Grouping

Oliver Ciupke

July 27, 1997

Abstract

Reengineering requires modelling and understanding a system on many levels of abstraction. *Grouping* is a concept for building abstract views on object-oriented systems. Grouping is defined formally based on graph theory. Visualizing a system containing groups and manipulating a corresponding model by tools can be defined straight forward.

1 The problem

During reengineering of large systems, it is necessary to be able to perform analysis and manipulation also on higher levels of abstraction. For larger systems, the standard views on a system (e.g. the class structure or a message trace diagram) are not abstract enough and often get crowded with too many details which are not needed to understand the system as a whole.

Some problems of software design themselves are located on higher levels than at which they are normally taken into consideration. E.g. the design on the class structure may look well designed, but there are too many dependencies between different packages.

2 Concept

When analyzing a system, we must have an exact way to come from elementary views to more abstract ones. They must be formally sound, so it is exactly known, what the meaning of an abstraction is and that it is possible to build tools dealing with abstractions.

What we need is a way to exactly describe how to come from a detailed description of a system to a more abstract one. We call this method *grouping*. Grouping means putting entities describing a common abstract concept into one more abstract entity, called a *group*.

If a system is represented as a graph, the entities and relationships between them are represented by nodes and edges. In this case, grouping means replacing nodes or edges by a node or edge representing the group.

Formally, grouping and related terms are defined as follows. Terms taken from graph theory are defined e.g. in [SS89], [Har69].

- A *group* is a subset of all entities under consideration (including groups themselves).
- A *grouping* is a surjective graph homomorphism, mapping the elements of a group to a single node (or a single edge).
- If the difference matters, one can distinguish *node groups* and *edge groups* or *node groupings* and *edge groupings*.

- The trivial grouping is the identity.
- The cardinality of a group may introduce a weight to the representing node or edge.

3 Examples

In principle, every set of entities under consideration can be grouped together. The same way, every entity can in principle be split up into further detailed parts of a system. But there are several groupings which are more often needed and most applicable to reach frequently useful abstractions. Examples for those are grouping

- Classes to packages
- Packages to packages of higher granularity, (in general packages may be grouped hierarchical, but in practice packages of different levels have often different names given, e.g. subsystems, program blocks, service blocks etc.)
- Objects to classes (or types)
- Dynamic method calls to pair of calling and called object (interval may e.g. be given by call and return of a surrounding call)
- Dynamic method calls within a certain interval of time
- Classes to files
- Files to directories
- Objects to processes

and many more.

4 Types of groups

Every entity has a type (e.g. class, object, method, file). These are types on the meta-level and not to be confused with the types declared in the program or in the specification of the programming language.

The type of a group is determined by the possible types of entities it can hold (which may be more than one).

A group may be represented by an entity of different type. The group of objects of a certain class is not this class, but it may be represented by it.

5 Operations on groups

There are special operations available on groups:

collapse: replace the set of entities contained in the same group by a representation of this group (e.g. in a certain view)

expand: replace the group by its elements (e.g. in a certain view)

When supported by a tool, those operations are often performed interactively by the user.

To perform an operation on each single element of a group or to filter a subset of elements from a group fulfilling certain properties (i.e. a predicate) there are operations well known from functional programming [BW88]:

map: takes an operation o and a set (or group) s and performs o on each element of s[1]

filter: takes a predicate p (a function returning a boolean) and a set (or group) s and returns a set s' containing those elements x of s, for which $p(x)$ is true

6 Visualization and tool support

Groups that are collapsed are shown as a node replacing the elements contained in the group. Relations of those elements are propagated to the group as required by grouping as a graph homomorphism.

Groups can also be visualized when they are not collapsed to a single node but expanded and all their elements are still visualized. This must be possible especially when groups are defined interactively. Possibilities for such visualizations are

- Drawing a shape (e.g. a box) around the elements to be grouped. This requires all elements of a group to be localized near each other.[2]
- Drawing all the entities contained in a group in the same way, e.g. in the same color or with the same shape.
- The group is shown as an additional node which is connected to its elements by its defining containment relation.

7 Future work

- Define problem patterns of reengineering related to grouping
- Specify requirements for tool support for grouping
- Enhance grouping capabilities of existing prototype tools

References

[BW88] Richard Bird and Philip Wadler. *Introduction to Functional Programming.* International Series in Computer Science. Prentice Hall, 1988.

[Har69] Frank Harary. *Graph Theory.* Series in Mathematics. Addison, 1969.

[SS89] Gunther Schmidt and Thomas Ströhlein. *Relationen und Graphen.* Mathematik für Informatiker. Springer-Verlag, 1989.

[1] some times "*map*" is referred as "*foreach*", but this term is often used with a different meaning in the area of concurrent programming

[2] This is the solution defined in the UML for packages.