Inżynieria oprogramowania, SUM Informatyka

PROJEKT DYDAKT
HARMONIX

Opis sposobu realizacji oraz możliwości rozbudowy

AUTORZY
Mateusz Bilski (mateusz.bilski@webmasta.pl)
Michał Furman (michal.furman@webmasta.pl)

PROWADZĄCY:
mgr Witold Rakoczy

AGH 2008

Spis treści

1. Środowisko programistyczne

1.1. Adobe Flex Builder 3 oraz Flex SDK

Do realizacji framework'u Harmonix wykorzystano narzędzie Adobe Flex Builder w wersji Professional Edition – Educational, która to jest dostępna w wersji darmowej dla studentów uczelni wyższych.

Jako SDK użyto wersji Adobe Flex SDK 3.3.

Powyższe narzędzia i biblioteki zostały pobrane odpowiednio z:

- **Adobe Flex SDK 3.3** (http://www.adobe.com/cfusion/entitlement/index.cfm?e=flex3sdk)
- **Adobe Flex Builder 3** (http://www.adobe.com/cfusion/entitlement/index.cfm?e=flexbuilder3)

Numer seryjny darmowej wersji studenckiej można uzyskać poprzez wypełnienie odpowiedniego formularza na stronach Adobe:

- **Adobe Flex Builder 3 Pro for Education** (https://freeriatools.adobe.com/flex/)

1.2. Zewnętrzne biblioteki

Do wszelkich animacji we framework'u Harmonix została użyta darmowa biblioteka Tweener, a dokładnie wersja 1.31.74 dla ActionScript 3.0. Biblioteka ta oferuje duże możliwości kontroli animacji przy bardzo intuicyjnej obsłudze.

Do pobrania z poniższej strony:

- **Tweener** (http://code.google.com/p/tweener/)

