ClassExpert: A knowledge-based assistant to support reuse by specialization and modification in Smalltalk

(Draft submitted to the Fourth International Conference on Software Reuse, 1996)

Krzysztof Czarnecki, Reinhard Hanselmann, Ulrich W. Eisenecker, Wolfgang Köpf

Daimler-Benz, Research and Technology, F3S/E, P.O. Box 2360, D-89013 Ulm, Germany

Telephone (+49) 731 505-4008 or (+49) 731 505-2858

Telefax (+49) 731 505-4210

E-mail {czarnecki, hanselmann, koepf}@dbag.ulm.DaimlerBenz.com, eisenecker@ mbgate.augusta.de

Index terms *—— software reuse, classification, reuse libraries, component retrieval, object-orientation, documenting classes, documenting frameworks, Smalltalk*

Abstract*. Smalltalk-80 is an object-oriented system promoting „programming by reuse“. However, the complexity of the Smalltalk class library makes it difficult for the non-expert user to find the problem-solving class. This paper describes ClassExpert, a tool that helps to retrieve classes matching the functional specification provided by the user. ClassExpert deploys an attribute-value classification scheme with taxonomies. This paper also shows how this scheme can be used to support reuse by specialization and modification.*

Introduction

Significant productivity improvements have been reported using object-oriented programming environments such as VisualWorks\Smalltalk.[1] One of the reasons for these improvements is their libraries of highly reusable classes. Unfortunately the size of these libraries makes it hard to find the desired class, especially for a beginner. For example, the class library of VisualWorks 2.0 has over 900 classes. In addition to this complexity problem, there is a vocabulary barrier and a potential vocabulary mismatch between the terminology used in the library and the terminology of the reuser [FLGD87]. Other problems are caused by the fact that not all concepts from the library are understood by the reuser (i.e. the problem of ill-defined needs, see [Henninger94]). In general, there is a problem of *documenting frameworks* which make up the library, so that users of all levels can efficiently utilize these frameworks in their projects (see [Johnson92] and [CampIslam93]). One standard Smalltalk tool which can help to find the desired class is the *SystemBrowser*. SystemBrowser has facilities such as categories of classes, categories of methods (i.e. protocols), cross-reference search facilities (senders or implementors of a method selector, instance variable references, class references etc.) to help the user to navigate in the class library. Although these facilities provide a remarkably detailed picture of the code, they do not solve the problem of finding the desired class within a short time. ClassExpert is a tool that enables the user to search for those classes which match the user-defined functional specification of the desired class. This article describes how ClassExpert works and how it can be used to document class hierarchies of object-oriented frameworks and to ex-

[1] VisualWorks is build on the ParcPlace Smalltalk system, a derivative of Smalltalk-80. VisualWorks is a registered trademark of ParcPlace Systems, Inc.

tend their functionality by specialization or modification. Although this tool is developed for Smalltalk, the idea of ClassExpert is directly applicable to other object-oriented systems.

ClassExpert

ClassExpert is a research prototype of a knowledge-based tool for class selection in VisualWorks / Smalltalk which was designed to address two problems:

1. how to find the "right" class, which is to be directly reused in a program, from a collection of functionally similar classes
2. how to find the appropriate superclass for the new class which should implement the needed functionality when extending the class library or framework.

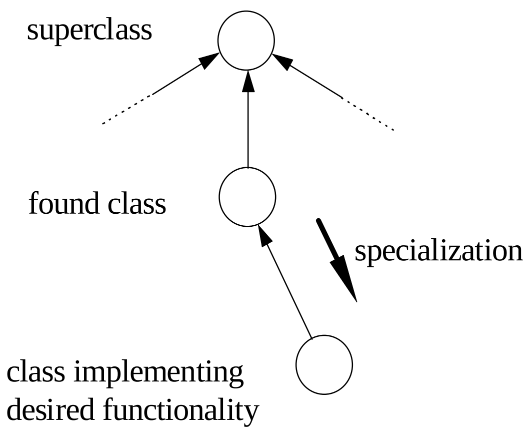


Figure 1 Extension by specialization

Extending the functionality of a framework can be achieved using *inheritance* or *aggregation*. In the case of inheritance, one can use two techniques: *specialization* or *modification* (see [BergEis95]. Specialization (see Figure 1) makes reuse of the code of the superclass which is provided by the framework (i.e. the found class) possible, whereas modification (see Figure 2), enables the user to reuse the code of both the superclass and the found class, which is a sibling in this case. The disadvantage of modification is the fact that the sibling has only a „copy-and-paste“ relationship to the class that implements the new functionality. This type of relationship causes maintenance problems since updates and bug fixes are not passed on along them automatically.

