Nomad

.NET Open Modular Application Developement Framework

Agenda

1. Temat pracy dyplomowej
2. Koncepcja realizacja projektu
3. Sposób prowadzenia prac nad projektem
4. Technologie
5. Literatura
6. Proponowane rozwiązania
7. Podsumowanie
8. Dyskusja?

Promotor

dr inż. Piotr Zielniewicz

Temat

Otwarta platforma dla modułowego tworzenia aplikacji

Uzasadnienie podjęcia tematu

* Złożone aplikacje okienkowe
  * Liczne powiązania między fragmentami kodu
* Interesująca nas technologia (C#, WPF)
* Brak rozwiązania dla wszystkich problemów
* Istnieją rozwiązania częściowe (Prism)
* Istnieją rozwiązania dla innych języków (OSGi)

Zakres – zadania szczegółowe

* Zaprojektowanie, zaimplementowanie i przetestowanie otwartego środowiska do modułowego tworzenia aplikacji opartego na architekturze Castle Windsor
* Zestaw podstawowych dodatków (plugin-ów) obejmujących współdzielone repozytorium dokumentów
* Moduł pobierania uaktualnień oprogramowania
* Opracowanie specyfikacji środowiska oraz stworzenie przykładowej aplikacji demonstracyjnej.

Planowane rezultaty

* Open-source'owy framework na licencji BSD, oferujący programistom całego świata:
  * RAD-owskie tworzenie modularnych aplikacji
  * Rozproszone zespoły programistyczne modułów
* Automatyczne repozytorium modułów:
  * Rozwiązywanie zależności
  * Aktualizacje modułów
* Publicznie dostępny portal projektu hostowany na codeplex/github
  * Wiki
  * Tutoriale
  * Przykładowe aplikacje
* Praca dyplomowa

