Mapping to the Windows Presentation Framework

This section maps the main IFML concepts to the .Net Windows Presentation Framework (WFP).

Windows Presentation Framework (WPF) is a part of .NET Framework by Microsoft that is meant to be the substitute of the old WinForms UI interface. It brings separation of concerns between interface and code-behind. This is made possible by detaching presentation defined using the XAML language from business logic written in C#.

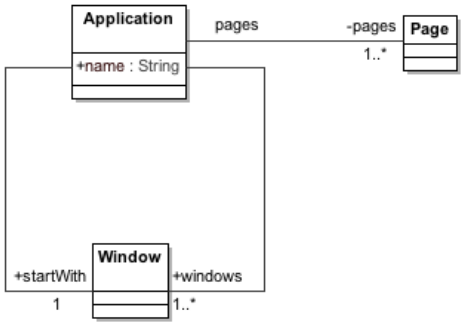

Figure 1: WPF metamodel, the Application element

In WPF the interface building blocks are nested. This generates a visual tree that is rendered by the framework.

The target application is modeled by the **Application** class which is the main container of all the elements of the model. It has a start window which is the first one to be opened at startup.

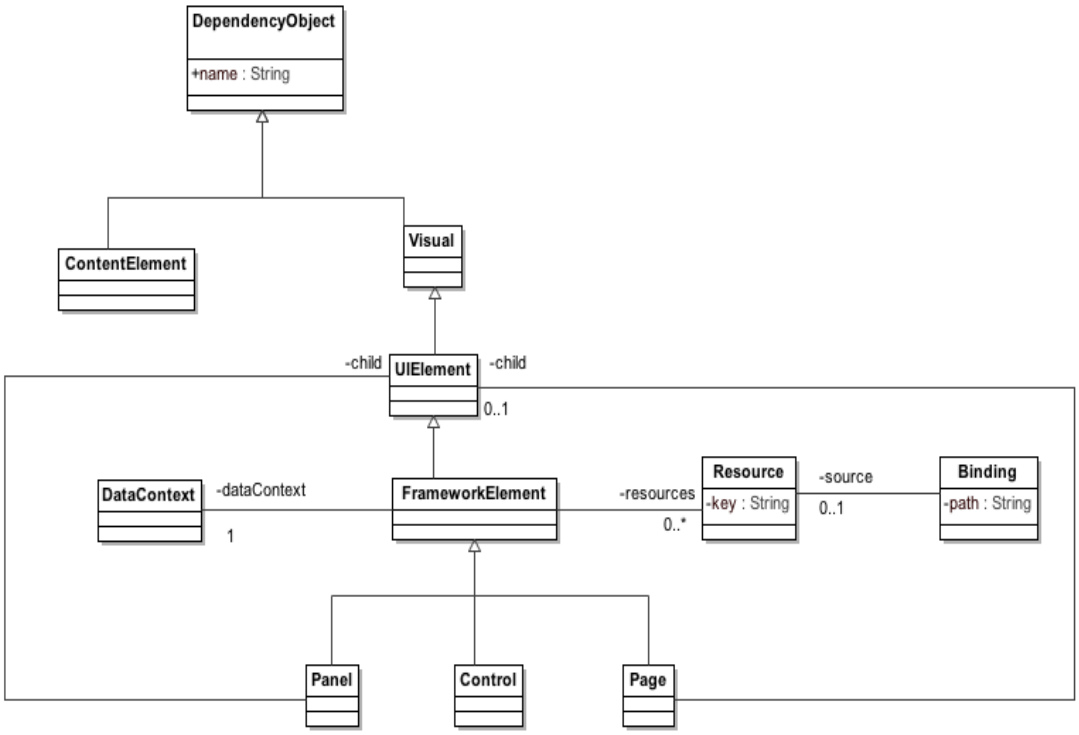

Figure 2: WPF metamodel, the DependencyObject element

All the visual objects inherit from **DependencyObject**, a class that allows the attachment of **DependencyProperty**. This lets define properties that may be shared among all the objects of the

framework and used as target for bindings.

DependencyObject can be split in two classes, **Visual** and **ContentElement**. Visuals elements are actually rendered by the framework, while **ContentElement**s are used to better define the layout of **Visual**s.

