Configuration Frameworks

P. Voborník

***Abstract*—**The traditional object-oriented programming frameworks are currently the most widely used method for creating large information systems. Configuration frameworks are their alternative or rather an extension, which are not yet widespread in practical use for desktop applications, despite their huge potential. This innovative method offers many advantages, such as deployment of new modules and whole versions without reinstallation on workstations and disrupting business, easy iterative development and maintenance etc. The paper presents this original approach on a concrete example of the test management administration interface and also shows possibilities of analogical processing other similar systems. Using a technology of the configuration framework for the development of a such systems could bring significant benefits to developers and to users, e.g. easy and fast installation and updates, low memory requirements, small application size, easy system maintenance without necessity of experts etc. As this article illustrates, creating of a unique completely original configuration framework for any specific needs is not so difficult, and it is worthwhile in many cases due to the subsequent savings.

***Keywords*—**Framework, configuration, XML, database, form, table, tree, information system, programming.

I. Introduction

Universal Testing Environment is an electronic online testing system designed for the creation, operation and administration of the tests, independently or in cooperation with LMS [1]. User part of the application (testing and administration interface) is created as the *Rich Internet Application* (RIA) at the *Silverlight*[1] technology. The system originated as a dissertation of the same name [2]. [3]

An integral part of this system is the administration interface that enables the assembly and settings of the tests, management of questions, users and groups, evaluating the results of testing, etc. While creating this part, the standard procedure, in which each window (page) of the application is created separately, was not used. Instead, an original configuration framework has been created for this purpose due to a multiple repetition of windows types. This enabled creating a predominant part of the administration interface using only the configuration data in a simple XML file (e.g. see [4]).

Framework is a software structure that assists in the development of another software product. The aim of a such structure is taking over the typical problems that are often encountered and fully automating their solution. The developer does not need to delay unnecessary overhead around well-known and already investigated issues and he can concentrate fully to implement new specific problems. [5]

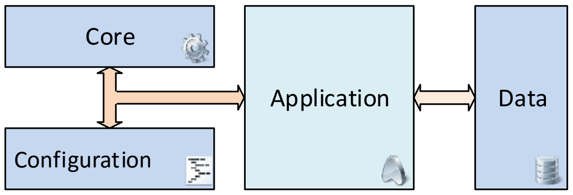


Fig. 1. Scheme of the configuration framework

Configuration frameworks are based on the idea that it is not necessary to program the whole content of the entire system, but only the framework (core) [6]. The framework generates each window dynamically, based on the configuration data (see Fig. 1). These can be stored in a short file or in a database. The structure of the application can be changed without its recompilation, update, and even without its shutdown.

Solution of the administration interface for Universal Testing Environment [2] will be described as an example of such framework. However, this principle can be advantageously used in many different projects containing repetitive types of windows. A classic example may be desktop information systems, which may consist of many dozens of windows, but only of a few kinds (mostly data overview in a table and detail of a record) [7].

II. The structure of configuration XML

The entry of configuration data which define the individual windows of the entire system can be done by many ways. The data of each window can be e.g. kept in a separate file, stored in a database as individual records or saved within a single file. The last method was chosen due to a relatively small scope of this application. Fig. 2 shows the basic structure of the XML configuration file beginning with the root element `<pages>`.

In this scheme, an asterisk (*) before an element name means that the element can be repeated multiple times at this point; an ellipsis (…) after an element name indicates its other branching (not visible in this scheme). From the elements `<grid>` and `<detail>`, only one can be used, exclusively. The `<grid>` element for overviews in a table and branched overviews, the other element, `<detail>`, for forms of detail. There are also special types of windows, which can have a completely

This research has been supported by Specific research project of University of Hradec Kralove, Faculty of Science in 2015.

P. Voborník is with the Department of Informatics, Faculty of Science, University of Hradec Králové, Rokitanského 62, Hradec Králové, 500 03, Czech Republic (e-mail: petr.vobornik@uhk.cz, orcid: 0000-0003-1841-3455).

[1] Silverlight is a software plugin for development lavishly furnished internet applications that run within a web browser. It is developed by Microsoft, executed using the plugin which is a smaller version of the .NET framework. [19]

different structure (e.g. a window for testing a question or a whole test). All these types will be described in more detail hereinafter.

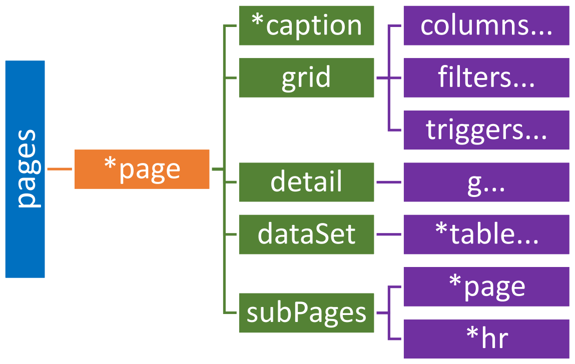


Fig. 2. The structure of the configuration XML which defining windows of the application

The definition of the individual windows can be divided into the following:

- What data are needed for the window
- How the window should look like (layout of visual components)
- Which other windows can be accessed from the current one
- Specific behaviour for certain events

A. Parameters

Parameters are values transferred between windows. For example, if we want to open a table with sub-overview of results only for the selected user from list of all users, the identifier (ID) of selected user must be transferred to the new window, to be able to filter and display only the desired data. There are also global parameters (e.g. `IdOrganization`, as an identifier of the organization whose data are currently administrated) that are automatically transmitted by the framework to each newly generated window. Transmission of other parameters depends on each individual window.

B. Data loading

The principle described in [8] was used for loading data. It is an original custom component that provides communication of a client application with the server. The component keeps the data in the memory structured into objects on the client side, and using the serialized communication [9], it is able to effectively and non-redundantly synchronize them with a central database on the server. The client application simply submits a request for data and then just waits for confirmation that the data are ready.

This component is created as universal and can be used in any project, however, it also holds the auxiliary methods that are specifically designed to support the configuration frameworks. More specifically, the possibility of composing of an object request for data from a text string or XML. A text string uses its own mini-language for querying the data in a simple way; XML version grants the possibility of assembling even more complicated queries including the support of parameters.

```
A_QUESTIONS(ID_GROUP=%IdTest%);A_RESULTS(
ID_TEST=21);A_TESTS(*);A_SOLVINGS
```

Code 1. Sample request for data from four different tables, the first two are filtered

Requests are separated for each table by a semicolon. A filter for limiting the desired set of data records can be written in parentheses after the name of the database table. This filter can also include parameters (embraced with the percent signs, e.g. `%parameter%`, see Code 1). The filter preferences can be composed by using operators `&` (and) and `|` (or). An asterisk (or nothing) means that all the records of the table will be loaded.

```
A_QUESTIONS[%Id%]:DATA,NOTE;A_RESULTS[1]:
TEST_DATA,RESULT;A_TESTS[1]:*;A_SOLVINGS[1]
```

Code 2. Sample request for delayed data from one record of four different tables

Square brackets (see Code 2) are used for loading a specific record by using its primary ID key which can be defined by the means of a parameter, or a value. Loading one particular record is performed when showing its detailed form. The so called “delayed data” can be downloaded only by this way (only for one particular record, see [8]). These values are usually more data-extensive (e.g. long description, XML data, article, etc.) and they are not loaded collectively for multiple records at once. This is partly due to the reduction of the volume of transmitted data and also because they are not displayed in tabular overviews due to their vastness, anyway.

