Polsko-Japońska Wyższa Szkoła Technik Komputerowych

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Robert Meisner
Nr albumu s3303

Generyczny system do zarządzania modyfikowalnymi widżetami, wyświetlanymi na pulpicie użytkownika

Praca magisterska
Napisana pod kierunkiem
Dr inż. Mariusza Trzaski

Warszawa, wrzesień, 2009

Streszczenie

Niniejsza praca skupia się na problemach związanych z tworzeniem systemów zarządzających mini-aplikacjacjami (widżetami). Jest to dokładne studium przypadku procesu wytwórczego modułowego oprogramowania, w którym poszczególne elementy są dostarczane przez zewnętrznych dostawców, nie mających dostępu do kodu źródłowego systemu. Praca przedstawia dokładną analizę funkcjonalności, jakie aplikacje tego typu powinny implementować. Autor pracy kładzie duży nacisk na opracowanie zasad tworzenia interfejsu graficznego dla tego typu rozwiązań.

Końcowym etapem pracy jest prototyp aplikacji. Pozwala on na uruchomienie trzech dostarczonych wraz z nim widżetów. Prototyp ten, oprócz zaawansowanego interfejsu graficznego wykorzystującego technologię WPF, implementuje rozwiązania dostarczane przez .NET, oferujące wsparcie w zakresie wykrywania nowych elementów, ich izolacji oraz zapewnienia bezpieczeństwa.

Spis treści

1. Wstęp

Systemy widżetów stają się coraz bardziej popularną formą uzupełniania braków w funkcjonalnościach oprogramowania oraz rozszerzania go o nowe możliwości. Widżety to mini-aplikacje dostarczające pojedyncze, proste funkcjonalności. Popularne wtyczki to: kalkulator, kalendarz, notatnik. Widżety są najczęściej wizualnie grupowane w kontenerze, znajdującym się na pulpicie lub przy krawędzi ekranu (sidebar). Dostęp do nich jest zwykle zapewniony poprzez klawisze skrótów (hotkey), lub poprzez najechanie określonego obszaru na ekranie kursorem myszki.

Widżety oraz aplikacje je hostujące to nie tylko miłe dla oka oprogramowanie, ułatwiające codzienne korzystanie z komputera, to także wiele problemów inżynieryjnych, z którymi spotykają się ich twórcy.

1.1. Cel pracy

Bezpośrednim celem pracy jest opracowanie cech, zasad i konwencji tworzenia oprogramowania charakterystycznych dla systemów zarządzających generycznymi mini-aplikacjami (widżetami) oraz dobór technologii oraz bibliotek programistycznych pozwalających na ich implementację.

Celem pośrednim pracy jest stworzenie aplikacji hosta pozwalającej programistom na tworzenie graficznych mini-aplikacji (widżetów) działających w ramach aplikacji hosta bez dostępu do kodu źródłowego tej aplikacji. Aplikacja hosta powinna izolować kod widżetu od innych aplikacji oraz narzucać zbiór uprawnień zdefiniowany przez użytkownika. Twórca widżetu powinien mieć jak największą swobodę w tworzeniu interfejsu graficznego widżetu oraz definiowaniu jego funkcjonalności. Aplikacja hosta musi zapewnić kanał komunikacji pomiędzy poszczególnymi widżetami.

Ponadto w ramach pracy zostaną przygotowane 3 przykładowe widżety:

- Zegar - pokazujący godzinę w wybranej przez użytkownika strefie czasowej
- Grafer - wyświetlający wykres funkcji podanej przez użytkownika
- Monitor Systemu - wyświetlający użycie zasobów w danej chwili. Powinien wyświetlać użycie takich zasobów jak: moc procesora, przepustowość sieci, pamięć operacyjna.

1.2. Przyjęte rozwiązania

Opracowany w ramach pracy prototyp będzie składał się z następujących elementów:

- Aplikacja hosta
- Widżet "Zegar"
- Widżet "Grafer"
- Widżet Monitor Systemu.

Wszystkie elementy zostaną opracowane z wykorzystaniem języka C# i platformy .NET w wersji 3.5. Do definicji interfejsu użytkownika zostanie wykorzystana technologia WPF (Windows Presentation Foundation) oraz język XAML. Do mechanizmu rozszerzeń zostaną wykorzystane biblioteki wchodzące w skład MAF (Managed Addin Framework), będące częścią .NET 3.5.

1.3. Osiągnięte rezultaty

Bezpośrednim rezultatem pracy będzie ustalenie dobrych praktyk i konwencji w tworzeniu systemów zarządzających widżetami, w tym także określenie technologii i bibliotek programistycznych pozwalających na kontrolę izolacji, bezpieczeństwa, wersjonowania, instalacji oraz konfiguracji poszczególnych widżetów. Konieczne będzie także zdefiniowanie kanału komunikacji na liniach host-widżet oraz widżet-widżet.

Pośrednim rezultatem pracy będzie aplikacja stanowiąca system zarządzania widżetami wraz z trzema przykładowymi widżetami.

1.4. Organizacja pracy

Praca rozpoczyna się od przedstawienia problemu związanego z projektowaniem oprogramowania oraz potrzeb, na jakie odpowiada oprogramowanie widżetów. Przedstawione zostaną wady oraz rozwiązania w istniejących aplikacjach tego typu. Porównane zostaną ponadto najpopularniejsze podejścia co do prezentacji danych oraz używanych technologii.

W rozdziale trzecim opisane zostaną narzędzia oraz biblioteki programistyczne wykorzystane w budowie prototypu aplikacji.

Czwarty rozdział przedstawia wymagania funkcjonalne stawiane przed aplikacjami tego typu. Mówi zarówno o funkcjonalnościach Aplikacji Hosta, jak i wspólnych funkcjonalnościach samych widżetów.

Rozdział piąty prezentuje analizę architektonicznych i implementacyjnych rozwiązań użytych w projekcie. Mówi także o przeszkodach, z jakimi Autor spotkał się podczas tworzenia pracy.

Zakończenie stanowi podsumowanie zalet i wad przyjętych rozwiązań w szóstym rozdziale oraz propozycje drogi dalszego rozwoju aplikacji w rozdziale siódmym.

2. System do zarządzania widżetami - stan sztuki

Widżet, jako mini-aplikacja dostarczająca prostą, pojedynczą funkcjonalność istnieje pod wieloma różnymi nazwami. W zależności od producenta oprogramowania widżety nazywane są gadgetami, add-in'ami, add-on'ami, plug-inami, komponentami czy też modułami. Na potrzeby niniejszej pracy, tam gdzie nazwa nie wynika z kontekstu, Autor przyjął nazwę „widżet” jako obowiązującą.

Na rynku istnieje wiele systemów zarządzania widżetami. Każdy z nich dostarcza podobną funkcjonalność, jednak drobne różnice sprawiają, że jedne są bardziej przyjazne dla użytkownika od innych. Każdy z opisanych systemów oprócz warstwy prezentacji widocznej dla użytkownika kryje wysublimowane mechanizmy zarządzające uprawnieniami, wersjonujące, aktualizujące, czy też izolujące poszczególne widżety. Z punktu widzenia inżynierii oprogramowania problemy modułowości, wieloplatformowości, skalowalności oraz generyczności, z jakimi spotykają się twórcy takiego oprogramowania, nie należą do trywialnych.

Każdy z systemów stara się znaleźć własne rozwiązanie powyższych problemów.

2.1. Dashboard - Apple

Dashboard to oprogramowanie stworzone dla systemu operacyjnego Mac OS X firmy Apple. Jego główną funkcjonalnością jest hosting mini-aplikacji.

Aplikacja tworzy częściowo przezroczystą warstwę nad wszystkimi oknami systemu. Warstwa ta jest domyślnie niewidoczna. Wybranie wcześniej zdefiniowanej kombinacji klawiszy pokazuje aplikację. Mini-aplikacje zwane widżetami, podobnie jak okna aplikacji, można przemieszczać, zmieniać kolejność, tworzyć, zamykać. Nowy widżet może zostać uruchomiony (umieszczony na częściowo przezroczystej warstwie) poprzez przeciągnięcie jego ikony z dolnego paska, gdzie widoczne są wszystkie dostępne widżety.

Ten system widżetów pozwala programistom niezwiązanym z tworzeniem samej aplikacji na dewelopment i dystrybucję pojedynczych widżetów we własnym zakresie.

Dashboard jest nieodłączną częścią systemu operacyjnego Mac OS X począwszy od wersji Tiger. Listę wyboru widżetów w ramach Apple Dashboard przedstawia Rysunek 1.

Rysunek 1. Apple Dashboard oraz lista wyboru widżetów

Źródło: Opracowanie własne

2.2. Microsoft Gadgets

Microsoft Gadgets to lekkie mini-aplikacje, zwykle o ograniczonym zakresie funkcjonalności. Gadget'y to odpowiedniki widżetów w Dashboard firmy Apple. Gadget'y dzielą się na trzy rodzaje:

- Gadgety webowe,
- Gadgety pulpitu,
- Gadgety urządzeń i Windows SideShow.

Przykładowe gadgety Microsoft Gadgets zostały zaprezentowane na Rysunku 2.

Rysunek 2. Przykładowe gadgety Microsoft Gadgets

Źródło: Opracowanie własne

Gadgety Webowe

Gadgety te są uruchamiane na stronach internetowych takich jak Live.com oraz Windows Live Spaces.

Widżety zwykle korzystają z usług internetowych w celu aktualizacji danych i zapisywania ustawień. Obciążenie związane z obsługą wielu klientów łączących się z daną usługą leży po stronie właściciela widżetu.

Użytkownicy mogą zmieniać kolejność widżetów, organizować je w zakładki oraz zmieniać schematy kolorów.

Gadgety pulpitu

Gadgety te są umieszczane na pulpicie w kontenerze (sidebar) „przyczepionym” do jego krawędzi. Sidebar domyślnie znajduje się po prawej stronie ekranu. Użytkownik ma możliwość zmianę jego położenia. Użytkownik może dodawać kolejne instancje widżetu do kontenera, które są układane automatycznie jedna pod drugą. Ich kolejność może zostać zmieniona przez użytkownika.

Gadgety urządzeń i Windows SideShow

Windows SideShow to nowa technologia stworzona przez Microsoft pozwalająca na wyświetlanie gadgetów na zewnętrznych wyświetlaczach (np. LCD). Technologia ta działa także gdy komputer jest w stanie uśpienia. Przykładowym zastosowaniem tej technologii może być odtwarzacz mp3 z wyświetlaczem na obudowie komputerowa. Mała konsumpcja energii przez wyświetlacz w połączeniu z odłączeniem urządzeń komputera, które nie są niezbędne do działania takiego gadgetu sprawiają, że komputer przenośny można zmienić w działający dziesiątki godzin odtwarzacz mp3. Po chwili ten sam wyświetlacz może pokazać prognozę pogody na jutro.

Odmienne nazewnictwo wywodzi się z lat osiemdziesiątych z nazewnictwa przyjętego przez twórców Amiga-OS - systemu operacyjnego komputerów Amiga. Chronologicznie patrząc słowo widżet jest starsze w kontekście niniejszej pracy. Dla ujednolicenia nazewnictwa Autor przyjął nazwy „widżet” oraz „wtyczka” jako obowiązujące.

2.3. Google Desktop

Google Desktop to system widżetów stworzony przez firmę Google. W przeciwieństwie do swoich konkurentów jest nie tylko kontenerem dla miniaplikacji, ale dodatkowo pozwala na tekstowe przeszukiwanie na komputerze użytkownika jego fotografii, emaili, dokumentów, historii obejrzanych stron i czatów.

Google Desktop został stworzony w trzech wersjach: Windows, Mac OS X i Linux.

Przykładowe widżety Google Desktop przedstawia Rysunek 3.

Rysunek 3. Widżety Google Desktop

Źródło: Opracowanie własne

Najnowsza wersja Google Desktop posiada następujące funkcje:

1) Indeksowanie plików

Pierwsze indeksowanie plików na komputerze, na którym uruchomiony został Google Desktop rozpoczyna się od razu po instalacji aplikacji. Po wstępnym indeksowaniu (100 000 plików) kolejne

pliki są indeksowane w miarę potrzeb. Wyszukiwanie plików jest dostępne od razu po instalacji. Domyślnie Google Desktop ma możliwość indeksowania kilkunastu formatów plików, między innymi plików poczty (Thundebird, Outlook), historii przeglądanych stron (Internet Explorer, Mozilla Firefox), dokumentów (formaty OpenDocument oraz Microsoft Office), archiwów rozmów (Skype, MSN, Google Talk, AOL) oraz kilkunastu typów plików multimediów.

2) Pasek boczny

Pasek boczny (Sidebar) znajduje się tylko w wersjach Google Desktop skierowanych dla systemów Windows i Linux. Jest kontenerem zawierającym gadgety. Sidebar może zostać umiejscowiony przy krawędzi pulpitu użytkownika.

Domyślnie sidebar zawiera następujące gadgety:

- Email - daje możliwość przeglądania poczty Gmail;
- Scratch Pad - umożliwia zapisywanie notatek w czasie pisania;
- Photos - wyświetla mini pokaz zdjęć z wcześniej zdefiniowanego katalogu;
- News - wyświetla nagłówki z serwisu Google News. Tematyka nagłówków dostosowuje się do upodobań czytelnika na podstawie najczęściej czytanych tematów nowości;
- Weather - pokazuje aktualną pogodę. Istnieje możliwość zdefiniowania lokalizacji;
- Web Clips - wyświetla nagłówki wcześniej zdefiniowanych źródeł RSS;
- Google Talk - w przypadku zainstalowanego Google Talk na komputerze użytkownika gadget ten pozwala na rozmowy z innymi użytkownikami z listy kontaktów.

Niezwykle przydatną funkcjonalnością są dymki informujące. W przypadku, gdy Pasek Boczny jest ukryty, a jeden z gadgetów chce coś zakomunikować użytkownikowi (np. nadejście nowej wiadomości), ma on możliwość wyświetlenia powiadomienia w formie dymka w prawym dolnym rogu ekranu.

Możliwe jest ukrycie Google Desktop. W tym przypadku Sidebar chowa się automatycznie, gdy kursor myszki nie znajduje się bezpośrednio na nim. Możliwe jest przywrócenie widoczności Paska Bocznego poprzez zbliżenie kursora myszy do krawędzi pulpitu, na której znajduje się Pasek Boczny.

Domyślnie Pasek Boczny zajmuje od 1/6 do 1/9 powierzchni ekranu (w zależności od rozdzielczości). Możliwa jest jednak zmiana szerokości paska przez użytkownika.

3) Quick Find

Podczas wyszukiwania plików przy użyciu Google Desktop, aplikacja wyświetla 6 najbardziej trafnych wyników w małym oknie umieszczonym poniżej pola tekstowego zawierającego słowa kluczowe wyszukiwania (patrz Rysunek 4). Okno jest aktualizowane na bieżąco w trakcie wprowadzania tekstu.

Rysunek 4. Okno Quick Find zawierające najbardziej trafne pozycje wyszukiwania

Źródło: Opracowanie własne

4) Gadgety i plug-iny

Możliwe jest dodawanie nowych gadgetów do aplikacji znajdujących się w specjalnie przygotowanym serwisie internetowym.

Google udostępniło API deweloperskiego bloga, aby ułatwić programistom tworzenie nowych gadgetów.

2.4. Yahoo! Widget Engine

Ten system widżetów został stworzony dla dwóch platform systemowych: Max OS X oraz Microsoft Windows. We wcześniejszych wersjach system ten nosił nazwę Konfabulator, lecz po przejęciu przez firmę Yahoo! przeszedł kompletny rebranding. Nazwa Konfabulator jest jeszcze używana jako nazwa silnika renderującego. Silnik ten używa interpretera XML w połączeniu z Javascriptem do wyświetlania widżetów.

W porównaniu do swoich konkurentów dany system widżetów wydaje się być bardzo dojrzałym i dobrze przemyślanym. Oferuje deweloperom rozbudowane API oparte o JavaScript. API to udostępnia podstawowe elementy takie jak: pola tekstowe, obrazy, timery. Ponadto pozwala na ściąganie stron z internetu, odczyt/zapis plików, czy animowanie poszczególnych elementów. Wbudowany interfejs daje możliwość uruchamiania kodu specyficznego dla systemu operacyjnego: skrypty powłoki i obiekty COM w Microsoft Windows i AppleScript w Mac OS X.

Wraz z wprowadzeniem wersji 4.0 Yahoo! Widgets dla deweloperów widżetów został udostępniony SQLite, pozwalający im na tworzenie i modyfikacje baz danych.

W wersji 4.5 wprowadzono obsługę WebKit i Adobe Flash, co pozwoliło na tworzenie widżetów z użyciem HTML i Javascriptu, jak i ograniczonej funkcjonalnie technologii Flash. Obsługa WebKit nie przewiduje używanie wtyczek, z tego powodu Flash jest obsługiwany z użyciem natywnych obiektów.

Jedną z cech, która sprawia, że system ten widocznie wyprzedza swoich konkurentów jest uruchamianie każdego widżetu w osobnym procesie. Zwiększa to znacznie stabilność całości systemu. W przypadku wystąpienia problemu z jednym z widżetów, może on zostać wyłączony bez wpływu na działanie reszty systemu. Minusem takiego rozwiązania jest jednak dodatkowa pamięć operacyjna konieczna do uruchomienia każdego procesu.

Widżety Yahoo! Widget Engine przedstawia Rysunek 5.

Rysunek 5. Widżety Yahoo! Widget Engine

Źródło: Opracowanie własne

2.5. Widżety desktopowe, mobilne, a webowe

2.5.1 Widżety desktopowe

Widżet desktopowy to wyspecjalizowana mini-aplikacja, posiadająca Graficzny Interfejs, wizualnie znajdująca się w ramach pulpitu (pulpity są elementem każdego popularnego systemu operacyjnego).

Historia widżetów desktopowych sięga roku 1984, gdy firma Apple wprowadziła do swojego systemu operacyjnego wbudowaną aplikację Desk Accessory. Aplikacja ta wprowadzała model, w którym możliwe było uruchomienie małych, prostych mini-aplikacji będących częścią systemu i korzystanie z nich równolegle z uruchomioną aplikacją główną. Dawało to wrażenie wielozadaniowości (multitasking) w systemie, który w rzeczywistości nie miał takich możliwości. Wraz z wprowadzeniem multitaskingu w systemach operacyjnych, wbudowane Desk Accesories zostały zastąpione samodzielnie działajaącymi aplikacjami. Wzrost wymagań użytkowników i ciągle zwiększająca się liczba aplikacji koniecznych do ich zaspokojenia sprawiły, iż koncepcja stworzona przez Desk Accessory ewoluowała i znalazła swoje odzwierciedlenie w sytsemach widżetów.

