Modelowanie i Analiza Systemów informacyjnych (MAS)

dr inż. Mariusz Trzaska
mtrzaska@pjwstk.edu.pl

Wykład 12

Tworzenie graficznych interfejsów użytkownika

Zagadnienia

- Wprowadzenie
- Biblioteki GUI
  - dla Javy,
  - dla C++,
  - Dla C#.
- Własna biblioteka GUI
- Implementacja GUI
  - Ręczne pisanie kodu,
  - Edytory wizualne,
  - Podejście deklaratywne.
- Podsumowanie

Wykorzystano: www.wikipedia.org,

Biblioteki do tworzenia GUI

- Wykorzystanie bibliotek GUI
  - Dostarczanych z językiem,
  - Firm trzecich,
  - Samodzielne stworzenie.
- Kryteria wyboru
  - Łatwość użycia,
  - Modyfikowalność,
  - Wydajność,
  - Przenaszalność,
  - Cena,
  - Estetyka.

Biblioteki dla języka Java

- Możliwości
  - AWT,
  - Swing,
  - SWT,
  - Java FX
- Ocena dostępnych rozwiązań
  - Łatwość użycia,
  - Modyfikowalność,
  - Przenaszalność,
  - Estetyka.

Biblioteki dla języka Java (2)

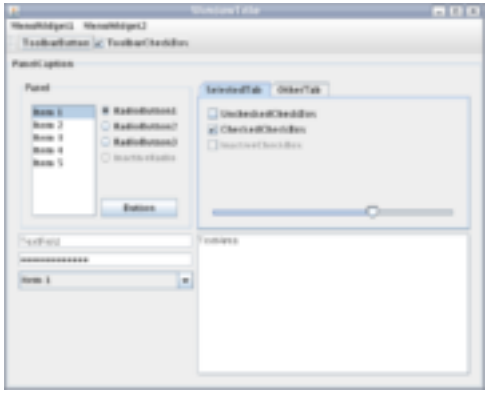
- AWT (*Abstract Window Toolkit*; *http://java.sun.com/products/jdk/awt/*),
  - Rok wydania: 1995,
  - Wykorzystanie natywnych widgetów danej platformy (różny wygląd na różnych OS),
  - Podstawowy zestaw widgetów (przyciski, pola tekstowe, menu, itp.),
  - Obsługa zdarzeń,
  - Interfejs pomiędzy natywnym systemem, a aplikacją Java,
  - Managery rozkładu,
  - Obsługa schowka i *Drag&Drop*,

Biblioteki dla języka Java (3)

- AWT (Abstract Window Toolkit) – c.d.,
  - Dostęp do urządzeń wejściowych takich jak mysz czy klawiatura,
  - Natywny interfejs AWT umożliwiający generowanie obrazu bezpośrednio do powierzchni widgetu,
  - Dostęp do zasobnika systemowego (nie na wszystkich systemach),
  - Zdolność do uruchamiania niektórych aplikacji systemowych z poziomu Javy, np. obsługa poczty czy przeglądarka internetowa.

Biblioteki dla języka Java (4)

- Swing (http://www.javaswing.net/,),
  - Rok wydania: 1997,
  - Wygląd oraz zachowanie widgetów określone przez Javę (taki sam wygląd i zachowanie na różnych OS). Są rysowane przy wykorzystaniu Java2D,
  - Rozszerzony (względem AWT) zestaw widgetów,
  - *Look&Feel* aplikacji korzystającej ze Swinga jest określany przez wybrany temat graficzny (*pluggable look and feel*),
  - Niezależność od platformy,
  - Rozszerzalność,
  - Zorientowany na komponenty,



Biblioteki dla języka Java (5)

