The MPS Language Workbench
As described in [1]: MPS (since 2003, http://www.jetbrains.com/mps/) is an open-source language workbench developed by JetBrains. Its most distinguishing feature is a projectional editor that supports integrated textual, symbolic, and tabular notations, as well as wide-ranging support for composition and extension of languages and editors. MPS realizes the language-oriented programming paradigm introduced by Sergey Dmitriev [2] and has evolved into a mature and well-documented tool. It is used by JetBrains internally to develop various web-based tools such as the Youtrack bugtracker. It has also been used to develop various systems outside of JetBrains, the biggest one probably being the mbeddr tool for embedded software development.

Questionnaire Language base assignment
The following were our main design decisions for the LWC 2013 assignment:
• The LWC 2013 assignment left the choice for a runtime open, so we chose to use Java as a runtime target, because it is supported out-of-the-box in MPS in the form of BaseLanguage (Java modeled in MPS).
• The runtime (basically manually written Java code, used as a library by the QL generator) has been created in BaseLanguage (which model-to-text transforms directly to java code).
• QL has been designed as a standalone DSL. Models created in QL are model-to-model transformed to API calls to the runtime in BaseLanguage. QLS (QL Styling) has been created as a language extension to QL.
• The variability specified by QLS is realized by generating a specific version of the runtime that is used by the code generated from QL, e.g. a yes/no question (possibly to specify in QLS as either a checkbox or a radio button) will be created by a call to a createYesNoQuestion function, which can be implemented differently, dependent on what is specified in a model in the QLS language.

MPS scalability and teamwork features

Scalability in terms of model size
• MPS allows for creation of stubs in order to treat certain pieces of a system separately.
• mbeddr (one of the more complex language collections created in MPS) has worked in real-life projects with hundreds of thousands of lines of code (however this is quite hard to map to number of model elements, as one line of code can be an expression that has dozens of model elements). This is discussed in [3]

Scalability in terms of teamwork
• MPS allows for diff and merge on editor-level, so it is possible to see a diff in the IDE as if in a plain-text diff and perform merges similar to plain-text merges in the IDE. This makes diff and merge very usable in concurrent development.
• MPS provides out-of-the-box connection to popular version control backends, like git, svn, and cvs. In the case of git, even a merge-driver is provided out-of-the-box.
• MPS provides facilities to run builds and tests (including generation) non-interactively from the command-line, which makes it possible to use in automated systems, such as continuous integration, continuous deployment, or automatic lifecycle management.
• MPS allows for self-documented model elements:
  ◦ metaclasses (or concepts) are self-documented
  ◦ AST annotations allow implementation of non-intrusive documentation and meta-information (such as requirements traceability)

Scalability in terms of model size and teamwork
• MPS allows for cross-referencing between models. This provides scalability in both directions:
  ◦ models can be split into configuration items that can be edited concurrently.
  ◦ large models don't need to be monolithic but can rather be split up in smaller models.
• MPS encourages the creation of modular languages and provides orthogonality (language extension and composition). This provides scalability in both directions again:
  ◦ teams can develop languages concurrently, because they each language can be created separately and then languages can be combined.
  ◦ because the metamodeling can be done in very orthogonal way, there is a strong separation of concerns; one is able to focus only on the aspects of one specific metaclass (or concept) and the system takes care of composition.

Demo description
The rough agenda for our demonstration at the LWC would be (showing some of the most interesting features; each item will take about 2 minutes to present):
  1. Overview of QL/QLS design, including showing the modularity of aspects on metaclasses (or concepts)
  2. Show a very big questionnaire model and performance in editor and generator
  3. Show a diff/merge
  4. Show some basic versioning scenarios, including resolving a conflict
  5. Show a build in MPS from the console
  6. Show some self-documented (meta)model items
  7. Show cross-referencing between models

References