Widżety dostarczają wyspecjalizowanych, prostych funkcjonalności. Ważna cechą widżetów desktopowych jest ich zdolność do działania bez stałego połączenia z internetem. Są łatwo dostępne poprzez ich lokalizację oraz użycie klawiszy dostępu (hotkey).

2.5.2 Widżety mobilne

Funkcjonalnie widżety, jak i systemy widżetów tworzone na aparaty komórkowe, nie różnią się niczym od swoich „pulpitowych" odpowiedników. Widżety mobilne starają się wykorzystać jak najbardziej ergonomicznie ograniczony ekran aparatu telefonicznego, zwykle jeden widżet zajmuje całą przestrzeń ekranu.

Sposób nawigacji i prezentacji danych może się także różnić od widżetów desktopowych. Jest on najczęściej dostosowany do możliwości nawigacyjnych aparatu. Istnieje parę silników widżetów zbudowanych w oparciu o Java ME, lecz brak ustandaryzowanych API kontrolujących aparaty telefoniczne sprawia, że tworzenie takich systemów działających na telefonach różnych producentów jest niezwykle trudne i wrażliwe na kolejne aktualizacje natywnego oprogramowania telefonu.

Powstało także parę silników opartych o AJAX, które rozwiązują częściowo problem wieloplatformowości, posiadają one jednak mały podzbiór funkcjonalnej potęgi oferowanej przez tak rozbudowane środowisko, jakim jest Java ME.

Rosnąca popularność mobilnych widżetów jest łatwo zrozumiała z powodu dostępności medium, jakim jest w dzisiejszych czasach przenośny aparat telefoniczny. Trudności w projektowaniu interfejsu użytkownika i w wyświetlaniu dużych ilości danych przez pojedyncze widżety w urządzeniach mobilnych o niewielkich przekątnych ekranu sprawiają, że coraz częściej wersje mobilne widżetów to ograniczone funkcjonalnie odpowiedniki swoich desktopowych pierwowzorów.

Firmy tworzące nowe mobilne silniki widżetów powinny projektować i wdrażać je biorąc pod uwagę cztery czynniki: sposób dystrybucji, model biznesowy, oprogramowanie strony serwera oraz środowisko klienckie.

2.5.3 Widżety webowe

Przeglądarki internetowe takie jak Internet Explorer i Firefox mogą być używane jako część infrastruktury systemu widżetów. Sieć jest doskonałym środowiskiem do dystrybucji widżetów, gdyż nie jest potrzebna jakakolwiek interakcja z użytkownikiem w celu ich instalacji.

Widżet webowy to przenośny kawałek kodu, który może zostać zainstalowany i wykonany na oddzielnych stronach HTML bez potrzeby jego kompilacji. Idea ta wywodzi się z pojęcia ponownego użycia kodu. Inne pojęcia służące do opisu widżetów webowych to: gadget, badge, moduł, webjit, capsule, snippet, mini i flake. Zwykle korzystają z technologii DHTML, Javascript lub Adobe Flash.

Widżety webowe okazały się posiadać olbrzymi potencjał komercyjny, głównie jako potencjalny kanał marketingowy. Pojęcia buzz marketing oraz marketing wirusowy są często kojarzone z widżetami webowymi. Ich wirusowy sposób dystrybucji zdecydowanie zmniejsza koszty dotarcia do potencjalnego klienta.

Platformy dystrybucji widżetów takie jak iGoogle i Live promują najpopularniejsze widżety pozwalając ich twórcom na dotarcie do jeszcze większej grupy klientów.

Pierwszym znanym widżetem webowym była mini-gra Trivia Blitz stworzona w 1997 roku przez Uproar.com (w latach 2000-2001 wiodąca firma na rynku gier online). Aplet gry pojawił się na 35000 stronach internetowych, począwszy od prywatnych stron na Geocities do stron CNN. Dewelopment gry został wstrzymany po tym jak Uproar.com zostało przejęte przez Vivendi Universal.

Pokazało to jednak jak olbrzymi potencjał leży w tej formie dystrybucji widżetów.

Idealnym środowiskiem dla widżetów webowych od lat są serwisy społecznościowe, takie jak Grono czy też Facebook, przede wszystkim z powodu udostępnionych przez te serwisy rozbudowanych API dla deweloperów, katalogów widżetów oraz łatwości dzielenia się ciekawymi treściami z innymi użytkownikami serwisu. Nie jest to ostateczna, najbardziej doskonała forma reklamy jak zauważa Ben Kunz w Business Week [1], ale z całą pewnością jej potencjał jest olbrzymi.

Widżety webowe przedstawiają dosyć poważne ryzyko w zakresie bezpieczeństwa. Są one często częścią serwisów, do których użytkownik ma pełne zaufanie i łatwo mogą wprowadzić użytkownika w błąd i uzyskać niepożądany dostęp do jego danych. Przykładem takiego zachowania był widżet „Secret Crush” rozprzestrzeniany w serwisie społecznościowym Facebook, zauważony przez Fortinet na początku 2008 roku, który skłaniał użytkowników do instalacji programu wyświetlającego reklamy Zongo [2].

2.5.4 Podsumowanie

Pomimo różnej formy dystrybucji widżetów, sposób działania i cel w każdym z trzech przypadków jest podobny: dostarczenie pojedynczej funkcjonalności w dostępny sposób. Największą siłą oprogramowania opartego o widżety jest dywersyfikacja dostawców poszczególnych widżetów, a co za tym idzie ich ciągła konkurencja, co z kolei sprawia, że poszczególne produkty stają się z dnia na dzień coraz lepsze. Nie bez znaczenia jest olbrzymia dostępność i generyczność systemów widżetów - jest to bezpośrednia przyczyna wzrostu popularności aplikacji w tej formie w ciągu ostatnich lat.

2.6. Wady istniejących rozwiązań

Pomimo olbrzymiej różnorodności rozwiązań, dostępnych technologii oraz lat doświadczeń istniejące rozwiązania nie są wolne od wad. Część wad istnieje z powodu ograniczeń narzucanych przez systemy operacyjne, część z kolei jest wynikiem wyboru „mniejszego zła".

Jednym z głównych problemów, które większość systemów zdaje się ignorować, jest izolowanie aplikacji od widżetów. Izolacja jest tutaj rozumiana jako sandboxing, czyli ograniczenie składowym mini-aplikacjom dostępu do zasobów komputera. Aplikacje te powinny mieć dostęp tylko do ograniczonych, z góry zdefiniowanych zasobów, ponadto niezdolność do kontynuowania przez nie pracy nie powinna mieć wpływu na stabilność i działanie całości systemu. Izolacja taka z punktu widzenia inżynierii oprogramowania jest niezwykle ważnym elementem systemów, w których mamy do czynienia z kodem, co do którego mamy ograniczone zaufanie. W przypadku systemów widżetów, gdzie poszczególne elementy systemu pochodzą od różnych dostawców, zaufanie do wybranych składowych musi być ograniczone. Jedynym istniejącym systemem widżetów, w którym zastosowano najwyższy poziom izolacji (osobne procesy i domeny aplikacji dla każdego widżetu) jest Yahoo! Widgets.

Ograniczone zaufanie do widżetów jest powodem kolejnego problemu, z którym borykają się twórcy tego oprogramowania, czyli bezpieczeństwa. Podobnie jak do poprzedniego zagadnienia, również w kwestii bezpieczeństwa podejście różni się w zależności od producenta oprogramowania. Jednym z systemów, który zupełnie ignoruje ten temat jest Apple Dashboard [3]. Widżety Apple Dashboard mogą posiadać uprawnienia nie większe od uprawnień użytkownika, który z nich korzysta, przy czym to sam widżet za pomocą pliku Info.plist decyduje, jakie uprawnienia zostaną mu przyznane. Ten plik XML znajduje się w zasobach każdego poprawnego widżetu i zawiera informacje na temat inicjalizacji widżetu, jego nazwy, wersji, dostawcy. Zawiera także 3 wartości Bollean o nazwach: AllowFileAccessOutsideOfWidget, AllowNetworkAccess oraz AllowSystem.

Te trzy parametry decydują o tym, czy widżet ma dostęp do plików poza katalogiem widżetu, czy ma dostęp do sieci oraz czy ma dostęp do linii poleceń.

Pomimo tego, że uprawnienia widżetu są ograniczone uprawnieniami użytkownika, który go uruchomił, nie zabezpiecza go to przed twórcą złośliwego widżetu, który wciąż może uzyskać pełny dostęp do plików użytkownika i ukraść je z użyciem sieci. Twórca widżetu ma praktycznie możliwość przejęcia pełnej kontroli nad komputerem użytkownika oraz uzyskania dostępu do poufnych informacji, takich jak hasło i wiadomości email.

Problem ten dotyczy prawdopodobnie większości aplikacji, z których w dzisiejszych czasach korzystamy, jednak łatwość instalacji widżetów, ich dostępność i nieświadomość zagrożenia ze strony użytkowników są wystarczającymi powodami, aby poważnie traktować zagrożenia z nich wynikające.

Podczas tworzenia aplikacji nie bez znaczenia jest ile pamięci operacyjnej i dyskowej ta aplikacja będzie zajmowała oraz w jakim stopniu może ona używać mocy obliczeniowej procesora. Podobnie jest w przypadku aplikacji widżetów. Im wymagania te są mniejsze, tym lepiej. Niestety im bardziej rozbudowana oraz miła dla oka jest aplikacja, tym więcej zajmuje ona zasobów. Dodatkowo wydaje się oczywiste, że zasoby powinny być zwalniane, gdy z danej funkcjonalności (widżetu) już nie będziemy korzystać. Projektanci aplikacji widżetów są zmuszeni iść na wiele kompromisów. Najbardziej „zachłanną" aplikacją z wymienionych w poprzednich rozdziałach jest Yahoo! Widget Engine. Spowodowane jest to świadomym wyborem, jako że przyjęte rozwiązanie problemu izolacji poszczególnych widżetów pociąga za sobą konsekwencje w postaci większej ilości zasobów koniecznych do uruchomienia każdego widżetu. Pomimo, że jest to wada tej aplikacji, jest ona spowodowana wyborem „większego dobra", jakim jest zwiększona stabilność całej aplikacji.

Ostatnią z wad zaobserwowanych przez Autora jest zbyt ubogie API dostępne dla twórców widżetów. Zbyt mała liczba dostępnym elementów oraz brak możliwości korzystania z zewnętrznych bibliotek sprawiają, że dewelopment widżetów staje się czasochłonny i trudny. Zdaniem Autora, idealnym rozwiązaniem jest pozwolenie użytkownikowi na tworzenie widżetu w sposób mu znany, tzn. tak, jak każdą inną okienkową aplikacę. Odnosi się to zarówno do interfejsu, jak i do komunikacji z zewnętrznymi bibliotekami. Oczywiście zbyt duża swoboda w tworzeniu widżetu budzi obawy dotyczące bezpieczeństwa i izolacji, jednak Autor uważa, że obawy te powinny zostać rozwiane nie na poziomie kodu widżetu, lecz aplikacji hosta. Aplikacja hosta powinna być odpowiedzialna za sandboxing i bezpieczeństwo, bez ograniczania API programisty.

3. Narzędzia użyte w pracy

3.1. Microsoft Visual Studio 2008

Microsft Visual studio to zestaw narzędzi programistycznych (Integrated Development Environment - IDE) stworzonych przez Microsoft. Jest on używany zarówno do tworzenia aplikacji konsolowych, jak i posiadających graficzny interfejs użytkownika (GUI). Możliwe jest tworzenie w nim aplikacji okienkowych, stron www, aplikacji webowych z użyciem kodu natywnego jak i zarządzanego na platformach obsługiwanych przez Microsft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework oraz Microsoft Silverlight.

Visual Studio posiada edytor kodu obsługujący IntelliSense oraz refaktoring kodu. IntelliSense to implementacja auto-uzupełniania kodu w czasie pisania. Jest to jeden z najbardziej rozbudowanych systemów tego typu dostępnych na rynku. Uwzględnia relacje pomiędzy obiektami klas projektu, przestrzenie nazw i powiązania bibliotek, przez co znacznie przyspiesza tworzenie oprogramowania.

Wbudowany debugger potrafi pracować na poziomie kodu, jak i maszyny.

Edytor graficznego interfejsu użytkownika pozwala na szybkie i efektywne tworzenie nawet rozbudowanych formularzy i wizualnych fajerwerków. Obsługuje używanie wtyczek, przez co możliwe jest łatwe rozszerzenie już istniejącej funkcjonalności. Możliwości wtyczek są ograniczone tylko wyobraźnią ich autorów. Zakres funkcjonalności wtyczek jest olbrzymi: od kompletnych systemów kontroli wersji (Subversion, Visual SourceSafe), przez wizualne edytory specyficznych języków programowania, do narzędzi wspierających proces wytwórczy oprogramowania (klient Team Foundation Server: Team Explorer).

Visual Studio obsługuje użycie języków oprogramowania poprzez wykorzystanie usług językowych, które pozwalają na użycie niemal każdego języka programowania przez programistę i debugger. Domyślnie Visual Studio obsługuje C/C++ (przez Visual C++), VB.NET (przez Visual Basic .NET), i C# (przez Visual C#). Obsługa innych języków takich jak Ruby i Python musi być instalowana jako osobne usługi językowe.

3.2. C# i .NET 3.5

C# jest językiem programowania korzystającym z wielu paradygmatów, takich jak: programowanie imperatywne, funkcyjne, generyczne, obiektowe i komponentowe (patrz [6]). Język

ten został stworzony przez Microsoft w ramach projektu platformy .NET i zatwierdzony jako standard przez Ecma (ECMA-334) i ISO (ISO/IEC 23270).

Z założenia C# jest prosty, nowoczesny, generyczny i obiektowy. Twórcą języka jest Anders Hejlsberg, twórca Borland's Turbo Pascal'a. Obiektowo zorientowana składnia oparta o klasy została zapożyczona z języka C++.

Najnowsza wersja języka 3.0 została wydana wraz z .NET 3.5 w 2007 roku. Następna wersja oznaczona jako 4.0 wciąż jest w trakcie tworzenia.

Niektóre cechy języka c#:

- Podobnie jak Java posiada hierarchię klas, gdzie klasa System.Object jest elementem nadrzędnym. Dotyczy to także typów prostych, takich jak: int, double czy bool;
- W założeniach języka przyjęto, że odśmiecaniem, czyli usuwaniem nieużywanych obiektów z pamięci, zajmuje się środowisko uruchomieniowe. Nie jest więc konieczne zwalnianie pamięci manualnie przez programistę, tak jak ma to miejsce w języku C, lecz zajmuje się tym Garbage Collector;
- Wielodziedziczenie jest niedozwolone;
- C# jest o wiele bezpieczniejszy w zakresie konwersji typów niż C++. Konwersja typów bez konieczności manualnego rzutowania możliwa jest tylko w przypadku konwersji uznanych za bezpieczne (np: rzutowanie do klasy, po której następuje dziedziczenie);
- Umożliwia tworzenie rozbudowanych mechanizmów refleksji i atrybutów klas. Dzięki temu możliwe jest tworzenie oprogramowania, którego części kodu nie są znane w czasie jego kompilacji. Ma to szerokie zastosowanie w bibliotekach typu ORM (Obiektowo-Relacyjne Mapowanie), czy też w narzędziach analizy kodu AOP (Programowanie Aspektowe). Atrybuty klas są wzorowane na adnotacjach z języka Java od wersji 1.5.

Platforma programistyczna Microsoft .NET Framework obejmuje środowisko uruchomieniowe (Common Language Runtime – CLR) oraz biblioteki dostarczające podstawowe funkcjonalności dla tworzonego oprogramowania. Na aplikację składa się kod programisty w połączeniu z bibliotekami dostarczonymi przez platformę .NET. Wszystkie aplikacje uruchamiane w ramach CLR są zarządzane przez maszynę wirtualną.

Wirtualna maszyna zarządza wykonaniem aplikacji napisanych specjalnie pod platformę .NET w taki sposób, że programista nie musi się przejmować możliwościami procesora maszyny, na której aplikacja jest uruchomiona. CLR dostarcza także mechanizmy bezpieczeństwa, zarządzania pamięcią i obsługi wyjątków.

Podstawowe cechy platformy .NET to:

- Kompatybilność ze starszymi rozwiązaniami - interakcja pomiędzy różnymi wersjami aplikacji jest często konieczna. .NET dostarcza rozwiązań pozwalających na dostęp aplikacji do funkcji wykonywanych poza platformą .NET. Dostęp do komponentów COM jest zapewniony przez przestrzenie nazw System.Runtime.InteropServices i System.EnterpriseServices. Dostęp do dodatkowych funkcjonalności jest możliwy dzięki usługom P/Invoke;
- Uproszczone wdrażanie - .NET zawiera zestaw narzędzi pomagających w zarządzaniu instalacją oprogramowania, dających pewność, że nie będzie ono kolidowało z wcześniej zainstalowanym oprogramowaniem oraz że podlega odpowiednim regułom bezpieczeństwa;
- Bezpieczeństwo - architektura .NET ma na celu rozwiązanie wielu problemów bezpieczeństwa aplikacji, takich jak przepełnienie bufora, które jest często wykorzystywane przez złośliwe oprogramowanie. Dodatkowo .NET dostarcza jeden wspólny model bezpieczeństwa dla aplikacji tworzonych w jego ramach;
- Przenośność - teoretycznie architektura platformy .NET sprawia, że może ona być niezależna od platformy systemowej. Oznacza to, że każdy program napisany w ramach platformy .NET powinien móc zostać uruchomiony na każdej platformie systemowej, na której istnieje implementacja .NET. Jednak Microsoft stworzył dotychczas tylko implementacje pod Windows, Windows CE i XBox 360 (powstała wersja implementacji .NET 1.0, jednak nie była ona kontynuowana, a jej licencja pozwalała tylko na użycie w celach edukacyjnych). Dokładne dokumentacje Common Language Infrastructure (zawierająca podstawowe biblioteki klas Common Type System i Common Intermediate Language), języków C#, C++/CLI zostały zgłoszone jako standardy do ECMA i ISO, co przynajmniej w teorii pozwala firmom zewnętrznym na stworzenie implementacji .NET w ramach innych platform. Jedną z takich implementacji jest Mono. Implementacja ta jest wieloplatformowa (Mac OS X, Linux, BSD, Windows). Na stan dzisiejszy (Lipiec 2009) posiada ona pełną implementację .NET w wersji 2.0 oraz część bibliotek z wersji 3.5. Projekt jest na bieżąco rozwijany i sponsorowany przez firmę Novell.

