ACTA UNIVERSITATIS AGRICULTURAE ET SILVICULTURAE MENDELIANAE BRUNENSIS

Volume 62 33 Number 2, 2014
http://dx.doi.org/10.11118/actaun201462020315

UNIVERSAL STYLE SHEET LANGUAGE ENVIRONMENT MODIFICATION FOR THE BUSINESS USE

Jiří Brázdil[1], Jiří Šťastný[1]

[1] Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic

Abstract

BRÁZDIL JIŘÍ, ŠŤASTNÝ JIŘÍ. 2014. Universal Style Sheet Language Environment Modification for the Business Use. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis,* 62(2): 315–320.

This paper deals with the description of USSL – Universal Style Sheet Language environment. USSL style sheet language is platform-independent and its primary focus is the declarative notation of the appearance of GUI libraries used by imperative programming languages. The implementation of the software support for wxWidgets library is made, because this library has no support for the separate declarative notation of the appearance via style sheet language. The separation of the appearance enables us to reuse and standardize the appearance notation and the independent development of the appearance. In this way it is possible to achieve consistent appearance of applications of specific set or even all of company software products. However, the first proposal of the USSL has several disadvantages which restrict the possibilities for practical use in business or other environment. These disadvantages are: the lack of @import rule for importing other style sheets, USSL only supports basic set of selectors compared with selectors of other style sheet languages for desktop environment such as Qt QSS and GTK+ GtkCssProvider, the lack of styling of the cursors, it is impossible to put down URL. The placement of widgets and its borders are not solved either. This paper contains suggestions for solving these issues.

Keywords: CSS, declarative approach, styles, wxWidgets

INTRODUCTION

The separation of appearance and presentation interface of GUI enables us to recycle appearance, to standardize the notation and independent development of appearance. It is possible for example in software development to achieve consistent appearance of applications of one group, or even all of company's products – from specialized applications to for example to office suite software.

It is also possible changing the goal platform – GUI library – to approach the original appearance, because the appearance is written down separately in multi-platform language. If the style sheet contains platform-dependent properties, the additional changes are required.

USSL style sheet language is platform-independent, although it is possible to write down specific properties for specific platforms, its primary focus is to write down appearance for GUI libraries used by imperative programming languages in declarative manner. Application to wxWidgets library is made, because this library has no support for declarative notation of the appearance using style sheet language, and design of this style sheet language is also inspired by Qt QSS and GtkCssProvider, which are style sheet languages for specific libraries, and other similar projects and possibilities of appearance notation of chosen libraries, which don't support separate declarative notation of appearance.

The list of issues of first design is described by Brázdil (2012a):

- the @import rule for importing other style sheets,
- the property, state and sub-control selectors,
- the cursors styling using properties for specific libraries,
- the URL notation,

- the inheritance in the value propagation mechanism.

The paper also introduces the original design of language and suggests the solution of issues.

MATERIALS AND METHODS

The design of language for appearance notation must accomplish these requirements:

1) It supports appearance notation for multiple libraries for GUI creation, and also covers the possibility of defining the platform-specific properties,
2) it is human readable,
3) its syntax is based on an existing language,
4) it has similar possibilities of selection elements like CSS selectors.

Based on discussion on these requirements, the syntax of CSS language was chosen. XML language syntax was abandoned because of its lesser readability, CSS syntax was found good readable according to Khedker (1997), who also analyzes the readability of programming languages.

Lie (2005) states that a style sheet language has six required components:

1) syntax,
2) selectors,
3) properties,
4) values and units,
5) value propagation mechanism,
6) formatting model.

The item 1) – the syntax is based on CSS, but this language, as mentioned by Brázdil (2010), has its own disadvantages:

- it is not possible to address parent elements,
- it cannot evaluate expressions,
- does not provide notation of constants nor variables.

To solve the two last disadvantages, it was proposed to use the enhanced syntax of the Sass[1] Language. The original design does not include the `@import` rule for importing other style sheets.

The support of the item 2) – selectors – is solved with the intersection of supported selectors of QSS[2] and GtkCssProvider[3]. These are universal, type, class, ID, child and descendant selectors.

The properties – item 3) – the notation of their names comes through the following normalization as it is mentioned by Brázdil (2012a):

- the capital letters are converted to lower-case letters,
- undersores, spaces and others are converted to hyphens (–),
- so called Camel case notation, e. g. `fontFamily` is converted as the capital letters are converted to lower-case letters and between words the hyphens are inserted, i. e. `font-family`,
- the prefixes of values, that result from the name of the library – `wx-`, `q-` etc., e. g. `wxNO_BORDER` to `no-border`, eventually the name of the type `wxFONTFAMILY_MODERN` to `modern`, are removed,
- the exceptions exist, e. g. normalization of wxWidgets to wxwidgets.

