Goals

LWC2014 will be held April 8 2014, before Code Generation 2014. The main focus will be to compare Language Workbenches (LWBs) on the following points:

- Creating a language and execution engine (generator, interpreter) for a given case
- Showing how the LWB supports multiple (many) engineers working on the same models
- Showing how the LWB deals with large models

All solutions must be based on a language workbench, i.e. a tool developed specifically to develop, maintain (and apply) (domain-specific) modeling languages. DSLs based on reflection or meta programming or just “clever APIs” are out of scope.

This Call for Participation describes the details of the assignment for the workshop, and requirements for participation.

Assignment

The assignment for the LWC 2014 edition consists of two parts: a base part, in which two languages need to be developed, and a focus assignment, in which these two languages are used to show how the participating language workbenches solve scalability issues related to team size and model size.

Base assignment

The base of the assignment is the Questionnaire Language and Questionnaire Style Language we used in LWC 2013. This part of the assignment is included because it shows the core functionality of a language workbench. Participants should implement this assignment, keeping in mind a number of constraints:

See Appendix A at the end of this CfP for the details of the base assignment.

- The solution must contain a full language implementation: language definition, IDE support, interpreter or generator.
- The language must implement the concepts presented in the assignment (both QL and QLS).

- By November 29th, the organizers will provide a reference implementation of QL and QLS that runs on a target platform of HTML5 + CSS + JavaScript. Participants are encouraged to comply to this reference implementation, or, if they choose another target technology, provide the same functionality and a similar look and feel to the reference implementation.

LWC2014 focus assignment

The focus of LWC2014 is on demonstrating scalability in two directions:

- People working concurrently on a single model and/or language definition ()
- The size of models, in terms of elements (i.e. large questionnaires, or complex layouts in QLS)

This part of the assignment in included because using LWBs in practice will imply that the models and meta-models created are accessed by multiple team members, and that models will grow significantly larger than the average demo model. This will have an impact on the features a language workbench provides to support practical work (version control, model integration, storage and access facilities, …)

By November 29th, the organizers will provide examples of concrete tasks that submitters may use to demonstrate team support and model size scalability.

Selection process and criteria for participation

In contrast to previous instances of LWC, LWC 2014 requires participants to submit a short report well before the workshop, containing the following four items:

- A short *technical and functional* description (<0.5 pages) of the used LWB. Note that participating tools should be language workbenches, that is, tools that are designed specifically for efficiently defining, integrating and using domain specific languages in an Integrated Development Environment (IDE). Embedded DSLs, as well as fixed-language or library-based solutions do not meet this criterion.
- A short description (<0.5 pages) of how the base assignment of building QL and QLS was addressed.
- A short *technical and functional* description (<0.5 pages) of how the LWB handles scalability and teamwork.

- A short *technical and functional* description (<0.5 pages) of how the LWB and its handling of scalability and teamwork will be demonstrated in 15 minutes at the workshop.

The reports will be reviewed by the program committee, and only authors of accepted reports will be invited to present their working solution at the LWC workshop on April 8th.

Important dates

- Reference implementation and scalability examples provided by PC: **November 29, 2013**
- Report deadline: **January 3rd** (e-mail reports to submission@languageworkbenches.net)
- Workshop invitation of accepted submitters: **January 24, 2014**
- Workshop: **April 8, 2014**

Appendix A - details of the base assignment

The assignment: Questionnaire Language (QL)

Forms-based software for data collection has found application in various areas, including scientific surveys, online course-ware and guidance material to support the auditing process. As an overall term for this kind of software applications we use the term "questionnaire". In this LWC'13 assignment the goal is to create a simple DSL, called QL, for describing questionnaires. Such questionnaires are characterized by conditional entry fields and (spreadsheet-like) dependency-directed computation.

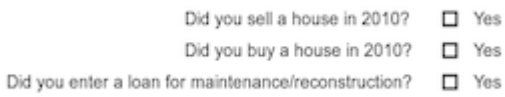
Example

The following example presents a possible textual representation of a simple questionnaire.