Koncepcja realizacji

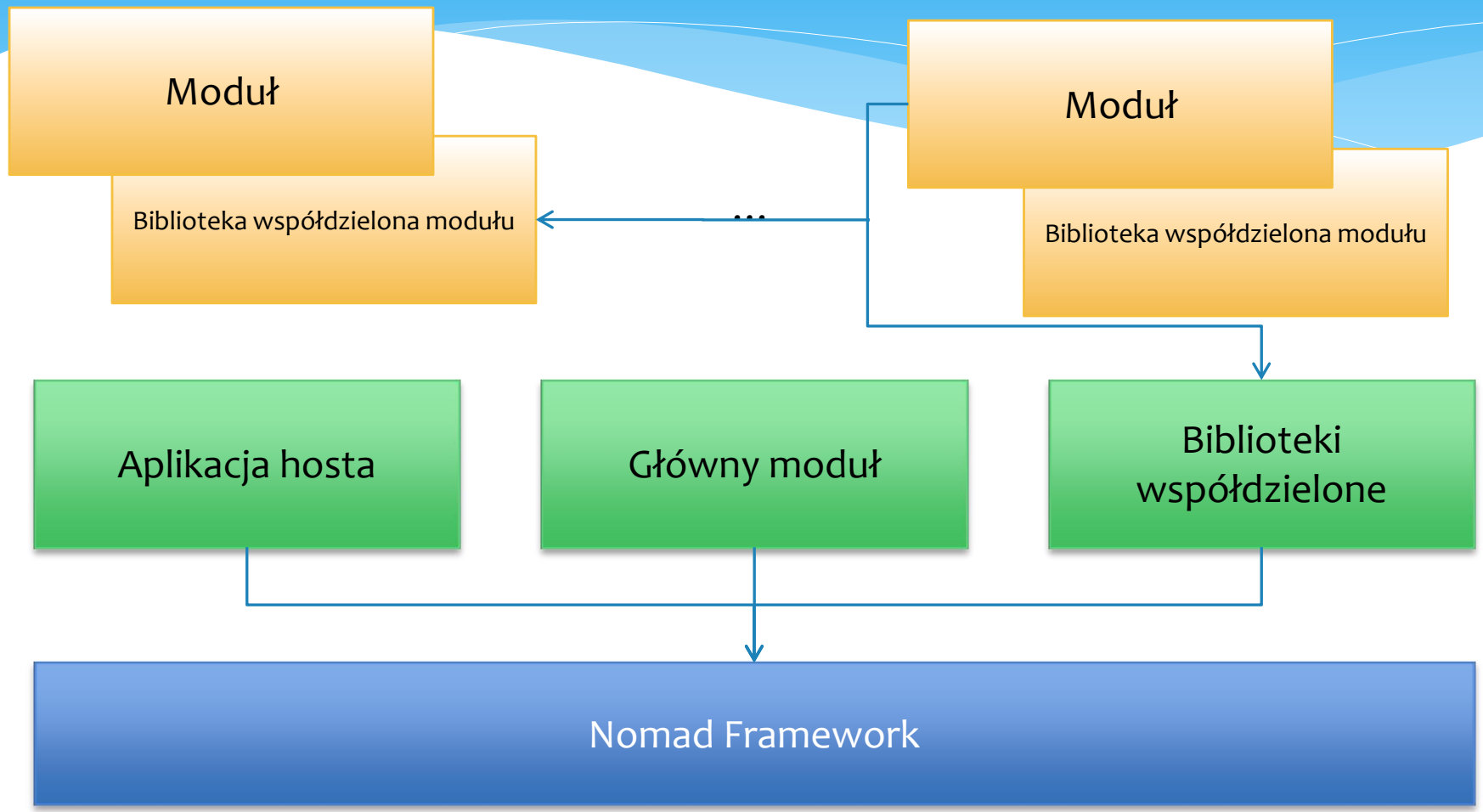
Koncepcja realizacji

Regiony

Komunikacja

Moduły

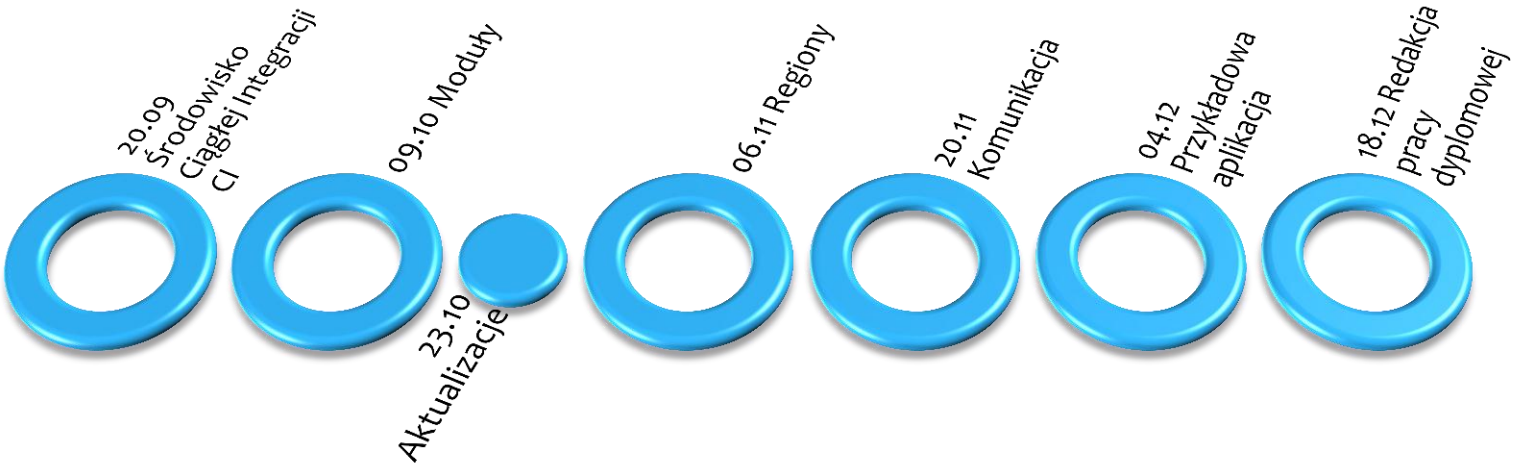
Koncepcja realizacji



Harmonogram prac

* Zwinne (Agile) podejście do prowadzenia projektu
* Elementy metodyki SCRUM
  * Sprinty
  * Przeglądy kodu
  * Feature Backlog
  * Szacowanie pracochłonności
  * Tygodniowe raporty

Harmonogram prac



Czynniki ryzyka

Technologia

- Brak istniejącego framework'u typu *plugin-model*
- Bogactwo technologii .NET
- Korzystanie w jednym projekcie z wielu różnych technologii wokółprojektowych

Ograniczenia

- Termin oddania pracy w formie papierowej

Środowisko deweloperskie i zespół

- Pierwsza praca dyplomowa tego zespołu
- Pierwszy duży projekt programistyczny tego zespołu

Omówienie stosowanych technologii

Platforma .NET

* WPF
* WebServices (*)
* AppDomains
* Wielojęzykowość

Castle Windsor

* Kontener Inversion of Control
* Dynamiczne rozwiązywanie zależności między obiektami
  * Obiekt deklaruje jakich usług potrzebuje do poprawnego działania
  * Kontener IoC wybiera i dostarcza implementacje tych usług
* Centralne zarządzanie cyklem życia obiektów

SandCastle

Narzędzie do generowania dokumentacji kodu:

* Zgodnej ze stylem MSDN
* Bazującej na dokumentacji kodu źródłowego
* Rozszerzalnej o dodatkowe dane w plikach XML

http://sandcastle.codeplex.com

Inne technologie .NET

* Log4net, NLog
* NUnit
* Moq
* Psake
* White
* Wipflash

Technologie wspierające zarządzanie projektem

Trac

Narzędzie portalowe oferujące:

* Zintegrowane wiki
* Śledzenie zmian w repozytorium kodu
* Przydział zadań (ticketów)
  * Organizacja czasu pracy (roadmapa)
  * Śledzenie postępów prac programistycznych
  * Zgłaszanie potrzeb oraz błędów
  * Rozdzielenie odpowiedzialności

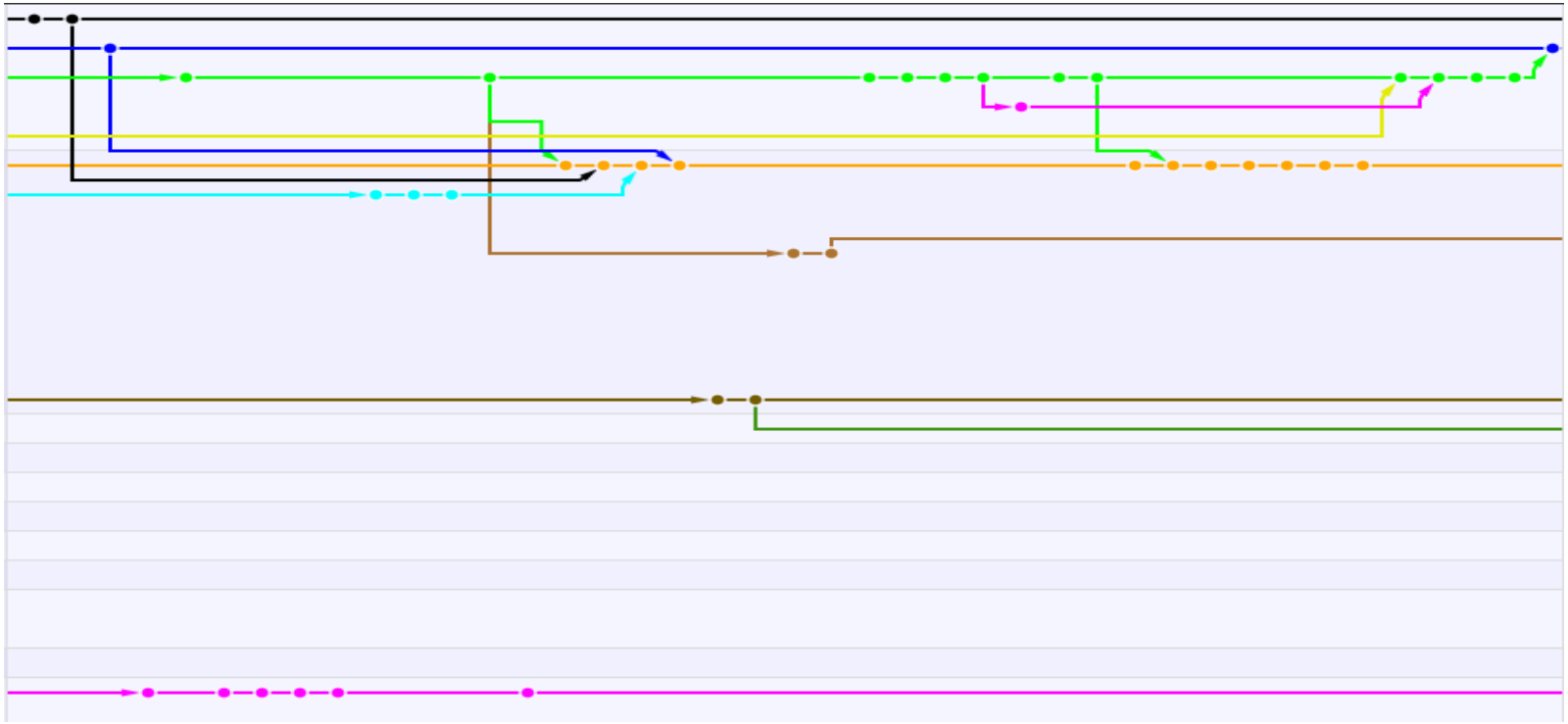
Git

* Rozproszony zystem kontroli wersji
* Zalety
  * Pełna kopia prac na dwóch serwerach
  * Udogodnienia dotyczące współpracy
    * Wersjonowanie kodu
    * Łatwa wymianę kodu i pracy
    * Możliwość rozwiązywania konfliktów
    * Możliwość tworzenia lokalnych backupów