Because not all properties of the record will be displayed in the detail form every time, their enumeration can be specified in the context of the query mini-language after a colon, or followed by an asterisk (or nothing) to load all the delayed values. Queries for specific records or for bulk data can also be combined into a single query string.

These queries can be put either to the data attribute in the `<dataSet>` element or split into multiple XML sub-elements (see Code 3).

If a more detailed breakdown of the query into XML is used, the requests for each table are defined in the `<table>` elements, where again the `data` attributes may include detailed queries in the described mini-language. Only the name of the database table can also be put into the `name` attribute and it is possible to itemize the filters into `<filter>` sub-elements. A `refresh` attribute indicates which data to load from the server when the function for data refresh is called from the window.

```
<dataSet>
 <table name="A_RESULTS" refresh="1">
  <filter param="IdOrganization">ID_LIMIT.ID_PERIOD.
   ID_ORGANIZATION.ID=%IdOrganization%</filter>
  <filter param="IdUser">ID_USER.ID=%IdUser%</filter>
  <filter param="IdTest">ID_TEST.ID=%IdTest%</filter>
  <filter param="IdLimit">ID_LIMIT.ID=%IdLimit%</filter>
 </table>
 <table name="A_USERS">
  <filter param="IdOrganization">
   Contains:Organizations~ID_ORGANIZATION.ID=
   %IdOrganization%</filter>
  <filter param="IdUser">ID=%IdUser%</filter>
 </table>
 <table name="A_TESTS">
  <filter param="IdOrganization">
   Contains:Organizations~ID_ORGANIZATION.ID=
   %IdOrganization%</filter>
  <filter param="IdTest">ID=%IdTest%</filter>
 </table>
 <table name="A_COMPUTERS" />
 <table name="A_TEST_LIMITS">
  <filter param="IdLimit">ID=%IdLimit%</filter>
  <filter param="IdTest">ID_TEST.ID=%IdTest%</filter>
 </table>
 <table name="A_PERIODS" refresh="1">
  <filter param="IdTest">
   Contains:Limits~ID_TEST.ID=%IdTest%|Contains:
   LimitsResults~ID_TEST.ID=%IdTest%</filter>
  <filter param="IdLimit">Contains:Limits~ID=%IdLimit%|
   Contains:LimitsResults~ID=%IdLimit%</filter>
 </table>
 <table name="A_ORG_GROUPS">
  <filter param="IdOrganization">ID_ORGANIZATION.ID=
   %IdOrganization%</filter>
 </table>
</dataSet>
```

Code 3. Sample definition of data requests for overview of results

Individual parts of the filter are connected with the `and` operator. A `param` attribute in the `<filter>` element not only indicates which parameters are to be replaced by their values, but it also specifies that if the window does not know any of these parameters, this condition will not be included in the filter. It is also the main difference between the XML query and the query string, where all the filter conditions are always used. This enables the same window with a completely different content to be opened. In this case (see Code 3) it is a summary of the results of the testing, which can be filtered only for a particular test, for a particular user, for the group of users, for a particular term, etc., and for their combinations (e.g., “results of the user A from test B”).

III. Design definition of individual window types

Only a small group of window types usually exists in most systems, which covers the vast majority of the system. Tables and details are the most common. However, a similar procedure can be applied to any other type of windows.

A type of window, which is defined in the `<page>` element, is determined by its attribute `type`. This framework supports the following windows types.

- `grid` – table overview
- `tree` – branched overview
- `detail` – detail form
- `designFrame` – design of a question
- `testerFrame` – testing of a test
- `upload` – uploading of files

A. *Table overviews*

Table overviews are windows usually containing data from a certain database table, interconnection of tables or views, displayed in a tabular form (grid), i.e. as columns and rows. Data in the table should offer the ability to sort, filter, group, summarize, search, etc. These functions are usually directly provided by a component (in this case DXGrid[2], see [10]) used on the client side. Window type *detail* is used to view more details of a specific record (row) and its editing (see chap. 3.2). A *detail* needs to be opened with the `Id` parameter, which is the ID of the selected record in the table overview. Also other windows with sub-overviews for the selected records can be opened (e.g. to open the overview of results of selected user from the list of users).

For the definition of the table overview, it is usually enough to define which columns it should contain and how to represent its data (see Code 4 and its result in Fig. 3). For example, numbers can have a specific display format (currency, percentages, decimals, integers), as well as dates and times; boolean values (yes/no) may be showed as checkboxes.

```
<grid table="Result">
 <columns>
  <col name="TestName" caption="Test"
       hideIfParam="IdTest" />
  <col name="UserName" caption="User"
       hideIfParam="IdUser" />
  <col name="PeriodName" caption="Period" visible="0" />
  <col name="Start" caption="Start"
       formatString="dd.MM.yyyy HH:mm" sort="0D" />
  <col name="End" caption="End" formatString="HH:mm" />
  <col name="TestTimeSpan" caption="Duration"
       formatString="HH:mm:ss" />
  <col name="Score" caption="Achieved result"
       formatString="#0.00%" />
  <col name="ScoreBonus" caption="Penalties"
       formatString="#0.00%;-#0.00%; " ro="0" />
  <col name="ScoreFinall" caption="Total result"
       edit="Score" />
  <col name="ComputerName" caption="Computer" ro="0" />
  <col name="IpStart" caption="IP" visible="0" />
  <col name="ResultsShowCount" caption="Views" />
  <col name="Note" caption="Note" edit="Memo" ro="0" />
  <col name="IsArchived" caption="Archive" ro="0" />
 </columns>
 ...
</grid>
```

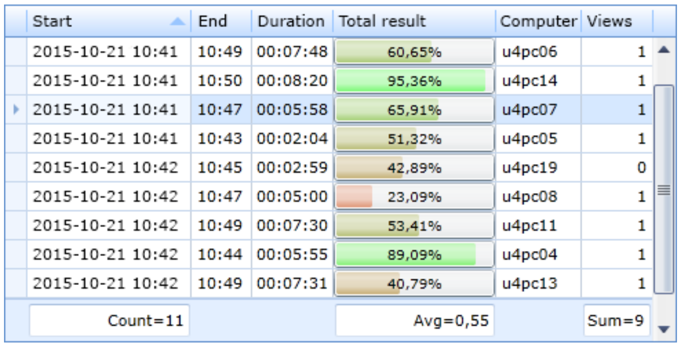
Code 4. Sample column definitions for overview of results

[2] DevExpress Grid – https://www.devexpress.com/Products/NET/Controls/Silverlight/Grid/

The settings of individual columns are determined by the attributes of the `<col>` element. Their enumeration is as follows:

- `name` – the name of the property of an object, the value of which will be displayed in the column.
- `caption` – is a caption in the column header.
- `formatString` – specifies the format string for non-text data. These strings are directly supported both by the programming language (C#, see [11]) and by the used grid component, so it is sufficient to set it to the column and the data will be displayed in this format.
- `ro` – determines if the column is read-only (1 = yes – default) or enabled for direct editing within the table (0 = no).
- `edit` – determines the type of editor that is used not only for direct editing of data, but also enables their proper formatting for a mere display. If it's not stated directly, it is derived from the data type of the property displayed in the column. This enables, for example, using different editors for the same data types (e.g. single-line or multiline editor "Memo" for a text string), and also custom editors and data displayers (e.g. "Score") can be created.
- `sort` – enables setting the default sorting for data in the table by selected columns. The first character determines the priority of a sorting and the second (last) character determines the direction (A – ascending, D – descending).
- `visible` – defines, whether the column is displayed or hidden by default. Users can manually display hidden columns or hide the ones they do not need to see.
- `hideIfParam` – hides the column if the specified parameter of a window exists. It is thus possible e.g. to hide a column with the name of a test in the overview of test results (if it is displayed only for a single test).