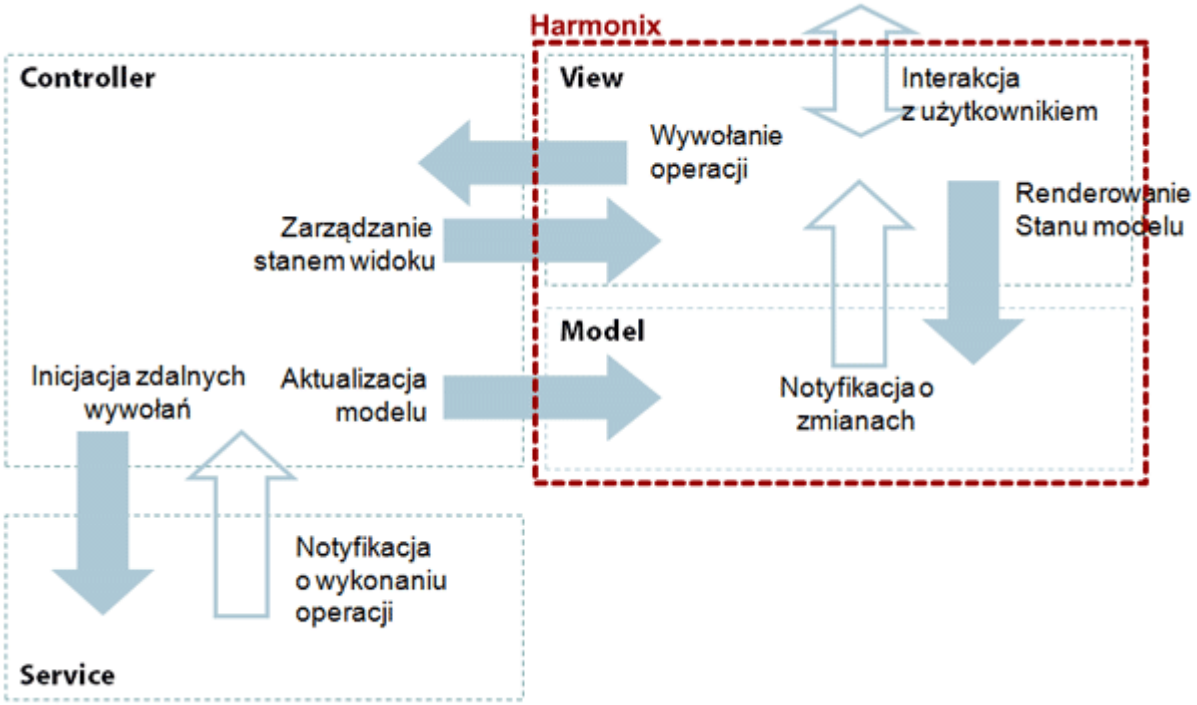
2. Koncepcja rozwiązania

2.1. Rozwiązanie w oparciu o mechanizmy framework'u Flex i modelu MVC

Większość aplikacji wykonanych w technologii Flex opiera się na modelu MVC i wykorzystuje mikroarchitektury takie jak Adobe Cairngorm czy PureMVC, dlatego też framework Harmonix był tworzony z myślą o wykorzystaniu go w takiej właśnie architekturze.

Harmonix swoim zakresem obejmuje renderowanie danych z modelu oraz interakcję z użytkownikiem. Obsługuje logikę w obrębie swoich komponentów, czyli wykonuje operację na danych lub obiektach wizualnych, które renderuje.

Poniższy wykres dokładnie obrazuje funkcje modelu MVC w aplikacjach Flex'owych i zakres, który obejmuje biblioteka Harmonix:



2.1.1. View

Rolę widoku w bibliotece Harmonix spełniają kontenery, czyli komponenty wizualne renderujące inne komponenty (tzw. *dzieci)* lub dostarczone kolekcje danych.

2.1.2. Model i bindowanie danych

Renderowane dane w komponentach biblioteki Harmonix zostają dostarczone przez referencje do kolekcji danych odpowiedniego typu. Dane są bindowane do kontenera, dzięki czemu nie ma konieczności dostarczania logiki obsługującej zmiany w kolekcji, ponieważ kontenery w bibliotece Harmonix są „świadome" zmian w kolekcjach danych, które renderują. Rozwiązanie takie jest zaczerpnięte bezpośrednio z samego framework'u Flex'a, który w dokładnie ten sam sposób obsługuje zmiany danych w komponentach takich jak *DataGrid* czy *List*.

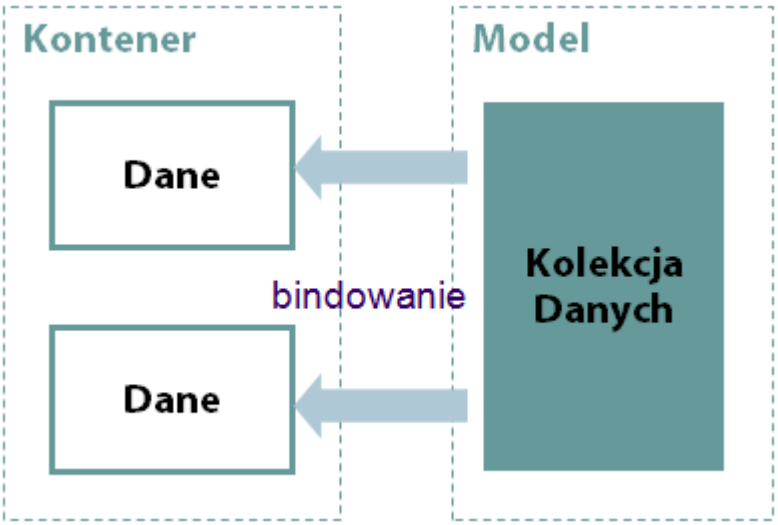
2.1.3. Controller

Istnieje możliwość przeciążania kluczowych metod w komponentach biblioteki Harmonix w celu delegacji operacji na danych do kontrolera w rozumieniu modelu MVC. Implementacja logiki w komponentach pozwala na użycie komponentów typu kontenery danych i obiektów wizualnych bez konieczności implementacji dodatkowego kontrolera, ponieważ ta zawiera już podstawowe operacje na danych, typu dodawanie lub usuwanie elementów.