The main subclass of **Visual** is **UIElement** which is used as common superclass to define nesting among elements of the UI.

The main subclass of **UIElement** is **FrameworkElement** which is the one that allows to define **Resource**s and the **DataContext**. **Resource**s are objects related to the **FrameworkElement** organized as a dictionary; they are used by the framework to enhance and better define layout and behavior of the interface. **DataContext** can be associated through a **Binding** to another object to define the source of all the contained **Binding**s, not otherwise specified.

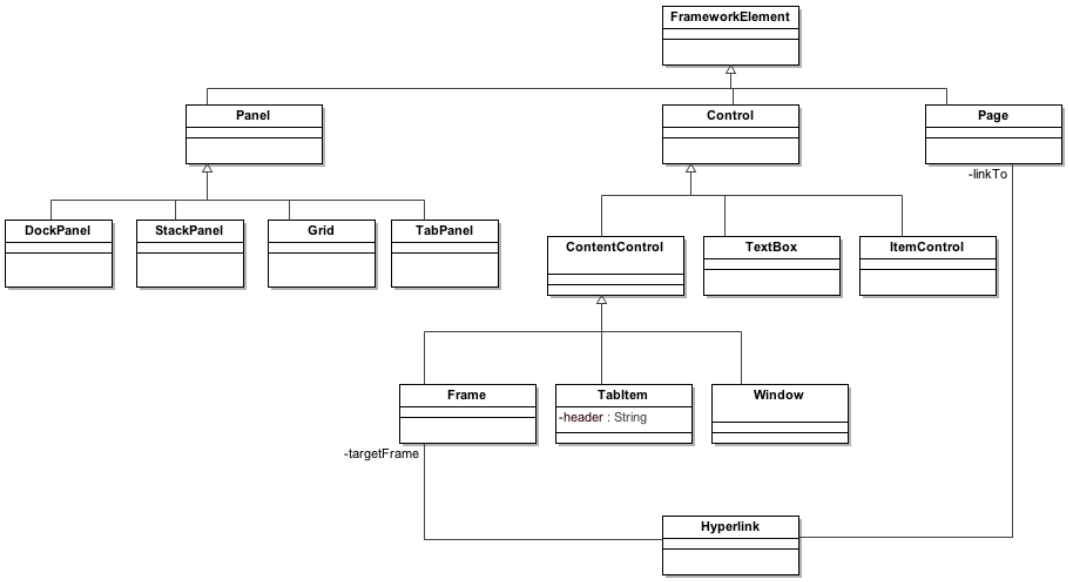

Figure 3: WPF metamodel, the FrameworkElement element

FrameworkElements can be divided in **Panel**s, **Page**s and **Control**s.

Panels are UI elements which can contain more than one child. They are classified by behavior:

- **DockPanel**: this container tries to minimize space wasting by expanding all the children to fit all the available space.
- **TabPanel**: it defines a XOR behavior (one by one), allowing to select the child to display through a tabbed header.
- **StackPanel**: it put all the children in a stack, queuing them one after another.
- **Grid:** it features a m by n grid in which all the children are placed. The coordinates of the cell in which the child resides is defined by the attached properties **Grid_Column** and **Grid_Row**.

Pages are one-child containers that allow navigation in a **Frame**.

Controls include **TextBox**es, **ContentControl**s and **ItemsControl**s.

ContentControls are **Window**s, **UserControl**s, **TabItem**s and **Frame**s.

- **Window**s are the outer containers of all **UIElement**s and have at most one child.
- **TabItem**s are one-child containers that allow to define the header used by a **TabPanel**.

- **Frame**s are controls that can dynamically navigate through **Page**s using **Hyperlink**s or explicit navigation.

ItemsControls are meant to dynamically define their children applying a template to items to be retrieved by an **ItemsSource**.

The IFML model is mapped to a WPF application as one window (the startup one) that contains a frame in which it's possible to navigate within pages.

All the first level ViewContainers are mapped to pages; to bypass the limitation related to the one-child nature of pages in WPF, ViewContainers with one child are mapped directly, while the ones with more children are mapped to pages with a grid as a child.