3.3. Managed Addin Framework (MAF) i System.Addin

Deweloperzy oprogramowania od dawna spotykają się z problemem tworzenia oprogramowania łatwo skalowalnego, pozwalającego na rozszerzenie funkcjonalności bez ryzyka destabilizacji pracy całej aplikacji.

Platforma .NET dostarczyła narzędzi pozwalających rozwiązać ten problem u podstaw w formie API refleksji i Domen Aplikacji (AppDomain). Managed Addin Framework (czasem określany jako System.Addin) jest zbudowany właśnie z użyciem tych podstawowych narzędzi.

Dostarczane usługi wysokiego poziomu pozwalają na dynamiczne wykrywanie, wczytywanie, zabezpieczanie i interakcję z zewnętrznymi bibliotekami.

Managed Addin Framework definiuje programistyczny model zbudowany na szczycie platformy .NET, który pozwala aplikacjom dynamicznie ładować i komunikować się z generycznymi komponentami w czasie wykonywania programu. Dostarcza rozwiązanie pozwalające na tworzenie wersjonowanego, składającego się z niezależnych od siebie modułów oprogramowania.

Model MAF

Model proponowany przez MAF składa się z serii segmentów, które tworzą kanał komunikacji (Pipeline). Kanał jest odpowiedzialny za komunikację pomiędzy add-in'ami, a aplikacją hosta. Jest to symetryczny model komunikacji. Poszczególne segmenty pomiędzy hostem i add-in'em dostarczają warstw abstrakcji koniecznych do wersjonowania i izolacji wtyczek.

Schemat 1 przedstawia poszczególne segmenty kanału komunikacji (Pipeline).

Schemat 1. Kanał komunikacji (Pipeline)

Źródło: Opracowanie własne

Biblioteki znajdujące się w poszczególnych segmentach nie muszą znajdować się w tej samej domenie aplikacji. Każda wtyczka może zostać załadowana do nowej lub istniejącej Domeny Aplikacji (AppDomain), a nawet do domeny aplikacji hosta. Możliwe jest załadowanie wielu wtyczek do tej samej Domeny Aplikacji, co pozwala wtyczkom na dzielenie zasobów oraz kontekstu bezpieczeństwa. Model proponowany przez MAF obsługuje i sugeruje użycie opcjonalnej strefy izolacji. Strefa ta może dostarczać izolacji na poziomie domeny aplikacji lub procesu. Segment zawierający kontrakty, znajdujący się w centrum kanału komunikacji, jest ładowany do domeny aplikacji hosta jak i do domen każdej z wtyczek. Kontrakt definiuje wirtualne metody, których host i add-in używają do wymiany informacji między sobą.

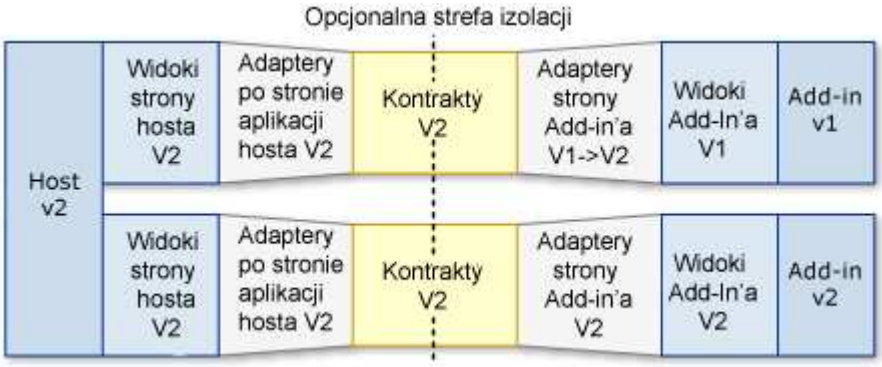
Aby przekroczyć strefę izolacji, typy wymiany danych muszą posiadać możliwość serializacji, lub być kontraktami. Typy, które nie spełniają tych warunków muszą zostać przekonwertowane do kontraktów przez odpowiednie adaptery.

Segmenty z widokami zawierają abstrakcyjne klasy bazowe lub interfejsy dostarczające aplikacji hosta i add-in'om widoki opisujące ich wzajemną strukturę.

Niezależne wersjonowanie

MAF pozwala aplikacji hosta i poszczególnym wtyczkom na niezależne wersjonowanie. W wyniku tego możliwe są implementacje poniższych scenariuszy:

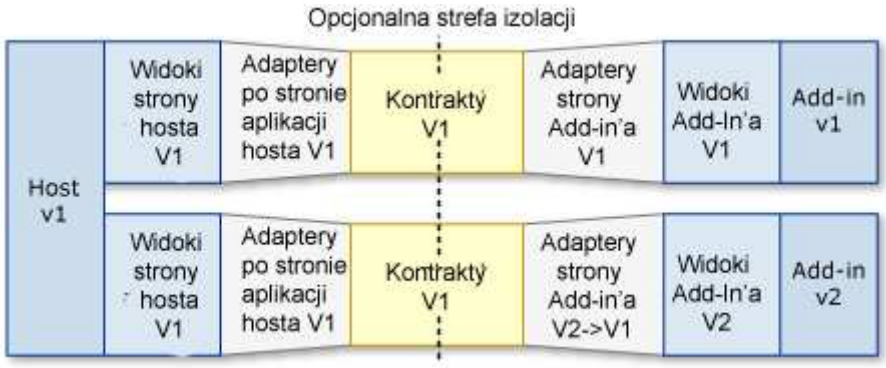
- Zbudowanie adaptera pozwalającego aplikacji hosta na komunikację z wtyczką stworzoną dla starszej wersji aplikacji hosta (patrz Schemat 2);



Schemat 2. Wersjonowanie. Add-in dla starszej wersji hosta

Źródło: Opracowanie własne

- Zbudowanie adaptera pozwalającego aplikacji hosta na komunikację z wtyczką stworzoną dla nowszej wersji aplikacji hosta (patrz Schemat 3);



Schemat 3. Wersjonowanie. Add-in dla nowszej wersji hosta

Źródło: Opracowanie własne

- Zbudowanie adaptera pozwalającego aplikacji hosta na komunikację z wtyczką stworzoną dla innej aplikacji hosta.

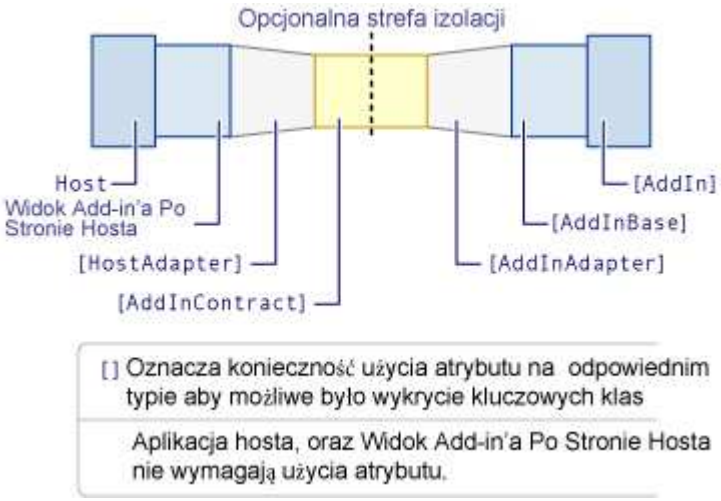
Wykrywanie i Aktywacja

Budowa kanału komunikacji (Pipeline) jest wykrywana przez API MAF poprzez identyfikację atrybutów klas.

Poniższe segmenty kanału komunikacji potrzebują następujących atrybutów na klasach, które je identyfikują:

- Add-in'y potrzebują atrybutu [AddInAttribute],
- Kontrakty potrzebują atrybutu [AddInContractAtribute],
- Adaptery po stronie AddIn'a potrzebują atrybutu [AddInAdapterAttribute],
- Widoki AddIn potrzebują atrybutu [AddInBaseAttribute],
- Adaptery po stronie aplikacji hosta potrzebują [HostAdapterAttribute].

Zależności te przedstawia Schemat 4.



Schemat 4. Atrybuty klas w poszczególnych segmentach kanału komunikacji

Źródło: Opracowanie własne

Wtyczki są aktywowane poprzez użycie żetonów reprezentujących wtyczki. Żetony te są przechowywane w składzie informacji, składającym się z dwóch plików będących buforami

podręcznymi: składu kanału informacji (pipeline store) oraz składu wtyczki (add-in store). Dzięki buforom podręcznym nie jest konieczne przeprowadzanie procesu wykrywania ponownie za każdym razem, gdy chce się uzyskać dostęp do danego segmentu modelu MAF.

Poziomy izolacji i odrębne procesy

Model proponowany przez MAF obsługuje parę poziomów izolacji pomiędzy wtyczką a hostem oraz pomiędzy wtyczkami. Zaczynając od najsłabszej izolacji, poziomy są następujące:

- Wtyczka jest uruchamiana w tej samej domenie aplikacji, co host. Scenariusz ten nie jest zalecany, ponieważ brakuje w nim izolacji oraz tracona jest możliwość dynamicznego unload'owania wcześniej załadowanych bibliotek;
- Wiele wtyczek jest uruchamianych w ramach tej samej domeny, różnej od domeny aplikacji hosta;
- Każda wtyczka jest uruchamiana we własnej domenie, co stanowi najpopularniejszy poziom izolacji;
- Wiele wtyczek jest uruchamianych w ramach jednej domeny aplikacji w odrębnym procesie;
- Każda wtyczka jest uruchamiana we własnej domenie i własnym odrębnym procesie. Scenariusz ten gwarantuje najwyższy poziom izolacji.

Kontrakty

Pierwszym etapem w tworzeniu kanału komunikacyjnego jest zdefiniowanie kontraktów, które muszą implementować interfejs IContract. Jeżeli host i wtyczka są ładowane do osobnych domen aplikacji, pomiędzy nimi znajduje się strefa izolacji. Kontrakt to niewersjonowalny interfejs definiujący protokół komunikacji i wymiany danych przez strefę izolacji. Użycie kontraktów chroni przed przekraczaniem strefy izolacji przez implementację aplikacji hosta i add-in'ów, co mogłoby powodować problem niespójności wersji.

Wymagania kontraktów

Kontrakty muszą spełniać szereg wymagań. Ich spełnienie daje pewność, że wszystkie typy danych uczestniczących w komunikacji mogą być wersjonowane, są bezpieczne i mogą przekroczyć strefę izolacji pomiędzy aplikacją hosta a wtyczką.

Kontrakt musi implementować IContract i może używać tylko poniższych typów:

- Innych kontraktów implementujących interfejs IContract,

- Prymitywnych typów danych (int, double, bool),
- Typów oznaczonych jako Serializable i zdefiniowanych w bibliotece z kontraktami,
- Typów oznaczonych jako Serializable i zdefiniowanych w bibliotece Mscorlib.dll, takich jak Int32 i DateTime,
- Zamkniętych, oznaczonych jako Serializable typów referencyjnych. Przykładem takiego typu danych jest String,
- Enumeracji zdefiniowanych w Mscorlib.dll lub bibliotece kontraktów,
- Obiektów typu AddInToken,
- Tablic powyższych typów, oprócz kontraktów. Przesyłanie kolekcji możliwe jest tylko poprzez użycie kolekcji implementującej generyczny interfejs IList<T>. W celu przesłania takiej kolekcji konieczna jest konwersja do typu IListContract<T> (MAF dostarcza gotowe adaptery pozwalające na taką konwersję).

Aby zbudować kompletny kanał komunikacyjny, kontrakt reprezentujący wtyczkę musi zostać oznaczony atrybutem AddInContractAttribute.

Kolejnym etapem jest stworzenie widoków i adapterów po obu stronach kanału komunikacyjnego. Te segmenty dostarczają widoki na wspólne struktury danych oraz adaptery pozwalające na konwersję widoków do kontraktów oraz kontraktów do widoków.

Widoki

Widoki po stronie aplikacji hosta i add-in'a to biblioteki zawierające interfejsy i abstrakcyjne klasy reprezentujące ich widoki na siebie nawzajem oraz typy danych używane w komunikacji pomiędzy nimi. Widoki nie są zależne w żaden sposób od kontraktów użytych w komunikacji pomiędzy nimi. Widoki dostarczają separacji pomiędzy implementacjami aplikacji hosta i add-in'ów. Taka architektura pozwala na zmianę adapterów i kontraktów bez wpływu na aplikację hosta lub którejkolwiek z wtyczek. Warunkiem koniecznym do stworzenia poprawnego kanału komunikacji jest oznaczenie widoku reprezentującego add-in atrybutem AddInBaseAttribute. Oznaczanie poszczególnych klas jest konieczne dla poprawnego wykrycia biblioteki zawierającej add-in. Jak już zostało wspomniane, przy okazji Odkrywania i Aktywacji widok reprezentujący aplikację hosta nie wymaga żadnych atrybutów, ponieważ jest on przekazywany do metody FindAddIns zajmującej się wyszukiwaniem add-in'ów.

Adaptery

Adaptery stron aplikacji hosta i add-in'ów to biblioteki zawierające klasy będące adapterami używanymi do konwersji z i do widoków i kontraktów. Określenie „strona” odnosi się tutaj do strony

kanału aplikacji, tak jak przedstawione jest to na Schemacie 1. W zależności od kierunku komunikacji adapter dokonuje konwersji z widoku na kontrakt lub z kontraktu na widok. W przypadku zapytań w obu kierunkach, konieczne są adaptery po obu stronach kanału komunikacji.

Istnieją dwa rodzaje adapterów:

1) Adapter Widok-Do-Kontraktu (view-to-contract adapter)

Klasa w bibliotece adapterów odpowiedzialna za konwersję widoku do kontraktu. Konwersja jest zwykle dokonywana poprzez przekazanie widoku w konstruktorze klasy. Adapter taki jest kontraktem, więc może przekroczyć strefę izolacji. Klasa taka musi dziedziczyć ContractBase i implementować dany kontrakt.

2) Adapter Kontrakt-Do-Widoku (contract-to-view adapter)

Klasa w bibliotece adapterów odpowiedzialna za konwersję kontraktu do widoku. Klasa ta implementuje lub dziedziczy odpowiedni widok, do którego ma się odbyć konwersja. Pola widoku są ustawiane poprzez konwersję pól kontraktu przekazanego do adaptera w konstruktorze.

Podobnie jak to miało miejsce w kontraktach i widokach, konieczne jest oznaczenie kluczowych klas odpowiedzialnych za komunikację odpowiednimi atrybutami. Adapter odpowiedzialny za konwersję widoku add-in'a do kontraktu po stronie add-in'a oznaczany jest atrybutem AddInAdapterAttribute, natomiast adapter odpowiedzialny za konwersję widoku aplikacji hosta do kontraktu po stronie hosta atrybutem HostAdapterAttribute.

Adaptery nie muszą być publiczne.

Tworzenie Add-In'a

Gdy zostanie już zbudowany kompletny kanał komunikacji wraz z aplikacją hosta można przejść do tworzenia add-in'ów. Każdy add-in implementuje abstrakcyjny widok, opisujący metody w nim dostępne. Przykład 1 prezentuje przykładowy add-in:

Przykład 1. Sposób tworzenia Add-In’a

```
namespace MyAddIn
{
   [System.AddIn.AddIn("Clocker", Description = "Zegar z wyborem strefy czasowej",
Publisher = "Robert Meisner",Version="1.0.0.1")]
   public  class MyAddinCode :WidgetBase.WidgetBase
   {
      public override void init()
      {
       ...
      }
      public override void dispose()
```

```
    {
     …
    }
  }
}
```

Add-in ten implementuje widok **WidgetBase,** dostarczając ciała metody init i dispose. Klasa ta jest oznaczona atrybutem **[AddIn]**, który pozwala na podanie nazwy, wersji, opisu i innych danych umożliwiających jego identyfikację.

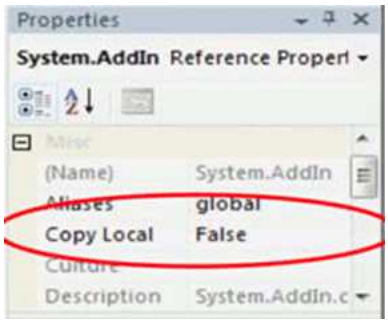
Struktura katalogów wdrożenia

W celu poprawnej identyfikacji każdego z komponentów, MAF wymusza z góry ustaloną strukturę katalogów wdrożenia (Rysunek 6). Każda część modelu MAF znajduje się w osobnym katalogu w katalogu głównym kanału komunikacji (domyślnie jest to **APPBASE**, czyli miejsce, w którym znajdują się pliki wykonywalne aplikacji hosta).

Rysunek 6. Struktura katalogów wdrożenia projektu MAF

Źródło: Opracowanie własne

Dokładne nazwy katalogów są wymagane (wielkość liter nie ma znaczenia). Każdy katalog przechowuje pojedynczą część kanału komunikacji. Są one ładowane dynamicznie przez MAF w momencie ładowania bibliotek add-in'a. Widok aplikacji hosta zawsze znajduje się w tym samym katalogu, co pliki wykonywalne hosta. W czasie tworzenia projektu w Visual Studio ważne jest ustawienie katalogów kompilacji tak, aby odzwierciedlały one powyższą strukturę. Dodatkowo wszystkie referencje pomiędzy poszczególnymi bibliotekami muszą być oznaczone jako **CopyLocal = false**, aby lokalne kopie bibliotek nie były umieszczane w poszczególnych katalogach.



