Applying Multi-dimensional Separation of Concerns to Software Visualization

Piotr Kaminski

Department of Computer Science, University of Victoria
PO Box 3055, STN CSC, Victoria, BC, Canada V8W 3P6
pkaminsk@csc.uvic.ca

Abstract

Software visualization tools have so far not taken advantage of the recent advances in multi-dimensional separation of concerns. To integrate the two, it is necessary to define a representation for concerns and concern spaces that fits these tools and can be implemented as a graph. If successful, this will enrich the structure of system models, allowing new kinds of visualizations and ultimately benefiting both communities.

1 Introduction

In recent years, research related to multi-dimensional separation of concerns has grown by leaps and bounds. At the same time, the software visualization community seems to have remained largely unaware of this progress and lags behind integrating the new concepts into its program understanding tools. Although such an integration is not a trivial endeavour, I contend that it would have significant benefits for both communities.

In this position paper, I first provide a quick overview of both the multi-dimensional separation of concerns (2.1) and software visualization (2.2) fields, concentrating on generally accepted ideas and approaches. I then expose some of the issues related to modeling concerns (3.1) and concern spaces (3.2) within program understanding tools, and propose some solutions. After touching on a possible implementation model (3.3) and some of the exciting research opportunities presented by visualization of software units organized into multi-dimensional structures (3.4), I conclude with my opinion on the expected benefits of this work (4).

2 Background

This section provides short overviews of both fields. If you are familiar with either one, you can safely skip the related subsection.

2.1 Multi-Dimensional Separation of Concerns

Extolling the advantages of separation of concerns has become somewhat of a bromide in the field of software engineering, though this doesn't make it any less true. However, the rewards promised by proponents of this approach have for the most part failed to materialize. Program comprehensibility, reusability and evolvability remain poor.

Some contend [8] that this is because current programming languages are limited to separating concerns in only one or two [3][4] dimensions at a time—the "tyranny of the dominant decomposition." For example, object-oriented programming only allows for concern decomposition and modularization by separating code between appropriately encapsulated classes. Not all features can be conveniently encapsulated in a class, though, resulting in scattering of concerns among classes (higher coupling), and tan-

gling of concerns within a single class (lowered cohesion).

The proposed solution is to provide mechanisms for separating concerns within multiple concern spaces *simultaneously*. Each concern space would group concerns of the same kind, for example classes, function points, aspects, roles, versions, etc. Work products (including especially the code, but also other artifacts) would then be decomposed into atomic units that would be categorized within each concern space. If necessary, means to recompose the original artifacts from the separated units should also be provided.

There is, as yet, no widespread agreement on the structure of a concern space and the ways to recompose the various artifacts, especially when the concern spaces interact in complex ways.

2.2 Software Visualization

Software visualization is concerned with producing visual representations of both static artifacts and their dynamic behaviour. While some visualization tools have specific objectives, such as optimizing running time or locating memory leaks, most simply aim to improve the user's overall understanding of a system.

A popular approach in this last category is exemplified by the Rigi tool [9]. It transforms all artifacts into a graph, where each vertex represents a unit and each edge some relationship between two units. The graph is represented on-screen by a stylized node-and-arc diagram, and various layout algorithms can be applied to clarify the structure of the system being examined.

The granularity of the information represented in the graph varies with the parser employed to extract it. Many tools consider functions and variables to be atomic units, and function calls and variable access to be atomic relationships. Finer granularity is possible if the tool's backing store can scale appropriately.

Of course, being able to only observe the atomic units and relationships will result in a "can't see the forest for the trees" problem for all but the smallest systems. To counteract this, all tools have the ability to regroup atomic units into compound ones, and to express higher-level relationships between them. Some of this recomposition is done automatically during parsing (according to the original artifact's dominating decomposition), and further modularization can be performed manually by the user within the environment. However, in current tools, all aggregation must take place along one axis: the standard "subsystem" dimension defined in the classical separation of concerns method.

3 Modeling Concerns

Being able to aggregate units independently along multiple axes should help the user better organize the information available about a system. Naturally, there are challenges to be faced in the construction of such a tool.

3.1 Concerns

How should the tool represent concerns? In current tools, synthetic (user-created) modules are simply new compound units within the integrated system model. This seems like a good lead to follow, since it allows concerns to be treated uniformly with other units, for visualization or for further structuring.

Both existing parsed units and new synthetic units should be valid concern candidates. This is necessary because some concerns might already by present in the system model (e.g. classes), and only need to be associated with the right "content" units. Forcing the user to create synthetic concerns in this case would result in one unit having two representations within the system.

The tool should also support numeric concerns, both discrete (integers) and continuous (real numbers, time), whose usefulness has been demonstrated in [6].

3.2 Concern Spaces

How should the tool represent concern spaces? A concern space is akin to a mathematical dimension. It consists of a domain of units (a subset of all the units in the system) and a range of concerns and the mapping of the units to those concerns.

In [7], a concern is defined as a predicate over units that indicates whether or not a unit pertains to that concern. According to this definition, a unit in the system model becomes a concern if and only if it is included in the range of a concern space. A single unit represents a separate concern for each concern space it's a member of. (This makes sense if you think of moments in time as concerns. Depending on the concern space, a given moment in time may actually express a different concern.)

