Software Reuse in Specification-Based Prototyping

Luqi
J. McDowell

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

luqi@cs.nps.navy.mil

Abstract

This paper explains the mechanisms for retrieving reusable software components used by CAPS, a computer-aided prototyping system for embedded and real-time software systems. The software retrieval system has been designed to provide retrievals with both high precision and high recall by exploiting the specifications associated with these prototypes. This speeds up software reuse by enabling the system to reduce the amount of information that a designer must examine to find an appropriate reusable component.

Keywords: prototyping, component specification, software retrieval, normalization, syntactic & semantic matching

1 Introduction

This paper addresses the design and implementation of the automated reusable software retrieval system in the Computer Aided Prototyping System (CAPS) [?]. The purpose of CAPS is to speed up prototyping for large Ada programs that can have hard real-time constraints. CAPS addresses these goals via software reuse, partial code generation, automatic construction of real-time schedules, and computer-aided design tools. CAPS supports automated retrieval of software components based on specifications expressed in the prototyping language PSDL [?]. These specifications serve a dual purpose: to document the requirements and design of a prototype, and to support accurate retrieval of appropriate reusable software components.

2 Relation to Previous Work

Almost all of the tools developed to assist in reusing software components use one or more of the following three approaches: interactive browsers, automated retrievals based on informal specifications, and automated retrievals based on formal specifications. Browsers are easy to implement and they are provided as backup methods by many systems, including CAPS, but they rely on the user's knowledge of the structure of the library and can require examination of the entire library in the worst case. Automated retrieval facilities based on informal specifications are also provided by many systems. The most popular variants are keyword searches (CAPS, the Operation Support System OSS), multi-attribute queries based on faceted classification (CAPS, DRACO, RAPID, OSS, the Reusable Software Library, the Common Ada Missile Packages project CAMP), and natural language searches (Reusable Software Library). CAPS also supports retrievals based on formal specifications. Formal specifications can support more accurate retrievals than informal specifications, although retrievals can be potentially time consuming. The CAPS system alleviates this problem via a layered set of increasingly refined filters, so that the more time consuming methods are applied only to relatively small sets of components.

3 Software Base Functions

The CAPS software base management system must perform three main tasks: query by specification, component browsing, and component transformation. The ability to query the software base to find software components satisfying a given PSDL specification is an essential part of the rapid prototyping method supported by CAPS. Component browsing gives the designer the ability to locate and view components in a manner other than by PSDL query, and provides interim bottom-up guidance for developing decompositions until automated assistance for this function can be developed. Component transformation is required once a reusable component is located to materialize any needed generic instantiations in a form consistent with the coding conventions of the CAPS execution support system.

3.1 Query by Specification

The CAPS software base stores components in an object-oriented database and uses PSDL specifications as the basis for high recall queries. Each stored component consists of a PSDL specification, a PSDL description of the implementation, the implementation code, and a normalized version of the PSDL specification. The syntax and semantics of the PSDL specification are used to direct the search for a component.

Figures 1 and 2 summarize the steps necessary to store components in the software base and to retrieve them using a given query specification. Components to be stored must first pass through syntactic and semantic normalization (see Figure 1). The normalization processes transform the component's PSDL specification to facilitate later matching. Syntactic normalization involves primarily format changes and statistical calculations while semantic normalization involves specification expansion and transformations.