Rysunek 7. Opcje referencji pomiędzy bibliotekami w projekcie MAF

Źródło: Opracowanie własne

3.4. Windows Presentation Foundation (WPF)

Windows Presentation Foundation, w skrócie WPF, to graficzny silnik służący do renderowania interfejsu użytkownika w aplikacjach okienkowych nowej generacji (patrz [6]). WPF został wydany wraz z Platformą .NET w wersji 3.0. Jest to duży krok w podejściu do projektowania i implementacji GUI. WPF został zbudowany zupełnie niezależnie od dotychczasowego, przestarzałego silnika renderującego GDI. WPF dostarcza jednorodny model programistyczny pozwalający na budowanie aplikacji oraz dostarczający bezwzględną separację interfejsu użytkownika i logiki biznesowej. WPF jest zbudowany w oparciu o DirectX, który umożliwia sprzętową akceleracje rysowania, co pozwala na zaawansowane wizualne efekty jak przezroczystości, gradienty, filtry i transformacje.

Architektura

Budowa WPF zawiera w sobie zarówno elementy napisane w kodzie zarządzanym, jak i niezarządzanym, jednak całe API udostępnione deweloperom jest w kodzie zarządzanym. Silnik kompozycji odpowiedzialny za rysowanie interfejsu użytkownika działa w kodzie niezarządzanym bez udziału .NET ze względu na wydajność. Silnik ten nosi nazwę Media Integration Layer (MIL) i znajduje się w milcore.dll. Korzysta on z możliwości DirectX i dostarcza podstawową obsługę zarówno dla powierzchni 2D, jak i 3D. Wszystkie kodeki odpowiedzialne za poprawne wyświetlanie

zawartości multimedialnej są także zaimplementowane w niezarządzanym kodzie i znajdują się w windowscodecs.dll.

Podstawową biblioteką dla każdego programisty rozpoczynającego pracę w technologią WPF jest PresentationCore. Dostarcza ona wraper dla MIL napisany w zarządzanym kodzie. Ponadto implementuje podstawowe usługi konieczne dla każdej aplikacji WPF. Do tych usług zalicza się system przetwarzania wiadomości używający obiektu Dispatcher, implementującego rozbudowany system zdarzeń i usługi mogące używać systemów układu interfejsu użytkownika (np. wymiarowanie elementów).

WPF udostępnia system własności (property) dla obiektów dziedziczących klasę DependencyObject. System ten monitoruje zależności pomiędzy konsumentami własności i może wykonywać różne zdarzenia na podstawie zmian w danych własnościach. Własności w WPF obsługują powiadamianie o zmianach (change notification), które wywołują wbudowane zachowania (behaviour), gdy tylko zajdzie jakakolwiek zmiana. Możliwe jest tworzenie własnych zachowań propagujących powiadomienie o zmianie własności do odpowiednich obiektów. System zarządzający układem interfejsu użytkownika używa takich zachowań do wyzwalania przeliczania układu UI w przypadku zmiany własności. Taka architektura pozwala na użycie paradygmatu deklaratywnego programowania, gdzie prawie wszystko, począwszy od zmiany kolorów i pozycji, do animacji elementów, odbywa się poprzez zmianę własności. Dzięki takiemu systemowi możliwe jest pisanie aplikacji WPF w XAML - deklaratywnym języku znaczników, poprzez łączenie słów kluczowych i atrybutów bezpośrednio z klasami WPF i własnościami.

Elementy interfejsu użytkownika są ekstensją klasy Visual. Obiekty Visual dostarczają interfejsu dla drzewa kompozycji zarządzanego przez Media Integration Layer (MIL). Każdy element WPF dodaje jeden lub więcej węzłów kompozycji do drzewa. Węzły te zawierają instrukcje dotyczące rysowania, takie jak instrukcje transformacji wizualnej czy też przycinania obiektów. Cała wizualna część aplikacji jest w rzeczywistości zbiorem węzłów kompozycji przetrzymywanych w podręcznym buforze w pamięci komputera.

Okresowo MIL trawersuje po strukturze drzewa, wykonując instrukcje dotyczące rysowania w każdym węźle. Powoduje to rysowanie każdego elementu na powierzchni DirectX, która następnie jest przedstawiana na ekranie. MIL używa algorytmu malarza, który rysowanie rozpoczyna od elementów w tle, dzięki czemu łatwo osiągnąć złożone efekty, takie jak np. przezroczystość czy rozmycie. Proces rysowania jest wspomagany przez GPU.

Wszystkie aplikacje WPF starują z dwoma wątkami: jeden zarządza UI, a drugi działa w tle, obsługując renderowanie i przerysowywanie (są one wywoływane przez WPF automatycznie bez potrzeby interwencji dewelopera). Wątek UI przechowuje obiekt Dispatcher'a (instancja

DispatcherObject), który z kolei przechowuje kolejkę operacji do wykonania na interfejsie użytkownika.

Tworzenie układu interfejsu użytkownika jest podzielone na dwie fazy: Mierzenie (Measure) i Porządkowanie (Arrange). Faza Mierzenia rekursywnie odpytuje wszystkie elementy i ustala, jaki będzie ich rozmiar. Faza Porządkowania rekursywnie układa wszystkie elementy w stosunku do swoich rodziców w drzewie, w zależności od użytego algorytmu układu.

XAML

XAML (Extensible Application Markup Language) jest językiem opartym na XML. Sukces języków opartych na znacznikach (HTML) w tworzeniu aplikacji webowych sprawił, że architekci WPF wprowadzili nowy język do opisywania interfejsu użytkownika - XAML.

Niewątpliwą zaletą XAML jest to, że jest on w pełni deklaratywnym językiem. W deklaratywnych językach programowania programista opisuje zachowania i integruje poszczególne komponenty bez użycia programowania proceduralnego.

Mimo, że rzadko spotyka się aplikacje napisane wyłącznie w XAML'u, to wprowadzenie tego języka zrewolucjonizowało sposób tworzenia aplikacji okienkowych na platformie Windows. Używanie XAML do tworzenia interfejsów użytkownika rozdziela model logiczny od widoku, co przez środowisko inżynierów oprogramowania jest uznane za dobrą praktykę. W XAML elementy i atrybuty są mapowane do odpowiedniego API jako klasy i własności. Wszystkie elementy w WPF mogą być oprogramowane zarówno za pomocą języków proceduralnych/obiektowych (C#, VB.NET), jak i XAML.

3.5. Thriple i FishEye Panel

W dzisiejszych czasach nie wystarczy już dostarczyć użytkownikowi funkcjonalnej aplikacji, ponieważ oprócz niezawodności i spełnienia wymagań funkcjonalnych użytkownik oczekuje miłego dla oka, estetycznego, ciekawego interfejsu.

Programiści, ku uciesze użytkowników, prześcigają się w coraz to bardziej wysublimowanej prezentacji danych i nawigacji. Także w tym projekcie estetyka wykonania jest nie bez znaczenia. Aby aplikacja sprostała dzisiejszym, wysokim standardom Autor użył dwóch bibliotek: Thriple oraz FishEye Panel. Obie biblioteki są dostarczane na licencji OpenSource, wraz ze źródłami, co pozwoliło dokonać pewnych zmian i usprawnień w ich działaniu.

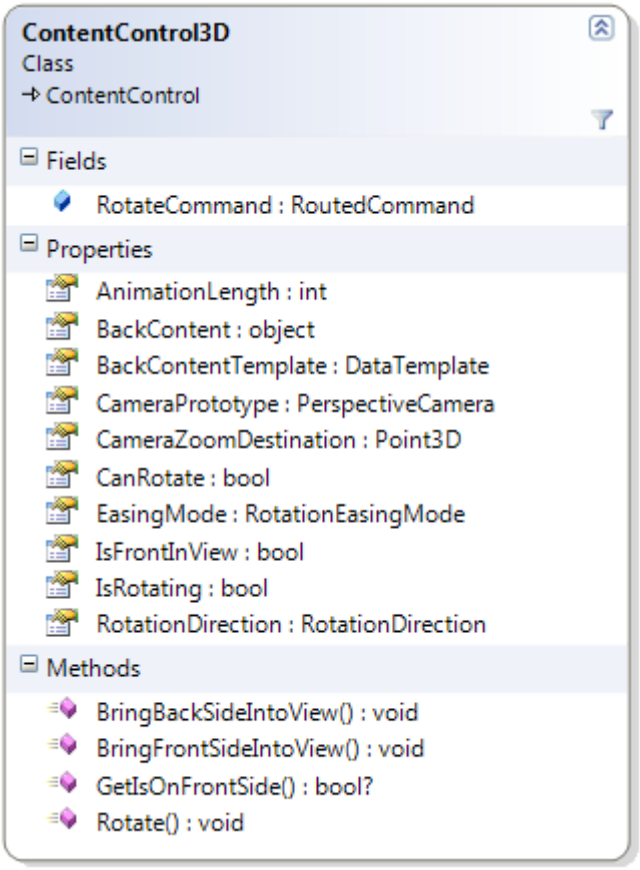
Thriple

Thriple jest biblioteką kontrolek 3D i paneli gotowych do użycia w aplikacjach WPF. Zawiera proste w użyciu komponenty pozwalające na dodanie efektów 3D do interfejsu użytkownika. Każdemu komponentowi towarzyszy prosta aplikacja pozwalająca programistom na szybką i efektywną naukę korzystania z nich.

Biblioteka jest dystrybuowana w wersji skompilowanej (jako dll) oraz jako otwarte źródła. Zawiera dwa komponenty: ContentControl3D oraz Panel3D.

Na potrzeby niniejszej pracy Autor zaadaptowana i użył tylko kontrolki ContentControl3D.

ContentControl3D daje możliwość obracania treści w przestrzeni 3D. Zawiera wiele własności pozwalających na kontrolowanie animacji obrotu, jej kierunku oraz efektu złagodzenia (easing effect).



Rysunek 8. Atrybuty i metody klasy ContentControl3D

Źródło: Opracowanie własne

Panel 3D jest panelem układu pozwalającym na ułożenie poszczególnych treści w prostej linii w przestrzeni 3D. Posiada metodę MoveItems pozwalającą na animację ruchu elementów w panelu.

FishEye Panel

To panel napisany przez Paul'a Tallet'a. Inspiracją były HyperBar (przykład z Expression Interactive Designer) oraz pasek Dock'a z systemu Mac OS X.

Różnica w jego działaniu, w stosunku do swoich pierwowzorów, polega na innym sposobie skalowania elementów w panelu. Elementy najbliżej kursora myszki powiększają się, natomiast pozostałe w tym samym momencie się kurczą, robiąc miejsce dla pozostałych.

Sprawia to, że Panel pozostaje stałej szerokości. Nie było to zachowanie pożądane w przypadku dużej liczby elementów. Animacja zachowywała się niestabilnie w przypadku, gdy domyślna szerokości elementów przekraczała szerokość panelu. Autor rozwiązał powyższy problem wprowadzając auto-skalowanie szerokości panelu w zależności od szerokość wszystkich elementów, jednocześnie zachowując efekt kurczenia.

4. Zakres funkcjonalności aplikacji

Kompletny system widżetów to nie tylko aplikacja zarządzająca widżetami, ale także możliwie duża baza widżetów dla niej stworzonych. Opis funkcjonalności, jakie taki system musi implementować, musi uwzględniać zarówno Aplikację Hosta jak i Widżet.

4.1. Aplikacja Hosta

We wszystkich systemach widżetów centralną jednostką zarządzającą jest aplikacja hosta (w skrócie host). Twórcy takich systemów uznali, że gwiaździsta architektura zarządzania najbardziej pasuje do systemów tego typu. Model ten stał się standardem w branży. Funkcjonalność hosta we wszystkich aplikacjach tego typu jest podobna, jednak użyte rozwiązania do jego implementacji są różne.

Pierwszym zadaniem aplikacji hosta, po jej uruchomieniu, jest wykrycie dostępnych widżetów w systemie. Zwykle pliki widżetów znajdują się w jednym katalogu na dysku. Host przegląda katalog i wykrywa istniejące w nim widżety. Po ich wykryciu dodaje je do listy dostępnych widżetów. Użytkownik ma możliwość uruchomienia danego widżetu po wyborze go z listy. Lista widżetów także przybiera różne formy - od popularnych list wyboru, po animowane w przestrzeni 3D karuzele.

Twórcy systemów widżetów często ukrywają interfejs graficzny, co sprawia wrażenie, że każdy widżet jest samodzielnym bytem - aplikacją działającą niezależnie od żadnej innej. W rzeczywistości jednak aplikacja hosta zawsze kontroluje działanie poszczególnych widżetów, izoluje je i dostarcza API do komunikacji.

Aplikacja musi zarządzać cyklem życia widżetów, ich tworzeniem, jak i zamykaniem. Zamknięcie aplikacji hosta musi również prawidłowo zamknąć wszystkie widżety.

4.1.1 Wykrywanie widżetów

Identyfikacja widżetów przebiega w różny sposób, w zależności od sposobu ich implementacji. Widżety mogą być wykrywane poprzez nazewnictwo plików np. plik nazwany widget.dll w katalogu o nazwie wtyczki zawiera bibliotekę widżetu lub plik uruchomieniowy. Jest to jednak bardzo naiwna identyfikacja, nie pozwalająca w rzeczywistości wykryć, czy dany plik jest poprawnego formatu i tym samym konieczny jest osobny mechanizm weryfikujący.

Kolejnym rodzajem wykrywania istniejących widżetów są pliki konfiguracyjne, zwykle w formacie XML. Zawierają one wszelkie informacje na temat wtyczki, takie jak nazwa, lokalizacja

pliku uruchomieniowego, wersja, opis, czy też informacje o oczekiwanym zbiorze uprawnień przez widżet. Host szuka plików konfiguracyjnych o danym rozszerzeniu i na ich podstawie buduje listę. Systemem implementującym takie rozwiązanie jest Apple Dashboard.

W przypadku skompilowanych widżetów do bibliotek lub plików wykonywalnych możliwe jest użycie bardziej wyrafinowanych metod wykrywania. Użycie mechanizmów refleksji pozwala na jednoczesną weryfikację poprawności struktury widżetu, jak i na jego wykrycie niezależnie od nazwy pliku. Managed Add-in Framework dostarcza narzędzi pozwalających w łatwy sposób wykryć biblioteki zawierające klasy implementujące dany interfejs znajdujący się w danym katalogu. U swoich podstaw wykorzystuje on mechanizmy refleksji.

4.1.2 Aktywacja widżetów

Aktywacja widżetów to nic innego jak ich uruchomienie, rozpoczęcie cyklu życia. W przypadku systemów, w których widżety to skompilowane biblioteki, niezbędne jest przygotowanie środowiska uruchomieniowego przed aktywacją. Środowisko to powinno dostarczać pożądany zbiór zasobów oraz ograniczać uprawnienia widżetu. Sandboxing jest jedną z kluczowych funkcjonalności, które powinna zawierać każda aplikacja hosta.

Sandboxing to popularna technika tworzenia ograniczonych środowisk uruchomieniowych, które mogą być używane do uruchamiania niezaufanych programów. Sandbox (piaskownica) ogranicza lub redukuje poziom dostępu dla programów, które zawiera - jest rodzajem kontenera. Przykład 2 prezentuje sposób wykorzystania techniki Sandboxing'u.

Przykład 2. Sandboxing

Jeżeli proces P uruchamia proces Q w piaskownicy wtedy zwykle uprawnienia Q są ograniczone do podzbioru uprawnień P. Za przykład możemy wziąć sytuację, w której P jest uruchomiony z pełnymi uprawnieniami administratora w danym systemie operacyjnym. P ma dostęp do wszystkich innych procesów systemu. Q znajdujący się w piaskownicy, stworzonej przez P, ma dostęp tylko do tych procesów, które znajdują się w tej samej piaskownicy. Pomijając możliwe luki w oprogramowaniu piaskownicy, zakres możliwych szkód spowodowanych przez wadliwy proces Q jest ograniczony.

Sandboxing był w centrum zainteresowania twórców wszelkich systemów informatycznych od dawna. W 1971 roku w publikacji zatytułowanej „Protection" [4]. Butler Lampson dostarczył abstrakcyjny model podkreślający właściwości istniejących mechanizmów kontroli dostępu oraz ochrony. Wiele z tych mechanizmów było równoważnych semantycznie z pojęciem sandboxing'u. Lampson użył sformułowania „domena" jako odniesienia do ograniczającego środowiska. Podobnie twórcy platformy .NET nazwali środowiska uruchomieniowe w ramach CLR domenami aplikacji (Application Domain, AppDomain).

Widżety pochodzą często z nieznanych źródeł, od niezaufanych dostawców, a łatwość ich instalacji, dostępność i nieświadomość użytkowników zmusza deweloperów do zachowania szczególnych środków ostrożności.

Najwyższa formę ochrony w systemie Windows zapewnia uruchamianie każdego widżetu w osobnym procesie we własnej domenie aplikacji. Jako jedyni z twórców popularnych systemów widżetów, taką formę ochrony przyjęli deweloperzy aplikacji Yahoo! Widgets.

Podobny poziom ochrony zapewniają systemy, w których widżety są oparte o języki skryptowe takie jak: Javascript czy LUA. Aplikacja hosta może implementować interpreter tych języków i obsługiwać tylko pewien bezpieczny podzbiór ich funkcjonalności.

Uruchomiony widżet ma możliwość inicjalizacji. Inicjalizacja pozwala wtyczce na ustawienie swojego stanu początkowego. Następnie jej zadaniem jest dostarczenie swojej graficznej prezentacji. Metody dostarczenia prezentacji graficznej przez widżet zostały opisane w rozdziale 4.2.5.

4.1.3 Lifetime Management widżetów

Podstawową funkcją każdej aplikacji hosta jest zarządzanie cyklem życia (lifetime management) widżetów. Cykl życia rozpoczyna się w momencie uruchomienia widżetu. Host musi być świadomy, które widżety są uruchomione, a które zakończyły swoje działanie. Aplikacja hosta, aby być zdolną do kontrolowania cyklu, musi przechowywać referencje do poszczególnych widżetów. Utrata referencji powoduje utratę możliwości kontroli. W środowisku .NET wszelkie obiekty, do których nie ma żadnych zewnętrznych referencji są usuwane przez kolektor śmieci (Garbage Collector). Cykl życia to maszyna stanów, w której pewne zdarzenia zmieniają miejsce, w jakim dany widżet znajduje się w cyklu. Przykładem jest zamknięcie widżetu przez użytkownika. Zdarzenie to powoduje przejście widżetu do stanu „nieaktywny”. Wiąże się to z szeregiem akcji, jakie musi podjąć aplikacja hosta takich jak zapisanie tego faktu w plikach konfiguracji czy zwolnienie pamięci po instancji widżetu.