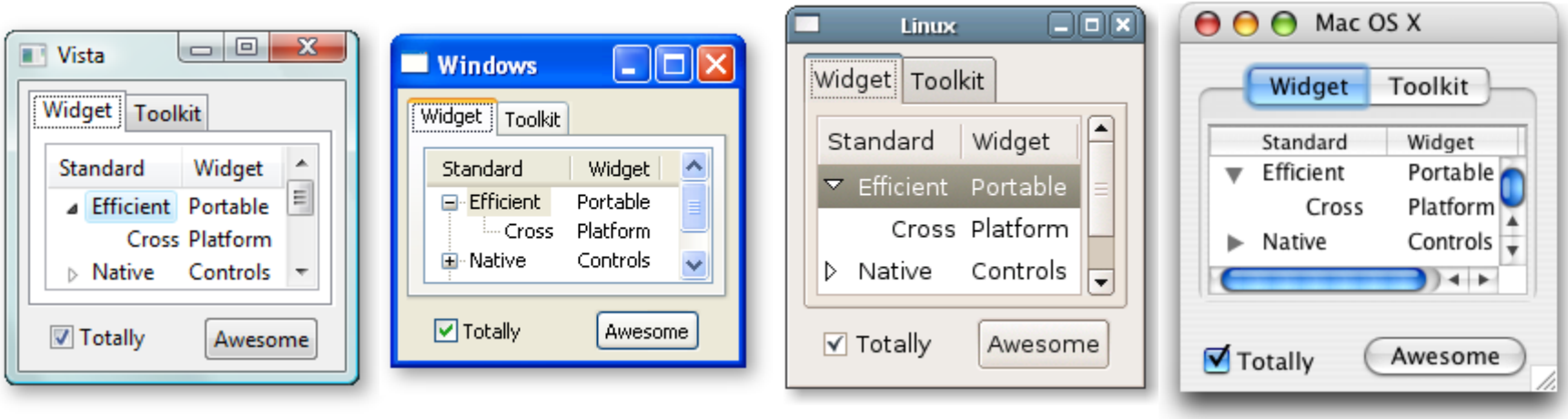
- Swing (http://www.javaswing.net/,) – c.d.,
  - Łatwa modyfikowalność:
    - Wykorzystanie gotowych elementów przy rysowaniu: *border*, *inset*, *decorations*,
    - Łatwe określanie własności (np. *border*),
    - Renderers,
  - Konfugurowalność (m. in. w czasie wykonania – *run-time*),
  - „Lekki" *UI*. Wykorzystanie własnych mechanizmów renderujących.
  - Częściowe korzystanie z AWT, m.in. `component.paint()`,

Biblioteki dla języka Java (6)

- Swing (http://www.javaswing.net/,) – c.d.,
  - Wykorzystanie MVC (*Model-View-Controller*). Większość widgetów ma przypisany własny model (określony przez interfejsy języka Java), który definiuje sposób postępowania z danymi.
  - Biblioteka dostarcza podstawową implementację różnych modeli.
  - Zdarzenia:
    - „fizyczne”, np. kliknięcie przyciskiem myszy,
    - „logiczne” (semantyczne, związane z modelem), np. dodanie wiersza danych do tabeli, zmiana zaznaczenia wiersza. Popularnym zdarzeniem jest `ActionPerformed`.

Biblioteki dla języka Java (7)

- SWT (Standard Widget Toolkit; *http://www.eclipse.org/swt/*)
  - Stworzony przez IBM, a aktualnie rozwijany przez fundację Eclipse,
  - Wykorzystuje natywne możliwości systemu operacyjnego do rysowania kontrolek,



Biblioteki dla języka Java (8)

- SWT (Standard Widget Toolkit; http://www.eclipse.org/swt/) – c.d.
  - Eclipse Foundation: *Celem SWT jest dostarczanie wspólnego API udostępniającego natywne widgety na różnych platformach.*
  - Wydajność względem Swing:
    - Szybszy przy rysowaniu,
    - Wolniejszy przy pobieraniu danych (wykorzystuje JNI - *Java Native Interface*).
  - Programy korzystające z SWT są przenośne, ale wymagają dedykowanej implementacji SWT dla każdego OS (innych nawet dla Windows x86 oraz x64). Nie zawsze jest ona dostępna dla wszystkich platform na których istnieje JVM.
  - Bardzo dobra praca pod Windows. Na innych platformach potencjalne problemy.