These attributes can be defined differently for another system, the support of any additional attributes can also be added.



Start	End	Duration	Total result	Computer	Views
2015-10-21 10:41	10:49	00:07:48	60,65%	u4pc06	1
2015-10-21 10:41	10:50	00:08:20	95,36%	u4pc14	1
2015-10-21 10:41	10:47	00:05:58	65,91%	u4pc07	1
2015-10-21 10:41	10:43	00:02:04	51,32%	u4pc05	1
2015-10-21 10:42	10:45	00:02:59	42,89%	u4pc19	0
2015-10-21 10:42	10:47	00:05:00	23,09%	u4pc08	1
2015-10-21 10:42	10:49	00:07:30	53,41%	u4pc11	1
2015-10-21 10:42	10:44	00:05:55	89,09%	u4pc04	1
2015-10-21 10:42	10:49	00:07:31	40,79%	u4pc13	1
Count=11			Avg=0,55		Sum=9

Fig. 3. Sample overview of test results (some columns are hidden for clarity)

The general rule is that more extensive definitions should have a shortcut and the default value for omitted attributes should cover the highest possible proportion of cases.

Filters

Data from the database tables are loaded into the table overviews. Which part of this database table is necessary for the overview, is exactly defined by the filters for each overview in the `<dataSet>` sub-elements (see Code 3). Nevertheless, these filters are not applied when displaying data. Therefore, there is no risk that some data are missing in the overview. It may happen that there are some records in the overview that do not belong there (e.g. if an overview of user A results is displayed as the first and an overview of user B results is displayed as the second, the results of both users will be included in the second overview, because they draw from the same data source that is not further filtered on the client side). For this reason, there is a `<filters>` section included in the overviews (`<grid>`). This enables defining the filtered data downloaded onto the client side (see Code 5).

```
<filters>
 <filter param="IdUser">IdUser=%IdUser%</filter>
 <filter param="IdTest">IdTest=%IdTest%</filter>
 <filter param="IdLimit">
  IdLimit=%IdLimit%
 </filter>
</filters>
```

Code 5. Sample of filter definitions of data for overviews on the client side

The individual filters are entered in a `<filter>` elements the value of which is the condition of the filter. The `param` attribute indicates which parameter is used in a condition and thus it is required. If this parameter is not passed when creating of the overview, the filter will not be applied. The conditions of individual filters are connected by an `and` operator.

Triggers

Element `<triggers>` can only be used for table overviews, because a new record is usually created only from them, even though it is opened in detail form. When a new record is created, some of its values is possible to pre-set automatically. Section `<triggers>` takes care of it (see Code 6).

```
<triggers>
  <insert property="IdTest" param="IdTest" />
  <insert property="IdGroup" param="Id"
          only="file" />
  <insert property="Period" new="Period">
    <insert property="IdOrganization"
            param="IdOrganization" />
    <insert property="From" value="now" />
    <insert property="To" value="now+600" />
  </insert>
</triggers>
```

Code 6. Sample of triggers from a branched overview of questions in a test

Only `<insert>` elements can be used as sub-elements which are handled when a new record is creating. These elements may contain the following attributes.

- `property` – the name of the property of an object, to which the value will be set
- `param` – a parameter name of the current window whose value will be set to the object property
- `value` – a constant value which will be set to the property of the new record ("now" set the current date and time)
- `new` – only for properties of the type of data object – it creates the new object into this property (the name of its class is the attribute value)
- `only` – restrictions of validity of the element to folders or to items (only for branched overviews, see below)

In the case of the `new` attribute when a new object is created for the property, the `<insert>` element may be also branched out to other sub-elements `<insert>`, which then sets the values for the new sub-object.

B. *Branched overviews*

Branched overviews are similar to table overviews, but they display data in a tree (hierarchical) structure (e.g. user groups + users or questions in groups).

Two types of items are distinguished in branched overview – a *folder* and an *item* (sometimes *file*). Folders have a minimum of three values: ID, id referring to the ancestor and the name. Thanks to the first two values the data can be easily organized into a hierarchical structure[3] and the name allows uniform clearly displaying of this data. Items can then be assigned into folders in two ways: directly (1:n, see Fig. 4 top) or via a link table (m:n, see Fig. 4 bottom). In the first case, each item will be only once in the list, in the second case it can be there multiple times, respectively not even once.

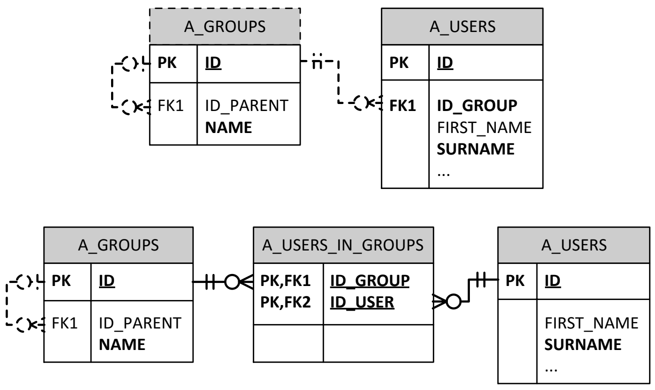
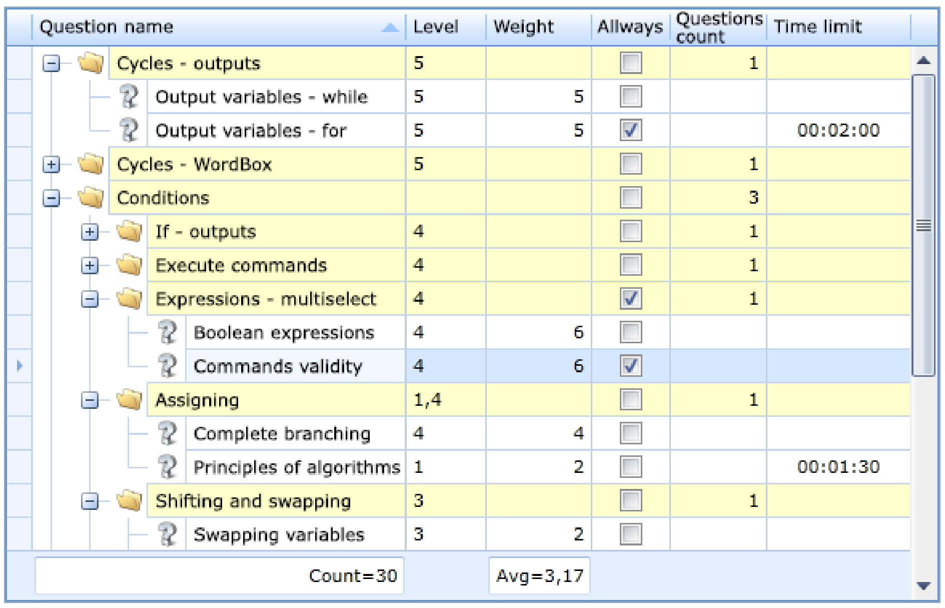


Fig. 4. Data model of classification of users into groups by binding of 1:n (top) and m:n (bottom)

Appearance of the window depends very much on the used component for displaying branched overview. The standard component from the Silverlight Toolkit[4] is capable of displaying only a single value (name), but not a comprehensive overview with more columns. The component from DevExpress[5] was used in this case, which allows to use a fully-fledged tabular overview including branched structures (see Fig. 5).