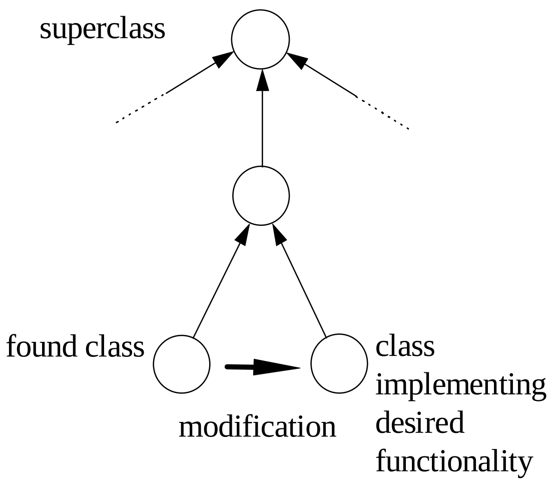


Figure 2 Extension by modification

Nevertheless, both specialization and modification promote code reuse and are of great interest to the programmer. Finally, extending functionality through aggregation also profits from methods for finding the problem-solving class.

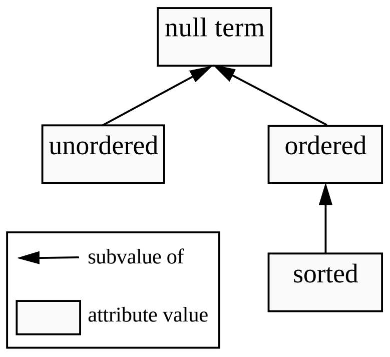


Figure 3 A sample taxonomy for the attribute "order" of collections

How does ClassExpert describe classes?

ClassExpert uses an attribute-value classification scheme, i.e. classes are described by attribute-value pairs. Attributes are also called *features*. Each attribute has a predefined vocabulary (as in the faceted scheme; see [OPB92]) and the terms are interconnected by generalization / specialization (*subvalue of*) and synonym relationships (see Figure 3). These vocabularies are called taxonomies. The root of a taxonomy is always the *null value* (in ClassExpert designated „not defined“). A sequence of attributes with their respective taxonomies defines a class-description format (i.e. the classification scheme). This description format takes advantage of the inheritance relationships between the Smalltalk classes being described, so that the values of the attributes are inherited according to these relationships. Also, it is possible to define categories which group functionally similar classes. All classes within a category are described by the same sequence of attributes; however, one class can belong to *several* categories.

