ASPECTS RELATED TO THE SPECIFICATIONS OF SOFTWARE COMPONENTS

Swathy vodithala
Assistant professor, Department of CSE
KITS , Warangal, India
Email: swathyvodithala@gmail.com

Dr. Suresh Pabboju
Professor and Head ,Department of IT
CBIT, Hyderabad, India
Email: plpsuresh@gmail.com.

Abstract— Reuse is the major area to be focused in CBSE, where developing the applications can be done by using off the shelf components. In order to reuse a software component the user must understand clearly about the component and this can be achieved only when it is defined unambiguously. This paper gives a brief description of how to describe a software component using its specifications. The retrieval and search process can also be minimized if we have an accurate description of the component. The specifications of a component include syntax, semantics and performance.

Keywords-: syntax, semantics, performance, behavior, specifications

I. INTRODUCTION

Software engineering is the application of a systematic and disciplined approach to the design, development, operation, and maintenance of software. Component-based software engineering (CBSE) (also known as component-based development (CBD)) focuses on the development of applications based on software components, so that the applications are easy to maintain and extend. A *software component* is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard. Software element contains a sequence of abstract program statements that describe the computation to be performed by a machine [1][7].

The software engineering area is playing an important role because of its dynamic nature .SE accepts even the other technical areas into it which strengthens this stream. The major area in software engineering is the software reuse. Software reusability is the use of existing software or software knowledge to construct new software .Reuse may be on design pattern, program elements or tools [6]. There are three major areas in software engineering which has to be focused when considering the components for software reuse[1]. These are described as

a) Classifying the components needed.

b) Describing the components wanted.

c) Finding the appropriate component.

The objective of presenting this paper is to describe the software components using specifications.

II. RELATED WORK

The specification of a software component must be unambiguous, clear and understandable to both the user and developer. The specification of a software component must be free of implementation details inorder to support a variety of components that implement it.

We use the word specification to mean an explanation of the structure , the behavior and the performance of a software component, i.e., its syntactic interface , semantics and the related complexity.

A. Syntactical specification

A component may be either a function or a module. One of the important ways to specify the syntactical structure of a software component is based on signature[3]. Every function can be defined with the prototype or declaration. This prototype is treated as a signature of the function.

Example: Consider the signatures presented in Figures 1 and 2 for a stack of integers and a queue of integers, respectively [5].

Create: => Stack

Push: Stack x Integer =>Stack

Pop: Stack => Stack

Top: Stack => Integer

Empty: Stack => Boolean

Figure 1: Signature of a Stack

Create*:* =>Queue

Enqueue: Queue x Integer:=>Queue

Dequeue: Queue =>Queue

Front: Queue => Integer

Empty: Queue => Boolean

Figure 2: Signature of a Queue

The names push and enqueue are synonyms and they can be even replaced by the term Insert. Now, Consider,

Figure 3 which appears to be equally applicable as a signature for both stack and queue, primarily due to the neutral nature of the names used.

Create: Sequence

Insert: Sequence x Integer =>Sequence

Remove: Sequence =>Sequence

Current: Sequence => Integer

Empty: Sequence => Boolean

Figure 3: Signature of a sequence.

The signature matching mainly uses the names and datatypes for matching the functions or modules.The standard library or repository consists of many functions,and most of the functions have same signature .Let us consider the functions strcpy ( ) and strcat ( ), they have same signature. The user would be unhappy if we substitute strcpy() in place of strcat() because the strcat() will concatenate the given srings and strcpy() will copy the given string into another.That is why the signature of a component alone is not sufficient for describing the component completly.

B. Semantical specification .

The behavior of software is normally explained by one of the techniques given below:

1. Informal specifications
   a) comments embedded in code
   b) informal metaphors
2. Formal specifications
   a) formal mathematics.
      i) algebraic specifications
      ii) model based specifications
   b)predicate calculus

The code written must always be explained with the comments so as the user or developer can further reuse it efficiently. The documentation plays a vital role in order to maintain and support the software. But,the comments mentioned in the code are not mandatory and even the absence of it will make no difference for writing a sourcecode[9]. The comments are usually written in natural language and are not the part of compilation process.

A metaphorical description of software behavior is attractive because it is easy to read, but it is also inherently ambiguous because it is based on natural language.An example of a metaphorical description is “stacks are like piles of cafeteria trays.” This metaphor can be used to explain the visible effects of operations on stacks such as push and pop.The general approach is to imagine a parallel in the physical world that acts as a *model* of an object in the computer. A metaphorical specification is the other side of the coin from the fundamental idea of object-oriented design (Cox, 1986; Meyer, 1988) in which a software object is considered to model a physical object. In either case it is important to establish a one-to-one mapping between physical and software objects, and this mapping can be interpreted in either direction. Just as a stack can be explained in terms of cafeteria trays, so might a program simulating cafeteria trays use a stack[9].

Both the comments and metaphors are informal specifications to explain the behavior of software components.But in either of the cases we use the natural language description .A natural language description may be part of an acceptable reusable component specification but it cannot do the job alone.

Two major approaches to formal specification have been developed:

1. *algebraic* specification (Guttag *et al.*, 1985; Wing, 1987) and
2. *model-based* specification (Bjørner *et al*., 1987;Spivey, 1989).

Both are based on the idea that abstract component behavior can be explained through *mathematical modeling* of program objects by mathematical objects

C. performance specification

The semantic description (behavior) of a component is a complex task when compared with the syntactical specification .we can also describe the performance of a component rather than syntax and semantics. The performance specification includes the time complexity and space complexity of a particular component. The execution time duration and storage requirement of a procedure generally depends on its parameters [10].