Question name	Level	Weight	Allways	Questions count	Time limit
Cycles - outputs	5		☐	1	
Output variables - while	5	5	☐		
Output variables - for	5	5	✓		00:02:00
Cycles - WordBox	5		☐	1	
Conditions			☐	3	
If - outputs	4		☐	1	
Execute commands	4		☐	1	
Expressions - multiselect	4		✓	1	
Boolean expressions	4	6	☐		
Commands validity	4	6	✓		
Assigning	1,4		☐	1	
Complete branching	4	4	☐		
Principles of algorithms	1	2	☐		00:01:30
Shifting and swapping	3		☐	1	
Swapping variables	3	2	☐		
Count=30		Avg=3,17			

Fig. 5. Sample of the branched overview of questions from the test about Algorithms

```
<columns>
 <col name="Name" caption="Question name"
      ro="0" sort="0A" />
 <col name="Level" caption="Level" ro="0" />
 <col name="Weight" caption="Weight" ro="0" />
 <col name="IsAllways" caption="Allways" ro="0" />
 <col name="IsMix" caption="Mixing" ro="0"
      only="folder" visible="0" />
 <col name="QuestionCount" ro="0" only="folder"
      caption="Questions count" />
 <col name="IsTogether" caption="Keep together"
      ro="0" only="folder" />
 <col name="TimeLimtSpan" only="file"
      caption="Time limit" ro="0" />
 <col name="Index" caption="Order" ro="0"
      visible="0" />
</columns>
```

Code 7. Sample of column definitions in branched overview of test questions

Definition of the layout is almost identical to the classic overview (see Code 7), i.e. a division into columns, their relations to the data, header title, editor etc. Moreover, there is an option to add an attribute `only` and to set its value to "folder" or "file". This property limits the use only for a specific type of item, i.e. only for a folder (*folder*) or only for an item (*file*). The

[3] folders of the first level (root) have ID of ancestor set to null, in numeric (int) expressing it is zero

[4] Silverlight Toolkit – http://silverlight.codeplex.com

[5] DevExpress Tree View – https://www.devexpress.com/Products/NET/Controls/Silverlight/Tree_List/

value in the column of the second item type, then cannot be edit and always remains empty (null).

Another difference in the definition of branched overview is a necessity instead of a single data source (the name of the object class), the two data sources must be defined and the relationships between them. These additional attributes are therefore defined in the `<grid>` element.

- `folder` – the name of the class with data for folders (e.g. "QueGroup")
- `file` – the name of the class with data for items (e.g. "Question")
- `fileIdParent` – the name of the property of an item with a group identifier (e.g. "IdGroup")
- `fileName` – the name of the property of an item with its name (e.g. "Name")
- `fileIcon` – the name of the item icon
- `canMoveFilesToRoot` – permission or prohibition of moving items out of folders into the root
- `treeListItemSubType` – the name of the class which encapsulating both types of values into a single

Folders have standardized some properties apart from items. They are always derived from a common ancestor – `TreeBase` class. Therefore, all references are unnecessary to define in the specification, as is the case with items. Specifically, it is the properties with the reference to the ancestor (IdParent), the name (Name) and the icon.

```
public class QueuestionsAndGroups : TreeListItem
{
  public bool IsAllways
  {
    get {
      return IsFolder ? Folder.IsAllways
                      : File.IsAllways;
    }
    set {
      if (IsFolder)
        Folder.IsAllways = value;
      else
        File.IsAllways = value;
    }
  }

  public int? QuestionCount
  {
    get {
      return IsFolder ? Folder.QuestionCount
                      : null;
    }
    set {
      if (IsFolder)
        Folder.QuestionCount = value;
    }
  }
  ...
}
```

Code 8. Sample of the code of the `QueuestionsAndGroups` class that encapsulates the property `IsAlways` shared for folders and items, and the property `QuestionCount` that have only folders

Components of displaying hierarchical data usually accept only one data source that contains only objects of the same class. In addition, this class must be completely identical for all objects, not only inherited. This is a problem when combining folders and items, because they cannot be used directly.

A modification of design pattern Decorator [12, pp. 9-21] was used for the solution. Packing class has been created, which always holds only the reference to an encapsulated object and includes properties referring to its properties, which must be a common for a folder and for an item in the source list for a valid display. This object always exactly knows which of these two types encapsulates, and accordingly handles its properties (see Code 8).

C. Details

The table is no longer the dominant element in the windows displaying detailed information of one record, but it is the form consisting of labels and editors. These should be appropriately (visually, logically and intuitively) placed in the window so that their filling and editing is as easy as possible for the users.

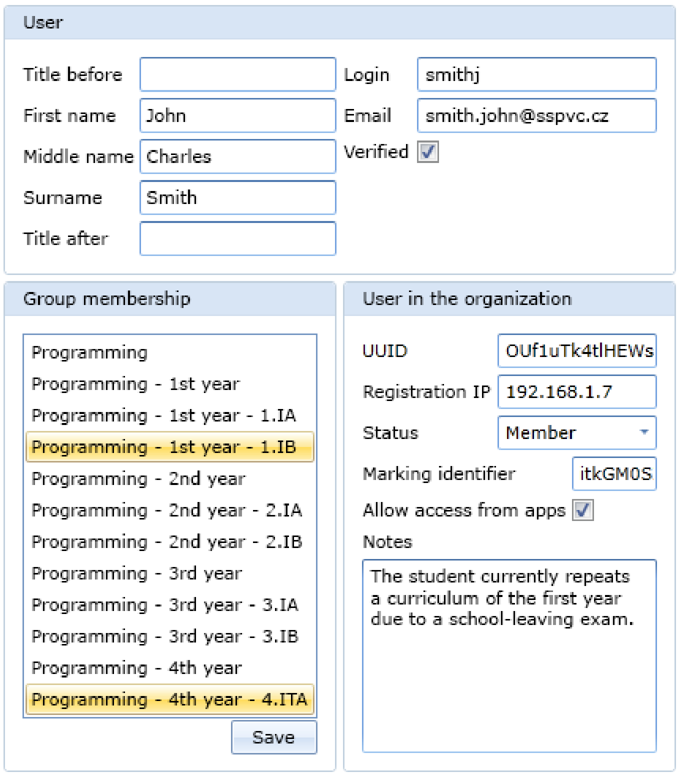
```
<detail table="User">
 <g>
  <g>
   <g o="h" caption="User">
    <g>
     <itm name="TitleBefore" caption="Title before" />
     <itm name="FirstName" caption="First name" />
     <itm name="MiddleName" caption="Middle name" />
     <itm name="Surname" caption="Surname" />
     <itm name="TitleAfter" caption="Title after" />
    </g>
    <g>
     <itm name="Login" caption="Login" ro="1" />
     <itm name="Email" caption="Email" ro="1" />
     <itm name="IsEmailVerify" caption="Verified"
          ro="1" />
     <itm name="IsArchived" caption="Archive" />
    </g>
  </g>
  <g o="h">
    <g caption="Group membership">
     <itm name="Groups" edit="List"
          source="OrgGroup" labelPos="top" />
    </g>
    <g subObject="OrgUser"
       caption="User in the organization">
     <itm name="Uuid" caption="UUID" ro="1" />
     <itm name="Ip" caption="Registration IP" ro="1" />
     <itm name="OrgState" caption="Status" />
     <g labelGrid="local">
      <itm name="MarkId" caption="Marking identifier" />
      <itm name="IsAppTrust"
           caption="Allow access from apps" />
     </g>
     <itm name="Note" caption="Notes" edit="Memo"
          vAlign="stretch" labelPos="top" />
    </g>
   </g>
  </g>
 </g>
</detail>
```