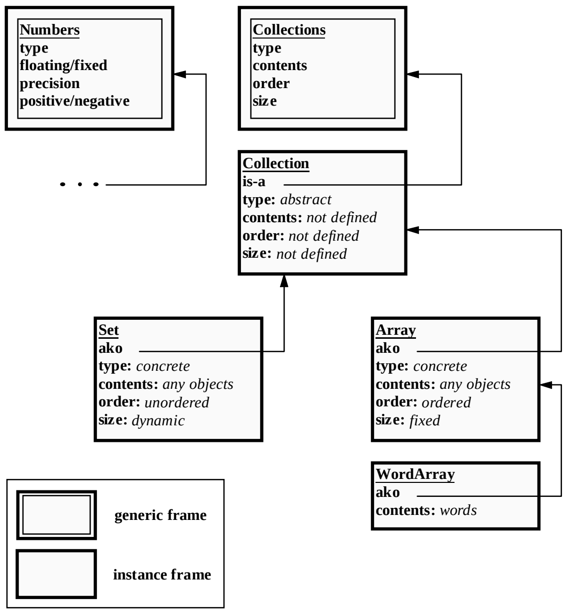
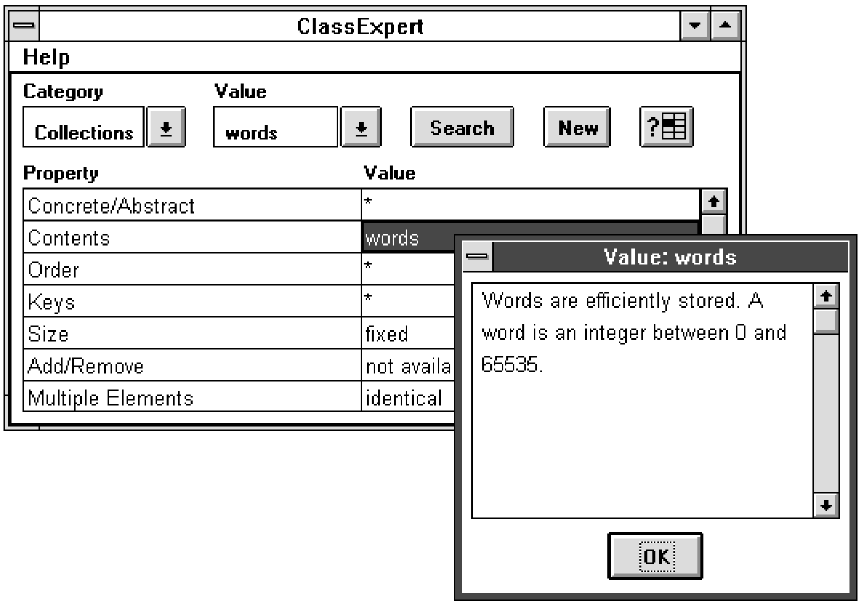


Figure 4 A fragment of a knowledge base describing Smalltalk collections



Property	Value
Category	Collections
Value	words

Property	Value
Concrete/Abstract	*
Contents	words
Order	*
Keys	*
Size	fixed
Add/Remove	not availa
Multiple Elements	identical

Value: words

Words are efficiently stored. A word is an integer between 0 and 65535.

Figure 5 Class Specification Entry Window

The components of ClassExpert

ClassExpert consists of a *knowledge base*, a *knowledge acquisition component* and a *dialog component*.

The **knowledge base** is implemented using frames (see [Minsky75]). A fragment of a knowledge base describing Smalltalk *collections* is shown in Figure 4. The attributes of a category as well as their correspoending taxonomies (there is one taxonomy per attribute) are defined in a *generic frame*. The classes belonging to a category are described in *instance frames* using the predefined attributes.

The **knowledge acquisition component** consists of a number of editors which support the definition and management of the frames. This component also includes scheme evolution facilities which help to extend and modify the metascheme of existing knowledge bases. The knowledge acquisition effort is reduced since the classes inherit their attribute values according to the class inheritance hierarchy.

The **dialog component** has a query window which enables the entry of the class specification as a table (see Figure 5). The matching classes are represented graphically in the *matching results window* as a sensitive tree (see Figure 6). Furthermore, the user can let the tool explain any of the categories, classes, attributes and values as well as directly browse through the detailed matching information (i.e. how good is the match?) and the found classes. He can also use the specification of the selected class as a new query. This technique is called *retrieval by reformulation* (see [Henninger94]).