2.2. Główne mechanizmy

Poniższe punkty objaśniają główne składowe biblioteki Harmonix, dzięki którym możliwe jest wykonanie założeń projektowych (*dokument „Cele i zadania projektu"*).

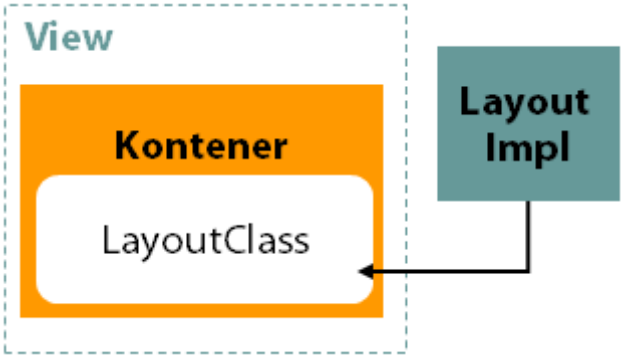
2.2.1. Kontener danych



Kontener danych to komponent (a dokładniej kompozyt zawierający inne obiekty) wizualny, za którego zawartość odpowiadają dostarczone dane.

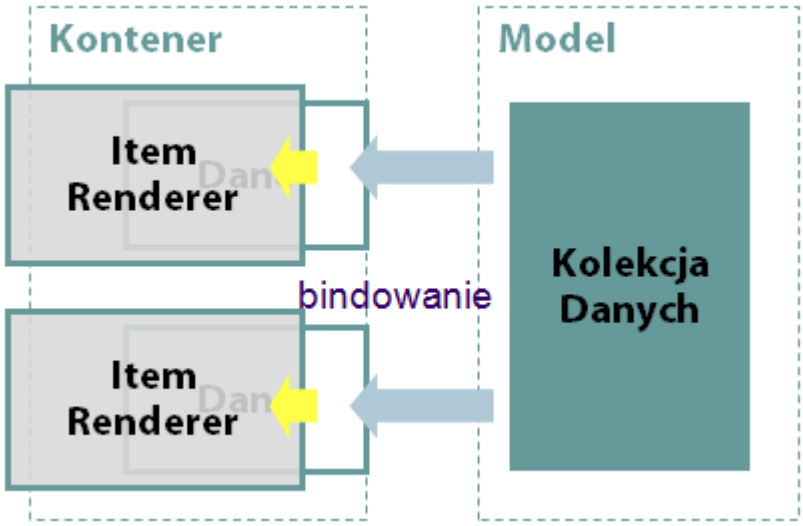
Rozwiązanie takie pozwala na prosty i intuicyjny sposób pokazywania danych, ich zmian w czasie rzeczywistym (z wykorzystaniem mechanizmu bindowania, który odpowiada za informowaniu o zmianach w kolekcji i automatyczne ponowne renderowanie aktualnych danych).

2.2.2. Layout



Kontenery w bibliotece Harmonix obsługują layout'y, czyli schematy renderowania obiektów wizualnych. Layout'y pozwalają na zmianę sposobu wyświetlania (przestrzennego ułożenia obiektów) dzieci. Layout'y mogą być podmieniane w kontenerze w czasie rzeczywistym co daje użytkownikowi możliwość wyboru sposobu wyświetlania obiektów zgodnie z jego upodobaniami.

2.2.3. ItemRender



ItemRenderer to we framwork'u Flex obiekt wizualny odpowiadający za renderowanie danych. Na dokładnie takiej samej zasadzie działają renderery danych w Harmonix. Każdy obiekt kolekcji danych ma swoją kopię ItemRenderer'a,

który odpowiada za wyświetlanie jego zawartości. Item renderer podobnie jak layout może być podmieniany w czasie rzeczywistym (runtime).

2.2.4. Drag & Drop

Do stworzenia operacji drag & drop zostały wykorzystane mechanizmy z Flex'a, ponieważ każdy komponent z założenia obsługuje te operacje, więc komponenty (kontenery) Harmonix implementują w odpowiedni sposób 2 z 3 etapów draggowania, czyli inicjację i upuszczanie (pominięto środkowy etap, czyli przeciąganie, które zostanie domyślne).