Biblioteki dla języka Java (9)

- SWT (Standard Widget Toolkit; http://www.eclipse.org/swt/) – c.d.
  - W przypadku gdy widgety na określonym OS nie udostępniają określonej funkcjonalności, SWT implementuje ją samodzielnie.
  - SWT nie wykorzystuje MVC, ale daje możliwość współpracy z biblioteką JFace (), która wspiera to podejście.
  - Ze względu na użycie natywnych widgetów, ich dostosowanie nie zawsze jest łatwe.
  - Konieczność ręcznego zwalniania zasobów wykorzystywanych przez widget: metoda `.dispose()`. W praktyce są to zwykle podklasy `Resource` takie jak `Image`, `Color` oraz `Font`.

Biblioteki dla języka Java (10)

- JavaFX
  - pierwsza wersja opublikowana 2008-12,
  - aplikacje desktopowe oraz RIA (*Rich Internet Applications*),
  - docelowo ma zastąpić Swing,
  - graficzny edytor oraz plik FXML (wariant XML),
  - klasy `Stage`, `Scene`, `Node` (części grafu tworzącego scenę),

Biblioteki dla języka Java (11)

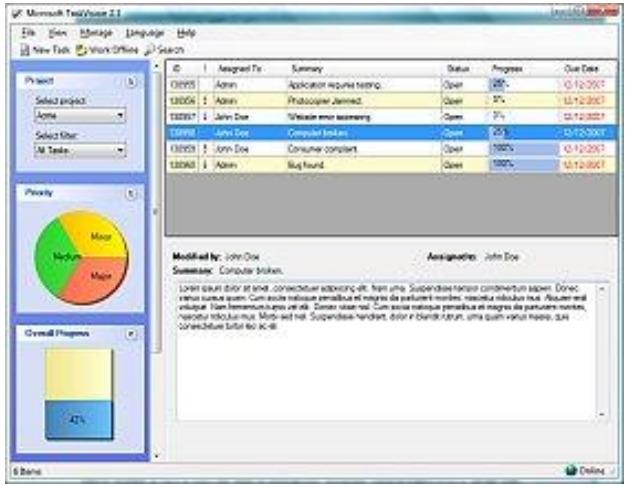
- JavaFX – *kont.*
  - sterowana zdarzeniami, ale z uwzględnieniem powiązanych własności (*bindable properties*),
  - zmiany wyglądu z CSS,
  - efekty specjalne (cienie, odbicia, rozmycie),
  - łatwe animacje,
  - wsparcie dla grafiki 3D.

Biblioteki dla języka C/C++

- Microsoft
  - Win32 API,
    - Bardzo niskopoziomowy sposób pracy,
    - Największa wydajność,
    - Wykorzystuje: GDI (*Graphics Device Interface*), *Common Dialog Box Library*, *Common Control Library*.
  - MFC (*Microsoft Foundation Class Library*),
    - „opakowuje” Win32 API,
    - Sposób pracy bardziej obiektowy, ale nadal pozostawia nieco do życzenia,
  - Produkty Borlanda: OWL (*Object Windows Library*), VCL (*Visual Component Library*).

Biblioteki dla języka C/C++ (2)

- Microsoft – c.d.
  - WinForms (*Windows Forms*; *http://windowsclient.net/*),
    - Działa również z MS C#,
    - Rozprowadzany razem z MS .NET,
    - Opakowuje Win32 API, ale robi to duuuużo lepiej niż MFC,
    - Bardzo duża liczba kontrolek,
    - Rozszerzany przez *User Interface Process Application Block – Version 2.0* (wprowadza m.in. odpowiednik MVC).



Biblioteki dla języka C/C++ (3)

- Microsoft – c.d.
  - WPF (*Windows Presentation Foundation*; http://windowsclient.net/).
    - Działa również z MS C#,
    - Najlepsze możliwości tworzenia GUI o unikalnym wyglądzie:
      - Tradycyjne kontrolki z tematami graficznymi,
      - Skomplikowany tekst,
      - Obrazki,
      - Wideo,
      - Grafika 2D,
      - Grafika 3D.
    - Bardzo duża liczba unikalnych kontrolek.