4.1.4 Wersjonowanie

Tworząc jakiekolwiek oprogramowanie należy przewidzieć fakt, że będzie ono rozwijane. Wersjonowanie to proces nadawania unikalnych nazw lub numerów wersji kolejnym stanom rozwoju danego oprogramowania.

Zmiany w aplikacji hosta mogą dotyczyć wielu dziedzin, od zmian w graficznym interfejsie, do zmian w sposobie komunikacji z poszczególnymi widżetami. Niektóre zmiany nie mają wpływu na funkcjonowanie całości, jednak nie można przyjmować, że tak będzie zawsze w przypadku tworzenia jakiegokolwiek oprogramowania, a w szczególności oprogramowania polegającego na kodzie dostarczanym przez zewnętrznych dostawców w formie modułów.

Projekt aplikacji hosta powinien przewidywać możliwość zmian i dostarczać mechanizmów odpowiednio reagujących w przypadku, gdy jeden z widżetów został napisany dla starszej wersji hosta. Duża część aplikacji jednak nie implementuje żadnej zaawansowane obsługi różnych wersji. Częstym zachowaniem jest informowanie użytkownika o możliwej niekompatybilności jednej z wtyczek.

Możliwym sposobem na implementację zaawansowanej obsługi scenariusza różnic wersji jest stworzenie adapterów po stronie hosta, pozwalających na obsługę zmienionych funkcjonalności lub ich braku w widżetach o wcześniejszej wersji. Takie rozwiązanie może być transparentne dla użytkownika. Widżety nie obsługujące pewnych funkcjonalności mogą mieć wyłączoną opcję ich uruchomienia. Taką sytuację przedstawia Przykład 3.

Przykład 3. Wersjonowanie widżetów i aplikacji hosta

Za przykład może nam posłużyć widżet A stworzony dla Hosta w wersji 1.0. Wersja Hosta na komputerze użytkownika to 1.1. Wersja 1.1 wprowadziła możliwość zmiany wielkości widżetów. Widżet A nie potrafi zareagować na żądanie zmiany wielkości przesłane przez Host. Host posiada jednak adapter pozwalający na wyłączenie tej opcji dla tego widżetu. Dodatkowo Host w wersji 1.1 poszerzył możliwość obracania widżetu z obracania tylko w przestrzeni dwuwymiarowej w wersji 1.0, do obracania w przestrzeni trójwymiarowej. Widżet A nie potrafi zachować się odpowiednio w przypadku, gdy żądanie obrotu będzie posiadało trzy współrzędne, a nie dwie. Adapter po stronie Hosta może jednak przeprowadzić rzutowanie trójwymiarowych koordynatów na dwuwymiarowe dla tej wtyczki.

Kolejnym możliwym problemem może być posiadanie przez użytkownika nieaktualnej wersji aplikacji hosta. W tym przypadku może on starać się korzystać z widżetów przeznaczonych dla wersji nowszej, niż posiadana. Trudno przewidzieć jakie zmiany będą wprowadzone w architekturze widżetów w nowszych wersjach systemu. Z tego powodu system powinien uniemożliwić użytkownikowi uruchomienie widżetów stworzonych dla nowszych wersji aplikacji hosta.

4.1.5 Konfiguracja

Aplikacja hosta musi przechowywać informacje na temat uruchomionych widżetów: ich położenia czy przyznanych uprawnień, aby przy każdym uruchomieniu aplikacji możliwe było przywrócenie jej stanu z ostatniego uruchomienia.

Ponadto host musi dostarczać możliwości zapisywania ustawień przez widżety nawet o najniższym zbiorze uprawnień. Pliki konfiguracyjne, przechowujące informacje na temat stanu aplikacji dotyczą zarówno aplikacji hosta, jak i pojedynczych widżetów. W obu przypadkach dostęp do nich powinien być nadany i kontrolowany przez aplikację hosta.

Poszczególne systemy widżetów przyjmują różne koncepcje w celu rozwiązania problemu konfiguracji. Popularnym sposobem jest domyślne zezwolenie każdemu widżetowi na dostęp do plików w katalogu, w którym widżet się znajduje. Wadą tego rozwiązania jest brak wpływu na wielkość plików wygenerowanych przez widżet. Nieprawidłowo działająca wtyczka mogłaby zająć zbyt duży obszar pamięci dyskowej. Taką strategię przyjęło wiele systemów widżetów, między innymi Apple Dashboard.

Ciekawe rozwiązanie proponuje Yahoo! Widget Engine. Począwszy od wersji 4.0 Widget Engine wprowadza ograniczoną obsługę SQLite - prostego systemu bazodanowego, jako alternatywnego miejsca przechowywania plików tekstowych i XML. Udostępnione funkcjonalności są podzbiorem tych z pełnej wersji SQLite, jednak dostarczają autorom widżetów wystarczającą funkcjonalność do większości zadań.

Autor pracy do rozwiązania problemu konfiguracji w opracowanym prototypie użył Isolated Storage przypisanego każdemu widżetowi. W systemie Windows, dzięki platformie .NET, możliwe było użycie Isolated Storage, czyli ograniczonej, izolowanej przestrzeni na dysku stworzonej dla danej aplikacji i użytkownika. Przestrzeń ta jest ograniczona do 20MB i w jej ramach widżet może tworzyć pliki bez obawy, że inne widżety uzyskają do nich niepożądany dostęp.

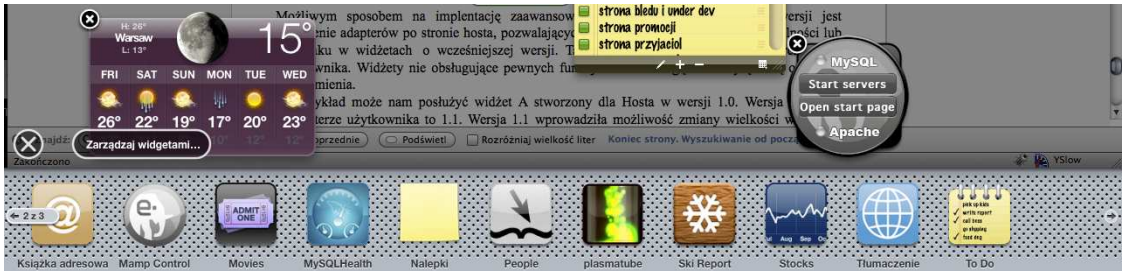
4.1.6 GUI Aplikacji hosta

Graficzny Interfejs Użytkownika jest niezwykle ważnym elementem systemów widżetów. Powinien on być interesujący, estetyczny oraz intuicyjny. W przeciwieństwie do aplikacji typowo użytkowych takich jak edytory tekstu, arkusze kalkulacyjne, gdzie wybór jest dyktowany ceną oraz zakresem funkcjonalności oferowanych przez oprogramowanie, potencjalni użytkownicy widżetów sugerują się głównie jakością interfejsu graficznego. Jest to spowodowane podobnym zakresem funkcjonalności oferowanym przez każdy z dostępnych na rynku produktów. Systemy widżetów są aplikacjami darmowymi, dopiero niektóre widżety, tworzone przez zewnętrzne firmy, posiadające unikalną funkcjonalność mogą być dostępne za opłatą. Oczywiste jest, że odpowiedzialność za tworzenie wtyczek o wysokiej jakości interfejsu graficznego jest zadaniem zewnętrznych dostawców, jednak same aplikacje hostujące w wielu przypadkach także zachęcają do korzystania z nich swoim wyglądem. Jedną z aplikacji przodujących w zaawansowanym interfejsie graficznym aplikacji hosta jest Apple Dashboard. Domyślnie aplikacja ta jest niewidoczna dla użytkownika, dopiero wciśnięcie wcześniej zdefiniowanej kombinacji klawiszy, lub użycie opcji systemu Mac OS X o nazwie aktywne narożniki ekranu, pokazuje użytkownikowi aplikację Hosta. Wraz z widżetami Interfejs programu zajmuje całą powierzchnię ekranu. Obszar, na którym znajdują się widżety jest półprzezroczysty (patrz Rysunek 9), więc widoczne są aplikacje znajdujące się pod nim .

Rysunek 9. Półprzezroczyste tło Apple Washboard

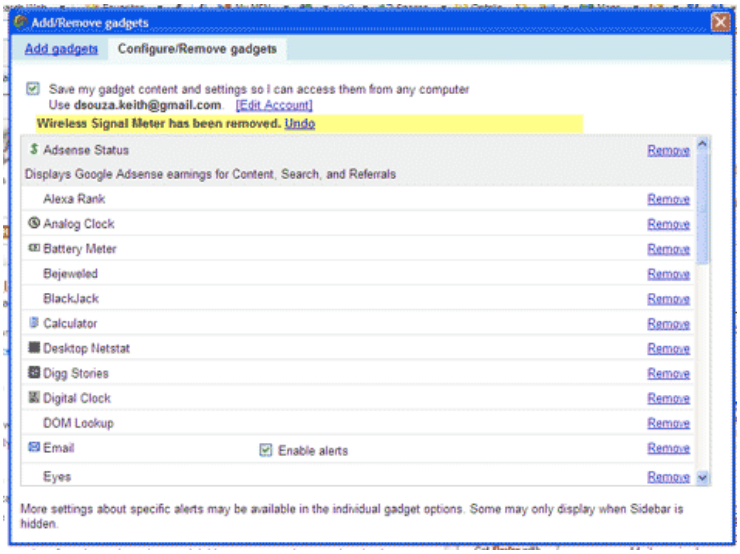
Źródło: Opracowanie własne

Warto zwrócić uwagę na listę widżetów dostępną po kliknięciu ikony plusa w lewym dolnym rogu ekranu. Nie jest to standardowa kontrolka listy oferowana przez system Mac OS X. Lista składa się z horyzontalnie ułożonych obok siebie ikon. Ikony te znajdują się na nieprzezroczystym tle (patrz Rysunek 10). Są one definiowane i dostarczane przez same widżety. Ikona powinna sugerować użytkownikowi funkcjonalność dostarczaną przez widżet. Rysunki 11 i 12 przedstawiają alternatywne sposoby prezentacji listy widżetów w Google Desktop i Microsoft Gadgets.



Rysunek 10. Lista widżetów w Apple Washboard

Źródło: Opracowanie własne



Rysunek 11. Lista widżetów w Google Desktop

Źródło: Opracowanie własne

Rysunek 12. Lista widżetów (Gadgets Manager) w Microsoft Gadgets

Źródło: Opracowanie własne

W przeciwieństwie do rozwiązania Interfejsu Graficznego przez Apple, wielu producentów zdecydowało się na umieszczenie widżetów w jasno zarysowanym wizualnie kontenerze. Aplikacje takie jak Microsoft Gadgets (patrz Rysunek 14), Yahoo! Widget Engine, Google Gadgets (Rysunek 13) dają możliwość umieszczenia widżetów w Panelu Bocznym (Sidebar), zajmującym tylko część ekranu użytkownika. Jego szerokość może być regulowana, a wtyczki w panelu układają się w formie

stosu, jeden na drugim. Nie można jednoznacznie stwierdzić wyższości jednego układu widżetów nad drugim - wybór zależy od preferencji użytkownika.

Rysunek 13. Widżety Google Gadgets w pasku bocznym oraz poza nim

Źródło: Opracowanie własne

Rysunek 14. Pasek boczny Microsoft Gadgets

Źródło: Opracowanie własne

Graficzny Interfejs aplikacji hosta sugeruje użytkownikowi rzeczywistą architekturę aplikacji. Wizualizuje jego pierwotną funkcję bycia kontenerem dla poszczególnych wtyczek. Brak Interfejsu,

jak to widzimy na Rysunku 15, może sugerować użytkownikowi, iż każdy pojedynczy widżet jest niezależną od żadnych innych aplikacją.

Rysunek 15. Widżety Yahoo! Widget Engine bez widocznego tła aplikacji hosta

Źródło: Opracowanie własne

Aplikacje hosta zajmujące część ekranu powinny pozwalać na określenie miejsca dokowania panelu bocznego. Dokowanie odbywa się zwykle poprzez przeciągnięcie powierzchni panelu bocznego w stronę danej krawędzi ekranu lub poprzez wybranie odpowiedniej opcji z okna ustawień.

Charakterystyczną cechą wszystkich systemów jest możliwość zmiany położenia poszczególnych widżetów. W przypadku panelu bocznego zmieniana jest ich kolejność w stosie. W przypadku luźnego rozmieszczenia możliwa jest zmiana ich współrzędnych położenia na ekranie.

Projekt prototypu powinien brać pod uwagę ergonomię, wygodę korzystania oraz estetykę interfejsu graficznego. Swoim interfejsem powinien zachęcać użytkownika, do korzystania z niego.

4.2. Widżet

Aplikacja hosta w każdym systemie widżetów ma olbrzymie znaczenie, jednak to poszczególne widżety dostarczają jej funkcjonalności interesujących użytkowników. Żaden system widżetów nie może istnieć bez rozbudowanej, na bieżąco aktualizowanej bazy wtyczek dla niego stworzonych.

Niezależnie od funkcji stworzonej przez autora wtyczki, podstawa, na jakiej są one budowane, musi implementować szereg wspólnych funkcjonalności.

4.2.1 Izolacja i Bezpieczeństwo

Pomimo faktu, iż to aplikacja hosta powinna być odpowiedzialna za izolację, sandboxing wtyczek, same wtyczki powinny prawidłowo obsługiwać ograniczony dostęp do zasobów. Wtyczka wymagająca większych uprawnień niż zostały jej przyznane powinna implementować zachowanie w pełni obsługujące taki scenariusz. Spełnienie tego warunku zwiększy stabilność i jest dobrą praktyką w systemach tego typu.

4.2.2 Instalacja, Wersjonowanie, Aktualizacja i Konfiguracja

Prosta instalacja każdego programu jest jednym z kluczowych czynników składających się na jego sukces. To samo odnosi się do mini-aplikacji, jakimi są poszczególne wtyczki. Ich instalacja powinna być prosta, intuicyjna, przyjazna użytkownikowi. Sposobem instalacji obsługiwanym przez większość aplikacji tego typu jest ręczne skopiowanie plików widżetów do katalogu, w którym są trzymane katalogi z plikami widżetów.

Drugim możliwym sposobem instalacji wtyczek, dostępnym np. w Microsoft Gadgets jest skojarzenie danego rozszerzenia pliku z aplikacją. Ten scenariusz przewiduje przygotowanie przez autora wtyczki archiwum w formacie ZIP lub CAB, zawierającego wszystkie jej pliki. Archiwum musi posiadać rozszerzenie .gadget. Uruchomienie takiego pliku i zgoda użytkownika na instalację rozpoczyna jej proces. Od tego momentu instalacja odbywa się bez jakiegokolwiek udziału użytkownika. Widżet po chwili jest dostępny do uruchomienia poprzez menedżera gadgetów (Gadget Manager).

Autor pracy zwraca uwagę na możliwość instalacji widżetów za pomocą wykonywalnych pakietów instalacyjnych (takich jak .exe, .msi). Taki sposób instalacji jest podobny do drugiego sposobu wspomnianego w tym rozdziale, z tą różnicą, że zamiast skompresowanego archiwum i obsługi instalacji przez aplikację hosta, mamy do czynienia z pakietami instalacyjnymi (w formatach exe lub msi), a za instalację widżetu jest odpowiedzialny on sam. Brak popularności tego rozwiązania w istniejących aplikacjach jest spowodowany koniecznością uproszczenia procesu przygotowania pakietu instalacyjnego przez autora wtyczki. Proces tworzenia wykonywalnego pakietu instalacyjnego wymaga wiedzy w tym zakresie od zespołu tworzącego wtyczkę oraz zwiększa koszty i czas tworzenia widżetów. Sprawia to, że ten typowy dla większości aplikacji, z którymi użytkownik ma na co dzień do czynienia sposób instalacji jest nie do przyjęcia w aplikacjach tego typu.

Każdy widżet powinien mieć możliwość wersjonowania. Zarówno aplikacja hosta, jak i widżet, powinny mieć możliwość określenia swojej wersji, zapewnioną przez system widżetów. W większości

istniejących systemów informacja na temat aktualnych wersji wtyczek jest przechowywana w plikach konfiguracyjnych. Wersjonowanie jest kluczowym kryterium niezbędnym przy określaniu konieczności aktualizacji do nowszej wersji.

W większości systemów widżetów to aplikacja hosta jest odpowiedzialna za zarządzanie aktualizacjami widżetów. Jest to możliwe po określeniu odpowiedniego protokołu ich aktualizacji. Taki scenariusz został zaprezentowany w Przykładzie 4.

Przykład 4. Aktualizacja widżetów przez aplikację hosta

System Widżetów A posiada zainstalowane widżety: W1, W2, W3. Przy każdym uruchomieniu, gdy połączenie z internetem jest dostępne, łączy się z usługą webową podając listę zainstalowanych wtyczek wraz z ich wersjami. Usługa w odpowiedzi zwraca listę widżetów, których aktualizacje są dostępne. Widżet następnie pyta użytkownika o zgodę na aktualizację. W przypadku odpowiedzi pozytywnej pobiera pliki instalacyjne widżetów i zastępuje nimi pliki starszej wersji.

Widżety nie znajdujące się w takiej centralnej bazie muszą być aktualizowane ręcznie przez użytkownika. Baza widżetów jest aktualizowana przez autorów poszczególnych wtyczek. Systemy takie jak Google Desktop, Yahoo! Widget Engine, Apple Dashboard oferują scentralizowane bazy widżetów.

Innym możliwym rozwiązaniem jest dostarczenie przez każdy widżet adresu URL prowadzącego do usługi pozwalającej na jego aktualizację. Protokół komunikacji z taką usługą musi być zdefiniowany przez autora systemu widżetów i dokładnie udokumentowany, aby twórcy wtyczek mogli ją zaimplementować. W takim przypadku cały ciężar obsługi aktualizacji rozkładany jest na wszystkich dostawców.