Also there is support for notation for specific properties for certain libraries using the `x-` prefix.

The item 4), the notation of values and units, is solved as follows: it is possible to notate both the integer number and decimal number, values of properties are normalized as it was mentioned above, the notation of logical values is realized by using the keywords `true` and `false`. The font is possible to notate using full or shorthand notation, the notation of cursor properties hasn't been supported yet.

The color notation is possible using the following alternatives, see Brázdil (2012a):

- (…) in the RGB model using six hexadecimal digits, (`#ffff00`),
- `rgb(r, g, b)` and `rgba(r, g, b, a)` notation,
- symbolic color notation using names (…).

Only pt units are supported, comment is the same as in CSS, i. e. multiline comment delimited with `/*` and `*/`, URL notation is not supported. The notation of variables and expressions comes from Sass language, see Sass Documentation (2012).

The value propagation mechanism is realized via default values and cascading, the addition of properties inheritance is planned in the future. Visual formatting model is supported, display is dependent on the goal platform.

The implementation of tools for handling this language for wxWidgets library is made. The implementation is described in Brázdil (2012a) in detail.

The first solution comes from enhancing wxWidgets classes. In C++ programming language a simplified lexical and syntactical analyzer of CSS-like language is implemented, the class which handles the functionality of cascading, the class for data types conversion to specific wxWidgets data types and enhanced wxWidgets classes. The enhanced classes are created as follows: for every widget class, which will have the support of reading declarative notation of appearance, the new class is created with this ability. The conversion class is required because it enables us to enhance the solution that can support other platforms in the future, e. g. create conversion filter, which converts the USSL style sheet to GtkCssProvider or QSS style sheet. The suggestion of such conversion is described in Brázdil (2012b).

1 http://sass-lang.com/
2 http://qt-project.org/doc/qt-4.8/stylesheet-syntax.html
3 http://developer.gnome.org/gtk3/3.0/GtkCssProvider.html

The second solution uses XML files with appearance specification. wxWidgets uses XRC format, GTK+ and Qt libraries use similar formats. The appearance and the structure of the GUI are specified in the XML file, it is possible to edit this file end rewrite properties, which are responsible for appearance, by others, which are loaded from external style sheet. The original implementation supports only the adjustment of XRC format for wxWidgets.

These solutions have several implementation issues: lack of inheritance, lack of selectors class, child and descendant, only pt unit of font size is supported, type selector acts like class selector, incomplete support of color notation and at last this style sheet language is not fully compatible with Sass language, or more precisely it's SCSS syntax variant.

The issues solution is suggested as follows:

The @import rule can be simply added to the solution and implemented the resolution of rule's conflicts.

The solution of styling of the cursor in the wxWidgets and other libraries is connected with the notation URI[4], eventually URL[5]. In CSS the syntax of the notation of URI according to BOS (2011) is following: The format of a URI value is 'url(' followed by optional white space followed by an optional single quote (') or double quote (") character followed by the URI itself, followed by an optional single quote (') or double quote (") character followed by optional white space followed by ')'. The two quote characters must be the same.

BOS (2011) also mentions the following example:

```
body{background: url("http://www.
example.com/pinkish.png")}
```

The support of inheritance of properties is a problem of implementation, it is connected to support of type, class, descendant and child selectors. Other selectors, which are not mentioned in the original design are property, state and sub-controls selectors.

It is possible to think about an addition of the state selector to the existing set of supported selectors, e. g. by calling the function `bool IsFrozen () const` of class `wxWindow` it is possible to find out the state of the widget, the example on state selector rewritten to style sheet language:

```
wxWindow:isFrozen { background-color:
rgb(255, 0, 0); }
```

The property selector is used e. g. in QSS and CSS, in QSS, it is related to a specific system of properties of Qt widgets. In wxWidget it would be necessary to implement such a system, or use the same mechanism as in the state selector, whereas it would be necessary to distinguish between the state of the widget and its properties, e. g. by calling of the function `bool HasScrollbar (int orient) const` of wxWidnow class is possible to find out if the window contains scrollbar with a certain orientation. The example rewritten to style sheet language follows:

```
wxWindow[HasHScrollbar=true]
{ background-color: rgb(255, 0, 0); }
```

The selectors of properties and states can be joined together and it is possible to syntactically use only the state selector, which is better from the universal usage point of view.

```
wxWindow:hasHScrollbar { background-
color: rgb(255, 0, 0); }
```

By using the selector of sub-controls it is possible to define a style only for a predefined part of the widget. The example for this is the button on the first page in the object of wxNotebook class:

```
wxNotebook::first-page wxButton
{ x-wxwidgets-window-style: no-border; }
```

What was not part of the original design, was the setting of the placement and the size of the widgets and their borders.