What kind of structure does a concern space have? We might want to limit the mapping to be injective (each unit maps to at most one concern), or perhaps even surjective (each concern is "covered" by at least one unit). We might also want an implicit "null" concern in each space, to which every unit in the system not present in the space's domain is automatically mapped. There are many possible tradeoffs between a strict structure that could facilitate composition at a later time, and flexibility, which increases the user's control over the model's organization.

One final note of interest is that mathematical dimensions are rarely unstructured sets. As a matter of fact, most are totally ordered sets. It might be useful to structure the concern range of a concern space in a similar manner. For example, a time-related concern space would benefit from having its concerns ordered chronologically [6]. A concern space of directories or classes, on the other hand, has an inherently hierarchical structure, as proposed in [5].

Allowing the user to impose a partial order on the collection of concerns can support all of these various structures. This scheme accounts for unordered and totally ordered sets, trees, and lattices, such as the one formed by Java interfaces.

3.3 Hypergraph Implementation

The concern space model proposed above might appear difficult to represent within a directed graph. This is indeed true, but a slightly more general graph model, a partially ordered hypergraph [1], can easily represent the required constructs.

In a hypergraph, an edge can connect any number of vertices. This allows for a natural model of collections, such as the range of concerns in a concern space. If we further consider each edge as a vertex, then relations such as the unit→concern mapping can be represented as a collection edge (set) of cardinality 2 ordered edges (mapping pairs).

This should be sufficient to convince you that the concern space structure I presented above is practical. For more information on a tool that implements these ideas, visit http://www.csr.uvic.ca/~pkaminsk/braque/.

3.4 Visualization of Concerns

Multi-dimensional separation of concerns greatly enriches the structure of the system under investigation. Effectively visualizing this structure is a challenge just beginning to be met.

One can imagine many static views that demonstrate the cross-cutting of various concerns through the system, such as in [2]. The three-dimensional views suggested in [5] could also prove interesting, and there are undoubtedly many other possibilities for ef-

fectively communicating the complex organization of a system to the user.

Recent tools have also started using animation [10] to convey the system's structure to the user more dynamically. Animating a transition between the visualization of two concern spaces could be an effective means of showing the differing organizations of units within the two spaces to the user.

4 Conclusions

In this paper, I have suggested a way to apply multi-dimensional separation of concerns to software visualization. Such a marriage would have clear advantages for software visualization: a richer organization structure imposed on the system model would increase a user's understanding of the system. This could allow maintenance programmers to better target and scope their work, and help evolve a system over time. Basically, it should bring all the benefits of true separation of concerns to existing software systems.

The multi-dimensional separation of concerns community also stands to gain from this arrangement. Software visualizations tools usually have powerful structural query facilities in addition to the graphical renditions they offer. Both could be used to great advantage when trying to reengineer existing software into a multi-dimensional shape. Mapping code units to concerns by hand is a time-consuming and difficult endeavour, as attested to by [7]. This activity could be assisted by and even partially automated with appropriate tool support. This next-generation tool would also be able to model software that has an explicit multi-dimensional structure, such as code written in AspectJ or HyperJ.

Although we'll need to keep a close eye on the additional complexity that multi-dimensional separation of concerns will introduce into software visualization tools, I believe that the benefits outweigh any disadvantages, and there is a lot exciting research waiting to be done in this area.

References

[1] C. Berge, "*Graphs and Hypergraphs*." North-Holland, Amsterdam, 1976.

[2] W. G. Griswold, Y. Kato, J. J. Yuan, "*Aspect Browser: Tool Support for Managing Dispersed Aspects*." Workshop on Multi-dimensional Separation of Concerns in Object-Oriented Systems, OOPSLA '99, 1999

[3] W. Harrison, H. Ossher, "*Subject-Oriented Programming (a critique of pure objects)*." In Proc. OOPSLA '93.

[4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, J. Irwin, "*Aspect-Oriented Programming*." In Proc. ECOOP '97.

[5] D. Kimelman, "*Multidimensional Tree-Structured Spaces for Separation of Concerns in Software Development Environments*." Workshop on Multi-dimensional Separation of Concerns in Object-Oriented Systems, OOPSLA '99, 1999

[6] V. Kruskal, "*A Blast from the Past: Using P-EDIT for Multidimensional Editing.*" Workshop on Multi-Dimensional Separation of Concerns in Software Engineering, ICSE 2000, 2000.

[7] A. Lai, G. C. Murphy, "*The Structure of Features in Java Code: An Exploratory Investigation*." Workshop on Multi-dimensional Separation of Concerns in Object-Oriented Systems, OOPSLA '99, 1999

[8] H. Ossher, P. Tarr. "*Multi-dimensional separation of concerns in hyperspace*." Technical Report RC 21452(96717)16APR99, IBM T.J. Watson Research Center, 1999.

[9] Rigi home page, http://rigi.csc.uvic.ca/

[10] SHriMP home page, http://www.csr.uvic.ca/~mstorey/research/shrimp/