Proper Placement of Derived Classes in the Class Hierarchy

Reda Alhajj  
*University of Calgary, Canada*

Faruk Polat  
*Middle East Technical University, Turkey*

INTRODUCTION

Users may derive new classes by defining views based on the current database contents. Some virtual classes are classified as brothers of existing classes, and others are either superclasses or subclasses of existing base and virtual classes. A base class is defined directly by the user using class definition constructs. A virtual class is classified as a brother of another class if it is derived from the latter class via a selection. To have a homogeneous system, virtual classes must be treated as first-class citizens in an object-oriented model.

In this article, we handle reusability maximization by investigating the proper location of a derived class in the hierarchy by adjusting the list of superclasses and the set of subclasses for a given class to increase inheritance instead of duplication. Our aim is to maximize inherited, and minimize locally defined characteristics by adjusting the list of superclasses to include classes that maximize inherited and minimize locally defined characteristics.

Depending on the specification of their superclasses and subclasses during query evaluation, virtual classes may be classified into four groups: (1) classes for which both superclasses and subclasses are specified; (2) classes for which only superclasses are specified; (3) classes for which only subclasses are specified; and (4) classes for which neither superclasses nor subclasses are specified. Reusability decreases from the first group to the fourth group. Two algorithms are presented to achieve the target of maximizing reusability when possible.

BACKGROUND

Shown in Figure 1 is a class hierarchy that will be referenced frequently in the article where illustrating examples are necessary. Given next is the basic terminology necessary to understand the rest of this article.

1. \( P_d(c) \): set of classes used in deriving class \( c \); \( P_d \) is empty for base classes because they are user-defined.
2. \( C_s(c) \): list of direct superclasses of class \( c \).
3. \( C_b(c) \): set of direct subclasses of class \( c \).
4. \( L_{\text{instances}}(c) \): set of object identifiers of objects added directly to class \( c \).
5. \( W_{\text{instances}}(c) \): set of object identities of objects in class \( c \) and its direct and indirect subclasses.
6. \( W_{\text{attributes}}(c) \): set of attributes that determine the state of objects in \( L_{\text{instances}}(c) \). It includes inherited and locally defined attributes.
7. \( L_{\text{attributes}}(c) \): set of additional attributes locally defined in class \( c \), that is, \( L_{\text{attributes}}(c) \subseteq W_{\text{attributes}}(c) \).
8. \( W_{\text{behavior}}(c) \): set of methods common to all objects in class \( c \). It includes both inherited methods and locally defined methods.
9. \( L_{\text{behavior}}(c) \): set of additional methods locally defined in class \( c \), that is, \( L_{\text{behavior}}(c) \subseteq W_{\text{behavior}}(c) \). For every virtual class \( c \), \( L_{\text{behavior}}(c) \) includes a method, called FindObjects(), which implements the query expression that decides on objects in \( W_{\text{instances}}(c) \).
10. \( \text{mc}(f, r) \): the header of method \( m \) such that, \( \text{mc} \in L_{\text{behavior}}(c) \), \( f \) is the list of domains for formal parameters, and \( r \) is the domain of the result.
Example 1: [Base classes] All the classes shown in Figure 1 are base classes. Next are the characteristics of some of them, where B, J, P, SI, RP, R, JP, and CP stand for Book, Journal, Proceedings, SpecialIssue, ResearchPaper, Researcher, JournalPaper, and ConferencePaper, respectively:

\[
\begin{align*}
C_p(B) &= \{\text{OBJECT}\} & C_b(B) &= \{P\} & P_d(B) &= \phi \\
L_{\text{attributes}}(B) &= \{\text{title:} \text{String}, \text{author:} \{R\}, \text{year:} \text{Date}, \text{subject:} \text{String}, \text{publisher:} \text{String}\} \\
L_{\text{behavior}}(B) &= \{\text{title()}, \text{title(t)}, \text{author()}, \text{author(p)}, \text{year()}, \text{year(y)}, \text{subject()}, \text{subject(s)}, \text{publisher()}, \text{publisher(p)}\}
\end{align*}
\]

\[
\begin{align*}
C_p(P) &= \{B\} & C_b(P) &= \{SI\} & P_d(P) &= \phi \\
L_{\text{attributes}}(P) &= \{\text{location:} \text{String}, \text{chairperson:} \{R\}, \text{committee:} \{R\}, \text{AcceptanceRate:} \text{Real}, \text{contents:} \{CP\}\} \\
L_{\text{behavior}}(P) &= \{\text{location()}, \text{location(l)}, \text{chairperson()}, \text{committee()}, \text{committee(p)}, \text{AcceptanceRate()}, \text{AcceptanceRate(i)}, \text{contents()}, \text{contents(c)}\}
\end{align*}
\]

Example 2: [Brother class] Given DBJournals as a brother of class J with: \(P_d(DBJournals) = \{J\}\). By definition, DBJournals has the following characteristics:

\[
\begin{align*}
C_p(DBJournals) &= \{\} & C_b(DBJournals) &= \{\} \\
L_{\text{attributes}}(DBJournals) &= \{\} & L_{\text{behavior}}(DBJournals) &= \{\text{FindObjects()}\} \\
L_{\text{instances}}(DBJournals) &= \{\}
\end{align*}
\]

Being its brother, DBJournals shares the characteristics of class J, and \(W_{\text{instances}}(DBJournals)\) is determined by invoking the method \text{FindObjects(DBJournals)}.