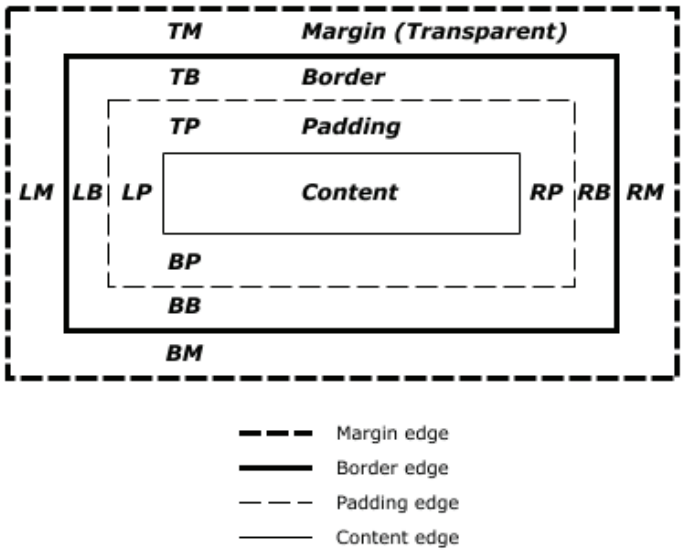
Bos (2011) states that the CSS box model describes the rectangular boxes that are generated for elements in the document tree and laid out according to the visual formatting model. (…) Each box has a content area (e.g., text, an image, etc.) and optional surrounding padding, border, and margin areas; the size of each area is specified by properties defined below.

The terminology is illustrated in Fig. 1: Box model. The meaning of abbreviations in the picture is following: the first letter stands for top, right, bottom, and left and the second letter is for padding, border, and margin.

wxWidgets library enables the display of single, double, raised, sunken suitable for static controls, themed or no borders around the widget. Some types are supported only on the certain platforms, see Smart (2012).

On the contrary, the Qt library supports the appearance notation using CSS-like language – QSS. According to Qt Project Hosting (2011), it is possible to use border, margin and padding properties in certain widgets. Similar possibilities are available within declarative notation of appearance within GTK+ library, The GNOME Project (2012) contains complete GtkCssProvider documentation.

4 http://en.wikipedia.org/wiki/Uniform_resource_identifier
5 http://en.wikipedia.org/wiki/Uniform_resource_locator



1: *Box model, source: Bos (2011)*

It is possible to add support for the border property to wxWidgets, other properties – margin and padding – are discussed bellow.

When we refer to the size of a window, we normally include the outer dimensions, including decorations such as the border and title bar. When we refer to the size of the client area of a window, we mean the area inside the window that can be drawn upon or into which child windows may be placed. A frame's client area excludes any space taken by the menu bar, status bar, and toolbar. (…) The coordinate system always has (0, 0) at the top-left corner, and window dimensions are in pixels (Smart, 2006).

Applied to the position and size of the widget, it is necessary to distinguish between an absolute and a relative position and using so-called layout manager.

The absolute positioning of the widgets within some container (window) appears not very well, if the user changes the size of the container. Furthermore, like Smart, (2006) mentions (…) unlike with print layout, an application's windows must often dynamically adapt to changes in size, font preferences, and even language. For platform-independent programming, the layout must also take into account the different sizes of individual controls from one platform to the next.

In the wxWidgets library it is possible to use absolute positioning of the widgets or to use so-called sizers.

The absolute positioning of the widget can be written down like this:

```
#button { x: 20; y: 30; }
```

In the C++ programming language the notation is as follows:

```
wxButton *button = new wxButton(panel,
ID_ADD_RECORD,    wxT("&Nový    záznam"),
wxPoint(20, 30));
```

By using sizers in the process of positioning the particular descendant of wxSizer class is chosen – wxBoxSizer, wxStaticBoxSizer, wxGridSizer wxFlexGridSizer or wxGridBagSizer.

