figure 6. A View after the Second Sibling-Order Compaction for *Transforms*.

The next view is interesting as it introduces mix-onion notations. The *leaves* in Figure 5 are of different types i.e., representing different abstracted information. For example, the lower-left onion node is an abstraction of generalizations among the classes while the one to its right is not. In this case, they cannot be combined and represented by a pure-onion notation as semantics are to be preserved. Therefore, a mix-onion node represents them (see Figure 6).

In the second level-order compaction, the reduction is applicable only to the lower-left *leaf* node. The other *leaf*

node is already at the root *parent* position. The generalization relation, under reduction, holds for both the partitions of the lower-left *leaf* node. Therefore, this level is reduced to a pure-onion notation.

The additional visualization space gained can be better utilized for elements under focus. For example, a set of relevant public operations of the classes in focus can be shown. Notice that the final sibling-order compaction is possible as both the leaf nodes are in pure-onion notations and thus the context is now abstracted in a single onion node (this view not shown with a figure).

3.2. Onion Graphs of *Projectors* – Multiple Inheritance

A partial UML class diagram for the *projectors* module is shown in Figure 7. Here, we have a case of multiple-inheritance. We give numeric labels to all the classes in this diagram[3] to facilitate discussion. Let us assume that the classes labeled 12, 13, and 18 are in the focus area. Therefore, the rest of the context forms an input to the selective-aggregation technique.

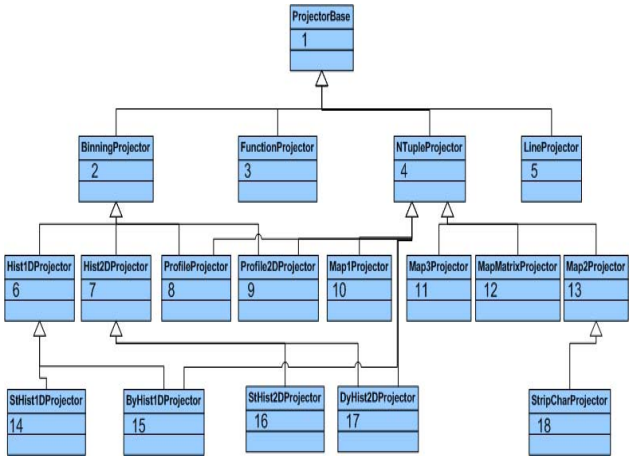

Figure 7. A Partial UML Class-diagram for *Projectors*– Multiple-Inheritance

The first step is the sibling-order compaction. The classes derived from multiple parents do not automatically become mix-siblings. In Figure 7 we have 4 classes (labeled 8, 9, 15, and 17) involved in multiple-inheritance. However, 8 and 9 are pure-siblings as both of their parents are identical. Also, some of the pure-siblings may be ignored for reduction on the basis of preserving the layout of the original diagram. Such is the case, for example, with pure-siblings 3 and 5. Therefore, only the siblings 8 and 9 are abstracted into a pure-onion node. The view obtained after this step is not shown.

[3] These labels are not part of the onion notations.

The next level-order compaction operates on the onion graph obtained after the sibling-order compaction. The view obtained after the first sibling-order compaction is similar to the previous one except that nodes 8 and 9 are merged and represented by a single pure-onion node (labeled 8 for simplicity). At this stage, the leaf-nodes are 3, 5, 8, 10, 11, 14, 15, 16, and 17. Leaf-nodes 3 and 5 have siblings from the parent 1 that are internal nodes. Leaf node 8 has siblings from the parent 2 that are internal nodes. Therefore, it does not participate in the level-order compaction. Nodes 10, 11, 14, 15, 16, and 17 are leaf-nodes with no internal siblings. Therefore, they take part in the level-order compaction.

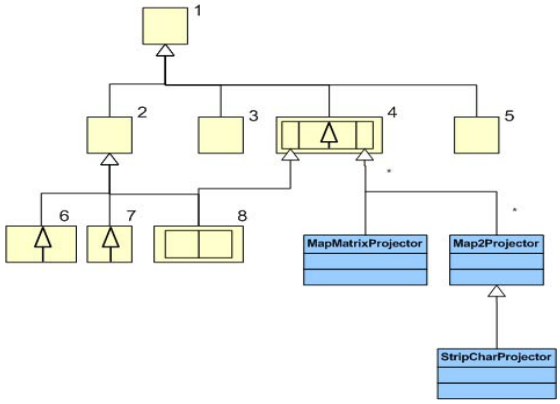

Figure 8. A View Obtained after the First Level-Order Compaction for *Projectors*.

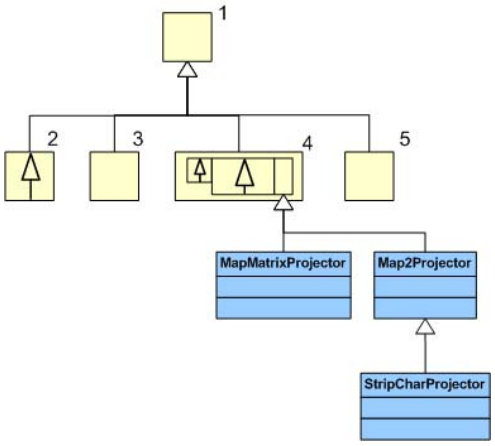

Figure 9. A View Obtained after the Second Level-Order Compaction for *Projectors*.