Code 9. Sample definition of the form for user detail

Distribution of components into logical units is realized by the groups, which are represented by a `<g>` element in the layout definition. This element may have several attributes. One is the `o`, which defines the orientation of the sorting of elements in the group, either vertically (`v`, default value) or horizontally (`h`). The second attribute is the `caption` that enables setting the title of the group. If the title is not listed, a header and a border of a group are also not displayed. It can be used for example for visual sorting of elements into multiple columns within a single parent frame. Another attribute, `subObject`, enables defining a group's data context[6] to the property of the edited object. This property is an object itself with its own properties. All items of this group will then refer to the properties of this sub-object (see Code 9 and its result in Fig. 6).

A `labelGrid` attribute in the group element `<g>` enables setting the value to "`local`", which excludes items of this group from a global system of aligning within the form and these items are aligned separately. The dimensions of the label from the other groups are therefore ignored and, vice versa, the other groups ignore this group. If only one item is to be handled like this, it is enough to wrap it with the `<g>` element without a `caption` attribute.

The whole detail form is always one main group, the content of which is generated recursively.



User

Title before		Login	smithj
First name	John	Email	smith.john@sspvc.cz
Middle name	Charles	Verified	✓
Surname	Smith		
Title after			

Group membership

- Programming
- Programming - 1st year
- Programming - 1st year - 1.IA
- Programming - 1st year - 1.IB
- Programming - 2nd year
- Programming - 2nd year - 2.IA
- Programming - 2nd year - 2.IB
- Programming - 3rd year
- Programming - 3rd year - 3.IA
- Programming - 3rd year - 3.IB
- Programming - 4th year
- Programming - 4th year - 4.ITA

User in the organization

UUID	OUf1uTk4tlHEWs
Registration IP	192.168.1.7
Status	Member
Marking identifier	itkGM0S
Allow access from apps	✓
Notes	The student currently repeats a curriculum of the first year due to a school-leaving exam.

Fig. 6. Sample of user detail form

Individual items of the form are composed by a label and an editor. Their definition is entered into the `<itm>` element (item) and they may have the following attributes.

- `name` – the name of the object's property, the value of which is displayed and can be edited by the item
- `caption` – the label before (or above) an item explaining its meaning
- `edit` – enables changing the editor (see below), otherwise the type of the editor is automatically chosen according to the data type of the object property
- `labelPos` – determines where a label of an item will be placed, in front of it (left) or above it (top)
- `formatString` – determines the mask for displaying and editing certain data types (date, time, numbers, etc., see *formatString* in overviews)
- `ro` – determines whether an item is read-only or not (default value)
- `width` – enables setting a specific width for an edit control (e.g. smaller width for the editors of codes or numbers and larger for the editors of names or addresses)
- `height` – enables setting a specific height for an edit control (appropriate e.g. for the editors of longer texts, "Memo")
- `minWidth` – enables setting a minimum width for an edit control, which must be respected by the editor even if the form would not fit into the size of the window and a horizontal scrollbar was thus displayed
- `minHeight` – enables setting a minimum height for an edit control
- `align` – horizontal alignment of the content in an edit control
- `vAlign` – vertical alignment of an edit control; it enables setting a "stretch" value which causes its vertical expansion in a free space of the window
- `hAlign` – horizontal alignment of an edit control
- `font` – font name for an edit control (can be changed e.g. for editing an XML code)
- `fontSize` – font size for an edit control

Some items with specific editors can take other attributes. This system supports the following edit controls.

- `Text` – short single-line text (standard for *string*, e.g. the name)
- `Memo` – long multiline text (e.g. a description)
- `Integer` – for integers
- `Float` – decimal (standard for *float* and *decimal*, e.g. score)
- `Check` – check box (standard for *boolean*, e.g. allowing access - yes/no)
- `DateTime` – date, date and time, according to *formatString* (standard for *DateTime*, e.g. launch time of the test)
- `TimeSpan` – time (standard for *TimeSpan*, e.g. duration of the test)
- `Combo` – selection menu; requires additional setup of `source` attribute for data source and a `display` attribute for determining the property for the text that is displayed in the menu (standard for data objects, e.g. selecting group for testing)

[6] data context is a property of a container (*Panel*) grouping of edit controls which determines the object, with its properties are these items linked and if necessary these properties are edited by them [20]

- `Enum` – same as `Combo`, only values for choice are automatically assembled from all possible values of the enumeration (standard for *enum*, e.g. user status in the organization)
- `Picture` – selecting an image from imported images (e.g. test icon)
- `PicturePreview` – image thumbnail resized to the desired dimensions or to the size of the window (e.g. the image previews in the list of imported sources)
- `List` – specific editor showing a list of items from another data source and allowing to add or remove these items to the m:n relationship between the edited object and the objects of the source (e.g. user's membership in groups)
- `TestSettings` – custom specific editor for editing test settings

D. Special types of windows

Special types of windows, unlike the previous ones, were created for a specific purpose and they are not re-usable for different data therefore it will be mentioned only in passing.

DesignFrame is a window designed for creating and testing of a specific test question. It works with XML code, which is transformed into the final QML[7] of which it can run fully functional question in a fictitious test with the predefined settings.[8]

TesterFrame is a window that is only an intermediary between the test application interface. It is opened within the window of the system administration with a full test. Here can be tested primarily test settings. The test can be launched either directly with the general settings as well as for its individual limits[9] for launching, always with their specific settings. In both cases, it is purely a testing launch, working outside the definition of time availability of the test and without saving results.

Upload is a window designed for uploading multimedia files (e.g. images) to the server.

Another window with specific properties is not a problem at any time to add for any additional needs. However, as can be seen, these cases occur really rarely, and the majority part of the application is covered by the universal windows.