Integracja zmian w kodzie

* Lokalne repozytorium
  * „Feature branch” – gałęzie dla fragmentu funkcjonalności
  * Liniowa historia vs. Graf
    * git rebase

Integracja zmian w kodzie



Integracja zmian w kodzie

* Dwie gałęzie na serwerze centralnym
  * **Stabilna** (master) – tylko kod, który działa (przeszedł testy jednostkowe)
  * **Integracyjna** (staging) – tutaj trafia nowy kod
* Automatyczne dołączanie stabilnego kodu do gałęzi stabilnej

Hudson – ciągła integracja

* Kontrola jakości kodu w skład której wchodzi:
  * Weryfikacja poprawności składniowej kodu
  * Weryfikację jakości kodu
    * kompletność dokumentacji
    * zgodność z kanonem kodowania)
  * Uruchomienie testów
  * Wygenerowanie dokumentacji
  * Przygotowanie paczki „release”

Testy funkcjonalne

* Duży fragment aplikacji
* UI
* Pliki
* Brak izolacji

Testy funkcjonalne

* Klasyczne podejście:
  * Aplikacja i testy – różne procesy
  * Interakcja: UI/pliki/baza danych
* Wady:
  * Potrzebowalibyśmy bardzo wielu aplikacji

Testy funkcjonalne

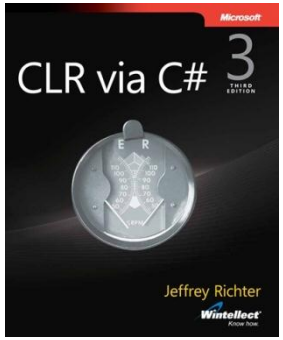
* Nasze podejście:
  * Aplikacja i testy w tym samym procesie
  * Komunikacja również przez pamięć
* Test może dynamicznie stworzyć testowaną aplikację

Testy funkcjonalne - UI

* Jedna testowana aplikacja w tle
* Wiele testowanych okien
* Każdy test może stworzyć inne okno
* Problemy z Continous Integration

Literatura

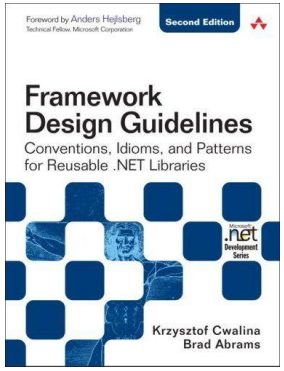
O tym **CO** robimy



Jeffrey Richter

CLR via C# (3rd edition)

Microsoft Press, 2010



Krzysztof Cwalina, Brad Adams

Framework Design Guidelines

Conventions, Idioms, and Patterns for Reusable .NET Libraries (2nd edition)

Addison-Wesley Professiona, 2005

O tym **CO** robimy

Composite Application Guidance

Microsoft

http://compositewpf.codeplex.com/

OSGi Service Platform Specifications

OSGi Alliance

http://www.osgi.org/

O tym **JAK** to robimy

* **Succeeding with Agile: Software Development Using Scrum**
  Mike Cohn, Addison-Wesley Professional, 2009
* **Refactoring to Patterns**
  Joshua Kerievsky, Addison-Wesley Professional, 2004
* **Implementation Patterns**
  Kent Beck, Addison-Wesley Professional, 2007
* **Continuous Integration: Improving Software Quality and Reducing Risk**
  Paul M. Duvall, Steve Matyas, Andrew Glover,
  Addison-Wesley Professional 2007

O tym **JAK** to robimy

* **Refactoring: Improving the Design of Existing Code**
  Martin Fowler, et al., Addison-Wesley Professional, 1999
* **Test Driven Development: By Example**
  Kent Beck, Addison-Wesley Professional, 2002
* **Head First Design Patterns**
  Elisabeth Freeman, et al., O'Reilly Media, 2004
* **Growing Object-Oriented Software, Guided by Tests**
  Steve Freeman, Nat Pryce, Addison-Wesley Professional, 2009