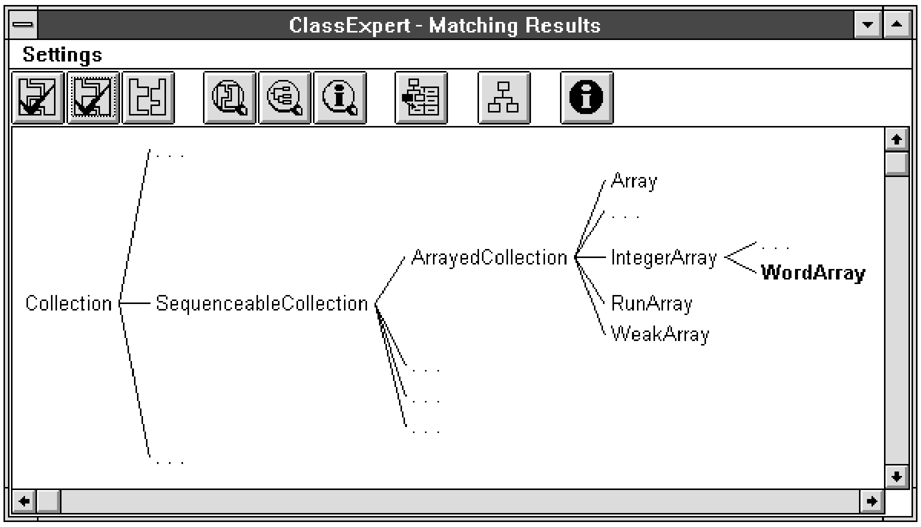


Figure 6 Matching Results Window

How to find the "right" class using ClassExpert?

ClassExpert supports three match levels which deploy the generalization / specialization relationships between the terms to provide the user with alternative solutions. These match levels are

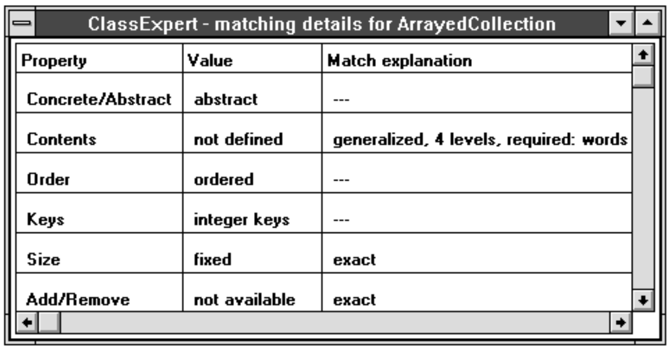
1. **exact match:** The found classes exactly match the user-provided specification of the needed class (i.e. all attribute values provided by the user exactly match the attribute values of a found class).
2. **generalized match:** The found classes are more general than the user-provided specification (i.e. the attribute values provided by the user exactly match the attribute values of a found class or are subvalues of the latter).
3. **partial match:** The functionality of the found classes partially matches the required functionality (i.e. some attribute values provided by the user do not match the attribute values of a found class).

The three match levels are accessible through the three leftmost buttons at the top of the result window (see Figure 6). The exact match level is designed to find the needed class to be reused in a program, whereas generalized and partial matches help in extending the functionality of the class library or framework. Classes found at the generalized match level are the prospective superclasses (extension by specialization, see Figure 1). On the other hand, classes found at the partial match level are good candidates for reuse through modification (extension by modification, see Figure 2).

The user interface

Much attention has been paid to the ergonomics of the user interface. The main design goal was to keep the interface as simple as possible in order to avoid the cognitive overhead. Therefore, the classification scheme has also been kept simple; for example, user-defined facet weights are not supported. Instead, the user can browse through the match results in order to determine how well a particular class matches his or her specification (see Figure 7). He can also open a browser on the class or read a short class description. The interface has been designed to conform to the Microsoft Style Guide [Microsoft91]. It also supports a *bubble help* explaining the components of the interface.

It is possible to invoke ClassExpert from



ClassExpert - matching details for ArrayedCollection

Property	Value	Match explanation
Concrete/Abstract	abstract	---
Contents	not defined	generalized, 4 levels, required: words
Order	ordered	---
Keys	integer keys	---
Size	fixed	exact
Add/Remove	not available	exact

Figure 7 Matching Details Browser

the Smalltalk on-line help through a hypertext link. This yields an excellent opportunity for integrating ClassExpert into other documenting techniques such as design patterns [Johnson92] as well as hypertext descriptions of frameworks, class categories, applications etc.