The layout algorithm used by sizers in wxWidgets is closely related to layout systems in other GUI toolkits, such as Java's AWT, the GTK+ toolkit, or the Qt toolkit. It is based upon the idea of individual windows reporting their minimal required size and their ability to be stretched if the size of the parent window has changed. This will most often mean that the programmer does not set the initial size of a dialog; instead, the dialog will be assigned a sizer, which will be queried about the recommended size (Smart, 2006).

The following example defines at first the orientation of the boxSizer object of wxBoxSizer class and then the setting of the style of button object, which is located within the boxSizer sizer:

```
#boxSizer { orientation: vertical; }

#boxSizer > #button { x-wxwidgets-
window-style: bu-right, no-border;}
```

It is possible to specify for the wxSizer class the minimal size of sizer, the minimal size of the item and the size of the borders.

In the process of implementing a computer application, it depends on the programmer if he chooses absolute positioning or layout management with the use of the sizers.

It is possible to use specific notation for the wxWidgets platform, the example, which is mentioned above can be rewritten as follows:

```
#boxSizer { x-wxwidgets-orient: vertical; }

#boxSizer > #button { x-wxwidgets-window-style: bu-right, no-border;}
```

In the original solution, which is implemented in C++ programming language, it is not possible to use it because the orientation of wxBoxSizer class sizer is set in the constructor, see Smart (2012). The support is possible by using XRC files, but it is worth mentioning that the programmer doesn't have to use XRC files. The solution of this issue is to use the adjustment of C++ source files right before the compilation, by this it is possible to create the support not only for wxWidgets, but also for other libraries.

The example of notation for Qt library follows:

```
#boxLayout { x-qt-orientation: vertical; }
```

As it is clear from the above example, it is possible to create a list of properties for frequently used layout managers and notate them by a universal notation:

```
#boxLayout { orientation: vertical; }
```

The support for other libraries, as it has already been mentioned, can possibly be solved by other ways if libraries support them:

- by the conversion of USSL style sheet to specific declarative format of notation, see Brázdil (2012b),
- by editing the XML files, where the structure and the appearance of the GUI are specified,
- by using reflection mechanism.

In the current state of the project it is not possible to deal with absolute positioning and editing of positioning with sizers. The mentioned issues can be solved by editing the source codes before compilation. As it was mentioned above, the creation of the layout of the widgets within the program window differs from layout of document page before printing, it is worth thinking about the fact if the creation of hierarchy and layout of the widgets is rather part of creation of GUI structure.

RESULTS

The grammars for lexical and syntax analyzers for enhancements mentioned in this paper for the USSL language were created. During this process the tool called Desátomat[6] was used for creating and editing the grammars.

The solution, which uses the wxWidgets widgets appearance information enrichment of the XRC files, described in Brázdil (2012a) was enhanced. Same as in the original solution, it is possible to use the script written in Python programming language[7] – the module for working with CSS language is used – cssutils[8] and xml.dom module from working with XRC files. The USSL style sheet language has same syntax as CSS and the XRC is a XML language, thus it is not necessary to write own modules, which can work with the USSL and XRC format. For using some enhancements of the language it is possible, like in the original solution, to use the pyScss[9] module from Python programming language.

The solution supports:

- Optional import of the style sheets, which are referenced from the main style sheet using @ import rule,
- URL,
- more properties (bitmap, orientation, pos, icon, …) and classes (wxBoxSizer, wxStaticBoxSizer, …),
- state, sub-control, child and descendant selectors,
- in connection with the possibility of position of widgets, the support for properties orient, size, border a pos was added from sizers, which supports them.

The solution has no support for either inheritance or for cursor styling. The cursor styling is not supported even in XRC, the type selector still acts like class selector.

The solution was tested on the same application, which has got the original solution, also the other XRC files were used. It was taken into consideration if the valid XRC was created and if the properties from USSL style sheet have effect on XRC file.

DISCUSSION

This paper contains the introduction to the language for declarative notation of appearance, which is described in Brázdil (2012a) in detail and its goal is to design a solution of issues of this language. The future work is rather in the implementation area – complex implementation of the solution described in this paper for enhancement of wxWidgets classes, solving issues of solution that uses modification of XRC files and usability testing on platforms other than wxWidgets, complete rewrite of the solution, so that it can be a part of a future release of wxWidgets library and implementation of useful tools for other platforms – mainly for Qt a GTK+ libraries and their declarative notation of appearance.