III. PROPOSED WORK

Based on the decomposition style of a program , there are different programming paradigms. They are

1. Procedural programming
2. Functional programming.
3. Abstract data type programming.
4. Modular programming.
5. Object oriented programming.
6. Generic programming.etc

Here, we consider the functional programming, which is based on the theory of mathematical functions. It does the computing of values based on expressions and functions[8]. There are different mechanisms to examine the behavior of a software component. In this paper we illustrate all the possible ways of examining the behavior based on formal specifications of a software component using simple examples.

A. Behavior by pre and post conditions:

Program operations are defined through *preconditions* and *post conditions*[2][4]. The meaning of this kind of specification is that, if the precondition is true when the operation is invoked, then the post condition is true when it returns. Preconditions and post conditions are assertions in predicate calculus with equality in which the variables are the mathematical models of the operation's formal parameters.

***Example*:** Consider addition of 2 numbers.

```
/*addition of two numbers */
int sum(int a,int b)
{
   return(a+b);
 }
```

Precondition(s): This function requires inputting two integer values say a and b.

Precondition set = {a>0,b>0,a=0,b>0,b=0,b<0}

Post condition(s): One of the conditions will be true after executing the function

Post condition set= { a+b>a, a+b>b , a+b<a, a+b<b, a+b=0}

B. Behavior using a regular expression:

***Example*:** Consider addition of 2 numbers.

R.E = a+b

Where a→[0-9]

b→$[0\text{-}9]^{+}$

C. Behavior using first order predicate

The behavior of a component can also be explained by using a first order predicate. The predicate is described using a self explaining hypothetical logic language[8].

Examples:

1. Searching of an element from an array

. →concatenation

iff →if and only if

For every element k and list L: is_in(k,L) iff

L=[k]

Or

L=L1.L2 and (is_in(k,L1) or is_in(k,L2)

is-in is a predicate and is-in(k,L1) can be read as k is an element which is present in the list L1.

2. Addition of two numbers

add (a,b)

3. Average of two numbers

add(a,b)∩div(a,b)

D. Behavior using mathematical predicates:

The behavior of a component can also be explained by using a first order predicate. The predicate which is described by using hypothetical logic language can be assigned to a relevant mathematical operator.

1. Searching of an element from an array

For every element k and list L: ∈ (k,L) iff

L=[k]

Or

L=L1.L2 ∩ (∈ (k,L1) U ∈(k,L2))

2. addition of two numbers

+(a,b)

3. average of two numbers

+(a,b)∩ /(a,b)

E. Behavior using mapping rule:

The behavior of a component can also be explained by using a mapping rule .The mapping rule gives the value of a range associated with the each value of its domain

***Example*:** Consider addition of 2 numbers

add: int→int

add(a,b) ≡ a+b(mapping rule)

IV. CONCLUSION AND FUTURE SCOPE

Software reuse is an important area because any application which is developed cannot be done from scratch .There is even nothing worst if we do not go for reuse to develop an application which is almost all available. Software reuse needs an efficient repository to store the components and the components that are stored must be well described. Based on the description of components it is possible to have a good search and matching mechanisms for using the component. There are many ways for describing the components such as based on syntax, semantics and performance .The syntax can be properly defined for any function or module, but the problem is with the description of semantics of component. There are many restrictions for describing the behavior of a component. The behavior can be described differently for different programming styles. The future scope is that to have a better description of a component behavior which works for any programming style.

REFERENCES

[1] "A Resolved Retrieval Technique for software Components" by Swathy vodithala, Niranjan reddy, Preethi. IJARCET volume 1,issue 4 june 2012.

[2] "Specification matching of software Components" Amy moormann zaremski and Jeannette m. wing

[3] "Signature matching: A key to reuse"Amy moormann zaremski and Jeannette m. wing.

[4] "Examining behavioral matching"Steven Atkinson.Software Verification Research Centre.Department of Computer Science University of Queensland

[5] "Component Classification and Retrieval Using Data Mining Techniques" Proceedings of National Conference on Challenges & Opportunities in Information Technology (COIT-2007) RIMT-IET, Mandi Gobindgarh. March 23, 2007. Achala Sharma, Daman Deep Kaur

[6] "An Integrated Classification Scheme for Efficient Retrieval of Components" Dr C.v.guru Rao and P.Niranjan. Journal of Computer Science 4 (10): 821-825, 2008 ISSN 1549-3636 © 2008 Science Publications

[7] Book: " Definition of a Software Component and Its Elements" by Bill Councill, George T. Heineman

[8] Book: Programming language concepts carlo ghezzi and mehdi jazayeri

[9] Reusable software components by Bruce W. Weide, William F. Ogden, and Stuart H. Zweben ,Department of Computer and Information Science,The Ohio State University,Columbus, Ohio 43210-1277

[10] performance specification of software components,murali sitaraman SSR'01, May 18-20, 2001, Toronto, Ontario, Canada. Copyright 2001 ACM 1-58113-358-8/01/0005…$5.00.

SWATHY VODITHALA received her B.Tech degree in computer science and engineering in 2005 from KITS, huzurabad (Jawaharlal Nehru Technological University),and M.Tech degree in software engineering from KITS, Warangal (kakatiya university) in 2011.She is pursuing her PhD from Osmania University. She worked with computer science department for 2 years in KITS, huzurabad as an Assistant Professor. At present, she is working as an Assistant Professor in computer science department at KITS, Warangal.