Szczegółowe rozwiązania problemów postawionych w pracy

Moduły

Plugin-model

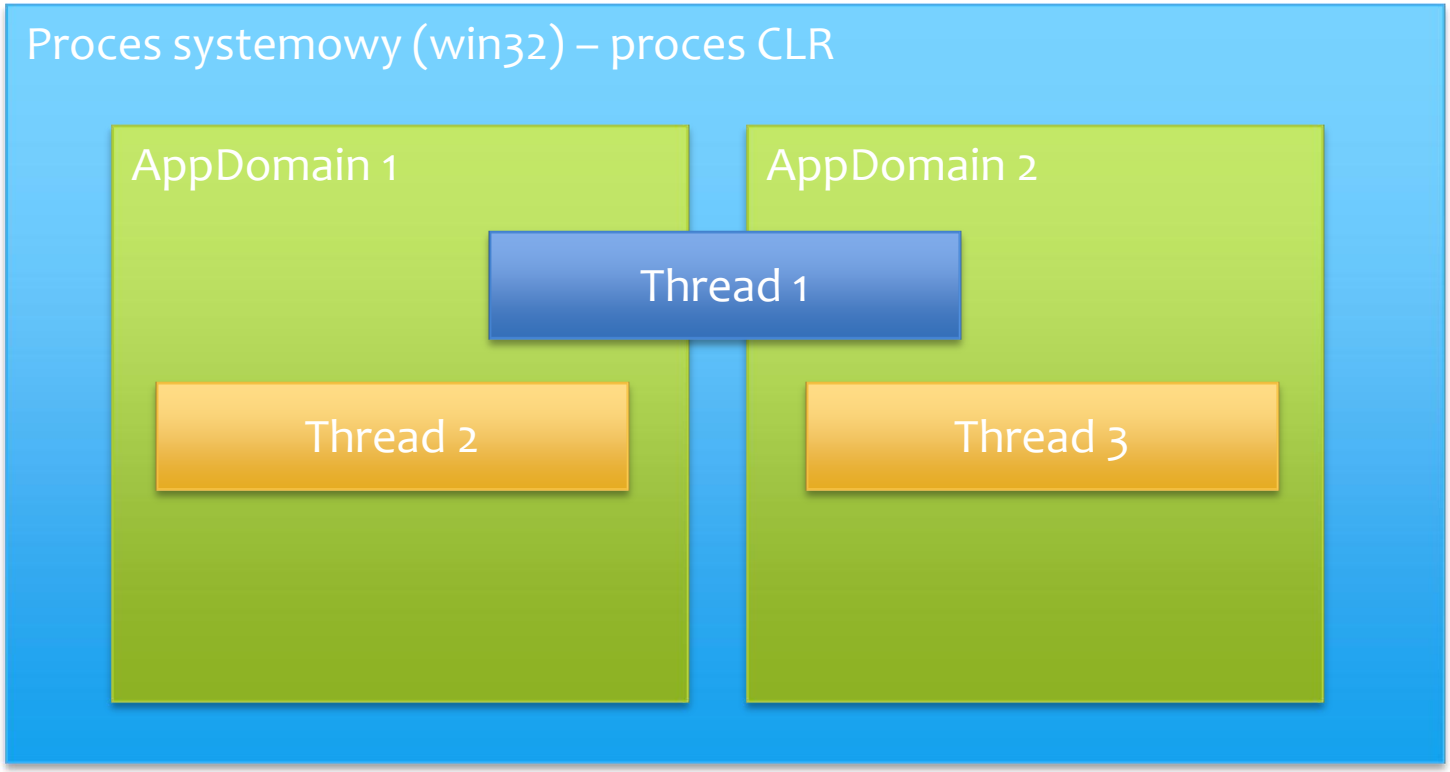
Izolacja modułów
cel

Cechy frameworku:

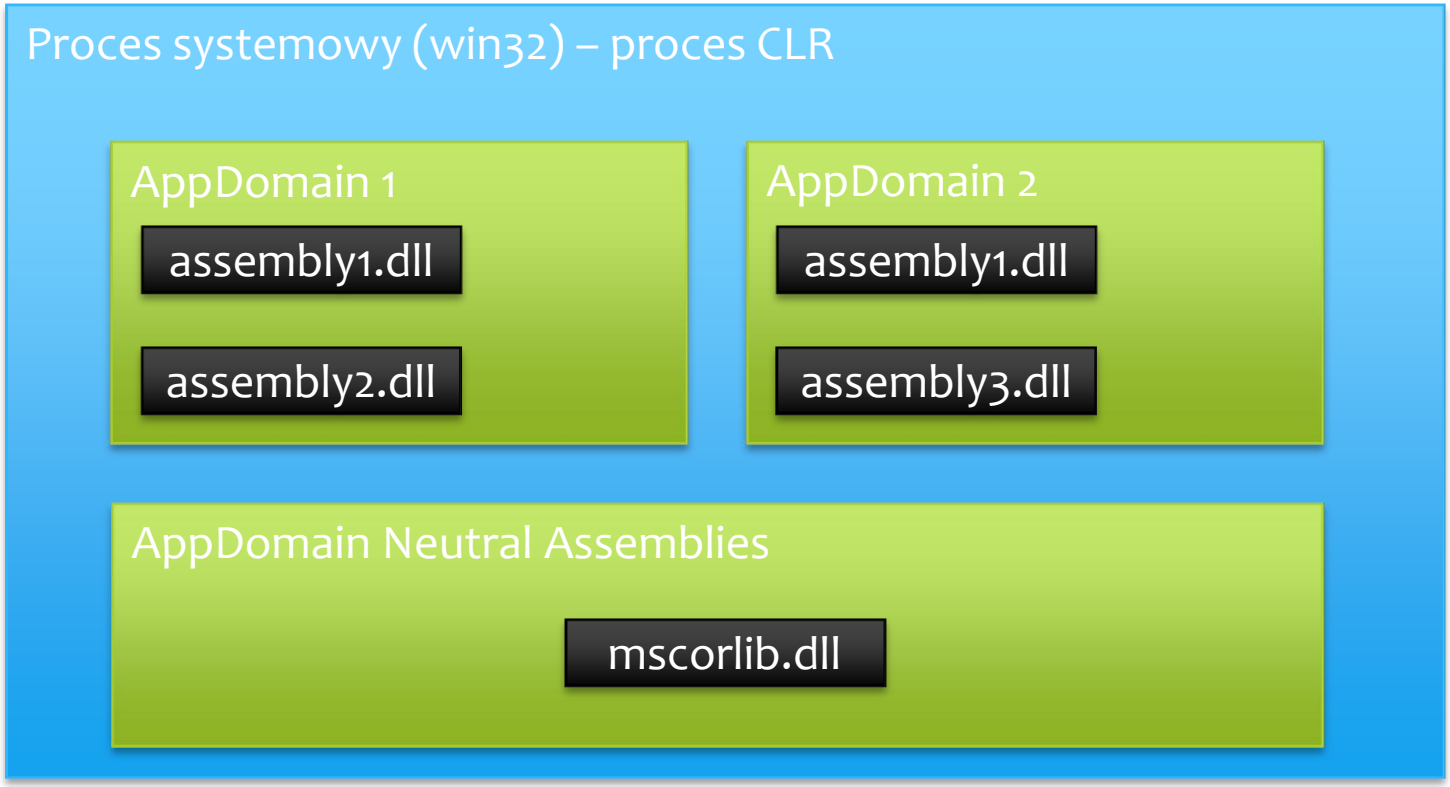
- Możliwość załadowania modułu podczas pracy aplikacji hosta
- Możliwość wyładowania modułów
- Możliwość uaktualnienia modułów i automatycznego restartu
- Komunikację modułów między sobą

Cele te uda się spełnić jedynie w przypadku izolacji modułów

Izolacja modułów
minimum technologiczne



Izolacja modułów
minimum technologiczne



Izolacja modułów
praca badawcza

Możliwe poziomy izolacji osobnego modułu:

* Na poziomie procesu systemowego
  * Zbyt wysoki narzut wydajnościowy na komunikację pomiędzy modułami / aplikacją główną
  * Pełny, niezależny kontekst procesu systemowego

Izolacja modułów
praca badawcza

Możliwe poziomy izolacji osobnego modułu:

* Na poziomie domeny aplikacji (AppDomain)
  * Niskie narzuty na komunikacje przy wykorzystaniu mechanizmów platformy .NET (Marshalling)
  * Współdzielenie przez moduły pewnych elementów środowiska wykonywalnego (np. kontekst graficzny systemu operacyjnego)

Izolacja modułów
praca badawcza

Fragmentaryczne wyniki testów wydajności mechanizmu *Marshalling* w porównaniu do lokalnego działania, na 10 000 instancjach:

* Utworzenie obiektów ~36000ms : ~9ms
* Wywołanie metody *toString()* ~800ms : ~200ms
* Wywołanie metody z parametrem (serializowalnym): ~940 ms : ~230ms
* Wywołanie metody z parametrem (marshalowanym): ~5640ms : ~1309ms

Wnioski:
Mechanizmy komunikacji między domenowej są niewystarczająco szybkie do implementacji frameworka typu *plugin-model* gwarantującego całkowitą izolację.