6 http://desatomat.cz/?lang=help
7 http://www.python.org/
8 http://cthedot.de/cssutils/
9 http://pypi.python.org/pypi/pyScss/

SUMMARY

This paper dealt with the description of USSL – Universal Style Sheet Language environment. However, the first proposal of the USSL has several disadvantages. This paper contains suggestions for solving these issues.

The grammars for lexical and syntax analyzers for enhancements mentioned in this paper for the USSL language were created.

The solution, which uses the wxWidgets widgets appearance information enrichment of the XRC files, was enhanced. The USSL style sheet language has same syntax as CSS and the XRC is a XML language, thus it is not necessary to write own modules, which can work with the USSL and XRC format.

The solution described in this paper supports optional import of the style sheets, which are referenced from the main style sheet using @import rule, URL, more properties (bitmap, orientation, pos, icon, …) and classes (wxBoxSizer, wxStaticBoxSizer, …), state, sub-control, child and descendant selectors, in connection with the possibility of position of widgets, the support for properties orient, size, border a pos was added from sizers, which supports them. However, the solution has no support for either inheritance or for cursor styling, the type selector still acts like class selector.

The future work is rather in the implementation area – complex implementation of the solution described in this paper for enhancement of wxWidgets classes, solving issues of solution that uses modification of XRC files and usability testing on platforms other than wxWidgets, complete rewrite of the solution, so that it can be a part of a future release of wxWidgets library and implementation of useful tools for other platforms – mainly for Qt a GTK+ libraries and their declarative notation of appearance.

REFERENCES

BOS, B., ÇELIK, T., HICKSON, I. and LIE, H. W., 2007: Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification: W3C Recommendation 07 June 2011. World Wide Web Consortium (W3C). *W3C* [online]. 7 June 2011 [cit. 2012-09-28]. Available from: http://www.w3.org/TR/2011/REC-CSS2-20110607.

BRÁZDIL, J. et al., 2010: Návrh stylového jazyka pro zápis vzhledu ve wxWidgets. In: *PEFnet 2010*. Brno: Mendel University in Brno, p. 40. ISBN 978-80-7375-450-1.

BRÁZDIL, J., 2012a: *Oddělení vzhledu a struktury prezentační vrstvy aplikací imperativních programovacích jazyků*. Diploma thesis. Brno: Mendel university in Brno.

BRÁZDIL, J., 2012b: Support for multiple platforms for Universal Style Sheet Language environment using conversion. [CD-ROM]. In: *PEFnet 2012*. Brno: Mendel University in Brno. ISBN 978-80-7375-669-7.

THE GNOME PROJECT, © 2005–2012: GtkCssProvider. *GNOME Developer Center* [online]. [cit. 2012-10-08]. Available from: http://developer.gnome.org/gtk3/stable/GtkCssProvider.html.

KHEDKER, U. P., 1997: *What Makes a Good Programming Language?* [online]. Pune, October 1997 [cit. 2012-09-25]. Available from: http://www.cse.iitb.ac.in/~uday/soft-copies/ppl.ps. Technical Report TR-97-upk-1. University of Pune.

LIE, H. W., 2005: Cascading Style Sheets [online]. *Series of dissertations submitted to the Faculty of Mathematics and Natural Sciences, University of Oslo*, No. 498. [cit. 2012-09-26]. ISSN 1501-7710. Available from: http://people.opera.com/howcome/2006/phd/css.pdf. Dissertation thesis. University of Oslo.

QT PROJECT HOSTING, © 2011: Qt Style Sheets Reference. *Qt Developer Network* [online]. [cit. 2012-10-08]. Available from: http://qt-project.org/doc/qt-4.8/stylesheet-reference.html.

SASS DOCUMENTATION, 2012: *Sass: {style with attitude}* [online]. 20 Aug 2012 [cit. 2012-09-26]. Available from: http://sass-lang.com/docs/yardoc/_index.html.

SMART, J., HOCK, K. and CSOMOR, S., c2006: *Cross-Platform GUI Programming with wxWidgets*. Upper Saddle River: Prentice Hall, 700 p. ISBN 0-13-147381-6.

SMART, J., ROEBLING, R., ZEITLIN, V. et al., 2012: *WxWidgets: Cross-Platform GUI Library* [online]. Documentation. July, 2012 [cit. 2012-09-28]. Available from: http://docs.wxwidgets.org/2.9.4/.

Contact information

Jiří Brázdil: xbrazdi1@node.mendelu.cz
Jiří Šťastný: jiri.stastny@mendelu.cz