The History of Software Localization

László Gulyás

There are different possible ways to arrange the developments of programming technology (from its very beginnings to nowadays' trends) into one logical and coherent line of evolution. One of these options is to discuss the subject as a process of increasing software localization. According to this idea the history of computer science is dominated by the classical *divide and conquer* rule. That is, facing the more and more complex problems to solve (which was enabled by the increasing power of the underlying hardware), programmers needed more and more localization of (concentrated focus on) their software elements. Whatever trivial this idea may seem at first, it has the distinctive power to unify different programming techniques and, at the same time, to separate ones which are close to each other but contain very important underlying conceptual differences.

According to this view, monolithic programs first got divided into separate *compilation units* and *subroutines* within each source file. Later on, the movement of structured programming further localized the 'behavior' of the software by introducing the concept of *block* and made the first step toward data-localization through *user-defined types*. The increasing acceptance of the *Abstract Data Type* concept has finalized the localization of data (pairing that of code or 'behavior') by *modules* on one hand, and *object-orientation* on the other. Recently, these concepts have been furthered by standardization and continuity in time in *component architectures*. The last step along the pathway of increased localization, up to now, is represented by *autonomous agents* (according to the *weak definition* of agency), where localisation of code and data is completed by that of its control.

In this paper, we will elaborate on the idea described above, reporting on a survey of programming languages (including C, Pascal, Module-2, Objective-C, C++, Ada, Java). This survey conceptualizes the different techniques the surveyed languages use to handle different aspects of software localization. As a result of this work, we construct a taxonomy of programming languages containing classes of increasing software localization. We also present concrete, implemented instances for the defined classes. These examples show that some of the localization aspects span across a variety of programming languages, while others are rather specific to some environments.

In addition to the classification above, our results contribute to the explanation of the driving forces behind popular trends of today's computer science. For example, as sketched above, they make a clear and logical connection between such techniques as object-orientation, autonomous agents and component architectures.