E. Sub-windows

Basic windows (e.g. an overview of the tests) can be opened directly from the main menu[10]. Other overviews are opening from these windows as their sub-windows. They also take over the parameters from the windows from which they were launched. Usually it is an identifier (ID) of any of the records (objects), whose detail form or sub-overview[11] to be opened.

```
<subPages>
 <page name="TestDetail" caption="New test"
       params="Id:ID;TestName:Name" for="new" />
 <page name="TestDetail"
       caption="Detail of the test"
       params="Id:ID;TestName:Name" />
 <hr />
 <page name="TestQuestionsGroups"
       caption="Test questions"
       params="IdTest:ID;TestName:Name" />
 <page name="TestLimits" caption="Test limits"
       params="IdTest:ID;TestName:Name" />
 <page name="Results" caption="Results"
       params="IdTest:ID;TestName:Name" />
 <hr />
 <page name="TesterFrame" caption="Try the test"
       params="IdTest:ID;TestName:Name" />
</subPages>
```

Code 10. Definition of sub-windows of the tests overview

Data of each sub-window, that can be opened from the current window, is defined in by `<page>` elements in the `<subPages>` element (see Code 10). Element `<hr>` may also been here, which add a separator (separating line) to the ribbon menu with buttons that link to each sub-window and thereby to divide them into thematic groups (see Fig. 7). Each `<page>` element contains the following attributes.

- `name` – window codename, identifying an element in the main branch (by the name attribute) of system defining XML, whose data will be loaded for the page
- `caption` – displayed name of the window and also the label on the button which to open it
- `params` – parameters passed to the sub-window in a special format (see below)
- `for` – value "new" means that before the window opens, a new record (object) of the same type as the source of the current list or detail is created, and this record with all his default properties will be transmitted to the preparing sub-window

`Params` attribute thus allows to specify the parameters that will be transmitted to the new sub-window. They are written in pairs separated by semicolons, and paired pairs are separated by a colon. The first of each pair specifies the name of the parameter under which it will be able to retrieve the sub-window. The second is the name of the properties of the current object[12] from which a transmitted value will be loaded.

[7] QML – Question Markup Language - XML based language for definition of test questions with rich interactive and randomly-changing content, see [2, pp. 39-66]

[8] editor of questions can directly be tested on http://app.alltest.eu/Design.aspx

[9] *test limit* – permit access to the test for launching it for specified tested users through the test interface (other part of the system) with set conditions (date and time of accessibility of the test, number of questions, time limit, mixing, level etc.)

[10] windows, that can be run directly from the main menu (i.e. not as sub-windows), has the `main` attribute set to 1 in element `<page>`

[11] sub-overview – an overview contains only the records related to a particular parent object (e.g. the results of only certain user)

[12] the current object – it is the object (record) in overviews currently selected (marked) in the list; in the detail form it is the edited record whose data are just retrieved

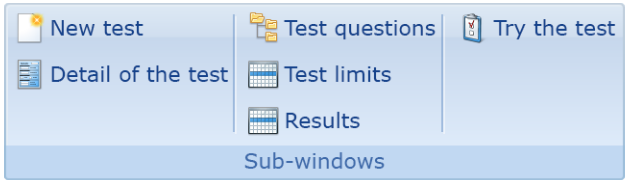


Fig. 7. Sample buttons for opening sub-windows from the table overview of tests

F. Headings

One of the attributes of the `<page>` element is the `caption` indicating the user name of the window (e.g. “Results”). This may however vary, according to other limitations that are defined by input parameters (e.g., “Results of John Smith”). For this purpose, all the windows can use the elements `<caption>`, which allow them to define the name depending on availability of parameters (see Code 11).

```
<page name="Results" caption="Results"
      type="grid" main="0">
  <caption param="UserName,TestName"
    caption="Results of the user %UserName% from
             the test %TestName%" />
  <caption param="UserName"
    caption="Results of the user %UserName%" />
  <caption param="TestName"
    caption="Results of the test %TestName%" />
  ...
</page>
```

Code 11. Sample of use of elements `<caption>` to define the user name of the window depending on the input parameters

The main attribute of the `<caption>` element has the same name `caption` and its defines its name. It may contain parameters (words from both sides introduced by a percent sign %), which will be replaced by the value of the parameter. Which `<caption>` element will be used for the window title depends primarily on the presence of all input parameters specified in the attribute `param` (if there are more, they are separated by a comma), and secondly on the order of elements. If the condition fits more elements, the first of them will be used to select the name.

IV. Configuration directly in the code using attributes

One of the special types of editors is the test settings (*TestSettings*) that allows to define individual values of the settings of the test[13] (see Fig. 9). There are more of these settings, they have a various type (yes/no, number, text, selection, etc.), and they are divided into groups for a clarity. There are several degrees of the same settings (default settings, test settings, settings of the one accessibility of the test and settings through by API), whereas apply that the higher overrides lower (see Fig. 8).

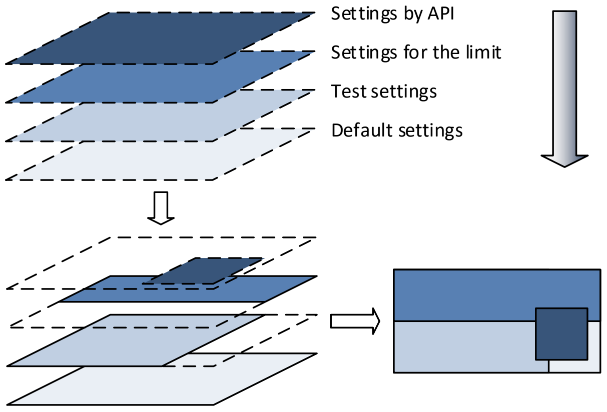
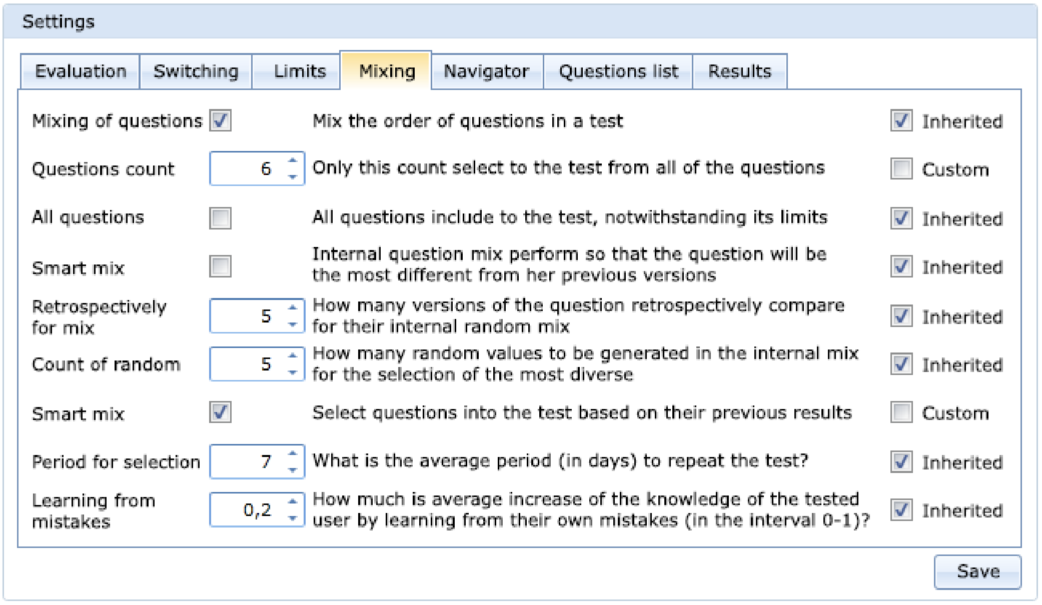


Fig. 8. Illustrative example of the overlap of various levels of the same settings

The form for this settings could be created by several ways. One of them is the above described method of creating the detail forms in configuration XML. This method has not been selected, due to its extensiveness and especially due to necessity of its reuse in different windows as their part. Another option was to create a special custom component (*UserControl*), with a fixed pre-defined items and this control using for editing of the whole setting. Since the setting is stored in XML format as one data item, this option is also suitable from this standpoint. However, this method has been used only in part.