The parent node 6 and its children 14 and 15 in Figure 7 are merged into a pure-onion node. The pure-onion node for this operation is labeled 6 in Figure 8 and marked as a leaf-node. Notice that nodes 14 and 15 are mix-siblings, however the other parent 4 of the node 15 also participates in this reduction. Therefore, nodes 14 and 15 are abstracted into a pure-onion node 6. Similar

actions occur for the parent node 7, and a pure-onion node 7 in Figure 8 represents the abstraction. The reduction operation that deserves the most explanation is one involving parent node 4 in Figure 7. Notice that the child node 8 cannot participate in this reduction, however, all the other children of node 4 can. Therefore, the abstracted node does not become a leaf after the reduction (see Figure 8). Furthermore, a pure-onion notation cannot represent the abstraction due to a need to preserve the relation semantics. Therefore, a mix-onion notation labeled 4 in Figure 8 represents this reduction (large middle node). Notice, that this node preserves the semantics of the relationship with other nodes and the layout.

The second sibling-order compaction operates on the onion graph presented in Figure 8. The leaf-node siblings 6 and 7 are merged and represented by a pure-onion node. No other sibling-order compaction is possible. The second level-order compaction operates on the onion graph obtained after the second sibling-order compaction. The onion graph obtained after this reduction is shown in Figure 9. Similarly, other reductions follow and are not discussed here.

4. Related Work

Broadly the research efforts on visualizing UML class diagrams can be put into three major categories: layout algorithms and abstracted views. Each is discussed separately.

4.1. Layout

A widely investigated approach is to address UML class model visualization as a pure-graph layout problem. An excellent source describing the graph-drawing algorithms is [2]. In [11] various graph-layout algorithms are examined with regards to scalability, aesthetic criteria, and cognitive factors. Researchers tried to identify aesthetic criteria that are most important for human perception with the goal of reducing the cognitive cost in performing a certain task [17, 25]. Minimizing edge-crossing and bends, and maximizing symmetry are identified to be the important aesthetic criteria for general graphs [18]. In another study path-length, path-continuity, and minimizing edge-crossings were recognized as essential criteria [25].

For UML class diagrams specific studies, subjects showed preference for fewer bends and crosses, shorter edge lengths, and an orthogonal structure [16]. However, it remains inconclusive as to which aesthetic criteria (with an exception of minimizing bends) are important for UML class diagram comprehension and future studies that considers semantic grouping of nodes and using secondary notations (e.g., color) [9] are suggested [19].

A UML class diagram layout algorithm, producing a balance of various aesthetic criteria and emphasizing the visibility of a particular structure via secondary notations, is presented in [10]. The quality of a design in the layout algorithm is presented in [6]. Also, there exist algorithms that incorporate other semantic information such as design patterns and architectural importance [1, 7].

4.2. Abstractions

Another abstraction-based approach, *focus+context* technique is applied to various graph-like structures [8, 11, 13, 21]. The most closely related work in context of UML class model is [15]. Our approach differs from the above work at the grass-root level. Our main motivation is reduction or elimination of selective edges (in context) while providing the semantic focus in UML notations. The work in [15] strives to achieve visual-space efficiency (possibly at the loss of pruned information below the lowest DOI value). Furthermore, our approach is not limited to a single class in the initial focus area but can contain multiple classes and relationships that are developer specified. The original layout is not preserved in the above work; rather it is continuously changing at each point of aggregation.

Another approach for abstracting UML class diagram is presented in [5]. The goal of this approach is to derive direct relationships between a given list of classes, and filter remaining elements. The intermediate classes and relations between a pair of classes are abstracted and represented by a single relation. In contrast, our goal is to provide a portion of class diagram in the context of the entire class diagram.

The nested graph view is an alternative for visually representing relations between nodes in a graph. Such a representation consists of nodes within a node. Note that a node may embody an abstracted graph. The SHrIMP tool [23] is an excellent representative utilizing nested graph in the domain of software visualization (though not applied in the domain of UML class models).

5. Conclusions and Future Work

A novel approach for focus+context views of UML class diagrams was presented. The combined view provides the focus and details in UML notation while the context information is presented in the form of onion notations. Furthermore, the views preserve the structure and semantics of the model from the UML as well as the developer's perspectives. We demonstrated the technique by a set of examples. Our visualization technique does achieve edge reduction, however formally assessing context information will require a user study. Anecdotally, during various informal discussions, a number of experienced UML users gave opinions that

support the notations usefulness. Additionally, a claim can be made that the previous research on fisheye and other context+focus views directly supports our hypothesis. However, the degree to which one can abstract into onion notation in the domain of UML is of great interest with respect to automation.