2.2.5. Schowek

Schowek został rozwiązany jako kolekcja dowolnych danych (wykorzystany fakt, że kolekcje we Flex'e nie są typowane), przy czym każda operacja kopiowania powoduje wyczyszczenie danych i umieszczenie w kolekcji nowych. Dodatkowo kontenery danych i obiektów wizualnych w framework'u Harmonix obsługują skróty klawiszowe, typowe dla operacji schowka, czyli *ctrl* + ***x***, *ctrl* + *c* oraz *ctrl* + *v*.

3. Sposób realizacji

3.1. Kontenery

3.1.1. LayoutContainer

Do realizacji głównego kontenera we framework'u Harmonix została utworzona klasa *LayoutContainer*. Dziedziczy ona bezpośrednio z Flex'owej *mx.core.LayoutContainer*, która daje możliwość ustawienia własnych reguł rozmieszczania graficznych dzieci kontenera, poprzez ustawienie atrybutu layoutObject.

Jako obiekt layout'u możemy ustawić dowolne klasy dziedziczące z klasy *mx.containers.utilityClasses.Layout*. Klasa ta jest ukryta we framework'u Flex poprzez parametr [*ExcludeClass*], dlatego nie jest normalnie widoczna dla developerów, nie jest również udokumentowana. Obiekt layout'u przyjmuje atrybut *target*, jako kontener, którego układem dzieci będzie zarządzał. Kontener nie ma wpływu na sposób ułożenia jego dzieci przez layout, który w pełni za to odpowiada.

Klasa *LayoutContainer* implementuje odpowiednio interfejsy:

- *IClipboardClient* – interfejs z framwork'u Hramonix, którego implementacja jest konieczna do obsługi operacji schowka (metody *copy, paste, cut*)
- *IFocusManagerComponent* – Flex'owy interfejs, którego implementacja jest wymagana, aby koponent mógł przyjmować „focus", czyli móc nasłuchiwać skrótów klawiszowych (we framework'u Flex tylko jeden komponent na scenie może mieć „focus" (zaznaczenie) w danym momencie)

Publiczne metody klasy *LayoutContainer*:

- *copy, paste, cut* – obsługa operacji schowka.
- *addChild, addChildAt* – dodawanie nowych dzieci do kontenera, odpowiednio na koniec i na odpowiedni index.
- *removeAllChildren, removeChild, removeChildAt* – usuwanie dzieci, wszystkich, pojedynczego i pojedynczego z odpowiedniego indeksu.
- *getSelectedItems* – zwraca tablice zaznaczonych obiektów

Metody *protected*, które mogą być wykorzystane przy rozszerzaniu komponentu LayoutContainer:

- *selectChild, unselectChild, isChildSelected* – zaznaczanie, odznaczanie i sprawdzanie czy element jest zaznaczony, poprzez przeciążanie tych metod możliwa jest zmiana sposobu zaznaczania dzieci kontenera
- *onChildClick, onChildMove* – handlery odpowiednio do kliknięcia na dziecko kontenera oraz do ruchu. Mogą być przeciążone w celu dodania dodatkowej funkcjonalności np. po kliknięciu lub ruchu elementu.
- onDragDrop, onDragEnter – handlery zdarzeń modelu drag & drop, odpowiednio do upuszczenia elementu oraz przeciągania elementu nad kontenerem.

Publiczne atrybuty, którymi można modyfikować działanie kontenera bez ingerencji w kod:

- *selectable, allowMultipleSelection* - włączanie zaznaczania dzieci kontenera
- *selectionColor* – kolor zaznaczenia elementu kontenera
- *clipboardEnabled* – włączanie operacji schowka dla kontenera
- *deleteEnabled* – włączanie opcji usuwania elementów przez przycisk „delete”
- *dragEnabled, dropEnabled* - włączanie odpowiednio przeciągania oraz upuszczania dzieci kontenera
- *horizontalGap, verticalGap* – ustawianie odstępów pomiędzy elementami
- *itemWidth, itemHeight* - ustawianie szerokości oraz wysokości dzieci kontenera
- *layoutClass* - klasa layout'u odpowiedzialnego za rozmieszczenie dzieci kontenera

3.1.2. DataContainer

Kontener danych, czyli klasa *DataContainer* to bezpośrednia pochodna klasy *LayoutContainer*, co zapewnia przejęcie jej wszystkich funkcjonalności. Komponent renderuje dane tworząc dla każdego obiektu z kolekcji danych, przekazanych do komponentu przez zbindowanie kolekcji (atrybut dataProvider), komponent typu przekazanego jako atrybut itemRenderer.