Mixing of questions	✓	Mix the order of questions in a test	✓ Inherited
Questions count	6	Only this count select to the test from all of the questions	☐ Custom
All questions	☐	All questions include to the test, notwithstanding its limits	✓ Inherited
Smart mix	☐	Internal question mix perform so that the question will be the most different from her previous versions	✓ Inherited
Retrospectively for mix	5	How many versions of the question retrospectively compare for their internal random mix	✓ Inherited
Count of random	5	How many random values to be generated in the internal mix for the selection of the most diverse	✓ Inherited
Smart mix	✓	Select questions into the test based on their previous results	☐ Custom
Period for selection	7	What is the average period (in days) to repeat the test?	✓ Inherited
Learning from mistakes	0,2	How much is average increase of the knowledge of the tested user by learning from their own mistakes (in the interval 0-1)?	✓ Inherited

Fig. 9. Sample of the test settings editor

[13] the functionality of the test settings editor can try on http://app.alltest.eu/Design.aspx

Component for editing of the test settings was created as universal so that all its content is automatically generated based on the configuration data. However, these data are not in an external XML file, but they are integrated directly into the application code. More specifically, the values required for the automated generation of the editor are written as program attributes [11, p. 449] for the definition of classes and their properties that are used for storing of loaded data objects in memory during editing (see Code 12). This procedure is based on methods of aspect oriented programming (see [13]), when a routine parts of the code are separated and encapsulated outside the main part of the code and they activate themselves only on the basis of certain symptoms [14], [15].

Captions and descriptions are assigned to classes and properties directly in the code and they then appear in the edit form. The editor is generated by the method with an input information about the type of major classes `TestSettings`, that has only 7 properties which are groups of individual settings. These properties are checked using the reflection [16, p. 489] and if their types (classes) contain attribute `XmlClass` (see Fig. 10), custom tab with label defined by this attribute is created for them in the form (see Fig. 9). Individual items linked with the properties of an object using the Data Binding[14] [17, p. 275] method is then generated on the panel under this tab. Caption and note are taken from the attribute `XmlProperty` and the type of item editor is chosen based on the data type of each property.

Checkboxes *Inherited* or *Custom* at right inform about the origin of each value. If the value is inherited (checked), then an information about the settings not saved into the XML at this level and is taken from a more general setting. If the value is custom (box is unchecked and his label says “Custom”) then it is stored into the XML with settings. When the box is checked again the item resets the value to the value of its ancestor.

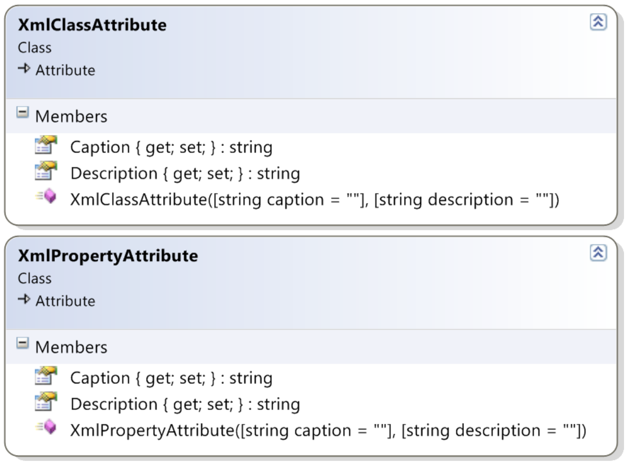

XmlClassAttribute
Class
Attribute
Members
Caption { get; set; } : string
Description { get; set; } : string
XmlClassAttribute([string caption = ""], [string description = ""])
XmlPropertyAttribute
Class
Attribute
Members
Caption { get; set; } : string
Description { get; set; } : string
XmlPropertyAttribute([string caption = ""], [string description = ""])

Fig. 10. Diagram of configuration classes `XmlClassAttribute` and `XmlPropertyAttribute`

Values of each setting item are determined in a way that each of them is trying to retrieve sequentially from the XML stored at different levels (see Fig. 8). If the value is found in one of the XML, a further search procedure is stopped and searching of the value for the next item begins.

```
[XmlClass("Mixing")]
public class Mixing : DepenencyObjectBase,
                      INotifyPropertyChanged
{
  private bool mixQuestions = true;
  [XmlProperty("Mixing of questions",
     "Mix the order of questions in a test")]
  public bool QuestionsCount
  {
    get { return mixQuestions; }
    set { mixQuestions = value;
          PropertyChanged("MixQuestions"); }
  }
  private int questionsCount = 0;
  [XmlProperty("Questions count",
     "Only this count select to the test …")]
  public int QuestionsCount
  {
    get { return questionsCount; }
    set { questionsCount = value;
          PropertyChanged("QuestionsCount");}
  }
  ...
}
```

Code 12. Sample code of the class `Mixing` which contains configuration attributes

A. Localization

Text labels in a specific language are stored in the configuration XML in addition to general information about the data, edit controls, links, etc. The same applies to the configuration directly in the code. That could be an issue if the system was meant to be multi-lingual.

The solution could be to enter key strings instead of specific texts. These keys would refer either to the classic items in the so-called *resources*[15] or to other external sources [18]. The specific texts for each label would be loaded from the resources, if necessary. The relevant file with resources would be assigned according to the user-selected national settings.

V. Conclusion

Configuration frameworks are an alternative to the traditional structural and currently the most widely used object-oriented programming frameworks. These frameworks in their pure form are not yet widespread in practical use for desktop applications, despite their huge potential. They offer many advantages, such as iterative development, deployment of new modules without reinstallation on workstations, distribution of new versions without disrupting business, easy development and maintenance etc.

[14] similar function of auto generating of a form based on the properties of the class also provides a component `DataFrom` from the *Silverlight Toolkit* plugin (http://silverlight.codeplex.com), see [20, pp. 78-81]

[15] Localizing Silverlight-based Applications – http://msdn.microsoft.com/en-us/library/cc838238(VS.96)

This approach could be very useful in the field of information systems, for their developers, administrators, and users too. For example, if a label has to be changed, or an item has to be added or removed from any of the window of desktop application, creation and distribution of a new version of the whole application or a library, which is complicated, laborious, administratively difficult and lengthy at the testing, is not required. A mere editing of the corresponding XML element stored e.g. in the global database would result in this action affecting all client workstations at the time the window next opens, without having to shut down the application. A plain text editor or a simple administrative web interface is sufficient for that action, rather than complicated and computationally intensive development environment or a compiler.

