Using Metainformation

Christoph Steindl[1], Markus Hof[1], Stefan Chiettini[1], and Ákos Frohner[2]

[1] Institute for Practical Computer Science, System Software Group
Johannes Kepler Universität Linz, A-4040 Linz, Austria
{steindl, hof, chiettini}@ssw.uni-linz.ac.at
[2] Department of General Computer Science
Eötvös Loránd University, Budapest, Hungary
Akos.Frohner@elte.hu

Abstract. In the past, systems have been monolithic and static. In the future, systems will be modular and dynamic. The use of metainformation helps to perform this evolutionary step or is an enabling technology for it.

1 Introduction

When we look at the historical development of programming languages, we see a continuing rise of the level of abstraction of machine details: from machine instructions, via assembly languages and higher programming languages to structured and object-oriented languages. The next step in this development is platform indepencence. Combined with network transparency, it allows to write programs that can run on any computer in a networked environment.

The composition of new software out of existing components is another promising application area: The programmer composes programs in a visual programming environment by sticking components together or glues them together using scripting languages. The components shall then cooperate, which they only can do if they have some knowledge about each other: their interfaces, properties, facilities and so on.

Metainformation seems to be the bridge between abstraction on the one hand and knowledge about each other on the other hand. *Metainformation makes some knowledge explicit that was previously only implicit.* It is also a means to make information available at run time that was usually only available at compile time. Metainformation is also the key to systems that are really extensible where only the required components are active at a time and where additional functionality can be added on demand.

Metaprogramming can exploit metainformation to several degrees: it can use metainformation to oberserve and manipulate itself and other running programs (*introspection*). It can use metainformation to explicitly call functionality that is normally hidden in the run-time system, e.g. creation of new objects, dynamic loading, linking, and unloading of components (*interception*). And it can use metainformation to change the behavior of language primitives at run time, e.g.

object creation and destruction, method dispatch, and access to simple attributes (*invocation*).

We believe it will be crucial for every computing system and especially programming language to offer a standardized access to metainformation. Many do so already (Lisp, Smalltalk, CLOS, Beta, Oberon-2, Java), and we think the implications and principles are well understood. However, we think that it is vital that this access is efficient in memory usage as well as in its run-time behavior. We see a wide field for further research and projects in this area: retrieving metainformation from legacy systems in an automatic or semi-automatic way; extending popular languages to handle metainformation; creating visual environments to support various aspects of this field.