Example 3: [Virtual classes] Consider the following characteristics of some example derived classes.

PP: (PrintedPublications) includes all books and journals.

\[
\begin{align*}
P_d(PP) &= \{B, J\} & C_p(PP) &= \{\text{OBJECT}\} & C_b(PP) &= \{B, J\} \\
W_{\text{instances}}(PP) &= W_{\text{instances}}(B) \cup W_{\text{instances}}(J) & L_{\text{instances}}(PP) &= \{\} \\
L_{\text{attributes}}(PP) &= \{\text{title:} \text{String}, \text{subject:} \text{String}, \text{publisher:} \text{String}\} \\
L_{\text{behavior}}(PP) &= \{\text{title()}, \text{title(t)}, \text{subject()}, \text{subject(s)}, \text{publisher()}, \text{publisher(p)}, \text{FindObjects()}\}
\end{align*}
\]

NP: (NewProceedings) includes only some characteristics from class P.

\[
\begin{align*}
P_d(NP) &= \{P\} & C_p(NP) &= \{\text{OBJECT}\} & C_b(NP) &= \{P\} \\
W_{\text{instances}}(NP) &= W_{\text{instances}}(P) & L_{\text{instances}}(NP) &= \{\}
\end{align*}
\]

\[
\begin{align*}
L_{\text{attributes}}(NP) &= W_{\text{attributes}}(B) \cup \{\text{location:} \text{String}, \text{chairperson:} \{R\}\} \\
L_{\text{behavior}}(NP) &= \{W_{\text{behavior}}(B) \cup \{\text{location()}, \text{location(l)}, \text{chairperson()}, \text{chairperson(p)}, \text{FindObjects()}\}\}
\end{align*}
\]

**MAIN THRUST**

Locally defined characteristics of class c include \(L_{\text{attributes}}(c)\) and \(L_{\text{behavior}}(c)\). The inherited characteristics of class c are \(W_{\text{attributes}}(c) - L_{\text{attributes}}(c)\) and \(W_{\text{behavior}}(c) - L_{\text{behavior}}(c)\). In this section, we present two algorithms to investigate adjusting for a given class: (1) its list of superclasses and (2) its set of subclasses. The first is necessary for classes in the third group, and the second is suitable for classes in the second group. However, both algorithms are required for classes in the fourth group.

Algorithm 1 investigates the possibility of maximizing reusability by adding some existing classes to \(C_p(c)\) or by pushing class c into the list of superclasses of some existing classes. We employ the following heuristic: having \(C_b(c)\) not empty helps in limiting our search to the direct and indirect superclasses of every class in \(C_b(c)\). Otherwise, we have to consider and investigate all classes in the hierarchy, which is the general case when \(C_b(c)\) is empty. For each class \(c \in C_b(c)\), Definition 1 is used to find the inheritance list of class \(c\), denoted \(IL(c)\), that includes all its direct and indirect superclasses. For the other case of having \(C_b(c)\) empty, the class hierarchy must be traced top-down and left to right in order to include all the classes present in the hierarchy inside \(IL(c)\). Algorithm 1 checks for the possibility of including inside \(C_p(c)\) any of the classes in \(IL(c)\), and the list of superclasses of the considered class is adjusted accordingly.

**Definition 1: [Inheritance List]** Given any class c, \(IL(c)\) is defined using:

1. For every \(c \in C_p(c)\) and \(c \in IL(c)\);
2. For every \(c \in IL(c)\), classes in \(C_p(c')\) are placed at the head of \(IL(c)\);
3. Nothing else belongs to \(IL(c)\).

Example 4: [Inheritance List] \(IL(SI) = [B, J, P]\) and \(IL(P) = [B]\).
Related Content

Dataveillance and Panoptic Marketspaces
www.igi-global.com/chapter/dataveillance-panoptic-marketspaces/11141?camid=4v1a

Co-creation and Collaboration in a Virtual World: A 3D Visualization Design Project in Second Life
www.igi-global.com/chapter/creation-collaboration-virtual-world/63672?camid=4v1a

Categorizing Post-Deployment IT Changes: An Empirical Investigation
www.igi-global.com/article/categorizing-post-deployment-changes/3368?camid=4v1a

The Impact of Network Layer on the Deadline Assignment Strategies in Distributed Real-Time Database Systems
www.igi-global.com/article/impact-network-layer-deadline-assignment/51163?camid=4v1a