ItemRenderer musi być pochodną klasy mx.core.UIComponent, ponieważ tylko takie obiekty mogą być wyrenderowane w kontenerach Flex'owych.

DataContainer nasłuchuje zdarzeń na swoim dataProviderze (event'u typu *CollectionEvent.COLLECTION_CHANGE*) i w zależności od rodzaju zmiany w kolekcji, wizualizuje zmiany w kontenerze, dodając, usuwając bądź zmieniając kolejność dzieci.

Publiczne metody klasy Data*Container*:

- *getSelectedItems* – zwraca tablicę zaznaczonych danych (obiektów z kolekcji danych).
- *removeDataChild* – usunięcie obiektu z kontenera oraz kolekcji

Publiczne atrybuty, którymi można modyfikować działanie kontenera bez ingerencji w kod:

- *dataProvider* – atrybut przez który przekazuje się kolekcję danych do wyrenderowania w kontenerze
- *itemRenderer* – przekazujemy klasę renderer'a, czyli komponent, który w specyficzny sposób pokaże dane znajdujące się w kolekcji.

3.2. Layout'y

Layout'y przekazywane do kontenerów typu *LayoutContainer* oraz *DataContainer* muszą dziedziczyć z klasy framework'u Harmonix typu *AbstractLayout*.

Abstrakcyjna klasa *AbstractLayout* implementuje wszelkie niezbędne mechanizmu do utworzenia layout'u takie jak nakładanie listener'ów na dzieci kontenera - nasłuchujących zdarzeń typu : ruch, kliknięcie, dodanie, usunięcie dzieci z kontenera docelowego, dzięki czemu klasy pochodne mogą przeciążając odpowiednie handler'y implementować odpowiednią dla siebie funkcjonalność.

Cała logika odpowiedzialna za rozmieszczanie dzieci w kontenerze znajduje się w metodzie *updateDisplayList*, której wywołanie następuje po każdej zmianie w komponencie (wynika to z cyklu życia komponentów framework'u Flex 3) w którym layout odpowiada za rozmieszczenie dzieci.

3.2.1. Serie (VerticalSeriesLayout i HorizontalSeriesLayout)

Serie czyli klasy *HorizontalSeriesLayout* oraz *VerticalSeriesLayout* dziedziczą bezpośrednio z klasy framework'u Harmonix typu *AbstractLayout*. Ustawienie dzieci kontenera odbywa się w metodzie *updateDisplayList*, gdzie przy każdej zmianie przeliczana jest nowa pozycja dzieci kontenera.

Do przeliczeń nowych brane są pod uwagę style kontenera, a w szczególności *paddingi* (odległości treści od boków kontenera) oraz przede wszystkim jego rozmiary, które mają bezpośredni wpływ na ilość kolumn i wierszy. Odległości pomiędzy rozmieszczanymi dziećmi wyciągane są bezpośrednio z targetu (kontenera, którego layout obsługuje) poprzez wartości *verticalGap* oraz *vorizontalGap,* jako że developer nie może bezpośrednio ustawiać parametrów layout, gdyż ten do kontenera (*LayoutContainer* lub *DataContainer*) jest przekazywany przez typ (z wykorzystaniem *mx.core.ClassFactory*).

W zależności od ilości dzieci w kontenerze włączana lub wyłączana jest obsługa scrolla.

3.2.2. CoverFlowLayout

Klasa *CoverFlowLayout* dziedziczy bezpośrednio z klasy framework'u Harmonix typu *AbstractLayout*. Podobnie jak w pozostałych layout'ach pozycjonowanie dzieci odbywa się w metodzie *updateDisplayList*.

