Graphical User Interface

for two-level transformation

Diogo Paulo da Fonte Lapa (38164)

Departamento de Informática da Universidade do Minho
Campus de Gualtar - Braga - Portugal

March 2, 2007

Contents

1 Preface

1.1 About the Documentation

The intended purpose of the documentation is to explain the main features of *2lt-gui* and to highlight features that may not be obvious to first time users. The documentation is not exhaustive.

1.2 Philosophy of *2lt-gui*

A data type can be represented by a couple of a type-level declaration and an instance of that type in the value-level. Changes in data types call for corresponding changes in data values.

A two-level data transformation consists of a type-level transformation of a data format coupled with value-level transformations of data instances corresponding to that format.

2lt-gui is a tool to perform two-level data transformations. The basic features are:

- Import of a data format specified in *XML* Schema, *SQL* and *Haskell*
- Specification of rules to apply to the format, by selecting from a palette of rules and rule combinators, or by writing a high-level script
- Running the rules
- Inspecting the result
- Apply the resulting data conversion function to data instances, either in backward or forward fashion
- Export data format and values to *XML*, *SQL* and *Haskell*

1.3 Terminology

Below is a table of terms that might be useful.

Term	Explanation
Rules	Combination of rules to apply to source data format
View	Expresses that a *Type a* (source) can be represented by a *Type b* (target), as witnessed by value-level functions that convert between these types
Source	Data format to transform
Target	Data format created from transformation
Result	Data instance resulting from data conversion
Generic data	Internal data representation used to calculate transformations

2 Layout

2lt-gui layout is based on a notebook. There are three different tabs: *Rules*, *View* and *Result*.

2.1 Rules

You can have only one tab of this kind. In this tab you can see the selected rules. To insert new rules use the *Rules* menu. For more information read section 3.2.

2.2 View

This is the main tab where you can see the source and the target. This is also the tab where you can apply data conversion functions to data instances. For more information read section 3.3. To add a new View tab go to $File \rightarrow New$.

2.3 Result

This is the resulting tab when you apply some data conversion function. This tab shows the resulting data instance from the conversion. Each time you apply a function, a new tab is created. In addition, this tab offers you the possibility to export the resulting data instance to one of the three formats: *XML*, *SQL* and *Haskell*.

3 How to use

3.1 Importing data format to source

When you import a data format it is parsed to a generic type representation. You can import a data format from three different formats:

- ***XML* Schema:** $Menu \rightarrow Type \rightarrow Import\ XSD$
- ***SQL*:** $Menu \rightarrow Type \rightarrow Import\ SQL$
- ***Haskell*:** $Menu \rightarrow Type \rightarrow Import\ Haskell$

The generic type representation is saved in the source. The source is used to derive the target type. For more detailed information of how to do this read section 3.2.

3.2 Selecting and applying rules

The rules define how you want to transform the source type.

You can set the rules either from selecting from a palette (*Rules* menu) or importing from a script ($Menu \rightarrow Import\ from\ script$). For more information about scripts see section **??**

3.3 Applying resulting data conversion functions

It's possible to transform data instances in forward and backward directions. Forward transforms source data instances to target data instances. Backward does the reverse way.

Again, you can read data instances from the three different formats. To use go to $Values \rightarrow Transform \ldots$

3.4 Exporting data format and instance

Finally, it's possible to export the result. Select the result tab and go to $Values \rightarrow Export \ldots$. If you want to export the result data type (target) you have to go to $Type \rightarrow Export \ldots$.

4 Rules Script

A rules script is a file with a high-level script defining a set of rules. All the rules in the script are defined in the *Two-level Transformation* Library. It is not possible to create new rules, only combinations of existing rules.

The syntax of the scripts is very similar to the *Haskell* source. However there are some differences:

- To represent strings it's possible to use either " or ' to encapsulate them.
- Nested strings are represented like in some script languages, you have to change between one notation or another (See script 2 in the examples).
- Rules receiving *Type* as argument, represent it as a string.