Izolacja modułów
realizowana koncepcja w *Nomad*

Kernel AppDomain

- Domena aplikacji dla mechanizmów wewnętrznych frameworka *Nomad*
- Główne elementy:
  - Nomad.Updater
  - Nomad.Configuration
  - Nomad.Filter

Modules AppDomain

- Domena aplikacji dla modułów oraz aplikacji hostującej
- Główne elementy:
  - Moduły
  - Aplikacji hostująca
  - Nomad.ModuleLoader
  - Nomad.Communication

Izolacja modułów
realizowana koncepcja w *Nomad*

* **Zalety:**
  * Rozwiązanie to jest wydajne
  * Wymagania w zakresie ładowania / wyładowywania / update'u / restart'u są spełnione
* **Wady:**
  * Nie ma pełnej izolacji – moduły mogą zaszkodzić sobie nawzajem
  * Wyładowanie jednego modułu pociąga konieczność wyładowania wszystkich modułów
  * Nie ma możliwości wykonania tzw. *Hot-swap* w czasie pracy aplikacji

Komunikacja inter-modułowa

Problemy komunikacji

* Moduł obsługuje zdarzenia zachodzące w systemie
  * Nieznane miejsca zajścia zdarzenia
  * Moduły pojawiają się w różnym czasie
* Moduł żąda wykonania zadania
  * Nieznany wykonawca
  * Wielu wykonawców
  * Brak wykonawcy
  * Asynchroniczność

Strony komunikacji

* Publikująca zdarzenie
* Oczekująca na zdarzenie

Metody komunikacji

* Synchroniczna
* Asynchroniczna
* W dedykowanym wątku

Proponowane realizacje komunikacji

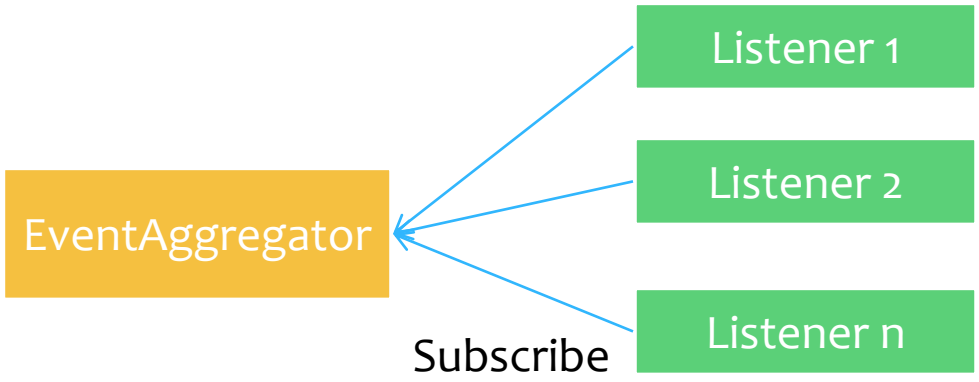
Event Aggregator

* Zapewnia komunikację (międzymodułową)
* Wyróżniamy dwie strony
* Różne metody komunikacji

EventAggregator – zalety

* Zdarzenie, którego nikt nie publikuje (jeszcze)
* Wielu dostawców

EventAggregator – publish/subscribe



EventAggregator

EventAggregator

EventAggregator - przykład

* Zapisanie się:
  * _eventAggregator.Subscribe<MessageType>(payload => HandlePayload);
* Publikacja:
  * _eventAggregator.Publish(sentPayload);

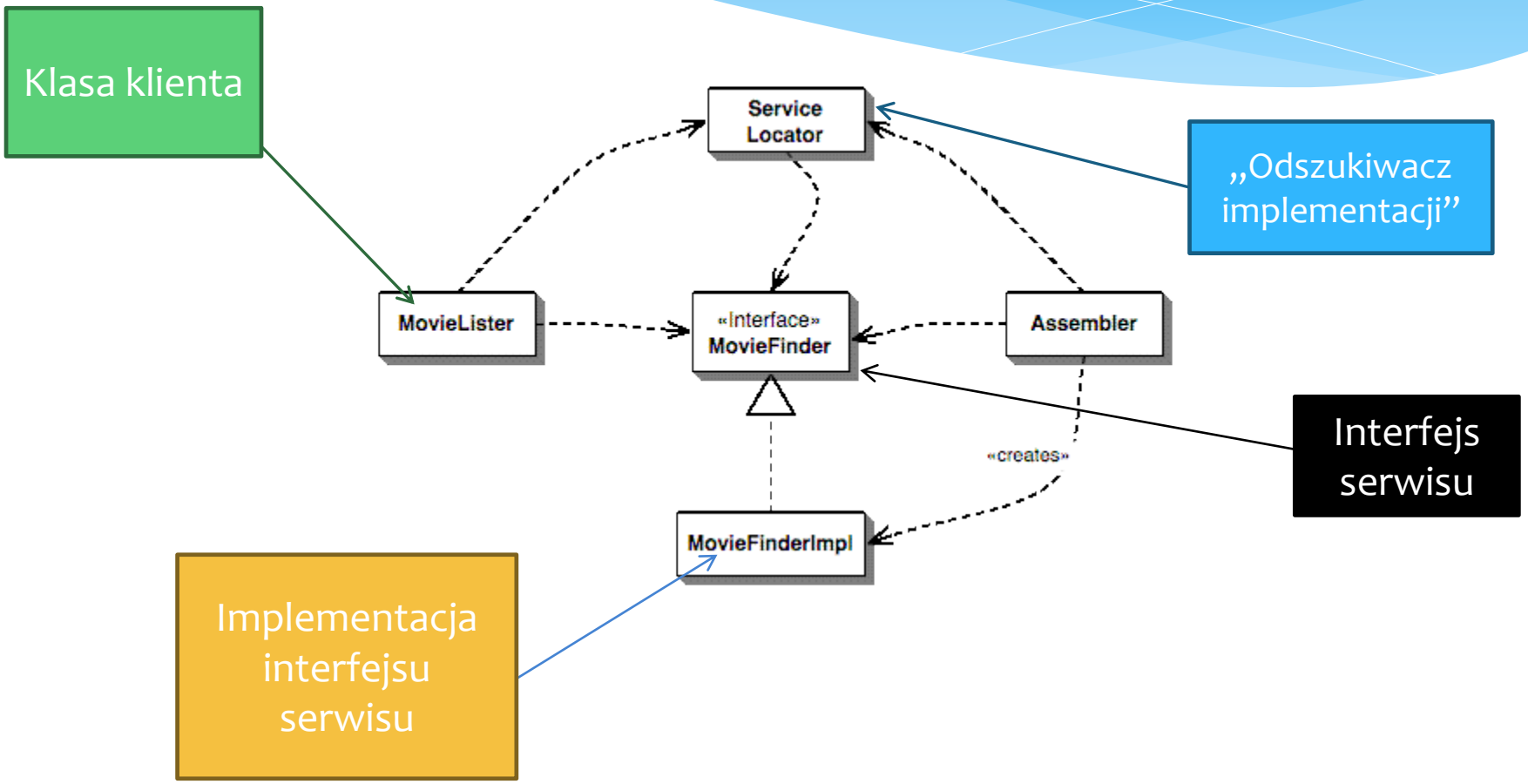
Service Locator

* Wzorzec projektowy używany do enkapsulacji czynności związanych z pozyskiwaniem określonej usługi
* Dokładny opis:
  http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceLocator.html
  http://martinfowler.com/articles/injection.html#UsingAServiceLocator
* Wprowadza dodatkową warstwę abstrakcji pomiędzy klasą realizującą usługę a klasą ją wykorzystującą