Related work

Systems deploying facet or attribute classification scheme for the retrieval of reusable components in general are described in [PrietoFreeman87], [Prieto-Díaz91], [OPB92], and [CKK91]. In contrast to these systems, ClassExpert was designed to support the retrieval of classes, so it can take advantage of the existence of the inheritance hierarchy. LaSSIE is a retrieval system which uses frames and automatic classification to aid the user in the maintenance of a big software system [DBSB91].

BRRR is a query tool for class selection in Smalltalk [Li93] and supports not only categories but also attribute-value classification scheme. In contrast to ClassExpert, BRRR does not support taxonomies (value hierarchies), which are crucial for reuse by specialization or modification. One important feature of BRRR is that the user can formulate queries at the method level.

A very similar approach to class retrieval to that of ClassExpert has been presented in [BergEis95]. This approach uses case-based reasoning (CBR) to retrieve Smalltalk classes whose functional specifications (in the form of attribute-value pairs with taxonomies) are stored in a case base. The authors of this approach used the CBR-tool INRECA, which uses quite complicated algorithms (e.g. kd-trees) and a lot of memory for the case base. For example, in the case of Smalltalk collections, the case base is about 100 times larger than the equivalent knowledge base of ClassExpert.

Conclusions

ClassExpert helps to

1. find the "right" class to be directly reused
2. find the "right spot" in the inheritance hierarchy to extend the functionality of a class library or framework by specialization or modification
3. document classes by providing a standardized vocabulary.

The primary goal of ClassExpert is to describe categories of functionally-similar classes. These categories could be so-called *one-class frameworks* (see [Deutsch89]), i.e. frameworks with one root abstract class, e.g. *collections* and *numbers*. But they could also be parts of more complex frameworks, e.g. a number of functionally-similar *views* of a concrete model-view-controller framework. The sample description scheme for collections depicted in Figure 4 shows that the attributes were chosen to adequately describe the functionality and differences of the collection classes. However in case of a set of very different classes, it is still possible to use one of the standardized sets of facets, e.g. the four REBOOT facets: *abstraction*, *operations,* *operates on*, *dependencies* (see [Karlson95, p. 99]).

Since ClassExpert documents classes and class hierarchies, it can be used in combination with other techniques, such as design patterns (see [Johnson92]), for documenting frameworks (compare to [CampIslam93]).

ClassExpert has been designed to

1. support class understandability
2. reduce the knowledge acquisition effort.

The first goal has been addressed by providing explanations and by keeping both the user interface and the classification scheme simple. For this reason, classification scheme of ClassExpert uses neither feature weights (i.e. attribute weights) nor term weights. They would complicate the scheme thereby increasing the knowledge acquisition effort. Nonetheless, feature weights can be used without having to define them explicitly. A possible approach would be to learn the values of the feature weights using the class hierarchy which is to be described itself. In this approach, the specification of each class from this hierarchy would be used to find the respective superclass. If the superclass proposed by the classification algorithm does not match the real superclass, the feature weights have to be adjusted. This approach represents an optimization problem and can be quite time-consuming. These algorithms are used in the field of case-based reasoning.

The second goal has been achieved by inheriting the attribute values from the superclass. However, the knowledge acquisition effort is significant. In contrast to syntactic features (such as method names, numbers of parameters etc.), functional features (e.g. *contents*, *order*, *size*; see Figure 4) cannot be extracted from the code automatically. On the other hand, a study presented in [BergEis95] shows that the use of semantic features for class retrieval results in better recall and precision than using syntactic features.

ClassExpert deploys functional knowledge at the class level. Examples of Smalltalk classes with more than fifty methods are no exceptions, so extending ClassExpert to apply its classification scheme to the method level seems to be desirable. The functional focus of the ClassExpert classification scheme also ignores the fact that object-oriented systems deploy inheritance to represent not only specialization but also other types of relationships, e.g. subtyping and implementation inheritance (see [Edwards93] and [Eisenecker95]). Nonetheless, functional features seem to describe the Smalltalk class hierarchy quite adequately [BergEis94].

The future work on ClassExpert will concentrate on improving its capabilities for documenting frameworks including implementation of class-typed and multivalued attributes to model class interdependencies, experiments with automatic learning of feature weights and more case studies involving documenting frameworks.