Ciekawym przypadkiem, w którym problem aktualizacji nie występuje, są widżety webowe. Dzieje się tak ponieważ poszczególne widżety są zawsze pobierane z serwera przez przeglądarkę internetową przy każdym odświeżeniu strony. Użytkownik zawsze posiada zainstalowaną ich najnowszą wersję. Problem niespójności wersji jest rozwiązywany przez autora wtyczki, bezpośrednio na serwerze. System ten jest najbardziej niezawodny. Pomimo, że rozwiązanie to jest najlepsze, dotyczy ono tylko aplikacji internetowych, a do jego sprawnej implementacji konieczny jest ciągły dostęp do internetu ze strony użytkownika.

4.2.3 Komunikacja

Niezwykle ważną cechą systemów widżetów jest ich generyczność i skalowalność. Cechy te znajdują swoje odzwierciedlenie w możliwości tworzenia nowych funkcjonalności oraz łatwym rozszerzaniu już istniejących. Kluczowa w tych przypadkach jest możliwość komunikacji pomiędzy widżetami, a aplikacją hosta. Protokół komunikacji pomiędzy aplikacją hosta, a widżetem powinien

być jasno określony, udokumentowany i udostępniony autorom widżetów. Autor pracy w opracowanym prototypie zaproponował wprowadzenie możliwości komunikacji pomiędzy poszczególnymi wtyczkami. W zaproponowanym przez autora modelu, to autor widżetu określa protokół komunikacji. Komunikacja ta przewiduje możliwość określenia typu wiadomości oraz przesyłania popularnych typów danych.

4.2.4 GUI widżetu

Jak już zostało zauważone rozdziale 4.1.6 Graficzny Interfejs Użytkownika odgrywa olbrzymią rolę w aplikacjach będących tematem niniejszej pracy. Jednak odpowiedzialność za graficzną atrakcyjność całości aplikacji nie leży po stronie twórcy systemu widżetów, lecz po stronie autorów wtyczek. Ta zależność zmusza autorów systemów do dostarczenia twórcom widżetów odpowiedniego API, pozwalającego na stworzenie GUI wtyczki. API to powinno w jak najmniejszy sposób ograniczać swobodę w tworzeniu interfejsu graficznego, dostarczać wystarczającą liczbę popularnych kontrolek do prezentacji danych, do realizacji jak największej liczby popularnych funkcjonalności. W istniejących aplikacjach ich twórcy dają autorom widżetów olbrzymią swobodę w tworzeniu interfejsu użytkownika (patrz Rysunek 16, Rysunek 17, Rysunek 18).

Rysunek 16. Różnorodność interfejsu użytkownika widżetów w Microsoft Gadgets

Źródło: Opracowanie własne

Rysunek 17. Różnorodność interfejsu użytkownika widżetów w Google Desktop

Źródło: Opracowanie własne

Rysunek 18. Różnorodność interfejsu użytkownika widżetów w Yahoo! Widget Engine

Źródło: Opracowanie własne

Swoboda jednak nie zawsze wiąże się z prostotą jej implementacji. W wielu przypadkach konieczna jest nauka nie tylko API, ale i specyficznego języka programowania przez programistów wtyczek, co zwiększa koszty i czas potrzebny na stworzenie pojedynczej wtyczki. Może się to przełożyć na małą ilość widżetów tworzonych przez zewnętrznych dostawców, co z kolei może stać się bezpośrednią przyczyną niezrealizowania jej biznesowych celów. Yahoo! Widget Engine dostarcza elastycznego API opartego na JavaScript'cie zawierającego wiele przydatnych funkcji dostępnych dla autorów wtyczek. Podstawowe elementy to pole tekstowe, obraz oraz timer. Inne dostępne funkcje to możliwość ściągania stron internetowych z internetu, zapis/odczyt z plików oraz obiekt animatora, pozwalający na animację poszczególnych elementów. Wbudowany interfejs pozwala na uruchamianie kodu specyficznego dla danej platformy, takiego jak obiekty COM w Microsoft Windows, czy też AppleScript w Mac OS X. Od wersji 4.0 dostępny jest obiekt Canvas pozwalający na programistyczne rysowanie oraz zapisywanie obrazu do plików formatu PNG lub JPG.

Oprócz prezentacji danych API powinno pozwalać na złożoną interakcję z użytkownikiem - powinno obsługiwać drag & drop, reagować na kliknięcia myszy oraz dawać możliwość wprowadzania danych za pomocą klawiatury.

5. Prototyp

Opracowany prototyp jest w pełni funkcjonalnym systemem zarządzania widżetami. W aplikacji tej Autor pracy położył duży nacisk na jej stabilność, łatwość tworzenia nowych widżetów oraz atrakcyjny interfejs użytkownika.

5.1. Architektura Host – Pipeline – Widżet

Architektura prototypu jest z góry narzucona przez Managed Addin Framework. Składa się z Aplikacji Hosta, Kanału komunikacji (Pipeline) oraz Widżetów.

5.2. PipeLine

Pipeline to inaczej kanał komunikacji pomiędzy widżetami, a aplikacją hosta. Ogólna struktura budowy jest analogiczna do tej przedstawionej w Rozdziale 3.3. W trakcie projektowania prototypu okazało się jednak, że konieczne będzie rozszerzenie proponowanego modelu o abstrakcyjną klasę bazową dla wszystkich widżetów, znajdującą się poza segmentem widoków add-in'a. Pozwala to na zdefiniowanie pewnej wspólnej struktury widżetów, określenie wersji oraz narzucenie pewnych zachowań. Klasa ta została nazwana WidgetBase. Dokładniejszy opis sposobu jej implementacji znajduje się w Rozdziale 5.4.

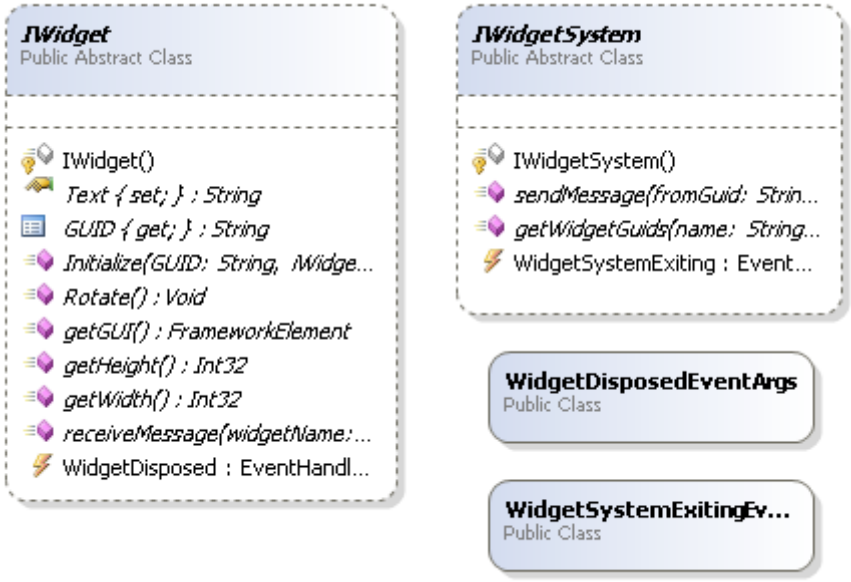
5.2.1 Widoki po stronie hosta

Segment widoków po stronie hosta zawiera abstrakcyjne klasy będące dla aplikacji hosta widokami na typy danych przesyłanych za pomocą kanału komunikacji. Klasy te opisują metody dostępne w tych typach danych.

W opracowanym prototypie Autor pracy umieścił cztery takie widoki:

- IWidget – widok widżetu z perspektywy Aplikacji Hosta;
- IWidgetSystem – widok Aplikacji Hosta na siebie samą;
- WidgetDisposedEventArgs – argumenty zdarzenia informującego aplikację hosta o zamknięciu widżetu;
- WidgetSystemExitingEventArgs – argumenty zdarzenia informującego widżety o zamknięciu aplikacji hosta.

Diagram klas przedstawiający te klasy przedstawiono na Rysunku 19. Widzimy na nim, że aplikacja hosta ma dostęp do metod widżetu, takich jak te odpowiedzialne za określenie jego wymiarów (getHeight(), getWidth()), inicjalizację (Initialize(..)), czy też komunikację (receiveMessage(..)).



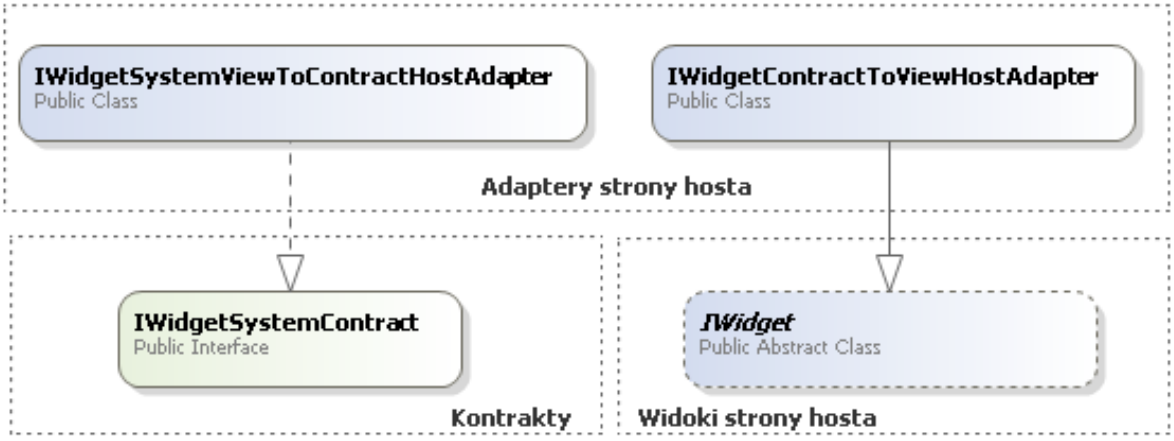
Rysunek 19. Diagram klas, abstrakcyjnych widoków po stronie hosta

Źródło: Opracowanie własne

Widoki te znajdują się w projekcie o nazwie HostView. Skompilowana biblioteka umieszczana jest w katalogu głównym aplikacji.

5.2.2 Adaptery po stronie hosta

Adaptery po stronie hosta to biblioteka zawierająca klasy odpowiedzialne za kontrolowanie komunikacji pomiędzy segmentem zawierającym kontrakty, a segmentem zawierającym widoki. Rolą tych adapterów jest konwersja widoków do kontraktów oraz kontraktów do widoków po stronie aplikacji hosta oraz ich wzajemna separacja. Rysunek 20 przedstawia uproszczony diagram klas oraz zależności pomiędzy adapterami strony hosta, a innymi segmentami.



Rysunek 20. Uproszczony diagram klas oraz zależności pomiędzy adapterami strony hosta, a innymi segmentami

Źródło: Opracowanie własne

IwidgetSystemViewToContract implementuje interfejs IwidgetSystemContract. Konwersja odbywa się w czasie tworzenia obiektu tej klasy. Do konstruktora tej klasy przekazywany jest widok, który następnie przechowywany jest jako referencja. Konwersja ma miejsce w momencie wywoływania na obiekcie tej klasy jego metody. Przykład 5 przedstawia jedną z metod służących do komunikacji na linii widżet-aplikacja hosta-widżet.

Przykład 5. Metoda służąca do komunikacji na linii widżet-aplikacja hosta-widżet

```
public bool sendMessage(string fromGuid,string toGuid, string messageName,
IListContract<double> doublesParams, IListContract<string> stringsParams)
{
            return _view.sendMessage(fromGuid,toGuid, messageName,
            CollectionAdapters.ToIList<double>(doublesParams),
            CollectionAdapters.ToIList<string>(stringsParams));

}
```

Metoda ta przyjmuje jako argumenty identyfikatory nadawcy i odbiorcy, nazwę wiadomości oraz dwie kolekcje z możliwymi do przekazania parametrami wiadomości. Typ IListContract jest typem dostarczonym przez Managed Add-In Framework i spełnia warunki nałożone na kontrakty, o których mowa w Rozdziale 3.3. Metoda ta jest pośrednikiem w komunikacji pomiędzy kontraktem a widokiem i tłumaczy zapytanie zrozumiałe dla kontraktu, na zrozumiałe dla widoku (stąd w nazwie ContractToView). Kolekcja IListContract jest niezrozumiała dla widoku, który przyjmuje jako argument typ IList. Do zamiany typów w tym przypadku Autor korzysta z klasy pozwalającej na konwersję kolekcji o nazwie CollectionAdapters. Jako, że obiekt bool zwracany przez metodę widoku jest typem prostym i spełnia warunki nałożone na kontrakty, nie jest konieczne przeprowadzanie żadnej konwersji i zostaje on zwrócony przez metodę adaptera. Ten sposób działania adapterów jest

wspólny dla obu stron kanału komunikacji. W przypadku adapterów „ContractToView”: dziedziczą one odpowiedni widok, przyjmują w konstruktorze dany kontrakt i dokonują tłumaczenia z zapytania zrozumiałego dla widoku na zapytanie zrozumiałe dla kontraktu. Takie zapytanie może przekroczyć strefę izolacji (patrz wymagania kontraktów - Rozdział 3.3).

W opracowanym prototypie adaptery te znajdują się w projekcie HostSideAdapters i są kompilowane do katalogu HostSideAdapters znajdującego się w katalogu głównym aplikacji.

5.2.3 Kontrakty

W projekcie o nazwie AppContracts znajdują się interfejsy odpowiedzialne za definiowanie protokołu komunikacji i wymiany danych poprzez strefę izolacji. Interfejsy te nazywamy kontraktami. Opracowany prototyp posiada dwa główne kontrakty pozwalające na przekazywanie referencji do obiektu aplikacji hosta (IWidgetSystemContract) oraz do widżetu (IWidgetContract). Na Rysunku 21 przedstawiono metody zadeklarowane w tych interfejsach.

Rysunek 21. Klasy i ich metody w segmencie kontraktów opracowanego prototypu

Źródło: Opracowanie własne

Kontrakty są wspólne dla obu stron kanału komunikacji i są używane do przekazywania danych przez strefę izolacji. W opracowanym prototypie strefa izolacji jest na poziomie procesów systemowych, czyli każdy widżet i aplikacja to osobny proces. Taka architektura była dyktowana założeniami przyjętymi przez Autora pracy, mówiącymi, iż niestabilna praca widżetu nie może wpływać na stabilność pracy innych widżetów, ani aplikacji hosta.

IWidgetSystemContract posiada następujące metody:

- sendMessage(..) – jako argumenty przyjmuje identyfikatory odbiorcy i nadawcy, nazwę typu wiadomości oraz dwie kolekcje double oraz string. Metoda ta służy do przesyłania wiadomości pomiędzy widżetami;

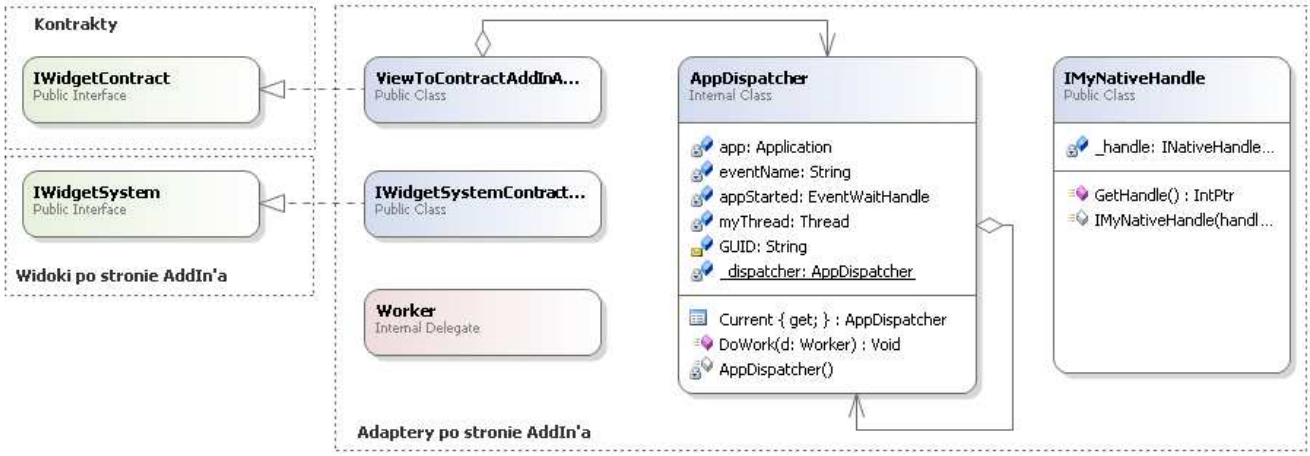
- getWidgetGuids(..) – jako argument przyjmuje nazwę widżetu. Metoda ta zwraca kolekcję identyfikatorów wszystkich uruchomionych instancji widżetu o danej nazwie. W ten sposób widżet może uzyskać informacje o innych uruchomionych widżetach w systemie.

Ważniejsze metody IwidgetContract:

- Initialize – pierwsza metoda wywoływana na obiekcie widżetu przez aplikację hosta od razu po jego aktywacji. Aplikacja hosta, z jej pomocą, przekazuje widżetowi jego unikalny identyfikator (GUID) oraz referencję do siebie samej. Widżet w odpowiedzi zwraca referencję do swojego Interfejsu Graficznego (GUI);
- Rotate – obraca widżet pokazując jego drugą stronę. GUI widżetów zostało rozszerzone o możliwość obrotu w przestrzeni 3D i możliwość zdefiniowania kontrolek znajdujących się z przodu i z tyłu widżetu;
- receiveMessage(..) - metoda jako argumenty przyjmuje GUID, od którego pochodzi wiadomość, typ wiadomości oraz dwie kolekcje typów double i string. Stwarza ona możliwość odbioru wiadomości przez widżet.

5.2.4 Adaptery po stronie widżetu

Adaptery po stronie widżetu pełnią analogiczną rolę jak te po stronie aplikacji hosta. Z powodu architektury wątków aplikacji WPF, gdzie wszystkie elementy GUI są zarządzane w jednym wątku, konieczne było stworzenie klasy AppDispatcher. Widżety, jako że są uruchamiane w osobnych procesach, posiadają własne wątki zarządzające Graficznym Interfejsem. W przypadku próby odwołania do elementu GUI z wątku innego niż ten, który go stworzył - CLR wyrzuca krytyczny wyjątek informujący, że taka operacja jest niedozwolona. Autor pracy spotkał się z tym problemem w momencie próby odwołania się do Interfejsu Graficznego widżetu z aplikacji hosta. Klasa AppDispatcher (patrz Rysunek 22) implementuje wzorzec singleton’a. Zawiera w sobie obiekt Application reprezentujący aplikację WPF. Z użyciem tego obiektu możliwe jest odwołanie się do obiektu Dispatcher i poprzez metodę Invoke synchroniczne wywołanie delegata Worker. Delegat Worker, w przypadku odwołania z aplikacji hosta, jest swoistą referencją do metody, która operuje na GUI widżetu. W ten sposób operacja na Interfejsie widżetu jest wywoływana w jego wątku głównym, a nie w wątku Aplikacji Hosta. Jako że obiekt klasy AppDisaptcher jest unikalny dla każdego uruchomionego widżetu, zawiera on także jego GUID.