Wyróżnianie jednego elementu podczas przewijania scrollem odbywa się na zasadzie obliczenia jego index'u biorąc pod uwagę ilość dzieci oraz szerokość scroll'owanej powierzchni (czyli wymiarów kontenera). Wyróżnione dziecko ma zmieniany index, na najwyższy, aby mogło być wyrenderowane pierwsze na *displayList*. Realizowane jest to poprzez wykorzystanie tablicy *indexedChildren,* która znajduje się w kontenerach framework'u Harmonix. Dzięki niej możliwa jest zmiana index'ów dzieci w stosie renderowania (*displayList*) zachowując

jednocześnie kolejność pokazywania dzieci taką jaka jest *dataProviderze* dla *DataContainer'a* lub w takiej jakie zostały dodane do LayoutContainer'a.

3.3. Schowek

3.3.1. ClipboardClient i IClipboardClient

Klasa *ClipboardClient* dziedziczy bezpośrednio z Flex'owej klasy *mx.events.EventDispatcher*. Klasa ta w konstruktorze otrzymuje jako parametr (*target*) komponent implementujący interfejs framework'u Harmonix typu *IClipboardClient* i na tym obiekcie nasłuchuje zdarzeń powiadamiających o operacjach schowka, czyli zdarzenia klawiatury (ctrl + x, ctrl + c, ctrl + v).

Po wystąpieniu zdarzenia kopiowane lub wycinane są zaznaczone elementy w kontenerze target i umieszczane a kolekcji schowka, która znajduje się w klasie *ClipboardManager*.

W momencie wklejania obiektów ze schowka, wykorzystywane są obiekty znajdujące się w kolekcji *ClipboardManager'a*, i umieszczane w docelowym kontenerze. Sposób umieszczania nowych elementów w kontenerze różni się w zależności od tego czy elementy schowka to dane czy obiekty wizualne.

Interfejs, który muszą implementować kontenery chcące korzystać z operacji schowka, zawiera metody, od których sposobu implementacji zależą operacje na danych lub obiektach wizualnych kontenera. Daje to możliwość własnego obsłużenia tych operacji, w sposób inny niż ma to miejsce w klasach *LayoutContainer* i *DataContainer*.

Metody interfejsu *IClipboardClient* to:

- *copy* – obsługa kopiowania elementów
- *cut* – obsługa wycinania elementow
- *paste* – obsługa wklejania elementów
- *clipboardEnabled* – flaga oznaczająca czy kontener chce w danym momencie obłsugiwać operacje schowka

3.3.2. ClipboardManager

Klasa *ClipboardManager* to singleton, który dziedziczy bezpośrednio z Flex'owej klasy *mx.events.EventDispatcher*. Zawiera kolekcję danych typu *mx.collections.ArrayCollection*, która przechowuje skopiowane dane (lub wizualne komponenty) z obiektów typu *ClipboardClient*.

Klasa jest sigletonem ponieważ schowek jest zawsze jeden w skali aplikacji. Nie implementuje ona żadnej logiki, gdyż jest jedynie kontenerem dla kolekcji kopiowanych danych.

Publiczne metody klasy *ClipboardManager*:

- *set/get clipboard* – pozwala na umieszczenie lub pobranie kolekcji przechowywanej w schowku
- *clearClipboard* – czyści kolekcję danych schowka
- *addToClipboard* – dodaje elementy do kolekcji danych schowka

3.4. Drag & Drop

Obsługa operacji drag & drop została zaimplementowana poprzez przeciążenie odpowiednich metod framework'u Flex. Każdy komponent (*mx.core.UIComponent* lub jego pochodne) we Flex'ie obsługuje operacje przeciągania i upuszczania elementów, wymagana jest jedynie implementacja niektórych metod pozwalających dokładniej kontrolować przebieg tych operacji:

- dragComplete – zakończenie operacji d&d
- dragDrop – upuszczenie elementu
- dragEnter – przeciągnięcie na komponent obsługujący operacje d&d
- dragExit – przeciągnięcie poza komponent obsługujący operacje d&d
- dragOver – przeciąganie nad komponentem obsługującym operacje d&d
- dragStart – rozpoczęcie operacji przeciągania elementu

Niektóre z tych metod zostały przeciążone w komponentach LayoutContainer oraz DataContainer, dzięki czemu jest możliwość przeciągania i upuszczania elementów tych kontenerów bez dodatkowej ingerencji w kod. Rozszerzając funkcjonalności kontenerów framework'u Harmonix można przeciążyć ponownie powyższe metody dodając dodatkową funkcjonalność.

