Optimized late binding: the SmallEiffel example
Olivier Zendra, Dominique Colnet

To cite this version:

HAL Id: inria-00563279
https://inria.hal.science/inria-00563279
Submitted on 4 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Optimized late binding: the SmallEiffel example.

Olivier ZENDRA, Dominique COLNET

{zendra, colnet}@loria.fr

LORIA
(INRIA - CNRS - Univ. of Nancy, France)
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy Cedex
FRANCE

Keywords
dispatch, late binding, type inference, specialization, SmallEiffel

Issue
Object-oriented languages imply a great number of dispatched call sites, whose cost is thus a crucial point and must be reduced as much as possible in order to get the best performance. However, an important constraint weighs on the compilers and the techniques they use to generate such efficient programs: the performance of the compiler itself has to be high enough to allow its use in incremental development environments.

Many dispatch techniques have been studied by Driesen et al. [DHV95]. Dynamic dispatch techniques (e.g. Polymorphic Inline Caches) rely on run time or profile-guided information [HCU91, AH96]. They consist of various forms of caching at run time [UP87]. Static dispatch techniques precompute their data and code structure at compile time. Most are variants of the VFT method, which has been shown not to schedule very well on modern processors [DH96, DMM96]. Improving dispatch techniques requires having type information on the (sometimes whole) system. Type inference and code analysis algorithms thus appear as extremely useful, and have been subject to much work [DGC95, Age95, BS96]. These analyses also allow other optimization techniques, such as customization [CU89, CU90].

Solution Approach
A two-pronged approach has been chosen, which is implemented in our Eiffel compiler, SmallEiffel. First, it consists in analyzing the compiled program in order to solve as many polymorphic call sites as possible into monomorphic ones. It is then easy to implement the latter much more efficiently than the former. Since even the most powerful type inference algorithm cannot solve all polymorphic calls sites, a second step consists in implementing the remaining polymorphic sites as efficiently as possible.

This solution is based on a simple, fast, whole system type analysis, that may give non optimum information, but does so at a reasonable cost [CCZ97]. A very good tradeoff is thus reached between compilation speed and the optimality of the generated executable. Call sites which can be resolved as monomorphic are implemented with static, direct calls.

Because the latter schedule very well on modern processors and are likely to continue doing so in future ones, we also implement the remaining polymorphic ones as static, direct calls to specialized routines. The various possible
these direct calls, unlike VFT-based ones, also enable further optimizations, such as inlining, which are crucial for speed. Although fairly simple, this system gives very good performance in terms of speed, and the increase in code size is very reasonable.

Of course, issues remain in terms of separate compilation and distribution of libraries without their source code.

References