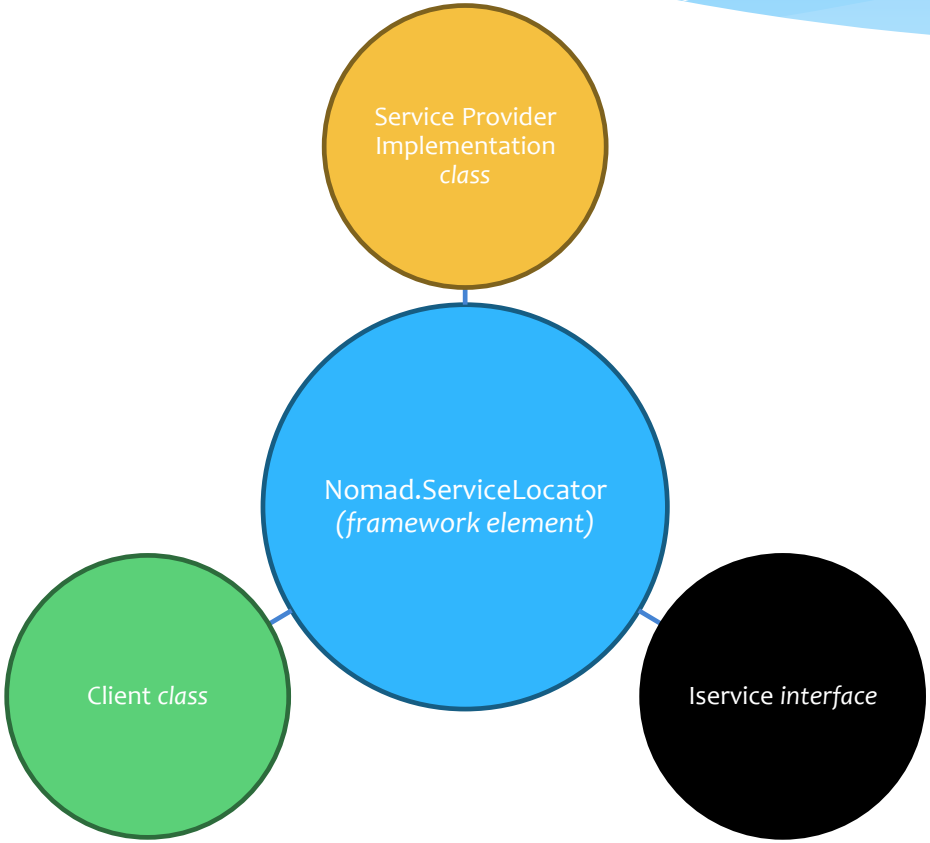
Service Locator - idea

* Dwa rodzaje akcji:
  * Zarejestrowanie obiektu jako implementacji serwisu
  * Pobranie serwisu przez klasę, która jest świadoma tylko interfejsu
* Cel istnienia w *Nomad*:
  * Umożliwia wystawianie usług pomiędzy modułami
  * Umożliwia korzystanie z usług dostarczanych z *Nomad*'em przez moduły

Service Locator – oryginalny diagram UML



Service Locator – implementacja *Nomada*



Service Locator – implementacja *Nomada*

Przykład kodu:

* Rejestracja
  * serviceLocator.Register<InterfaceType>(object)
* Pobranie
  * serviceLocator.Resolve<InterfaceType>()