If there is at least one first level landmark ViewContainer, the main window does not contain directly the frame, but a grid with two children: the frame and a StackPanel that contains Hyperlinks to all the landmarked pages.

All the sub-ViewContainers are mapped to grids; otherwise, if they are XOR, they are mapped to TabPanels whose children are surrounded by TabItems.

All the ViewElementsEvents of type SelectEvent that reference a ViewContainer are mapped to a StackPanel containing Hyperlinks to all the pages linked by outgoing NavigationFlows.

List ViewComponents are mapped to ListBoxes: if they have a ViewElementEvent of type SelectEvent with an outgoing NavigationFlow that links to another ViewComponent, they are also mapped to a ViewSource bound to a ObjectObservableCollection and to a grid which DataContext is bound to the ViewSource current item.

Forms are mapped to grids; their fields are mapped to TextBox (SimpleField) or ComboBox (SelectField).

Finally since the WPF metamodel is a direct mapping of the entities that compose the .Net framework for desktop applications, a simple model to text transformation is needed for generating a working application.

Mapping to Java Swing

This section describes an example of mapping from IFML to Java Swing in order to model very simple Java-based desktop application.

Java Swing is a Model-View-Controller GUI framework for Java application. Thus it allows to develop desktop application in Java decoupling the data viewed from the interface from the user interface controls through which it is viewed.

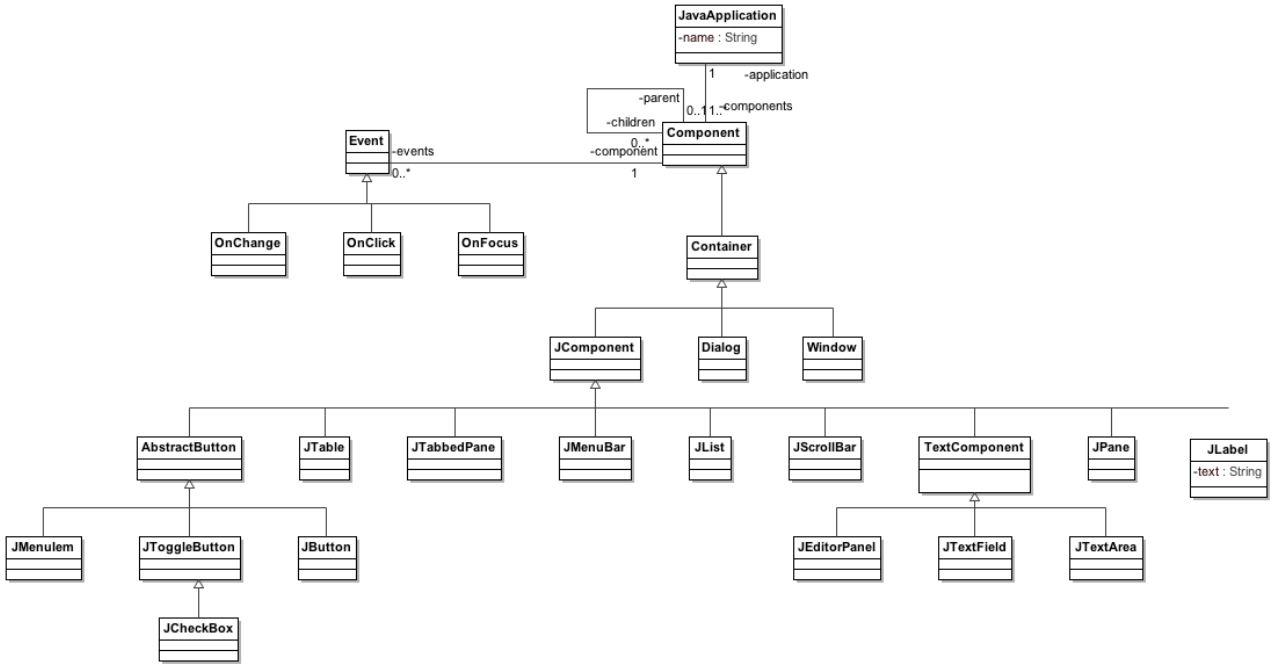

Figure 4: The Java Swing metamodel

The desktop application is described by the **JavaApplication** element, which contains all the **Component**s.