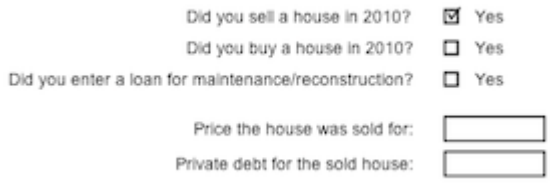
```
form Box1HouseOwning {
   hasSoldHouse: "Did you sell a house in 2010?" boolean
   hasBoughtHouse: "Did you by a house in 2010?" boolean
   hasMaintLoan: "Did you enter a loan for maintenance/reconstruction?" boolean
   if (hasSoldHouse) {
      sellingPrice: "Price the house was sold for:" money
      privateDebt: "Private debts for the sold house:" money
      valueResidue: "Value residue:" money(sellingPrice - privateDebt)
   }
}
```

This simple form should generate into a GUI which allows the following user interaction:

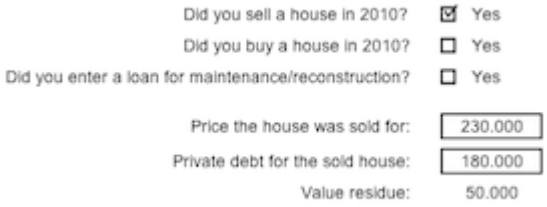
Step 1



Step 2



Step 3



The key things to observe:

- Questions become available in the GUI as soon as their enabling conditions are satisfied.
- Default styling and widgets is implied. For instance, boolean questions produce checkboxes, numeric fields are rendered as text-boxes, computed values are read-only.

Elements of QL

The snippet of fictional QL above is textual only to provide an example. QL should be easy to express as a graphical language as well. The main ingredients of QL can be summarized as follows.

Syntax

QL consists of questions grouped in a top-level form construct. First, each question identified by a name that at the same time represents the result of the question. In other words the name of a question is also the variable that holds the answer. Second, a question has a label that contains the actual question text presented to the user. (Note that technically this is a presentation issue that could be in a separate language for layout and styling, but to make QL standalone we need it

here. See below for more on the layout language.) Third, every question has a type. Finally, a question can optionally be associated to an expression: this makes the question computed.

A questionnaire consists of a number of questions arranged in sequential and conditional structures, and grouping constructs. Sequential composition prescribes the order of presentation. Conditional structures associate an enabling condition to a question, in which case the question should only be presented to the user if and when the condition becomes true. The expression language used in conditions is the same as the expressions used in computed questions. Grouping does not have any semantics except to associate a single condition to multiple questions at once.

For expressions we restrict ourselves to booleans (e.g., &&, || and !), comparisons (<, >, >=, <=, != and ==) and basic arithmetic (+, -, * and /). The required types are: boolean, string, integer, date and decimal and money/currency. NB: this set could be extended with other data types, such as an enumeration data type (e.g., "good", "bad" and "don't know"), or integer range (e.g. 1..5). The only requirement is that the data type can be automatically mapped to a widget.

Notes:

- If possible, the expression language for conditions and computed values should be reused.

Semantics

The output of a QL description should be a simple GUI program that shows questions as soon as they become enabled. (So the initial view should consist of all questions that do not have a enabling condition). The user should be able to fill in answers, to which the system responds with more questions to be filled in and/or with the display of additional computed results. After all questions have been filled in --- the fixed point has been reached --- the result of the complete questionnaire should be saved somehow (e.g., as XML, YAML, JSON, etc., or in a database).

The semantics of expressions and question variables is standard, except that an unbound variable (= unanswered question) has the special value`undefined`. Any expression referring to an undefined variable is undefined itself. In conditional context, `undefined` means `false`.

Rendering a QL questionnaire as a GUI should make sensible defaults for aspects of presentation. For instance, the widget used for a question can be derived from the type: checkboxes/radiobuttons for booleans, text fields for strings, spinlocks for integers etc.

Optional: layout and styling (QLS)

The base version of QL is rendered using default heuristics for the presentation aspect of a questionnaire. In this optional assignment the goal is to create a separate layout and styling language. This language can be used to group related questions in pages (with navigation links), sections and subsections, select specific font styles and colors for labels, and choose widgets other than the default choice. For instance, a boolean can be represented by a check box, two radio-buttons, a drop-down list etc. A (bounded) integer could be represented either as a text-field or as a slider. Dates could map to a date picker widget. Etc.

The QLS language *references* the QL language: the names of questions are used to identify the points where to apply a particular styling. This should be non-invasive: QLS styling is applied where needed, the rest of a QL program is rendered as if no QLS existed. In other words: the QLS language is optional for the assignment, but also optional for use.

Challenges

- Modularly integrate QLS into QL; the QL language implementation should be independent of QLS.
- Check QLS *against* QL; check that only defined questions are referenced and make sure that the presentation choices made in QLS are compatible with the type of questions in the QL program.