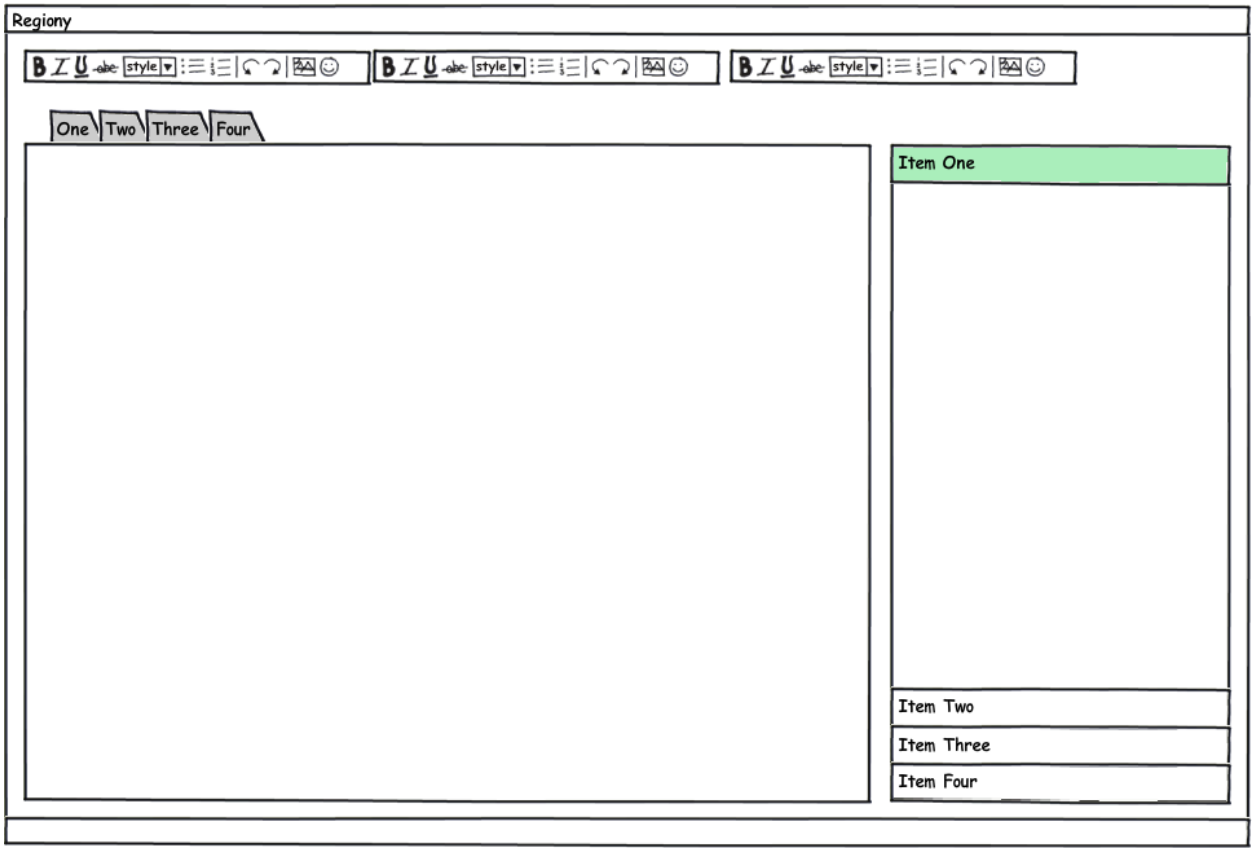
Regiony

Kompozycja interfejsu użytkownika

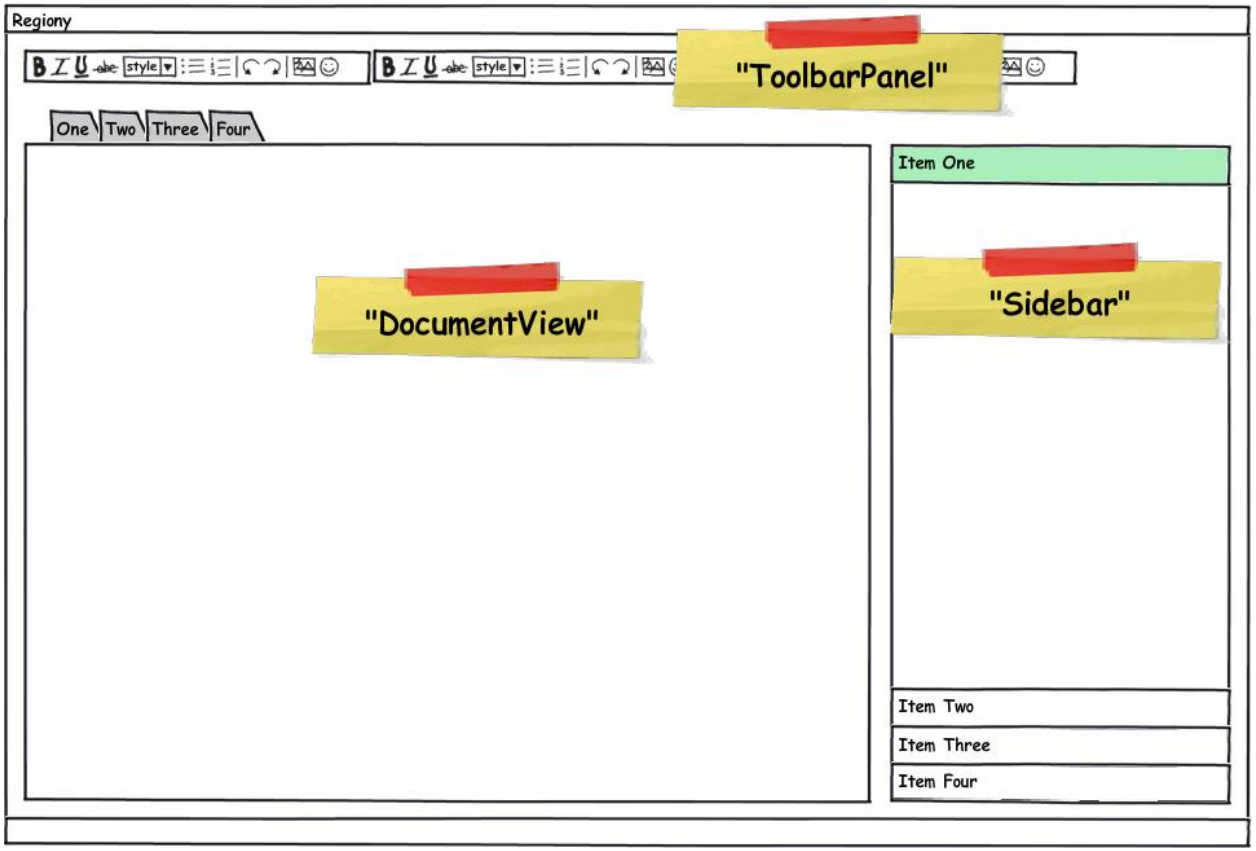
Region

* **Nazwane** miejsce, w którym moduły mogą umieszczać swoje widoki
* **Powiązany** z kontrolką interfejsu użytkownika
* **Nie może istnieć** bez tego powiązania

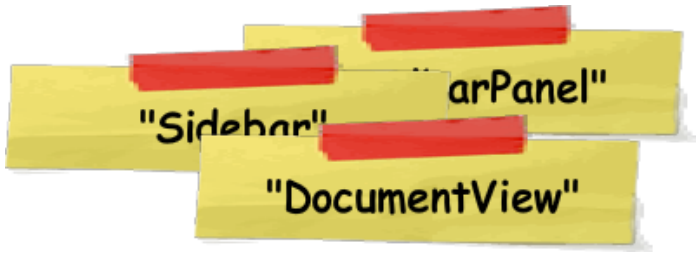
Regiony


Regiony
style
style
style
One
Two
Three
Four
Item One
Item Two
Item Three
Item Four

Regiony


Regiony
"ToolbarPanel"
One
Two
Three
Four
"DocumentView"
Item One
"Sidebar"
Item Two
Item Three
Item Four

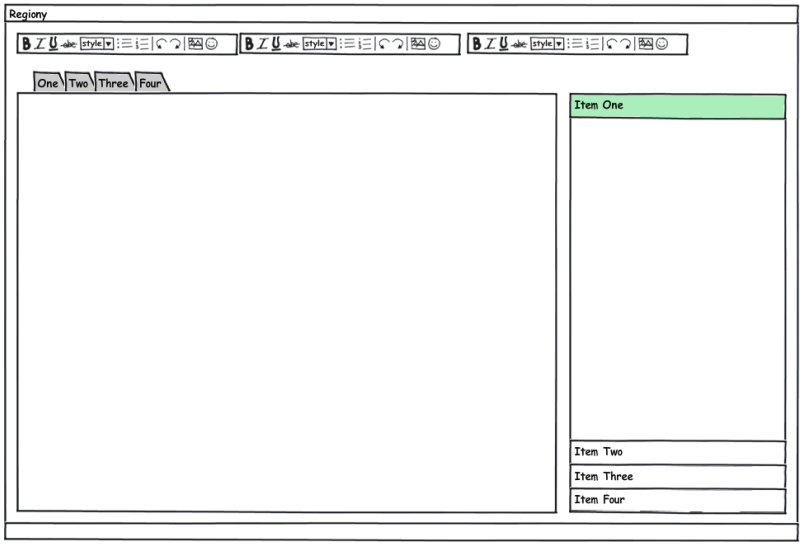
Regiony



Nazwy

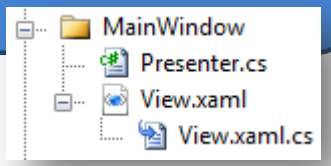
Shell.SharedLib

```
public class RegionNames
{
    public static readonly string DocumentView = "DocumentView";
    public static readonly string SideBar = "SideBar";
    // ...
}
```



