Modelling By the People, For the People

Steven Kelly (http://orcid.org/0000-0003-0931-157X)

MetaCase, Jyväskylä, Finland
stevek@metacase.com

Abstract. Domain-specific modelling has moved the focus of model content from the implementation domain to the problem domain. However, much research still sees most modellers as programmers – a tacit assumption that is exemplified by the use of IDEs as modelling environments. MDE research should instead be reaching out to make itself presentable to subject matter experts – as language creators, users, and research subjects. Every leap in developer numbers has been triggered by a new format, and we need another such leap now. Domain-specific MDE is ideally placed to step up and enable the creation of applications by people of all backgrounds.

Keywords. domain-specific modeling, productivity, programmer demographics

Introduction. Domain-specific modelling has moved the focus of model content from the implementation domain to the problem domain. However, much MDE research still sees most modellers as programmers – a tacit assumption that is exemplified by the use of IDEs as modelling environments, diagrams bearing a striking resemblance to UML, and structures straitjacketed into the hierarchical tree of XML. The false assumption seems to be "because I must build the language workbench in an IDE, experts must create their languages in that IDE, and modellers must model in the IDE". MDE research is stuck in a programmer mindset and an IDE from the last decade, when it should be reaching out to make itself presentable to subject matter experts, both as language creators and users.

Background. Every leap in developer numbers has been triggered by a new format, either in languages (machine code to assembly language to 3GLs) or devices (mainframes to PCs to mobile). 'Language leaps' came from research making software development easier, so that more people are able to successfully create applications. 'Device leaps' are different: rather than increasing the supply and decreasing the cost, they increased the demand. Language leaps have increased developer numbers much more than devices leaps: an order of magnitude as opposed to a 'mere' doubling.

Interestingly, the increase in the number of developers in the language leaps seems to mirror the increase in productivity of a given average developer, e.g. roughly five times faster with 3GLs compared to assembly language. Since the early 3GLs, there has not been an appreciable leap in productivity, so it seems fair to assume that we have made only incremental progress in making software development accessible to people for whom it was previously too hard. Learning and working in JavaScript is not much easier than learning and working in COBOL. Contrast that with how much easier either is than working directly in machine code.

Libraries, Frameworks and Languages. Components, libraries and frameworks are sometimes touted as offering a decisive advantage. A pre-existing component or library may save a company's time, but the application code still needs writing, and the productivity for that task is unchanged. A framework can make things easier for the programmer, letting him 'fill in the blanks' of its ready-built behaviour. However, the content entered 'in the blanks' is still largely 3GL code, so a framework alone will not make application creation available to a larger audience.

A domain-specific language, on the other hand, helps both in increasing productivity and in reducing the demands on the developer. The 'blanks' are filled in with domain concepts, not code. By abstracting away from the implementation technology, a DSL also often makes it possible to generate the same application for different formats: web server and client, desktop app, and mobile app.

A graphical or projectional DSL in particular generally constrains you to only be able to create legal models, as opposed to the illegal source code you could write while trying to use a framework. Among programmers there are clearly a disproportionate number who prefer text over graphics, compared to the population at large; conversely, a graphical DSL is thus often a better choice when reaching out to current non-programmers.

Conclusion. Good domain-specific modelling languages in non-IDE tooling have consistently shown productivity improvements of a factor of 5–10. Given the earlier link between productivity increase and increase in the number of people who can successfully develop software, there would seem to be a good possibility that a DSM approach could make application creation possible for an order of magnitude more people. Our own experience with MetaEdit+ tends to confirm this: we have clients where none of the modellers are programmers, clients where all are, and clients where there is a mix. The future could well reach the ideal of a balanced mix of good programmers and smart subject matter experts, all collaborating directly in the same set of models. MDE is ideally placed to step up and make possible the creation of applications of all formats, by people of all backgrounds.

Since the 1990s, much modelling research has focused on creating tools and languages, but with declining returns in terms of novelty, incremental benefit over current industrial use, and adoption. For tools, researchers need to throw off their "not invented here" attitudes, and reach out to investigate real industrial use of non-IDE based commercial domain-specific modelling tools, both language workbenches and fixed tools like Simulink and LabVIEW. For languages, rarely does a researcher have the domain knowledge and experience to create a good DSM language for a company. Rather, they should concentrate on being a catalyst, enabling a company to create its own language, studying that process, and incrementally improving it as they learn over several cases. In this area, more research is definitely needed and will be fruitful.