In the future, we plan to conduct user studies to validate our conjecture regarding context information. Also, we are developing tools (plug-ins to open source UML tools) to support automatic transformation of the UML class models in onion graphs.

6. References

[1] Andriyevska, O., Dragan, N., Simoes, B., and Maletic, J. I., "Evaluating UML Class Diagram Layout based on Architectural Importance", in Proceedings of 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'05), Budapest, Hungary, September, 25 2005, pp. 317-326.

[2] Battista, G., Eades, P., Tamassia, R., and Tollis, I., Graph Drawing Algorithms for the Visualization of Graphs, Prentive Hall, 1999.

[3] Bertin, J., Graphics and Graphic Info Processing, De Gruyter, 1981.

[4] Card, S. K., Mackinlay, J., and Shneiderman, B., Readings in Information Visualization Using Vision to Think, San Francisco, CA, Morgan Kaufmann, 1999.

[5] Egyed, A., "Automated Abstractions of Class Diagrams", ACM Transactions on Software Engineering and Methodology (TOSEM), 11, 4, October 2002, pp. 449-491.

[6] Eichelberger, H., "Nice class diagrams admit good design? " in Proceedings of ACM symposium on Software visualization, San Diego, California, 2003, pp. 159-168.

[7] Eichelberger, H. and Gudenberg, J., W., "UML Class Diagrams – State of the Art in Layout Techniques", in Proceedings of Visualizing Software for Understanding and Analysis (VISSOFT'03), Amsterdam, Sept. 22 2003, pp. 30-34.

[8] Furnas, G. W., "Generalized Fisheye Views", in Proceedings of Human Factors in Computing Systems, Boston, MA, 1986, pp. 16-23.

[9] Green, T. and Blackwell, A., "Thinking about visual programs", Thinking with Diagrams, IEE Digest No: 96/010, 1996.

[10] Gutwenger, C., Junger, M., Klein, K., Kupke, J., Leipert, S., and Mutzel, P., "A new approach for visualizing UML class diagrams ", in Proceedings of ACM symposium on Software visualization, San Diego, California, 2003, pp. 179-188.

[11] Herman, I., Melancon, G., and Marshall, S. M., "Graph Visualization and Navigation in Information Visualization: A Survey", IEEE Transaction on Visualization and Computer Graphics, 6, 1, January/March 2000, pp. 24-43.

[12] Kagdi, H., Maletic, J. I., and Sutton, A., "Context-Free Slicing of UML Class Models ", in Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM'05), Budapest, September, 25-30 2005, pp. 635-638.

[13] Lamping, J., Rao, R., and Pirolli, P., "A focus+context technique based on hyperbolic geometry for visualizing large hierarchies", in Proceedings of ACM Conference on Human Factors in Computing Systems, 1995, pp. 401-408.

[14] Munzner, T., "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space", in Proceedings VRML '95, San Diego, 1995.

[15] Musial, B. and Jacobs, T., "Application of Focus + Context to UML", in Proceedings Australian symposium on Information visualisation, Adelaide, Australia, January 01 2003, pp. 75-80.

[16] Purchase, H., "Effective information visualisation: a study of graph drawing aesthetics and algorithms", Interacting with Computers, 13, 2, December 2000, pp. 147-162.

[17] Purchase, H., McGill, M., Colpoys, L., and Carrington, D., "Graph Drawing Aesthetics and the Comprehension of UML Class Diagrams: An Empirical Study", in Proceedings of Australian Symposium on Information Visualisation, Sydney, Australia, 2001, pp. 129 - 137.

[18] Purchase, H. C., "Which Aesthetic has the Greatest Effect on Human Understanding? " in Proceedings of 5th International Symposium on Graph Drawing, Springer-Verlag, 1997, pp. 248-261.

[19] Purchase, H. C., Allder, J.-A., and Carrington, D. A., "User Preference of Graph Layout Aesthetics: A UML Study ", in Proceedings of 8th International Symposium on Graph Drawing, Springer-Verlag, 2001, pp. 5-18.

[20] Purchase, H. C., Cohen, R. F., and James, M., "Validating Graph Drawing Aesthetics", in Proceedings of Graph Drawing '95, Passau, Germany, September 20-22 1995, pp. 435-446.

[21] Rao, R. and Card, S. K., "The Table Lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information", in Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'94), 1994, pp. 318-322, 481-482.

[22] Sindre, G., Gulla, B., and Jokstad, H., "Onion Graphs: Aesthetics and Layout", in Proceedings of IEEE Symposium on Visual Languages, Norway, August 01 1993, pp. 287-291.

[23] Storey, M.-A. D., Best, C., and Michaud, J., "SHriMP Views: An Interactive Environment for Exploring Java Programs", in Proceedings of International Workshop on Program Comprehension (IWPC'01), Toronto, Ontario, Canada, May 12-13 2001, pp. 111-112.

[24] Ware, C., Information Visualization. Perception for Design, Morgan Kaufmann Publishers, 2000.

[25] Ware, C., Purchase, H., Colpoys, L., and McGill, M., "Cognitive Measurements of Graph Aesthetics", Information Visualization, 1, 2, June 2002, pp. 103-10.