ClassExpert has been implemented in VisualWorks 2.0 and is fully integrated in the system.

References

[BergEis95] Bergmann R. and Eisenecker U. W. Fallbasiertes Schließen zur Unterstützung der Wiederverwendung objektorientierter Software: Eine Fallstudie. In M. Richter and F. Maurer, editors, *Expertensysteme 95. Beiträge zur 3. Deutschen Expertensystemtagung (XPS-95)*, 1.-3. März, 1995 Kaiserslautern. Infix, Sankt Augustin 1995, pages 152 - 169.

[CampIslam93] Campbell R. H. and Islam N. A Technique for Documenting the Framework of an Object-Oriented System. In *Computing System*, Vol. 6, No. 4, Fall 1993, pages 363 - 389.

[CKK91] Cho W. G., Kim Y. W. and Kim J. H. CLIS: A software Reuse Library System with a Knowledge Based Information Retrieval Model. In *Artificial Intelligence in the Pacific Rim: Proceedings of the Pa-*

cific Rim International Conference on Artificial Intelligence, Hozumi Takanaka, editor, IOS Press, 1991, pages 402 - 407.

[Deutsch89] Deutsch L. P. Design Reuse and Frameworks in the Smalltalk-80 Programming System. In T.J. Biggerstaff and A. J. Perlis, editors, *Software Reusability*, Vol II, ACM Press, 1989, pages 55 - 71.

[DBSB91] Devanbu P., Brachman R., J., Selfridge P., G. and Ballard B. W. LaSSIE: A Knowledge-Based Software Information System. In *Communications of the ACM*, May 1991, pages 34 - 49.

[Eisenecker95] Eisenecker U. W. Multiple Inheritance Part III: Inheritance. To appear in *Overload, Journal of the C++ S.I.G. of the Association of the C & C++ User Group*, Issue 10, 1995.

[Edwards93] Edwards S. H. Inheritance: One Mechanism, Many Conflicting Uses. In *Proceedings of the Sixth Annual Workshop on Software Reuse (WISR6, 1993),* 1993.

[FLGD87] Furnas G. W., Landauer T. K., Gomez L. M., and Dumais S. T. The vocabulary problem in human-system communication. In *Communications of the ACM*, Vol. 30, No. 11, 1987, pages 964 - 971.

[Henninger94] Henninger S. Using Iterative Refinement to Find Reusable Software. In *IEEE Software*, September 1994, pages 48 - 59.

[Johnson92] Johnson R. E. Documenting Frameworks using Patterns. In *OOPSLA'92, ACM SIG PLAN NOTICES,* Vol. 27, No. 10, October 1992, pages 63 - 73.

[Karlson95] Karlson E.-A., editor, *Software Reuse: A Holistic Approach.* John Wiley & Sons Ltd., 1995.

[Li93] Li Y. Finding Reusable Components in Smalltalk-80. In *Computers Educ.*, Vol. 20, No. 1, 1993, pages 63 - 72.

[Microsoft91] Microsoft Corporation, *The Windows Interface - An Application Design Guide,* 1991.

[Minsky75] Minsky M. A framework for representing knowledge. In P.H. Winston, editor: *The Psychology of Computer Vision*, McGrow-Hill, New York, 1975, pages 211 - 277.

[OPB92] Ostertag E., Hendler J., Prieto-Díaz R., and Braun Ch. Computing Similarity in a Reuse Library System: An AI-Based Approach. In *ACM Transactions on Software Engineering and Methodology*, July 1992, pages 205 - 228.

[PrietoFreeman87] Prieto-Díaz R. and Freeman P. Classifying Software for Reusability. In *IEEE Software*, January 1987, pages. 6 - 16.

[Prieto-Díaz91] Prieto-Díaz R. Implementing Faceted Classification for Software Reuse. In *Communications of the ACM*, Vol. 34, No. 5, May 1991, pages 88 - 97.

We thank you for reading this paper. We are very interested in any comments you might have about it. Please let us know if you found anything not clear, redundant, etc.