Do wyzwalania operacji przeciągania elementów w kontenerach framework'u Harmonix zostało wykorzystane zdarzenie *MouseEvent.MOUSE_MOVE* na elemencie kontenera oraz Flex'owa klasa *mx.managers.DragManager*, której metoda *doDrag* inicjuje operacje przeciągania obiektu wizualnego.

3.5. Inne

3.5.1. ResizablePanel

ResizablePanel to bezpośrednie rozszerzenie Flex'owej klasy *mx.containers.Panel*. Do *rawChildren* (lista wizualnych obiektów w kontenerach Flex'owych, nie będących jednocześnie jego bezpośrednimi dziećmi, lecz obiektów z któch danych kontener się składa) Panelu zostały dodane dodatkowe obiekty *Button* do obsługi powiększania i minimalizacji komponentu. Operacje dodania tych button'ów odbywa się w metodzie *createChildren* – wynika to z cyklu życia komponentu Flex'owego.

Publiczne atrybuty, dzięki którym developer może zmieniać zachowanie komponentu bez ingerencji w jego kod:

- *currentSizeState* – ustawienie stanu komponentu (maksymalizacja, minimalizacja)
- *doubleClickResize* – włączanie i wyłączanie możliwości zmiany stanu komponentu przez jego dwukrotne kliknięcie

Zdarzenia, których można nasłuchiwać na komponencie *ResizablePanel:*

- *closePanel* – rozgłaszany przy zamknięciu Panel'u, pozwala na obsłużenie dodatkowych funkcjonalności w momencie zamykania komponentu.
- *maximized* – rozgłaszany w momencie maksymalizacji komponentu

- *minimized* - rozgłaszany w momencie minimalizacji komponentu

Metody typu *protected*, które rozszerzając komponent *ResizablePanel* można przeciążyć dodając dodatkową funkcjonalność:

- *maximize* – logika maksymalizacji komponentu
- *minimize* – logika minimalizacji komponentu

Style za pomocą, których możliwe jest ustawienie obrazków (skin'ów) dla przycisków panela:

- *closeButtonStyle* – przycisk zamykania
- *maximizeButtonStyle* – przycisk maksymalizacji
- *minimizeButtonStyle* - przycisk minimalizacji

4. Propozycje rozbudowy

4.1. Wizualizacja schowka

Jedną z możliwości rozszerzenia funkcjonalności schowka mógłby być komponent wizualny przedstawiający aktualną zawartość kolekcji danych w schowku.

Komponent taki jako dataProvider przyjmował by kolekcję clipboard z klasy framework'u Harmonix ClipboardManager. Kolekcja ta jest zaimplementowana jako kolekcja mx.collections.ArrayCollection, więc możliwe jest zbindowanie danych i natychmiastowe pokazywanie zmian w kolekcji.

Komponent ten mógłby rozszerzać klasę framework'u Harmonix DataContainer, którego itemRender zmieniałby się w zależności od typu obiektów w kolekcji schowka.

4.2. Obiekty dragProxy dla kontenerów

DragProxy to obiekty wizualne przedstawiające przeciągane elementy podczas operacji Drag&Drop. Obiekt dragProxy mógłby być zminimalizowaną kopią przeciąganego elementu lub jego tekstową interpretacją.

Domyślnie we framework'u Flex obiekt dragProxy jest kopią przeciąganego elementu z nałożonym efektem przeźroczystości.

4.3. Layout'y 3D

Jako, że kontener framework'u Harmonix LayoutContainer może przyjmować dowolny layout (pochodny z *mx.containers.utilityClasses.Layout*) , możliwe byłoby wykonanie layout'u trójwymiarowego, który korzystał by z nowych możliwości jakie oferuje Flash Player w wersji 10 (klasy flash.geom.PerspectiveProjection, flash.geom.Utils3D) lub dodatkowej biblioteki takiej jak Papervision3D lub Away3D.