Rysunek 22. Klasy segmentu zawierającego adaptery po stronie widżetu wraz z ich powiązaniami do innych segmentów

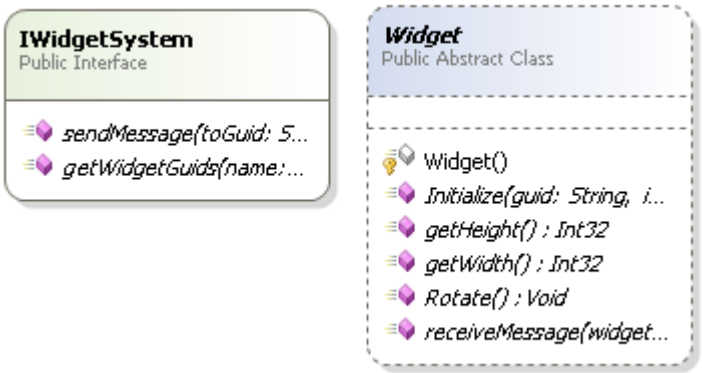
Źródło: Opracowanie własne

Adaptery po stronie widżetu znajdują się w projekcie AddInSideAdapters i są kompilowane do biblioteki umieszczanej w katalogu o tej samej nazwie, w katalogu głównym aplikacji.

5.2.5 Widoki po stronie widżetu

Funkcja widoków po stronie widżetu, w budowie kanału komunikacji jest analogiczna do tych po stronie aplikacji hosta. Są one widokami widżetu na obiekty biorące udział w komunikacji poprzez strefę izolacji.

Na Rysunku 23 przedstawiony został uproszczony diagram klas przedstawiający klasy znajdujące się w tym segmencie w opracowanym prototypie.



Rysunek 23. Klasy widoków po stronie widżetu

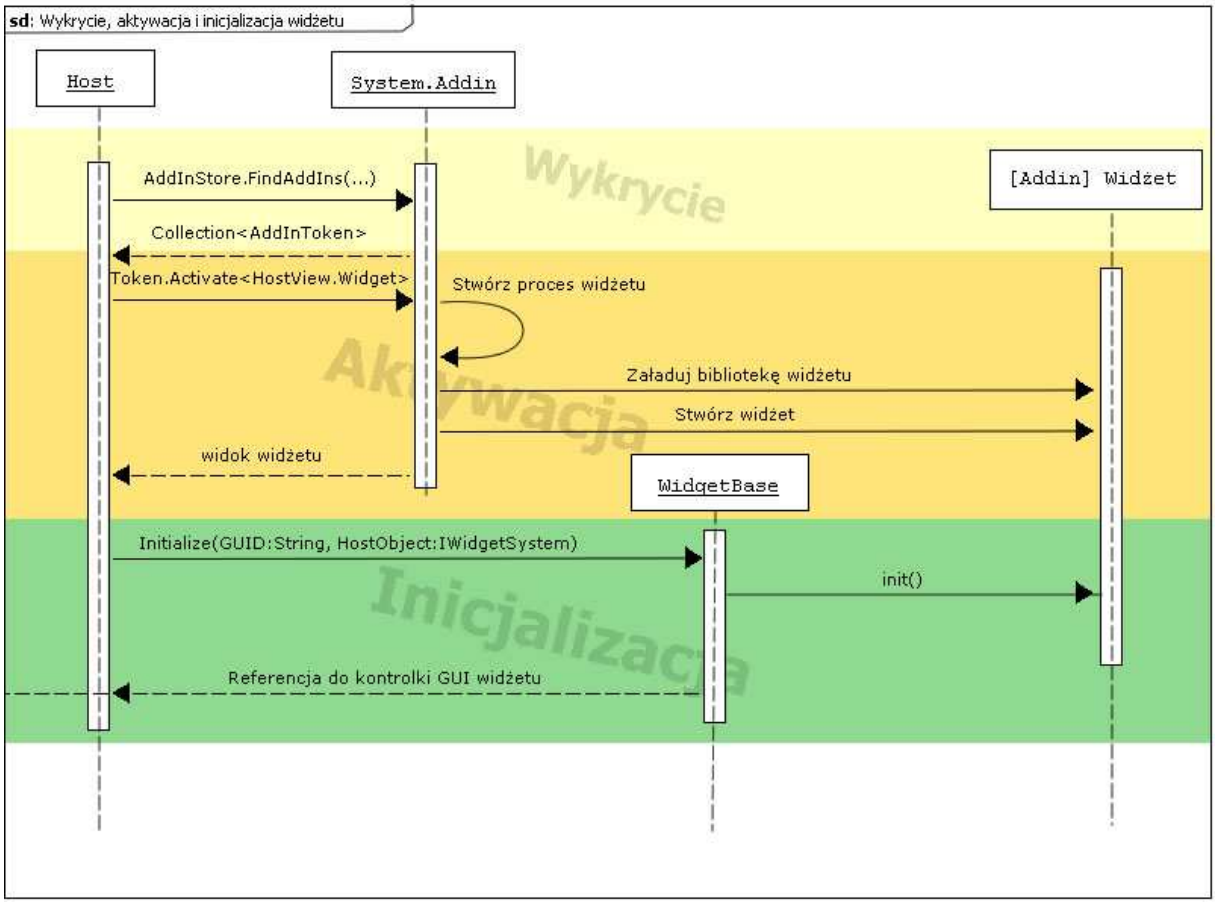
Źródło: Opracowanie własne

5.3. Host

Aplikacja hosta (w skrócie „Host”) pełni rolę zarządzającą w każdym popularnym systemie widżetów. Tak też jest w opracowanym prototypie. Jej zadaniem jest poprawne wykrycie i aktywowanie widżetu. Jest ponadto odpowiedzialna za poprawną, niezakłóconą komunikację pomiędzy wtyczkami i ich poprawne zamknięcie. Ważną cechą tej aplikacji jest umiejętność zapamiętania swojego stanu z ostatniego uruchomienia i przy ponownym uruchomieniu przywrócenie go. Stan aplikacji hosta to nie tylko jej własne cechy jak np. położenie, ale też lista uruchomionych widżetów i ich położenia.

5.3.1 Wykrywanie, aktywacja i inicjalizacja widżetów

Zanim widżet będzie gotowy do interakcji z użytkownikiem aplikacja hosta musi go wykryć, aktywować, a następnie inicjalizować. Na Rysunku 24 przedstawiono w formie diagramu sekwencji uproszczony przebieg tych trzech procesów.



Rysunek 24. Wykrycie, aktywacja i inicjalizacja widżetu

Źródło: Opracowanie własne

Prototyp w celu wykrycia i aktywacji widżetu używa wbudowanych narzędzi MAF. Zainstalowane widżety to te, których pliki o poprawnej strukturze znajdują się w katalogu Addins, w katalogu głównym aplikacji. Proces wykrycia zainstalowanych wtyczek odbywa się poprzez wywołanie metody FindAddins, która wyszukuje w domyślnym katalogu biblioteki zawierające klasę IWidget i zwraca obiekty typu AddInToken reprezentujące możliwe do uruchomienia widżety.

W przypadku, gdy konieczne jest uruchomienie instancji widżetu (poprzez ostatni stan zapisany w konfiguracji lub akcję użytkownika), aplikacja hosta wywołuje na obiekcie tokenu metodę Activate. W metodzie tej tworzy proces i odizolowane środowisko dla nowo powstającej instancji widżetu. W procesie tym załadowuje bibliotekę widżetu oraz wywołuje konstruktor widżetu, tym samym tworząc nowy obiekt. Następnie zwracana jest do aplikacji hosta referencja do obiektu widżetu.

Inicjalizacja odbywa się poprzez wywołanie na obiekcie widżetu metody Initialize znajdującej się w abstrakcyjnej klasie WidgetBase, którą dziedziczy każdy widżet. Na Rysunku 24 przedstawiono WidgetBase i obiekt widżetu jako dwa osobne byty, w celu podkreślenia faktu, że logika metody Initialize jest w pełni zdefiniowana w klasie WidgetBase i nie może być zmieniona przez autora widżetu. Następnie wywoływana metoda init() na obiekcie widżetu, musi zostać zaimplementowana przez twórcę wtyczki. Metoda ta pozwala autorowi widżetu na ustawienie jego stanu początkowego.

5.3.2 Zarządzanie widżetami

W opracowanym prototypie klasą bezpośrednio odpowiedzialną za zarządzanie widżetami jest WidgetsManager. Klasa jednocześnie implementuje widok IWidgetSystem. Tym samym jest główną klasą aplikacji hosta. Odpowiedzialna jest nie tylko za zarządzanie, ale i za komunikację z widżetami, w której sama uczestniczy. Każdy widżet posiada referencję do tej klasy, zapewnioną poprzez kanał komunikacji. WidgetBase przechowuje ekstensję widżetów oraz implementuje mechanizmy zarządzające ich cyklem życia.

5.3.3 Konfiguracja hosta

W dzisiejszych czasach każda nowoczesna aplikacja powinna mieć możliwość zapamiętania swoich ustawień pomiędzy poszczególnymi jej uruchomieniami. Autor pracy na potrzeby obsługi konfiguracji aplikacji hosta stworzył klasę AppConfigurationManager znajdującą się w przestrzeni nazw Meisner.Mgr.Widgetsystem.Configuration. Zajmuje się ona obsługą tworzenia, modyfikacji, odczytywania i usuwania zapisanych ustawień. Korzysta przy tym z wbudowanych w .NET bibliotek służących do obsługi konfiguracji aplikacji oraz z klas reprezentujących strukturę pliku konfiguracyjnego. Klasy te, wraz z opisem tego, co reprezentują zostały przedstawione poniżej:

- WidgetsListSection – kolekcja wykrytych tokenów widżetów;
- WidgetElement – token wykrytego widżetu;

- WidgetInstanceElement – uruchomiona instancja widżetu;
- WidgetInstanceList – kolekcja instancji widżetów.

Na Przykładzie 6 Autor zaprezentował plik konfiguracji posiadający trzy wykryte tokeny i cztery uruchomione instancje.

Przykład 6. Plik konfiguracji, posiadający trzy wykryte tokeny i cztery uruchomione instancje

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSections>
        <section name="WidgetsListSection"
type="Meisner.Mgr.WidgetSystem.Configuration.WidgetsListSection,
WidgetSystem, Version=1.0.3538.24857, Culture=neutral, PublicKeyToken=null"
/>
    </configSections>
    <WidgetsListSection lastAccess="false">
        <widgetSections>
            <clear />
            <add name="Clocker" assemblyPath="MyAddIn" version="1.0.0.1"
publisher="Robert Meisner" description="..." twoSided="false"
manyInstances="false" isRectangular="false" securityLevel="NoTrust">
                <instances>
                    <clear />
                    <add GUID="{8ea2ef20-86}" x="282" y="7" />
                </instances>
            </add>
            <add name="Grafer" assemblyPath="TestWidget" version="false"
                publisher="Robert Meisner" description="..."
                twoSided="true" manyInstances="false" isRectangular="true"
                securityLevel="FullTrust">
                <instances>
                    <clear />
                    <add GUID="{e67d798d-54 }" x="26" y="550" />
                    <add GUID="{78ef7a44- }" x="333" y="315" />
                </instances>
            </add>
            <add name="Monitor Systemu" assemblyPath="TestWidget2"
version="false" publisher="Robert Meisner" description="..."
twoSided="true" manyInstances="false"
isRectangular="true"securityLevel="FullTrust">
                <instances>
                    <clear />
                    <add GUID="{7d9f7f7e}" x="16" y="198" />
                </instances>
            </add>
        </widgetSections>
    </WidgetsListSection>
</configuration>
```

Widać w nim istnienie tokenów widżetów Grafer, Monitor Systemu oraz Clocker. Clocker posiada jedną uruchomioną instancję o GUID równym "{8ea2ef20-86}" i położeniu x="282" y="7". Istnieją jeszcze trzy instancje, z czego dwie należą do widżetu Grafer, a jedna do Monitor Systemu. Łatwo zauważyć, że dla poszczególnych tokenów są przechowywane ich charakterystyczne

informacje w postaci atrybutów: assemblyPath, version, publisher, description, twoSided, manyInstances, isRectangular="true" i securityLevel.

5.3.4 Graficzny Interfejs aplikacji hosta

Aplikacja hosta jest umieszczona bezpośrednio w oknie pulpitu systemu Windows. Oznacza to, że znajduje się ona bezpośrednio nad ikonami będącymi na pulpicie i zawsze pod oknami innych aplikacji. Taka implementacja daje użytkownikowi wrażenie, że system widżetów jest nierozłączną częścią pulpitu. Interfejs graficzny aplikacji hosta składa się z trzech wizualnie wyodrębnionych części, wymieniając od góry: z paska górnego (Rysunek 24), panelu zawierającego widżety oraz paska z listą widżetów(Rysunek 25).

Rysunek 25. Pasek górny aplikacji hosta

Źródło: Opracowanie własne

Pasek górny zawiera szereg przycisków pozwalających na kontrolę położenia, zmianę przezroczystości panelu z widżetami, oraz zamknięcie aplikacji.

Przyciski wymieniając od lewej:

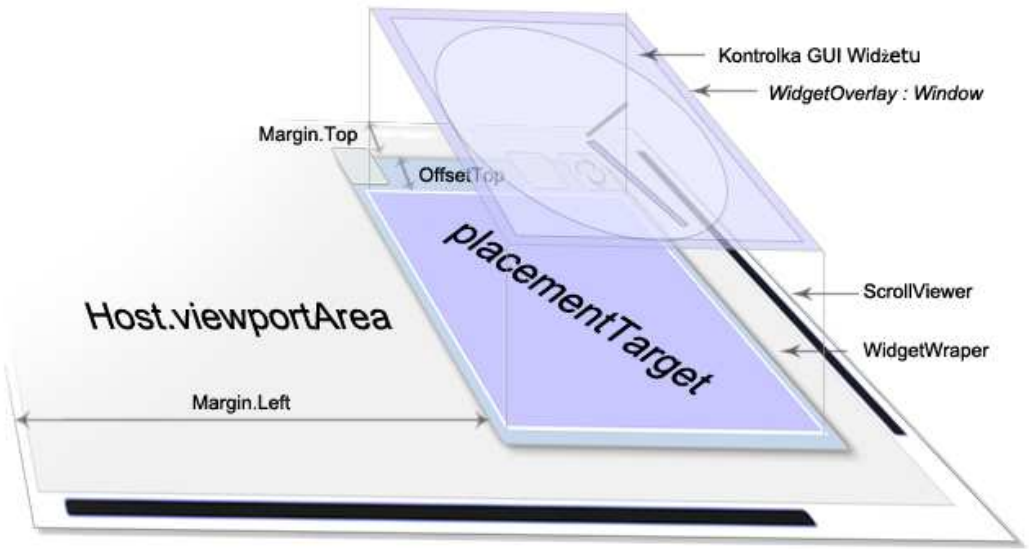
- Minimalizacja – pozwala na zminimalizowanie okna aplikacji hosta wraz ze wszystkimi widżetami. Po minimalizacji przy pasku notyfikacji wyświetlany jest dymek informujący o możliwości kliknięcia w ikonę notyfikacji i przywrócenie poprzedniego stanu aplikacji hosta;
- Szerokość – przycisk suwaka pozwala kontrolować szerokość aplikacji na ekranie. Wartości szerokości są procentowe ze skokiem co dziesięć procent. Minimalna wartość to 30% maksymalna 100%. Z prawej strony suwaka znajduje się przycisk „Ok”, służy on do akceptacji ustawionej szerokości i przeładowania okna;
- Przezroczystość – daje możliwość zmiany przezroczystości tła panelu zawierającego widżety. Możliwe są dwie wartości: brak przezroczystości oraz przezroczystość 50%. Przezroczystość pozwala na wyświetlenie ikon i okien znajdujących się pod aplikacją hosta;
- Horyzontalne dokowanie – kliknięcie w przycisk powoduje zmianę lokalizacji dokowania aplikacji hosta na przeciwległą horyzontalną krawędź ekranu. W przypadku aplikacji hosta zadokowanej do prawej krawędzi ekranu, aplikacja zmieni miejsce dokowania na lewą. Prototyp obsługuje wyłącznie dokowanie do krawędzi lewej i prawej;
- Zamknij – zamyka aplikację hosta wraz ze wszystkimi widżetami.

Rysunek 26. Pasek dolny zawierający listę widżetów

Źródło: Opracowanie własne

Pasek dolny aplikacji hosta zawiera graficzną, interaktywną listę dostępnych widżetów. Użytkownik poprzez podwójne kliknięcie myszy może stworzyć nową instancję wtyczki. Autor pracy dla uatrakcyjnienia pracy z systemem zaimplementował interaktywny efekt FishEye, uaktywniający się po najechaniu kursorem myszy na ikony listy widżetów. Ikony są definiowane przez autora widżetu poprzez dołączenie do plików widżetu pliku w formacie PNG o nazwie „icon.png".

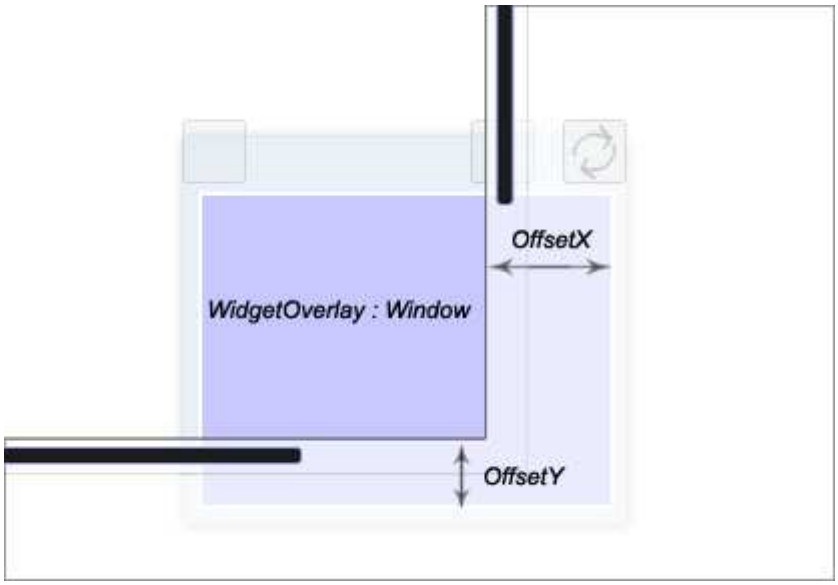
Panel zawierający widżety jest umieszczony w kontenerze ScrollViewer, dostarczającym mechanizmu pasków przewijania dla treści wykraczających poza jego ramy. Obiekt viewportArea określa powierzchnię, w której widżet powinien być widoczny dla użytkownika. Klasą przechowującą wtyczkę jest klasa WidgetWraper. Obiekty WidgetWraper posiadają swoją graficzną reprezentację. Składa się ona z trzech przycisków, ramki widżetu oraz powierzchni placementTarget, w ramach koordynatów której umieszczany jest Interfejs Graficzny przesyłany przez widżet. Interfejs ten umieszczany jest w oknie WidgetOverlay znajdującym się nad powierzchnią placementTarget i posiadającym jej wymiary (patrz Rysunek 27).



Rysunek 27. Budowa interfejsu wyświetlania widżetów

Źródło: Opracowanie własne

Poprzez system zdarzeń okno to monitoruje położenie i wymiary placementTarget i dostosowuje swoje w przypadku ich zmiany. Taka architektura została wymuszona przez istnienie problemu „Air Space” (opisany w rozdziale 6.1). Rozmiar okna WidgetOverlay nie jest definiowany tylko i wyłącznie poprzez wymiary placementTarget, ale i położenie krawędzi viewportArea. Jak widać na Rysunku 28 w przypadku przemieszczania się okna poza te krawędzie konieczne jest jego przycięcie o marginesy OffsetX i/lub OffsetY.



Rysunek 28. Widżet przekraczający ramy swojego kontenera

Źródło: Opracowanie własne

WidgetWraper w swojej reprezentacji graficznej posiada trzy przyciski:

- Przesuń (Rysunek 29) – poprzez przeciągnięcie kursorem myszki pozwala na zmianę pozycji wtyczki w ramach panelu zawierającego widżety. Położenie widżetów jest zapamiętywane przez konfigurację aplikacji hosta.

Rysunek 29. Przycisk „Przesuń”

Źródło: Opracowanie własne

- Obróć (Rysunek 30) – obraca widżet pokazując drugi Interfejs Graficzny. Interfejs oznaczony jako „Front” jest domyślnie pokazywany jako pierwszy, kliknięcie przycisku Obróć powoduje pokazanie Interfejsu „Back”. Efekt obrotu jest animowany w przestrzeni 3D.

Rysunek 30. Przycisk „Obróć”

Źródło: Opracowanie własne

- Zamknij (Rysunek 31) – zamyka instancję widżetu, aktualizując jednocześnie konfigurację aplikacji hosta.

Rysunek 31. Przycisk „Zamknij”

Źródło: Opracowanie własne

5.4. Widżet

5.4.1 Tworzenie nowego widżetu – WidgetBase

W opracowanym prototypie Autor w dużej mierze skupił się na prostocie tworzenia nowych widżetów. Klasa znajdująca się w bibliotece WidgetBase, o tej samej nazwie, dostarcza widżetom ich podstawowej funkcjonalności oraz deklaruje metody, których ciało może zaimplementować autor widżetu. WidgetBase tworzy ramę widżetu, dając jednocześnie dostawcom wtyczek dużą swobodę w jej wypełnieniu. Tworzenie nowego widżetu rozpoczynamy poprzez stworzenie nowej klasy implementującej klasę WidgetBase z biblioteki o tej samej nazwie. Implementacja wymaga zdefiniowania ciała dwóch metod: init() i dispose(). Autor wtyczki w metodzie init() powinien ustawić dwa atrybuty pochodzące z klasy WidgetBase: FrontWidget i BackWidget. Oba pola są typu FramworkElement, będącego podstawą wszystkich elementów GUI w WPF. FrontWidget jest pokazywany domyślnie jako pierwszy w momencie wyświetlenia wtyczki w systemie. Dopiero polecenie „Obróć” uruchamia efekt animacji i pokazuje BackWidget, jednocześnie ukrywając

FrontWidget. Metoda dispose() daje autorowi widżetu możliwość zdefiniowania operacji zakończenia pracy i jest ona uruchamiana w momencie zamknięcia aplikacji hosta.

Klasa widżetu musi ponadto być oznaczona atrybutem [AddIn]. Atrybut może przyjąć także parę parametrów, takich jak nazwa, opis, wersja, dostawca widżetu. Możliwe jest zdefiniowanie dodatkowych opcjonalnych ustawień poprzez atrybut QualificationData, co zostało opisane dokładniej w Rozdziale 5.4.3.

Wymiary widżetu są określane poprzez atrybut Size i także mogą być zdefiniowane w metodzie init().

Przykładową implementację widżetu, o wymiarach 230x225, pokazującego po jednym przycisku na każdej ze swoich stron, przedstawia przykład 7.

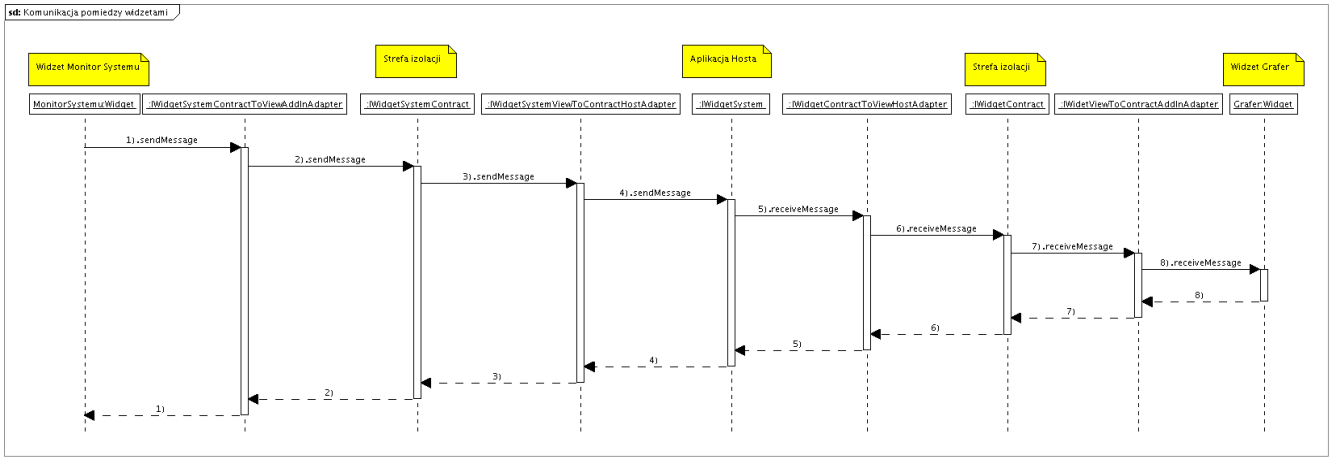
Przykład 7. Implementacja widżetu o wymiarach 230x225