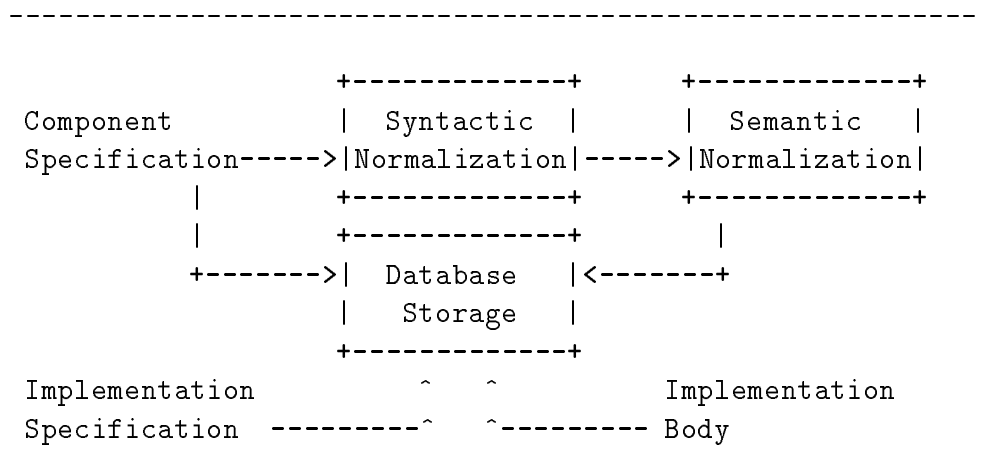

Fig. 1 Process for Adding a Software Component

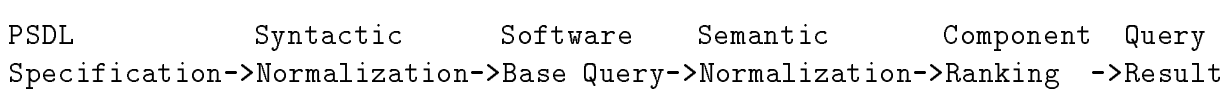

Fig. 2 Process for Retrieving a Software Component

Figure 2 shows the general process for component retrieval. A query for a library component is formed by constructing the PSDL specification for the desired component. The query specification is syntactically and semantically normalized and then matched against the stored specifications.

The retrieval process starts with a faceted classification step in which attributes that are derived from the PSDL specification are used as a multi-attribute index to select a subset of the database to serve as the starting point for the rest of the process. A major difference between the CAPS approach and other systems that use multi-attribute searches is that the attribute values are derived from the formal specification by a repeatable and completely automatic process. This ensures that components are eliminated from consideration only if they could not possibly satisfy the query.

After selection of a database partition based on the multi-attribute index, the partition is exhaustively scanned and passed through the syntactic matching filter.

The components that remain are then passed through several semantic matching filters. Syntactic matching of the query component takes place before semantic matching because syntactic matching is faster than semantic matching and is used to partition the software base quickly in order to narrow the list of possible candidates that the semantic matching algorithm must consider. Semantic matching is time consuming and must be applied to as small a candidate list as possible.

The main benefit of syntactic matching is speed whereas the advantage of semantic matching

is accuracy. Accuracy is required in order to reduce the number of reusable components that a designer will have to evaluate before making a selection. Many functions or types with different behaviors can have syntactically identical interfaces. Clearly we cannot rely on syntax alone to provide us a sufficiently fine grained search.

A semantic process alone would be unacceptable because semantic matching would have to be applied to every software base component causing the search process to be impractically time consuming. For a more detailed discussion of the semantic matching mechanisms used by the software base see [**?**].

3.1.1 Syntactic Matching

The purpose of syntactic matching is to rapidly eliminate from consideration those modules in the software base that cannot match the query specification's interface. This matching process [**?**] uses the type information in query module's PSDL interface specification to formulate a query.

The attributes of a PSDL specification p for a software component c that are important to the syntactic matching process are the following:

S(p)= ({In(t, n) : there are n>0 occurrences of type t as input parameters to c },
{Out(t, m) : there are m>0 occurrences of type t as output parameters from c },
{E : E is an exception defined in c},
{St : St is a state variable in c})

S(p) is the interface subset of the PSDL specification for module c and is the only part of the specification that pertains to the syntactic matching process.

Given a software base module c, and a query module q, along with their respective PSDL interface specifications S[c] and S[q] c is a syntactic match for q if and only if all of the following constraints are met:

(1) Exists f[i] : S[q] -> S[c] such that
[f[i](In[q](t, n)) = In[c](t', m)
and m = n
and (t = t' or t' is a generic match of t)
and f[i] is bijective]

(2) Exists f[o] : S[q] -> S[c] such that
[f[o](Out[q](t, n)) = Out[c](t', m)
and m = n
and (t = t' or t' is a generic match of t)
and f[o] is injective]

(3) if —ST[q]— > 0 then —ST[m]— > 0 else (—ST[q]— = 0
= —ST[m]—)

To improve efficiency, we use the matching rules to derive a set of module attributes that can be used to rapidly identify and reject modules with unsuitable interfaces. Some examples of these derived attributes include:

If the number of input parameters in S[q] is not equal to the number input parameters in S[c], then there can be no function f[i] to satisfy rule 1. Therefore S[c] can be eliminated from the search.

If the number of output parameters in S[q] is greater than the number of output parameters in S[c], then there can be no function f[o] to satisfy rule 2. Therefore S[c] can be eliminated from the search.

If S[q] has state variables defined (i.e. q defines a state machine) but S[c] has no state variables, then S[c] can be eliminated from the search.

The rules for the syntactic matching of type modules are similar to those for operator modules with the addition of a mapping function to map the operators of S[q] to the operators of S[c] and an additional check to ensure the generic parameter substitutions used for this mapping function are consistent for all operators in S[c].