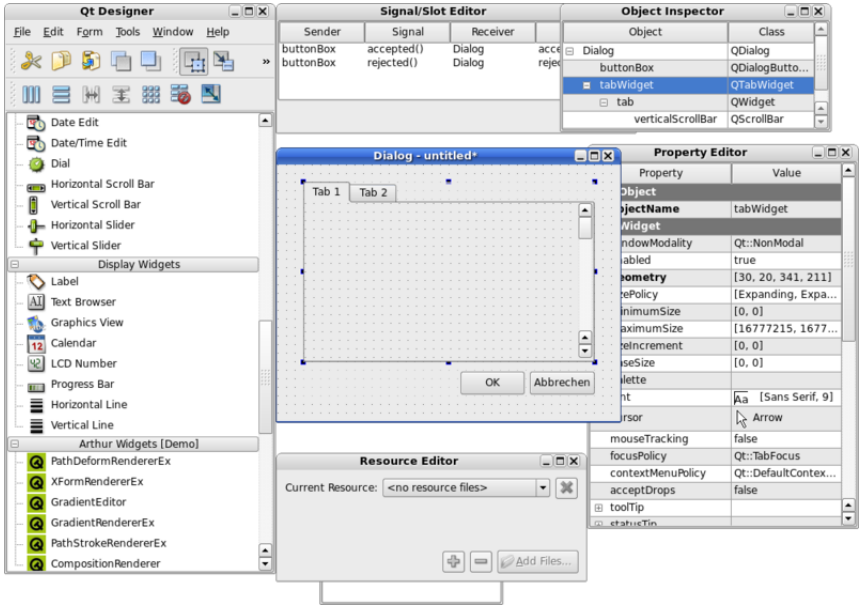
Biblioteki dla języka C/C++ (4)

- Microsoft – c.d.
  - WPF (*Windows Presentation Foundation*; http://windowsclient.net/) – c.d.
    - Umożliwia wykorzystanie deklaratywnego podejścia do tworzenia GUI – język XAML,
    - Nowe narzędzia dla grafików: *MS Expression Blend*.

	WinForms	PDF	WinForms + GDI	WMP	Direct3D	WPF/ *Windows Store* (Metro)
Forms, Controls	X		X			X
Complex text		X				X
Images			X			X
Video / Audio				X		X
2D Graphics			X			X
3D Graphics					X	X

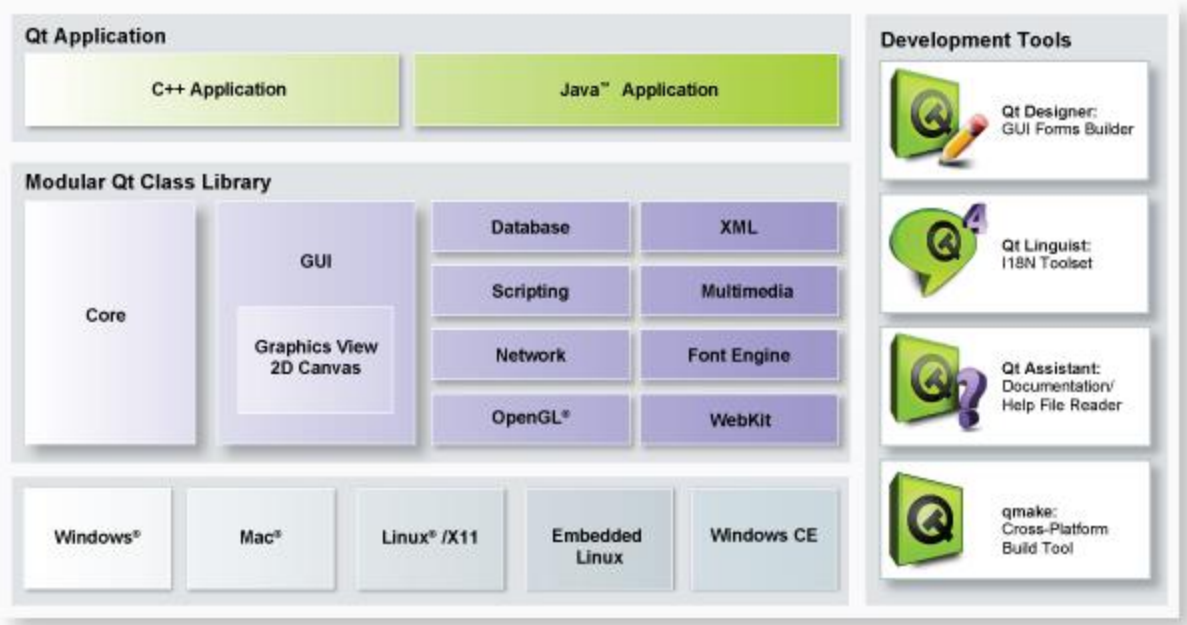
Biblioteki dla języka C/C++ (5)