Of course, the same immediate updates are allowed inherently with the online web systems created in HTML, but also in this case, the use of the configuration framework may be advantageous. Although larger systems tend to use frameworks that e.g. standardize the look and the layout on all pages, the layout of controls and their functionality are usually necessary to be solved programmatically on each page, separately, even if only by calling global methods. Complete abstraction of the entire user interface from the code determining the behavior of the system, windows (pages) and elements, presented in this article, is more demanding on the initial implementation, but it will pay off dramatically with as little as middle-large applications, during their further development and subsequent maintenance.

If such a system was based on the configuration framework, it could bring significant benefits to both its authors and users, e.g. small size of applications, low memory requirements, easy installation, uncomplicated update during runtime, the fact that even a non-expert can handle the framework maintenance etc. In order to simplify the generation of configuration data, a custom WYSIWYG editor could also be easily created.

Many ready solutions already exist for deployment on the web (e.g. DotNetNuke[16], Joomla[17], Drupal[18] and basically even the Moodle[19] and others), which use a similar principle. These systems usually come with a content management system that completely encapsulates the management of configuration data into more user-friendly form-based editors. However, these systems are not always suitable for all particular purposes. Configuration frameworks are then rather exceptional in the area of user-friendly thick clients (desktop applications). The situation is similar in the area of rich Internet applications that take advantage of a thick client at the architecture of a thin client such as Flash or Silverlight, used in this case. Modern universal Windows applications are kept up to date through Windows Store, but such an update may still take several days and reinstallation of the entire application is required. As shown above, creating a custom completely original framework for any specific needs is not difficult, and due to the acquired subsequent savings it is worthwhile in many cases.

References

[1] Š. Hubálovský and J. Šedivý, “Education of student’s project team cooperation using virtual communication supported by LMS system,” in *14th International Conference on Interactive Collaborative Learning (ICL2011) – 11th International Conference Virtual University (VU'11)*, Bratislava: Slovenská Technická Univerzita, 2011, pp. 456–459, ISBN 978-1-4577-1746-8.

[2] P. Voborník, *Universal Testing Environment*. Ph.D. dissertation, Faculty of Informatics and Management, University of Hradec Králové, Hradec Králové, Czech Republic, 2012. Available: http://download.petrvobornik.cz/docs/disertace.pdf.

[3] P. Voborník, “Universal Testing Environment as an External Tool of Moodle,” in *10th International Scientific Conference on Distance Learning in Applied Informatics (DiVAI 2014)*, Štúrovo, Slovakia: Wolters Kluwer, 2014, pp. 215–225, ISBN 978-80-7478-497-2.

[4] S. Jarzabek, P. Bassett, H. Zhang and W. Zhang, “XVCL: XML-based Variant Configuration Language,” in *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, Washington DC: IEEE Computer Society, 2003, pp. 810–811, ISBN 0-7695-1877-X.

[5] Z. Smištík, *Modern technology for developing web applications and their performance*. Thesis, Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic, 2008.

[6] P. Voborník, “Concept for development of large-scale applications through configuration frameworks,” in *Recent Advances on Systems, Signals, Control, Communications and Computers*, Budapest, Hungary, WSEAS Press, 2015, pp. 83–90, ISBN 978-1-61804-355-9, ISSN 1790-5117.

[7] S. Szénási, “Distributed Region Growing Algorithm for Medical Image Segmentation,” *International Journal of Circuits, Systems and Signal Processing*, vol. 8, no. 1, 2014, pp.173–181, ISSN 1998-4464.

[8] P. Voborník, “Effective object-relational mapping data transfer in the cloud computing,” in *Information Technology for Practice 2011*, Ostrava: VŠB-TUO, 2011, pp. 189–197, ISBN 978-80-248-2487-1.

[9] V. Strnadová, *Interpersonal communication*. Hradec Králové: Gaudeamus, 2011, 543 p., ISBN 978-80-7435-157-0.

[10] R. Ferdiana, “Agile Software Engineering Framework for Evaluating Mobile Application Development,” *International Journal of Scientific & Engineering Research*, vol. 3, issue 12, 2012, pp. 89–93.

[11] J. Liberty and D. Xie, *Programming C# 3.0*. Vol. 5, Sebastopol: O’Reilly Media, 2008, ISBN 978-0-596-52743-3.

[12] J. Bishop, *C# 3.0 Design Patterns*. Sebastopol: O'Reilly Media, 2007, ISBN 978-0-596-52773-0.

[13] I. Půdelka, *Aspect oriented programming and its support*. Thesis, Masaryk university, Brno, 2010.

[14] W. Schult and A. Polze, *Aspect-oriented programming with C# and .NET*. Washington, DC: IEEE Computer Society, 2002, pp. 241–248, ISBN 0-7695-1558-4.

[15] P. Voborník, “Tool and mechanisms for efficient transfer of data in cloud client-server applications,” in *Recent Advances on Systems, Signals, Control, Communications and Computers*, Budapest, Hungary, WSEAS Press, 2015, pp. 166–171, ISBN 978-1-61804-355-9, ISSN 1790-5117.

[16] J. Hilyard and S. Teilhet, *C# 3.0 Cookbook*. 3rd ed., Sebastopol: O’Reilly Media, 2007, ISBN 978-0-596-51610-9.

[17] J. Ghosh and R. Cameron, *Silverlight Recipes: A Problem Solution Approach*. New York: Apress, 2009, ISBN 978-1-4302-2435-8.

[18] J. Cardenosa, C. Gallardo and A. Martin, “Internationalization and localization after system development: A practical case,” in *Proceedings of the Fourth International Conference “Information Research and Applications” i.TECH 2006*, Varna, Bulgaria, Sofia: FOI-COMMERCE, 2006, pp. 207–214, ISBN 978-954-16-0036-8.

[19] T. Lammarsch, W. Aigner, A. Bertone, S. Miksch, T. Turic and J. Gärtner, “A Comparison of Programming Platforms for Interactive Visualization in Web Browser Based Applications,” in *International Conference Information Visualisation*, Washington DC, USA: IEEE Computer Society, 2008, ISBN 978-0-7695-3268-4.

[20] M. Moldaschl, *Rich internet application development*. Bachelor thesis, Vienna University of Economics and Business, Vienna, 2011.

[16] DotNetNuke – dotnetnuke.codeplex.com
[17] Joomla – www.joomla.org
[18] Drupal – www.drupal.org
[19] Moodle – www.moodle.cz

Petr Voborník was born in the Czech Republic in 1982. He received a master degree (Ing.) in Information Management and a doctoral degree (Ph.D.) in Information and Knowledge Management from the University of Hradec Králové in 2006 and 2012. He is now an Assistant Professor at the Department of Informatics, Faculty of Science, University of Hradec Králové. His current research interests include developing new algorithms and mini-languages for optimization of electronic testing for his Universal Testing Environment. He also participates in several researches as a programmer, he teaches and popularizes programming and he creates independent applications and games.