Abstract—End-user programmers such as scientists and engineers often adopt a visual domain-specific language due to its easy learnability, but then they later encounter problems when trying to create high-performance programs. In response, they typically have had to resort to learning and using general textual languages such as C or Fortran as a supplement or replacement for the visual language. This paper proposes a technique, called Smell-driven performance analysis, for helping end-user programmers to overcome performance problems without leaving the visual dataflow paradigm. The technique involves statically analyzing programs to heuristically detect areas with potential performance problems (“bad smells”), alerting end-user programmers about problems, and advising on how to fix those problems. We have implemented a prototype for applying this technique and conducted a user study in which end-user programmers diagnosed performance problems. The experiment showed our technique increased participants’ success rates at finding problems and decreased the time required for finding solutions. Qualitatively, 92% of participants said our technique was helpful, and they listed numerous specific benefits provided.

Keywords—end-user programming; performance; visual language

I. INTRODUCTION

Professional software engineers have long known that the performance of a program can have a powerful effect on its value. Some end-user programmers also share this concern about performance. For example, empirical studies have revealed that scientists tend to rely on textual general-purpose languages such as C++ or Fortran for high-performance computing, despite the availability of domain-specific languages designed for high usability [2][7][18].

LabVIEW is one example of a programming language that has high usability but sometimes leads to programs with inadequate performance. Its maker, National Instruments, claims LabVIEW is the “most widely used development environment for instrument connectivity and [hardware] test application,” particularly among engineers and scientists [9]. An independent survey of LabVIEW users investigated the reasons for this high level of adoption, finding that users appreciated LabVIEW primarily due to its visual dataflow language and secondarily due to its support for code reuse [21]. One respondent summarized, “The development time for LabVIEW is less than half that for C,” while another claimed to be “3X more productive than programming in C”. Yet at the same time, survey respondents sometimes found LabVIEW performance to be inadequate. They expressed concerns that LabVIEW offered no way to optimize usage of registers, memory, disk, and CPU. The researchers conducting the survey noted, “LabVIEW solves these problems by allowing the programmer to call code written in other languages”—in other words, the “solution” to the problem is to step outside LabVIEW.

These challenges are not unique to LabVIEW. Nardi lists HP VEE and Prograph as two other canonical examples of visual dataflow languages [12], where data input/output nodes are connected to one another via computation nodes and virtual dataflow “wires.” Scientists and engineers using these languages (e.g., [8][19]) also encountered performance problems. To solve such a problem, “Parts of the application were written in the C language and linked to the VEE application to obtain higher throughput” [8].

In short, these empirical studies reveal a way in which visual dataflow languages can present a “low ceiling” over what users can accomplish [11]. To date, researchers have mainly considered language ceiling from the standpoint of functionality. For example, one paper discussed how end-user programmers might encounter a ceiling when they try to create custom widgets [11], while another discussed the ceiling that people encounter when they go from animating 2D images to animating 3D images [16]. In contrast, the empirical studies above reveal a ceiling in terms of how well programs carry out functions, rather than in terms of what functions are carried out. Helping end-user programmers to overcome performance problems and break through this ceiling—without leaving the visual dataflow language—is an unexplored research challenge.

In this paper, we propose a new technique called Smell-driven performance analysis aimed at meeting this challenge. The technique uses the established concept of a “bad smell,” which is a heuristic for finding sections of code that function correctly but that have poor quality [4], and applies this concept in the context of visual dataflow languages favored by some end-user programmer populations. Specifically, Smell-driven performance analysis involves statically analyzing programs to heuristically detect areas with potential performance problems, alerting end-user programmers about problems, and advising on how to fix those problems.

Our technique implicitly depends on two hypotheses whose truth is non-obvious. First, we hypothesize that many performance problems are primarily the result of how end-user