- Qt (*http://www.trolltech.com/products/qt/*)
  - Pierwsze prace zostały rozpoczęte w 1991,
  - Przenaszalność (Unix, Linux, MacOS X, Windows, Windows CE, Java),
  - Wykorzystanie niestandardowych dodatków do C++, które są przetwarzane przez pre-procesor,
  - Wsparcie dla:
    - Wielu języków,
    - Baz danych SQL,
    - Przetwarzania XML,
    - Zarządzania wątkami,
    - Technologii sieciowych
    - Pracy z plikami.



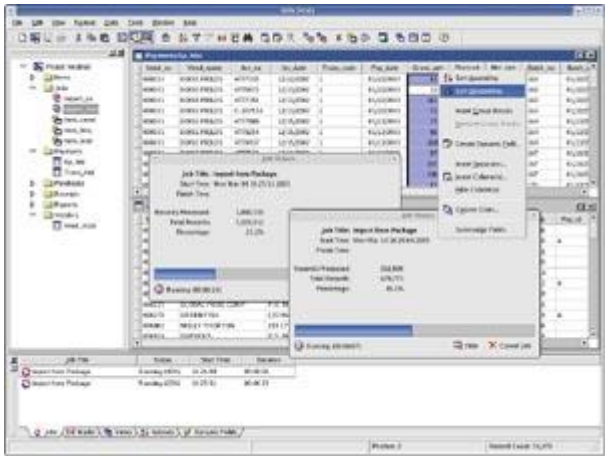
Biblioteki dla języka C/C++ (6)

- Qt – c.d.
  - Ostatnie wersje wykorzystują mechanizmy OS do wyświetlania kontrolek,
  - Kompilator meta dostarcza informacje, które nie są normalnie dostępne w C++. Dzięki temu biblioteka oferuje wsparcie m.in. dla asynchronicznego wołania funkcji.
  - Aplikacje, m.in:
    - KDE,
    - Opera,
    - Google Earth,
    - Skype,
    - Qtopia,
    - Photoshop Elements,
    - Virtual Box.



Biblioteki dla języka C/C++ (7)

- wxWidgets (*http://wxwidgets.org/*),
  - Pierwsze wydanie: 1992.
  - Przenaszalność: Mac OS, Linux/Unix (X11, Motif, and GTK+), OpenVMS, OS/2, AmigaOS.
  - Wykorzystuje natywne możliwości OS,
  - Zarządzanie zasobami,
  - Dodatkowe możliwości, m.in:
    - Praca z OpenGL,
    - Dostęp do baz danych ODBC,
    - Komunikacja sieciowa oparta na socket'ach.



Biblioteki dla języka C/C++ (8)

- GTK+ (*The GIMP Toolkit*, *http://www.gtk.org/*),
  - Przenaszalność,
  - Pierwotnie stworzony dla GIMP'a,

  - Różne silniki do wyświetlania kontrolek (m.in. emulujące popularne *L&F*),
  - Nowa wersja (GTK+ 2) zawiera ulepszony rendering kontrolek, nowy silnik tematów graficznych, wsparcie dla Unicode.
- Duuużo innych rozwiązań…

Biblioteki dla języka C#

- Microsoft WinForms,
- Microsoft WPF (*Windows Presentation Foundation*)
- wxWidgets (wiązanie do C#),
- GTK+ (wiązanie do C#),
- Mono (otwarta implementacja .NET włączając WinForms),
- Xamarin (*Forms* oraz *Native*).

