

A Rule-based Component Customization Technique for QoS Properties

Jia Zhou, Kendra Cooper and I-Ling Yen
Department of Computer Science
University of Texas at Dallas
{jxz023100, kcooper, ilyen}@utdallas.edu

I. Introduction

Component customization technique is an important part of component-based development (CBD) process. It allows the component to be effectively adapted and re-used in diverse applications. However, existing component customization techniques focus on customizing the functionality of a component. Embedded systems generally have strict quality of service (QoS) requirements such as timeliness, memory limitations, output precisions, etc. Currently, customization for the satisfaction of QoS requirements has not been widely addressed.

To address the problem, we propose the concept of configurable parameter. A configurable parameter is a constant or a variable defined in a component which, when adjusted, can provide QoS trade-offs. A QoS properties reconfigurable (QoS-R) component is a software component which can be reconfigured either statically or dynamically to provide desired QoS trade-offs. In this paper, we present our current results in developing a new, systematic code parameterization technique which focuses on identifying configurable parameters in components and converting them into QoS-R components.

II. Rule-based Code Parameterization Technique

The code parameterization process is divided into three steps [1]. The first step is to identify configurable parameters in a non-QoS-R component. In the second step, the non-QoS-R component is modified according to each configurable parameter identified so that the component becomes QoS-R component. Finally, QoS data of each component under different configurations are collected and stored in the repository. These QoS data record the impact of configurable parameters to QoS properties of the component and provide necessary information for component retrieval and composition analysis in future system development. In this paper, the focus is on configurable parameter identification.

A. Overview of Configurable Parameter Identification

The configurable parameter identification process analyzes the source code of a component based on the grammar of the programming language and converts the source code to a parse tree. Two sets of rules are defined to assist the identification process. The fact analysis rules are used to extract syntactic and semantic facts of grammar construct from the parse tree. The configurable parameter identification rules are used on these facts to evaluate every variable and constant identifier declared in the program to test if a variable or constant is a specific kind of configurable parameter.

B. Fact Analysis Rules

The basic facts in the program can be classified into identifiers, statements, and expressions.

Identifiers are the basic data objects of a program. Some of them may be the potential configurable parameters. A fact is used to express that an identifier is of a certain type. For example, *constant(d)* and *variable(d)* indicate that the identifier *d* is a constant or variable.

A statement is the basic execution unit of a program. A variety of different kinds of statements need to be identified in a program. For example, *assignment-statement(s, lhs, rhs)* indicates that statement *s* is an assignment statement where *lhs* and *rhs* are the left hand side expression and the right hand side expression of *s* respectively. *loop-statement(s, c, b)* indicates that statement *s* is a loop statement (i.e., *s* is one of for, while, do while statement) where expression *c* is the loop condition of *s* and compound statement *b* is the loop body of *s*.

Expression consists of identifiers and operators and is the constituent element of the statement. For example, *comparison-expression(e, lhs, rhs)* indicates that the expression *e* is a comparison expression (i.e., *e* is one of less than, greater than, less than or equal, greater than or equal, equal, not equal expression). Here *lhs* and *rhs* are

the left hand side expression and the right hand side expression of e respectively.

We also define a fact to express the relationship between identifiers, expressions and statements. We use *inside(d, e)* to express that grammar construct d appears in grammar construct e . Note here d may be a constant, variable or statement, and e may be an expression, statement or a compound statement.

C. Configurable Parameter Identification Rules

A set of configurable parameter identification rules is defined to identify different kinds of configurable parameters in a program. The configurable parameter identification rules use the basic facts extracted from the program by fact analysis rules as premises to generate conclusions. These conclusions express if a constant or variable in the program is a configurable parameter.

As an example, we present one rule: loop control variable. Loop control variables generally impact the program's Time-Quality trade-off. A larger loop control variable can improve the result's quality at the cost of increasing program execution time. A smaller loop control variable reverses the effect. The rule for identifying a loop control variable is as follows: for each loop statement, if the guard condition of loop statement is a comparison expression with a constant value on left hand side of inequality and some variables on right hand side of inequality (or vice versa) and if in body of loop there exists the same variables on left hand side of assignment statement, then the constant value on one side of inequality is a loop control variable. This can be defined as a predicate *loop-control-variable(d)*:

loop-control-variable(d) iff:

$constant(d) \wedge variable(d1) \wedge loop-statement(s, c, b) \wedge comparison-expression(c, lhs1, rhs1) \wedge (inside(d, lhs1) \wedge inside(d1, rhs1) \vee inside(d1, lhs1) \wedge inside(d, rhs1)) \wedge assignment-statement(s1, lhs2, rhs2) \wedge inside(s1, b) \wedge inside(d1, lhs2)$ is true.

D. Example

Newton's algorithm is used to illustrate how the fact analysis rules and configurable parameter identification rules are used. Newton's algorithm is a numerical method for finding the roots of a polynomial. The code for the algorithm is shown in Figure 1. Note that f is the input function for which the root is computed and df is the differentiation of f . The value of "epsilon" affects the Time-Quality trade-off. With a larger "epsilon", the algorithm generates a root with less precision but the computation takes less time. With a smaller "epsilon", the effect is reversed. This is a very common type of configurable parameter, namely, loop control variables

The fact analysis rules are applied first during the top

down scan of the program to extract some facts. For example, from line 2 "`#define epsilon 0.001`", the fact that identifier "epsilon" is a constant is obtained. This is expressed by *constant(epsilon)*.

When fact extraction finishes, the configurable parameter identification rules are used to analyze extracted facts and generate conclusions. The rule used is the **loop-control-variable** rule. When the rule is applied to on the identifier "epsilon", the rule finds that all the premises for the rule are true and it gives the conclusion that "epsilon" is a loop control variable.

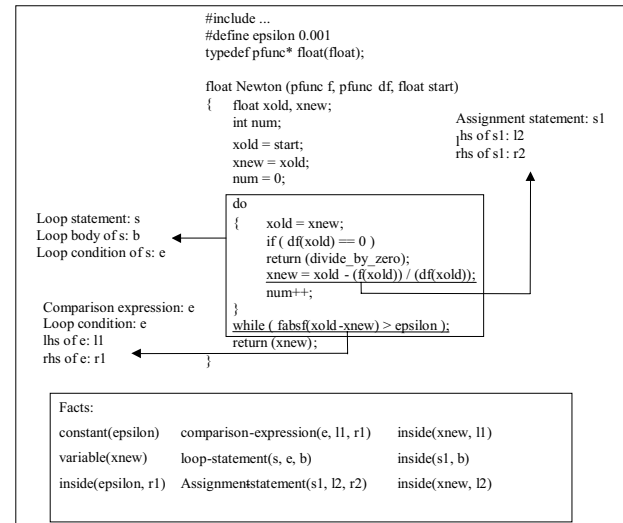


Figure 1. Newton's example

III. Conclusion

This paper presents an overview of a code parameterization technique and focuses on the configurable parameter identification process. This code parameterization technique can customize an existing component into one that supports the reconfiguration of QoS properties including time, space, and quality. The technique has a defined process, is rule based, and is semi-automated.

IV. Acknowledgement

This research is supported in part by the National Science Foundation under Grant No. EIA-0103709.

References

- [1] Kendra Cooper, Jia Zhou, Hui Ma, I-Ling Yen, Farokh Bastani, 'Code Parameterization for Satisfaction of QoS Requirements in Embedded Software', Proceedings of ERSA'03, Las Vegas, June 2003, pp. 58-64.