LWC11 - The Task

Version 1.0, July 25, 2010

Markus Völter et. al.

Abstract

Over the past few years, and actually the past year, a lot of new initiatives have surfaced in the area of creating so called language workbenches - aiming at facilitating the definition and use of DSLs and code generation.

We believe each of these has its own strengths and weaknesses, and none is 'the best' for every purpose. Still, a lot of people keep asking for the best workbench. Based on that, we are now planning to have a Language Workbench Comparison, in which we will be able to compare the strengths and weaknesses of these workbenches based on solutions for a predefined set of cases.

The idea for LWC11 originates from a group of people who met in the preparations for, and during CodeGeneration 2010: Markus Völter, Eelco Visser, Steven Kelly, Angelo Hulshout, Jos Warmer, Pedro J. Molina, Bernhard Merkle and Karsten Thoms.

Comparison Points

The task described below used for comparing language workbenches. We have tried to come up with a task that is as simple as possible, and still illustrates the core features of language workbenches, as described here:

Phase 0 - Basics

This phase is intended to demonstrate basic language design, including IDE support (code completion, syntax coloring, outlines, etc).

- 0.1 Simple (structural) DSL without any fancy expression language or such.
- 0.2 Code generation to GPL such as Java, C#, C++ or XML
- 0.3 Simple constraint checks such as name-uniqueness
- 0.4 Show how to break down a (large) model into several parts, while still cross-referencing between the parts

Phase 1 - Advanced

This phase demonstrates advanced features not necessarily available to the same extent in every LWB.

- 1.1 Show the integration of several languages
- 1.2 Demonstrate how to implement runtime type systems
- 1.3 Show how to do a model-to-model transformation
- 1.4 Some kind of visibility/namespaces/scoping for references
- 1.5 Integrating manually written code (again in Java, C# or C++)
- 1.6 Multiple generators

Phase 2 - Non-Functional

Phase 2 is intended to show a couple of non-functional properties of the LWB. The task outlined below does not elaborate on how to do this.

- 2.1 How to evolve the DSL without breaking existing models
- 2.2 How to work with the models efficiently in the team
- 2.3 Demonstrate Scalability of the tools

Phase 3 - Freestyle

Every LWB has its own special "cool features". In phase three we want the participants to show off these features. Please make sure, though, that the features are built on top of the task described below, if possible.

The Task

This section describes the task that is used to compare the language workbenches. The description is structured as follows: In black color we repeat the goal outlines above. In red we explain the actual task. In blue we provide examples to illustrate the task. Note that you are NOT required to use exactly the same syntax etc. as the blue example suggests (this is especially true if you use a graphical LWB ☺). It is merely intended as an example to clarify the task itself.

Phase 0 - Basics

- 0.1 Simple (structural) DSL without any fancy expression language or such.
  Build a simple data definition language to define entities with properties. Properties have a name and a type. It should be possible to use primitive types for properties, as well as other Entities.

```
entity Person  {
   string name
   string firstname
   date bithdate
   Car ownedCar
}

entity Car {
   string make
   string model
}
```

- 0.2 Code generation to GPL such as Java, C#, C++ or XML Generate Java Beans (or some equivalent data structure in C#, Scala, etc.) with setters, getters and fields for the properties.

- 0.3 Simple constraint checks such as name-uniqueness For example, check the name uniqueness of the properties in the entities.

- 0.4 Show how to break down a (large) model into several parts, while still cross-referencing between the parts. For example, put the Car and Person entities into different files, while still having the Person -> Car reference work.

Phase 1 - Advanced

This phase demonstrates advanced features not necessarily available to the same extent in every LWB.

- 1.1 Show the integration of several languages. Define a second language to define instances of the entities, including assignment of values to the properties. This should ideally really be a second language that integrates with the first one, not just "more syntax" in the same grammar. We want to showcase language modularity and composition here.

```
Person p = {
   name = "Voelter"
   firstname = "Markus"
   bithdate = 14.02.1927
   ownedCar = c
}

Car c = {
   make = "VW"
   model = "Touran"
}
```

- 1.2 Demonstrate how to implement runtime type systems. The initialization values in the instance-DSL must be of the same type as the types of the properties.
- 1.3 Show how to do a model-to-model transformation. Define an ER-meta model (Database, Table, Column) and transform the entity model into an instance of this ER meta model.
- 1.4 Some kind of visibility/namespaces/scoping for references. Integrate namespaces/packages into the entity DSL

```
package p1 {
       entity Person  {
              string name
              string firstname
              date bithdate
              Car ownedCar
       }
}

package p2 {
       import p1
       entity Car {
          string make
          string model
       }
}
```

and make sure in the instance DSL you can only assign values to the properties of the respective entity, i.e. make sure that writing

```
Car c = {
   birthdate = …
}
```

is illegal.

- 1.5 Integrating manually written code (again in Java, C# or C++). Integrate derived attributes to entities.

```
entity Person  {
   string name
       string firstname
       date bithdate
```

```
        Car ownedCar
        derived int age // somehow somewhere implement the code
                        // to calculate age
}
```

Note that if you want, you can also define or reuse an expression language that allows defining the algorithm for calculating the age directly in the model. Ideally, you will show both (manually written 3GL code as well as an expression language).

- 1.6 Multiple generators. Generate some kind of XML structure from the entity model.

Documenting your entry

The main point of LWC11 is to be able to explain and compare how languages and IDEs are implemented using the various available tools. It is hence important to document your approach. Here are some things to keep in mind:

- Make sure that the documentation is structured along the same phases (0-3) and tasks (0.1, 1.4, 2.4, etc.) so it is easy for people to find stuff in your documentation.
- You can either write a document or create PowerPoint slides. In any case, you must submit the documentation as PDF.
- The documentation should be accompanied by a ZIP file that contains the code for your solution, licensed in a way so that everybody can play with it. You should also clearly specify which version of the LWB was used to create the example, and where people can get it (at least an eval version).
- Remember that you don't just want to show how powerful your LWB is, you also want to convince potential users how simple and accessible it is. So make sure your documentation is understandable!