3.1.2 Semantic Matching

Semantic matching is based on test cases and symbolic inference. The set of components that pass the syntactic matching all have interfaces that are type-consistent with the query. These components are passed through a set of filters defined by test cases derived from the PSDL specification. Additional test cases and additional filter passes are generated until either the set of candidates is small enough or the last new test case did not eliminate any components. At this point the remaining candidates are likely candidates for satisfying the query. The final phase consists of a set of automated theorem proving techniques that attempt to conclusively show that one of the retrieved components does satisfy the query specification. These are based on algebraic specifications, term rewrite systems, and a fast but limited inference method.

3.2 Component Browsing

Although browsing by component name and keyword browsing are not the preferred methods for finding reusable components in a large software base, they are needed to allow users to familiarize themselves with the components in the software base and to allow the software base administrators to maintain them. The software base was designed and implemented to support both keyword queries and named look up. The result of a keyword query is a list of those components that possess one or more of the query keywords. The list is ordered with those components that satisfy the most query keywords coming first.

3.3 Component transformation

The goal of the software base is to provide to CAPS a component implementation that is an exact match for a query specification and meets the needs of the CAPS execution support system. To accomplish this, once a reusable software component has been located it must be transformed into a form that matches all of these requirements. This transformation involves changing parameter, type, and operator names of the library component to match those of the query specification as well as instantiating any generics.

The CAPS software base cannot directly generate implementation code because it is not language specific. It can generate an abstract representation of how the library component satisfies the

syntax and semantics of a query component. This representation can then be used by a translation tool specific to a particular implementation language to generate the implementation code. This method of component integration is preferable since components coded in additional implementation languages can be added to the software base as long as a translation tool to generate the final implementation for each implementation language is provided.

4 Position

We believe that specification-based software retrieval is feasible and necessary for effective reuse in large libraries, to prevent the designer from being swamped by mountains of irrelevant components.

References

[Luqi 88] Luqi and M. Ketabchi, "A Computer Aided Prototyping System", IEEE Software 5, 2 (March 1988), 66-72.

[Luqi 88b] Luqi, V. Berzins and R. Yeh, "A Prototyping Language for Real-Time Software", IEEE Trans. on Software Eng. 14, 10 (October, 1988), 1409-1423.

[McDo 91] J. McDowell, "A Reusable Component Retrieval System for Prototyping", M. S. Thesis, Computer Science, Naval Postgraduate School, Monterey, CA, Sep. 1991.

[Stei 91] R. Steigerwald, Luqi and J. McDowell, "A CASE Tool for Reusable Software Component Storage and Retrieval in Rapid Prototyping", Information and Software Technology 38, 11 (Nov. 1991).

[Luqi 87] Luqi, "Normalized Specifications for Identifying Reusable Software", Proc. ACM-IEEE Computer Society 1987 Fall Joint Computer Conference, p. 46-49, Dallas, TX, October, 1987.

[Gris 91] M. Griss, "Software Reuse at Hewlett-Packard", Position paper for Workshop on Reuse, OOPSLA'91, Oct. 6, 1991.

[Bigg 89] T. Biggerstaff and A. Perlis, "Software Reusability", Addison-Wesley, 1989.

[Berz 91] V. Berzins and Luqi, "Software Engineering with Abstractions", Addison-Wesley, 1991.

[Free 86] P. Freeman, "Tutorial: Software Reusability", IEEE Computer Society, 1986.

[wIIT 83] Proceedings, Workshop on Reusability in Programming, ITT Programming, Stratford, Connecticut, Sep. 1983.

5 Biography

Dr. Luqi received the B.S. degree from Jilin University, China, received the M.S. and Ph.D. degrees in Computer Sci- ence from University of Minnesota, and is currently an asso- ciate professor at the Naval Postgraduate School. She worked on software research & development for the Science Academy of China, University of Minnesota, University of Maryland, International Software Systems Inc., and etc. She is a technical consultant for the computer industry. Her research interests include rapid prototyping, real-time sys- tems, design of computer languages, software reuse, specifications, software configuration management, software development tools, and scientific computing. Her research is supported by the National Science Foundation, the Office of Naval Research and many other agencies. She received an Engineering Initiation Award and a Presidential Young Investigator Award from NSF. She has published more than fifty technical papers in professional journals and conferences as well as co-authored several books. She is an associate edi- tor for IEEE Expert and the Journal of Systems Integration. She has supervised more than twenty-eight M. S. and Ph. D. theses in software engineering.

Her current address is NPS CS/Lq, Monterey, CA 93943.