

DSLs as a silver bullet?

Jean Bezivin, ATLAS Group, INRIA & University of Nantes
jean.bezivin@univ-nantes.fr

I. Introduction

Since the time Brooks published his seminal work, several other tentative silver bullets have been proposed. Among the latest attempts, DSLs (Domain Specific Languages) may bring significant improvements or may alternatively cause considerable havoc in the software development scene. The lessons taught by Brook in 1987 are still of complete actuality and may be of great help if we are able to use them in the evaluation of this important DSL evolution. It is probably too late to avoid the proliferation of small dedicated languages but it is still time to organize and harness their collective usage in good practices.

II. Domain Specific Languages

In the post-object period (from the 90's to now), one of the main trends has been to find linguistic support for expressing separation and composition of concerns in software systems. The dream of a unique and unifying general purpose language (GPL) able to express everything from high level requirements to low level machine idiosyncrasies progressively came to its end in the 80's.

One of the last known GPL dinosaurs was ADA, as a successor of PL/1, and this failure was concomitant to many new tentative proposals more or less directly related to DSLs. One may quote the rapid extension of XML, of pattern languages, of aspect oriented programming, of model engineering and many more. The common feature of all these approaches is the recognition that there are at least some parts of the software system or some tasks in the development process that needs alternative languages as a support. Each brought its own myth with which it is often difficult to survive. In the case of the OMG MDA proposal, the myth is the hope of complete potential separation of business knowledge on one side and of platform knowledge on the other side that could be combined by model transformation into an executable integrated product. In the case of AOP the myth is to let intact the main programming substrate while adding a number of pluggable aspects for the representation of multiple concerns. In the case of XML the myth is to provide the XSD metalanguage to design a galaxy of small languages (schemas) allowing to express a potentially infinite number of concerns.

III. Advantages

The easy definition and implementation of small dedicated languages has been shown to bring a lot of advantages. Each such language is exactly adapted to the task at hand and the programmer does not have to learn useless features. To quote [Brooks, 1987]:

“Moreover, at some point the elaboration of a high-level language creates a tool-mastery burden that increases, not reduces, the intellectual task of the user who rarely uses the esoteric constructs.”

The program design, debug, document and other tasks may be achieved at a higher level of abstraction, not polluted by technical implementation details. Later this program may be transformed into concrete executable statements.

IV. Drawbacks

The rapid proliferation of DSLs of all sorts may rapidly bring chaos to the software world. Having thousand of languages invented daily for different purposes poses a new problem that had never been addressed. How to deal with the resulting fragmentation? Will the medicine be worse than the illness? What should be the policy of big organizations that will see their business units and even there individual engineers express themselves in a myriad of uncontrollable locally defined formalisms?

V. Lessons learnt from Brooks

Although Brooks did not specifically mention DSLs in his 1987 article, there are plenty of lessons that can be directly applied to the present situation. Each new technology (objects, components, aspects, etc.) brings

at the same time improvement and regression. Objects have for example partially helped to improve reusability and to structure the globe software organization but made the portability problem worse. The DSL technology is an obvious candidate to silver bullet status. The distinction between potential advantages and drawbacks, - between accidental and essential complexity - in the deployment of DSLs has rarely been observed in software technology evolution. If we want the contribution of DSLs to be globally positive, we need to show how the technology itself may help to harness its potential drawbacks and solve the generated problems. The idea here is to express the relations and interactions between various languages by other languages. In a move from modeling in the small to modeling in the large, the focus should be on expressing all relations of transformation, extension, composition and more by new coordination DSLs. In this way it should be possible to limit the accidental negative contributions of this technology and keep most of its positive essential contributions on the quality of software systems. Following the general framework sketched by Brook will be of significant help here.