Własna biblioteka GUI

- Dlaczego nie skorzystać z istniejących rozwiązań?
- Wybór modelu:
  - „Fizyczny” zdarzeniowy,
  - „Logiczny” (semantyczny) zdarzeniowy,
  - Mieszany: „fizyczno-semantyczny”,
  - Inny?
- Zestaw *widgetów*,
- Sposób rysowania kontrolek,
- Łączenie zdarzeń z ich obsługą.

Własna biblioteka GUI (2)

- Zwykle, w czasie implementacji takiej biblioteki, występuje konieczność pracy na bardzo niskim poziomie abstrakcji:
  - Przechwytywanie niskopoziomowych zdarzeń systemu operacyjnego: wciśnięcie/zwolnienie klawisza klawiatury, zmiana położenia kursora myszy, wciśnięcie/zwolnienie klawisza myszy.
  - Rysowanie kontrolek w oparciu o podstawowe elementy graficzne (linia, punkt, okrąg, bitmapa).
- Praca dość trudna. W miarę możliwości, lepiej używać istniejących rozwiązań.

Implementacja GUI

- Ręczne pisanie kodu źródłowego,
- Wykorzystanie specjalizowanego edytora,
- Podejście deklaratywne.

Implementacja GUI – ręczne pisanie kodu

- Największa kontrola nad efektem końcowym w kategoriach:
  - Funkcjonalności,
  - Wydajności,
  - Przenaszalności,
  - Użyteczności,
  - Estetyki.
- Niestety zwykle wymaga dość dużej wiedzy dotyczącej konkretnej platformy/biblioteki.

Implementacja GUI – ręczne pisanie kodu (2)

- Utrudnione (pod pewnymi względami) nanoszenie poprawek.
- Stosunkowo niewielka szybkość tworzenia GUI.
- Duża podatność na błędy.
- Metoda w większości wypadków nie powinna być stosowana.
- Chociaż czasami jest jedynym sposobem osiągnięcia zamierzonego efektu.

Implementacja GUI – edytory graficzne

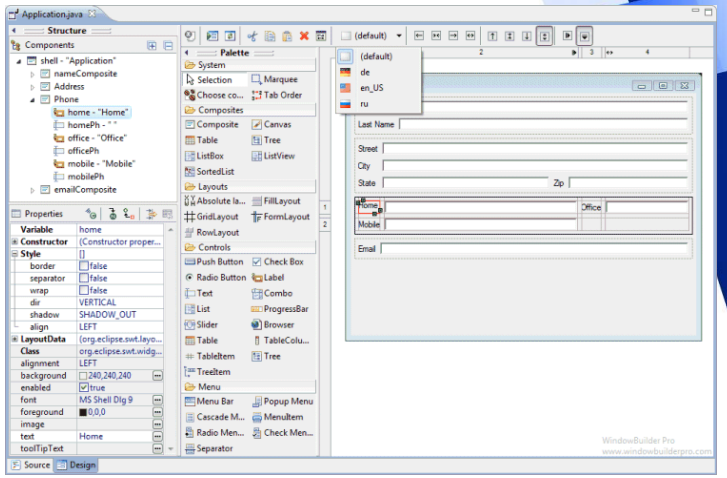
- Różna jakość istniejących rozwiązań:
  - Tylko generowanie. Ręcznie naniesione poprawki są tracone przy ponownym generowaniu kodu.
  - Praca dwukierunkowa (*bi-directional code generation*). Edytor odzwierciedla w projekcie graficznym, ręcznie (w kodzie źródłowym) naniesione zmiany.
- Duża szybkość tworzenia GUI.
- Mała podatność na błędy.
- Natychmiastowy podgląd efektu końcowego.