Kontener posiadający taki layout miałby możliwość pokazywania swoich elementów w przestrzeni wielowymiarowej. Taki sposób wizualizacji wiązał by się oczywiście z zaprojektowaniem takiego układu, który poza efektownym wyglądem oferował by również odpowiednią funkcjonalność oraz czytelność renderowanych danych.

5. Struktura projektu

5.1. Struktura pakietów oraz opis zawartości plików

5.1.1. Framework Harmonix

Główny pakiet framwork'u to **pl.flexable.harmonix**, w nim zawarte są następujące katalogi oraz klasy:

- **clipboard**
  - *ClipboardClient* – klient schowka
  - *ClipboardManager* – manager kolekcji danych schowka
- **common**
  - *Debug* – pomocnicza klasa debugująca
- **container**
  - *DataContainer* – kontener danych przyjmujący layout'y
  - *LayoutContainer* – kontener obiektów wizualnych przyjmujący layout'y
- **errors**
  - *HarmonixError* – błąd framework'u Harmonix
- **events**
  - *ClipboardClientEvent* – zdarzenie rozgłaszane przez ClipboardClient
  - *ClipboardManagerEvent* – zdarzenie rozgłaszane przez ClipboardManager
- **interfaces**
  - *IClipboardClient* – interfejs kilenta schowka
  - *ISelectableContainer* – interfejs kontenera posiadającego możliwość zaznaczania swoich elementów/dzieci
- **layout**
  - *AbstractLayout* – abstrakcyjny layout
  - *CoverFlowLayout* – layout typu cover flow
  - *HorizontalSeriesLayout* – layout typu horyzontalna seria
  - *LayoutMetrics* – metryki dla layoutów (przechowywanie padding'ów obsługiwanych kontenerów)
  - *VerticalSeriesLayout* – layout typu wertykalna seria
- **ui**
  - *ResizablePanel* - minimalizowalny i maksymalizowalny panel

5.1.2. Aplikacja HarmonixOverview

W głównym pakiecie znajdują się następujące katalogi i klasy:

- **examples**

- *ClipboardManagerExample* – przykład użycia schowka
- CoverFlowLayoutExample – przykład użycia layoutu typu cover flow
- DataContainerExample – przykład użycia kontenera danych
- LayoutContainerExample – przykład użycia kontenera layout'ów
- ResizablePanelExample – przykład użycia rozszerzonego panelu
- SeriesLayoutExample – przykład użycia layout'ów typu seria

- **renderers**
  - *DataItemRenderer* – item renderer wykorzystywany jako wizualizacja danych w kontenerze typu *DataContainer*
  - *ItemRenderer* – item renderer używany jako przykładowe elementy w kontenerze typu *LayoutContainer*
- harmonix_overview – główny plik aplikacji

5.1.3. Aplikacja HarmonTestDrive

Główny pakiet aplikacji to **pl.flexable.harmonix.samples.harmon**, w nim zawarte są następujące katalogi oraz klasy:

- **model**
  - *TestDriveModel* – model aplikacji (w rozumieniu architektury MVC)
- **util**
  - *ModelUtil* – metody przydatne do wypełniania modelu aplikacji losowymi danymi
- **view**
  - **renderers**
    - *CurseRenderer* – item renderer danego przedmiotu (wizualna reprezentacja *vo.CurseRenderer*)
    - *DayRenderer* – item renderer dnia miesiąca (wizualna reprezentacja *vo.Day)*
  - *CurseBox* – kontener zawierający listę przedmiotów
  - *HarmonogramBox* – kontener zawierający harmonogram (listę dni)
  - *TopBannerBox* – kontener zawierający górny banner aplikacji (logo + menu)
- **vo**
  - *Curse* – value object reprezentujący dany przedmiot
  - *Day* – value object reprezentujący dzień miesiąca

Główny plik aplikacji to *HarmonTestDrive*.