The **Component** element is the abstract description of the element of a graphical user interface. In particular a **Component** can have a set of child element and a set of **Event** used to enable the user's interaction. Furthermore an **Event** can be associated to a set of **Action**s

Every **Component** is a **Container.** In particular there are the **Window, Dialog**, **JComponent** elements. The first two are pure container while the last comprehends a set of elements that can contain other element or just show data.

The **JComponent** element is then specialized by a set of class that represent the actual GUI elements, for example there are: **AbstractButton,** that model the general button that is more specialized by the class **JToggleButton, JButton, JmenuItem; JTable,** that model a table, **JPane**, **JTabbedPane**, **JScrollBar**, **Jlist**, **Jlabel** and **TextComponent**, that represent the general component to edit text, which is further specialized by the class **JTextField**, **JTextArea** and JeditorPanel.

The IFML model is mapped to a **JavaApplication** element.

Each IFML::Window element is mapped to a **Window** element (in case of a modal window a **Dialog** is created instead).

Each not XOR sub-ViewContainer is mapped as a **JPane** (while a XOR container is mapped as a **JTabbedPane** with each of its child ViewContainer mapped as **JPane** element).

Forms are mapped as **JPane** elements, their fields are then mapped as **JTextField** (in case of SimpleField) or **JCheckBox** in case of multi selection field).

List are mapped as **JList** elements.

Details are mapped as **JTable** showing at each row an attribute of the DataBinding considered.

If events were defined, the corresponding **Event** is created and associated to the correct **Component.** In particular, in case of Select and Submit a **JButton** is created in order to trigger the event. If an Action was defined, a element of type **Action** will be created.

If one or more ViewContainer marked as "landmark" exist, a **JMenuBar** element will be created in each **Window**, containing all the **JMenuItem** element linking to the landmark ViewContainers.

Mapping to HTML

This section describes an example of mapping from IFML to HTML in order to model a very simple web application.

The web application is modeled by the **WebSite** class, which is the main container of all the other elements. In particular a WebSite is composed by a set of **Pages**. Then the metamodel describes in details the structure of each element.

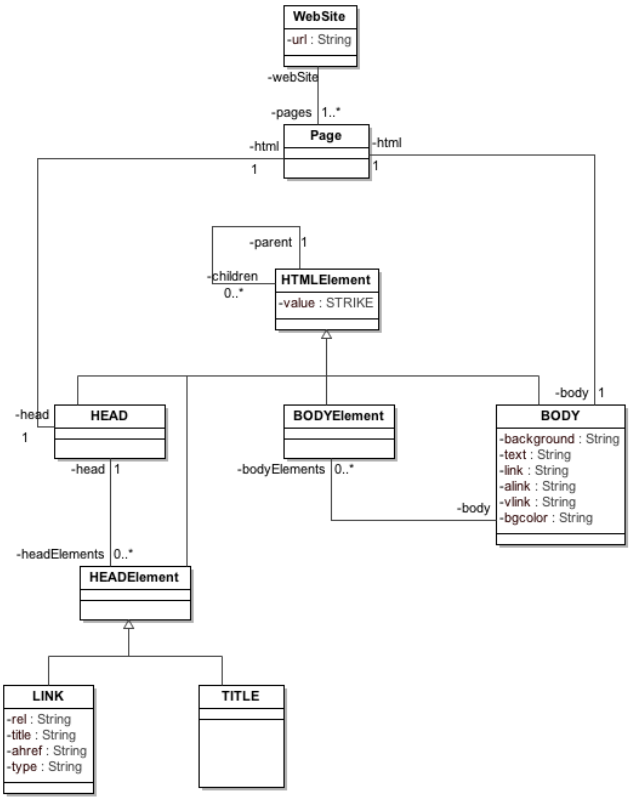

Figure 5: HTML metamodel, the Page and Head element

A Page is composed by a **HEAD** and a **BODY** (represeniting the <head> and <body> tags), the HEAD contains a set of **HEADElement** while the **BODY** a set of **BODYElement**, both of them inhertis from the general class **HTMLElement** and are abstraction of the concrete html tag.

