Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 182 (2007) 187–200
www.elsevier.com/locate/entcs

Component Updates as a Boolean Optimization Problem

Alexander Stuckenholz[1]

Department of Data Processing Technologies
FernUniversität in Hagen
Hagen, Germany

Abstract

Component updates always bear the risk of negatively influencing the operativeness of software systems. Due to improper combinations of component versions, dependencies may break. In practise this often turns out to be due to missing or incompatible interfaces and signatures (syntactical interface) but may also be caused by changes in behavior or quality. In this paper we model the problem of finding a well-configured system consisting of multiple component versions as a Boolean Optimization Problem. To achieve this, we introduce objective functions and constraints that lead to most recent, minimal systems and use Branch-and-Bound to restrict the search space.

Keywords: Component Based Software Development, Updates, Component and System Evolution, Compatibility, Boolean Optimization, System Synthesis

1 Introduction

Never change a running system! Every system administrator knows this rule to prevent unforeseen incompatibilities often causing breakdowns and sleepless nights. But sometimes parts like components have to be replaced by newer versions because of serious security holes, functional limitations or quality improvements.

By updating a system, thus replacing components by newer versions, dependencies may be added or removed. The system changes over time which is called *system evolution*. Most recent research investigating the source of defect of object oriented software systems (cf. [2]) indicate that missing components or wrong component versions are the most frequent reasons for configuration problems.

Nevertheless, there are almost no tools that are able to prevent these situations. Tools for automated configurations which combine compositional reasoning with

[1] Email: Alexander.Stuckenholz@FernUni-Hagen.de

1571-0661 © 2007 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2006.09.039

automated versioning and dependency analysis (cf. [17]) are missing in configuration management.

In this paper, we sketch a mechanism to construct well-configured component based systems by combining available component versions and respecting constraints like favoring recent versions, minimizing the count of components in a system and minimizing the count of replacements between updates.

The system is specially designed for situations in which new component versions are not exact substitutes of their predecessors, but introducing new dependencies or change parts, on which other components in the system rely. We assume that in such situations, a balanced configuration can be found by a smart combination of available component versions. This way, we do not require backward compatibility between component versions, which is the basis of most packaging systems in this area, but allow arbitrary evolution of involved components.

The paper is structured as follows. The next section introduces the original problem of incompatible component updates in more detail. Section 1.2 mentions related approaches like Linux packaging to solve the update-problem and the application of optimization methods. Section 2 introduces a simple component based system, which we will use as a running example for the rest of the paper. In section 3 we establish objective functions and constraints for the problem in the normal form of a Boolean Optimization Problem. Furthermore we mention the complexity of the combinatorial problem for finding well-configured system configurations. Section 4 addresses the search for solutions by calculating upper and lower bounds and their usage in branching the search-tree by a Branch-and-Bound algorithm. Section 5 mentions the real-world projects in which these methods are currently evaluated and presents the results attained so far. Finally section 6 summarizes the results and gives some prospects to open questions and further research.

1.1 The Problem

Component Based Software Development (CBSD) is defined as the planned integration of preproduced software components (cf. [3]). The main goal is to reduce development costs, shorten the time-to market by massive reuse and to increase product quality by frequently utilized software artifacts.

At the same time, even today many software developers spend their time reinventing the proverbial wheel, although the reuse of software components in an early development state would eliminate the necessity to redevelop similar functionality again. However, CBSD has certain drawbacks compared to conventional software architectures. Direct and indirect dependency relations between the components of a system arise when software components are reused in several application at the same time. Figure 1 clarifies this structural difference between monolithic in contrast to component based architectures.

In conventional, monolithic software architectures, all functions required by applications A and B are implemented internally. With such a structure, a substitution of an erroneous or non-performant function $F(x)$ is almost impossible, as all applications of the platform have to be analyzed for their usage of that or similar

Download English Version:

https://daneshyari.com/en/article/423512

Download Persian Version:

https://daneshyari.com/article/423512

Daneshyari.com