Implementacja GUI – edytory graficzne (2)

- Wydaje się, że wykorzystanie dobrego edytora GUI jest najlepszym sposobem tworzenia interfejsu.
- Rozwiązania warte polecenia:
  - Microsoft Visual Studio:
    - Języki: C#, C++,
    - Biblioteki: WinForms, WPF, MFC
  - NetBeans IDE

Implementacja GUI – edytory graficzne (3)

- Rozwiązania warte polecenia – c.d.:
  - Eclipse WindowBuilder Pro
    - SWT, Swing, GWT Designer
    - ~~Płatny (329,00 USD)~~



    - https://www.eclipse.org/windowbuilder/
  - IntelliJ IDEA IDE
    - *http://www.jetbrains.com/idea/*
    - Wersja bezpłatna (*Community*)

Implementacja GUI – podejście deklaratywne

- A może „komputer” sam stworzy dla nas GUI?
- Koncentrujemy się na tym:
  - co jest do zrobienia,
  - a nie jak to zrobić?
- Różne stopnie zautomatyzowania:
  - Deklaratywność **semantyczna**: określamy, które elementy modelu danych mają mieć *widgety*,
  - Deklaratywność **komponentowa**: definiujemy jakie *widgety* maja być stworzone.

Implementacja GUI – podejście deklaratywne (2)

- Wydaje się, że taki sposób pracy będzie zyskiwał na popularności.
- Szczególnie w przypadku GUI, które:
  - są dość standardowe (wygląd, funkcjonalność)
  - nie muszą być szczególnie dopracowane.
- Szukanie złotego środka pomiędzy:
  - Zaangażowaniem programisty
  - Generycznością (uniwersalnością) rozwiązania.

Implementacja GUI – podejście deklaratywne (3)

- Istniejące rozwiązania komercyjne, np. Microsoft XAML (*Extensible Application Markup Language*, kiedyś *Extensible Avalon Markup Language*),
  - Mocno wykorzystywany w .NET, a szczególnie w WPF i *Windows Store Apps* oraz w Xamarin;
  - Definiowanie:
    - Elementów GUI: 2D, 3D,
    - Łączenia do danych (*data binding*),
    - Zdarzeń,
    - Efektów specjalnych: obrót, animacja
  - Bezpośrednie przełożenie na kod C#.
  - Programista i tak musi się sporo napracować definiując te elementy.

Implementacja GUI – podejście deklaratywne (4)

- Microsoft XAML – c.d.
  - Przykładowy kod XAML

```
<Button Content="Click me">
        <Button.Margin>
                <Thickness Left="10" Top="20" Right="10" Bottom="30"/>
        </Button.Margin>
</Button>

<Page
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="MyNamespace.MyPageCode">

        <Button Click="ClickHandler" >Click Me!</Button>
</Page>
```

Implementacja GUI – podejście deklaratywne (5)

- Bardziej użyteczne podejście mogłoby być opisane poniższymi założeniami:
  - Programista określa elementy modelu danych (klasy), które powinny mieć stworzone GUI:
    - Atrybuty,
    - Metody.
  - System generuje formularze (okna z kontrolkami) umożliwiające tworzenie nowych instancji danych, ich edycje, itp.
  - Ewentualnie doprecyzowanie dalszych szczegółów:
    - Etykiety tekstowe,
    - Rodzaje konkretnych widgetów,
    - ...

Podsumowanie

- Współczesne języki programowania są dostarczane z dość wygodnymi bibliotekami do tworzenia GUI.
- Oprócz nich programista może wykorzystywać rozwiązania innych firm, które czasami są lepsze (łatwiejsze w użyciu, szybsze, przenośne, itp.).
- Graficzny Interfejs Użytkownika może być implementowany korzystając z kilku sposobów:
  - Ręcznie,
  - W specjalizowanym edytorze,
  - Deklaratywnie.
- Wydaje się, że aktualnie najatrakcyjniejsze jest podejście zakładające wykorzystanie edytora.
- Niemniej, w przyszłości może się to zmienić na rzecz rozwiązań deklaratywnych.