The **HEADElement** comprehends the **TITLE** and **LINK** tags, while the **BODYElement** comprehend all the html tags used for creating web pages (**P**, **TABLE**, **FORM, DIV, A** etc..).

In order to allow the nesting of tags, the **HTMLElement** class has a reference to a set of children **HTMLElement.**

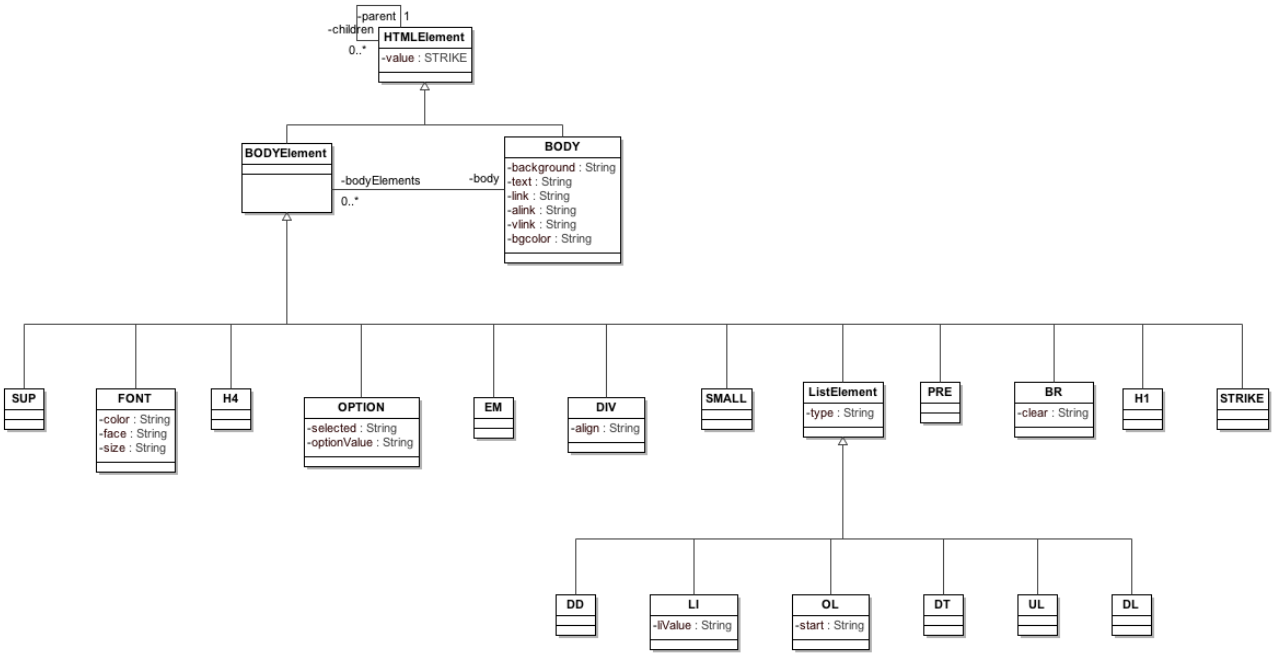

Figure 6: HTML metamodel, a fragment of the BODY element

The IFML model is mapped to a **WebSite** element.

Every first level ViewContainer is mapped to a **Page** element, in particular the one marked as "home" will be named "index".

Each sub-ViewContainer will be mapped to a **DIV** element.

Each NavigationFlow not associated to a SystemEvent is mapped to a **A** element. If an Action is present, its name will be appended at the end of the link.

Forms are mapped into **FORM** element and their fields are mapped to corresponding **INPUT** elements.

Details are mapped into a **UL – LI** elements, in which each list item is a attribute of the data binding considered.

Lists are mapped into **TABLE**, in which the first row is composed by the field of the corresponding data binding. If a SelectEvent is associated to the component, then a last column is added which contains a **A** element.

If one or more ViewContainer marked as "landmark" exist, a **DIV** element containing all the **A** element linking to the landmark ViewContainers will be created in each **Page**.