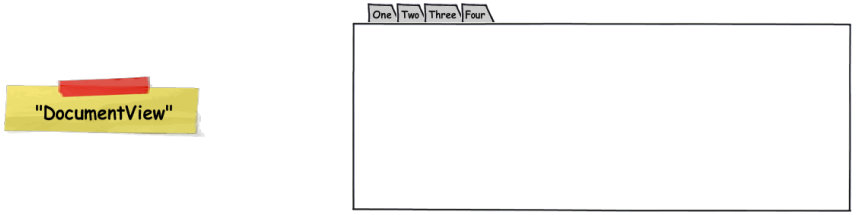
Widok

Shell



Regiony a kontrolki

* Dodatkowe wymagania na widok
  * np. implementacja interfejsu
* Inne sposoby interakcji
  * np. przekazywanie informacji o aktywowaniu dokumentu
* Inne sposoby wyświetlania
  * np. dostarczona przez aplikację (lub moduł) konfiguracja toolbarów



RegionAdapter

* Utworzenie **Regionu**
  * Specyficzne kontrolki mogą mieć własne implementacje (np. toolbar?)
* Spięcie Regionu i kontrolki
* Dodanie zachowań

Spięcie regionu z kontrolką

* Proste
  * TabControl.ItemsSource = Region.Views
* Złożone
  * np. dla Toolbaru, wynikające z ograniczeń kontrolki

Zachowania

* Reakcja na zdarzenia z kontrolki
  * Przekazanie do widoku
  * IActiveAware
* Reakcja na uaktualnienie danych widoku
  * Uaktualnienie kontrolki
  * IHaveTitle

Użycie

* Rejestracja regionu

```
var region = regionManager.AttachRegion(tabControl, RegionNames.DocumentView);
```

```
<TabControl Name="Documents" Regions:Properties.RegionName="Documents">
```

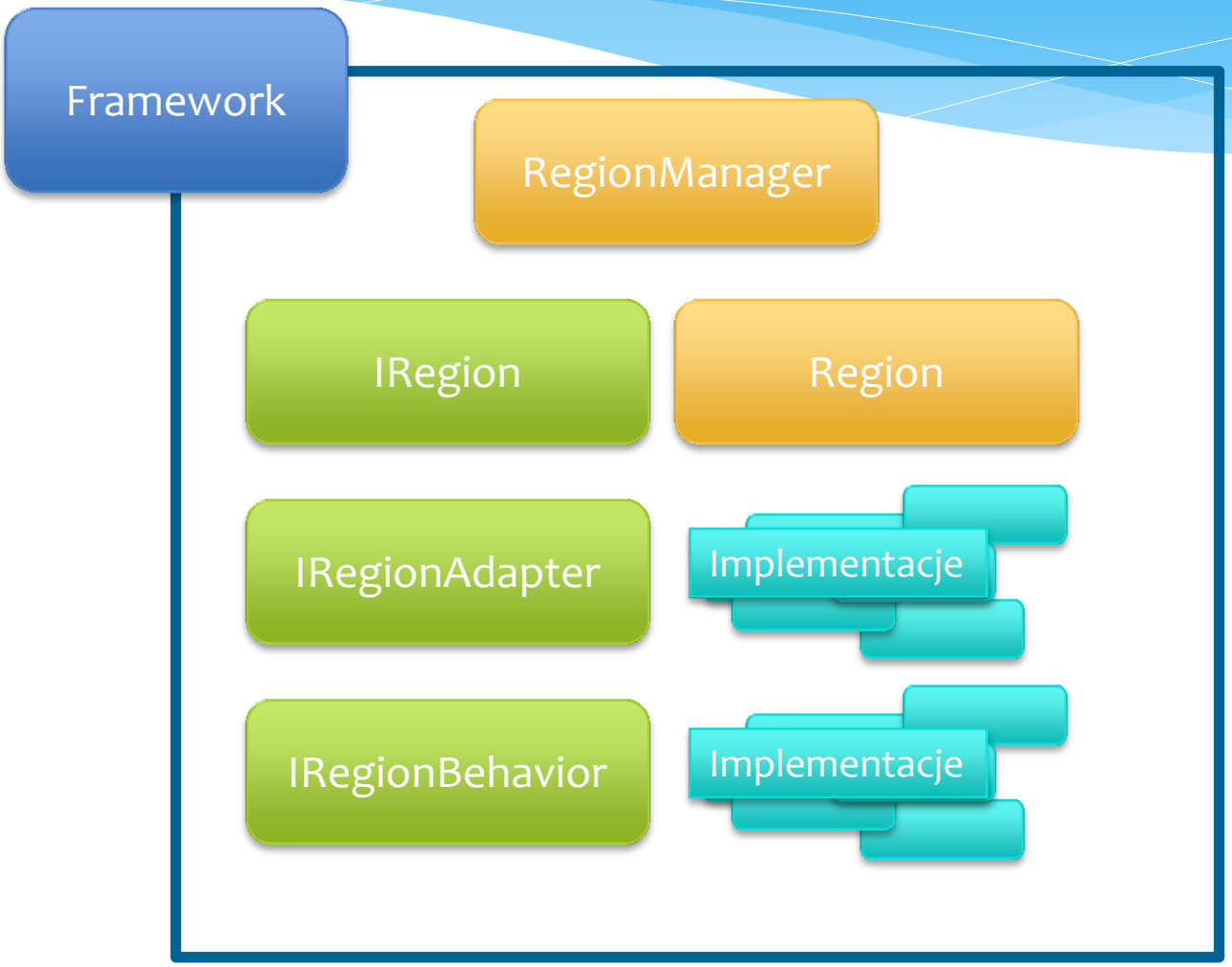
* Dodanie widoku

```
RegionManager.Regions[_regionName].Add(View);
```

RegionManager

* Zarządza regionami
* Zna sposób wyboru adaptera dla kontrolek
  * Niekoniecznie bezpośrednio
* **Entry point** tej części frameworku
* Niekoniecznie jeden
  * Hierarchiczne RegionManagery
  * Kontekstowe RegionManagery

Kawałek architektury



Mechanizm aktualizacji

Aktualizacje

* Definiowane repozytorium
* SOAP / WebRequests
* Aktualizacja bez nadmiarowej interakcji z użytkownikiem

Manifest modułu

* Zależności do innych modułów
* Zestaw plików modułu

Mechanizmy Kontroli Wiarygodności

* Wiarygodność modułów
  * zaufani wystawcy
  * podpisane pliki (RSA)
  * podpisany manifest

Dziękujemy za uwagę

Pytania?