```
[System.AddIn.AddIn("Widżet Testowy", Description = "Przykladowy widżet",
Publisher = "Robert Meisner",Version="1.0.0.0")]
[System.AddIn.Pipeline.QualificationData("Shape", "Rectangle")]
public  class MyAddinCode :WidgetBase.WidgetBase
    {


        internal static Button front = null;
        internal static Button back = null;

        public override void init()
        {

           front = new Button() { Content = "Przycisk z przodu widżetu." };
            back = new Button() { Content = "Przycisk z tyłu widżetu." };
            this.FrontWidget =front;
            this.BackWidget=back;
            this.Size = new Size(230,225);
        }
        public override void dispose()
        {

        }

    }
```

5.4.2 Komunikacja pomiędzy widżetami

Opracowany prototyp pozwala na wzajemną komunikację pomiędzy widżetami. Komunikacja jest kontrolowana przez aplikację hosta. Rysunek 32 przedstawia diagram sekwencji opisujący drogę komunikacji. Widzimy, że w procesie uczestniczą dwie strefy izolacji, co oznacza, że komunikacja odbywa się pomiędzy trzema procesami aplikacji. Wiadomość jest wysyłana od nadawcy do aplikacji hosta, która przekazuje ją do adresata. W komunikacji pomiędzy nimi biorą udział odpowiednie

adaptery, pozwalające na konwersję typów pomiędzy poszczególnymi segmentami kanałów komunikacji.



Rysunek 32. Przesyłanie wiadomości pomiędzy widżetami

Źródło: Opracowanie własne

Metoda dostępna dla widżetów służąca do przekazywania wiadomości do innego widżetu nazywa się sendMessage i jest wywoływana na obiekcie Aplikacji Hosta. Jako parametry przyjmuje GUID adresata, typ wiadomości, oraz dwie kolekcje typów double i string. Rolą widżetu chcącego udostępniać możliwość przesyłania wiadomości jest określenie protokołu tej komunikacji. Autorzy przy dystrybucji widżetu powinni dołączać opis protokołu wraz z jego dokumentacją. Dostawca wtyczki może dodać możliwość odbioru wiadomości od innych wtyczek poprzez przysłonięcie metody OnMessageReceived, jako parametry przyjmuje ona nazwę widżetu nadawcy oraz jego GUID, typ wiadomości oraz dwie kolekcje typów double oraz string.

5.4.3 Konfiguracja Widżetów

W opracowanym prototypie Autor pracy umożliwił swobodę w tworzeniu własnych mechanizmów konfiguracji. Ograniczenia wynikają tylko i wyłącznie z praw dostępu nadanych widżetowi przez użytkownika.

Widżety posiadają jednak możliwość zdefiniowania pewnych wspólnych właściwości w momencie ich uruchomienia. Ustawienia te są deklarowane przez widżet poprzez atrybuty jego klasy głównej. Atrybuty te są obsłużone z użyciem klasy QualificationData i odczytywane przez aplikację hosta w momencie wykrycia widżetu.

Możliwe ustawienia i ich wartości:

- **SecurityLevel** – poziom zaufania wymagany przez widżet. Możliwe wartości:

- NoTrust (wartość domyślna) – brak zaufania, dostęp do Isolated storage i nadane prawa wykonywania i GUI
- FullTrust – pełny poziom zaufania. Widżet, jeśli użytkownik wyrazi na to zgodę otrzyma pełny dostęp do komputera, ograniczony jedynie prawami użytkownika

- **TwoSided –** definiuje, czy widżet ma posiadać dwie strony i być wyświetlany w przestrzeni 3D z możliwością obrotu. Możliwe wartości:
  - Yes (wartość domyślna)
  - No
- **MultipleInstances –** definiuje, czy widżet może być uruchamiany więcej niż raz. Przykładem w prototypie widżetu, który może być uruchamiany więcej niż raz jest Zegar, z kolei przykładem, w którym takie zachowanie nie jest wskazane jest Monitor Systemu. Możliwe wartości:
  - Yes (wartość domyślna)
  - No
- **Shape –** określa kształt widżetu. Możliwe dwie wartości:
  - Rectangle (wartość domyślna) – Wtyczka ma kształt prostokąta ograniczony przez jego atrybut Size.
  - Ellipse – Wtyczka ma kształt elipsy ograniczonej przez jego Size.

Przykładową deklaracja powyższych ustawień przedstawia przykład 8.

Przykład 8. Konfiguracja widżetu

```
[System.AddIn.AddIn("Grafer", Description = "Widget rysuje wykres o
funkcji wielomianowej o podanym wzorze.", Publisher = "Robert Meisner")]
[System.AddIn.Pipeline.QualificationData("SecurityLevel", "FullTrust")]
[System.AddIn.Pipeline.QualificationData("TwoSided", "Yes")]
[System.AddIn.Pipeline.QualificationData("MultipleInstances", "Yes")]
[System.AddIn.Pipeline.QualificationData("Shape", "Ellipse")]
public class Grafer : WidgetBase.WidgetBase { ... }
```

5.4.4 Bezpieczeństwo widżetu

Widżety mogą być uruchamiane przez użytkownika w dwóch trybach poziomu zaufania: Brak Zaufania i Pełne Zaufanie.

W trybie Braku Zaufania zbiór uprawnień jest tworzony poprzez wydzielenie części uprawnień z dostarczonego przez .NET zbioru o nazwie „Internet” i rozszerzenie go o dodatkowe możliwości. Zbiór ten jest domyślnym zbiorem uprawnień dla wszystkich aplikacji uruchamianych z Internetu w systemie Windows. Rozszerzenie uprawnień jest konieczne, aby możliwe było poprawne

uruchomienie aplikacji okienkowej jaką jest widżet i stworzenie izolowanej przestrzeni na dysku do przechowywania plików tymczasowych wtyczki. Kod odpowiedzialny za stworzenie uprawnień w trybie braku zaufania:

```
PermissionSet psWidget = GetNamedPermissionSet("Internet");
psWidget.AddPermission(new
SecurityPermission(SecurityPermissionFlag.UnmanagedCode));
IsolatedStorageFilePermission isolatedStoragePermission = new
IsolatedStorageFilePermission(PermissionState.Unrestricted);
isolatedStoragePermission.UsageAllowed =
IsolatedStorageContainment.AssemblyIsolationByUser;
isolatedStoragePermission.UserQuota = 1024 * 20;
psWidget.AddPermission(isolatedStoragePermission);
psWidget.AddPermission(new UIPermission(PermissionState.Unrestricted));
```

Jak widać, uprawnienia te dają możliwość uruchamiania niezaufaneggo kodu, dają dostęp do Isolated Storage wtyczce, oraz pozwalają na używanie GUI

W trybie Pełnego Zaufania wtyczka dostaje uprawnienia równe uprawnieniom uzytkownika, który uruchomił system widżetów. Zwykle wiąże się to z nieograniczonym dostępem do danych użytkownika i sieci.

5.4.5 Interfejs użytkownika widżetu

Domyślnie każdy widżet może zdefiniować dwa interfejsy graficzne dostępne dla użytkownika. Dostęp pomiędzy nimi użytkownik uzyskuje poprzez wywołanie metody Rotate(). Efekt animacji obracania w przestrzeni 3D sprawia wrażenie, że wtyczki mają dwie strony. Autor pracy przyjął, że interfejs dostępny od razu po uruchomieniu aplikacji znajduje się z przodu widżetu, natomiast niewidoczny z tyłu. Intencją Autora jest, aby interfejs z tyłu służył większości widżetów do tworzenia formularzy konfiguracyjnych, tak też zaprojektowane są trzy widżety dostarczone z prototypem. Pomimo tego, twórcy widżetów nie są w żaden sposób zobligowani do takiego projektowania swoich mini-aplikacji. Interfejs wtyczek powinien być projektowany z użyciem bibliotek Windows Presentation Foundation (WPF). Autor pracy uznał, że użycie technologii WPF najbardziej współgra z polityką firmy Microsoft dotyczącą tworzenia GUI w najnowszych wersjach systemu Windows i daje największe możliwości co do swobody tworzenia profesjonalnego, interaktywnego interfejsu graficznego. Autorzy wtyczek nie są ograniczeni przez API systemu i mogą używać wszystkich kontrolek, styli i templatek dostępnych w WPF oraz na platformie Windows.

5.5. Widżety obecne w prototypie

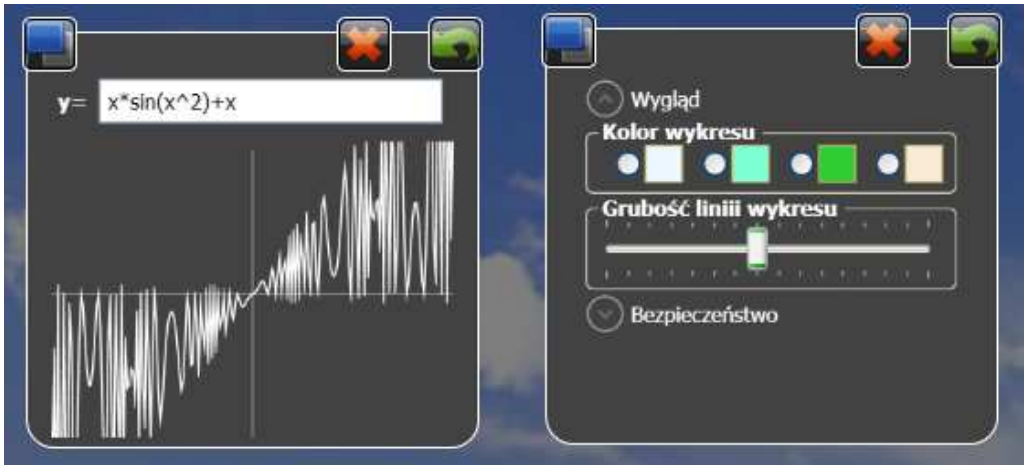
Opracowany prototyp systemu zawiera trzy widżety: Grafer, Monitor Systemu oraz Zegar.

5.5.1 Grafer

Widżet ten służy do rysowania wykresu funkcji wielomianowej o wzorze podanym przez użytkownika (patrz Rysunek 33). W części przedniej widżetu znajduje się pole tekstowe i obszar rysowania wykresu. Wzór funkcji wprowadzany jest za pomocą klawiatury w pole tekstowe znajdujące się przy górnej krawędzi widżetu. Wykres jest odświeżany w trakcie wprowadzania tekstu. W części tylniej znajduje się panel konfiguracyjny widżetu składający się z dwóch części: Bezpieczeństwo i Wygląd. W części wygląd użytkownik może zmienić kolor linii wykresu oraz jej grubość. W części Bezpieczeństwo użytkownik może zezwolić innym widżetom na komunikację z Graferem. Logika obsługi komunikacji jest oparta na wzorcu maszyny stanów.

Grafer może udostępnić swoją funkcjonalność rysowania wykresów innym widżetom, jeśli użytkownik wyrazi na to zgodę. Oferuje on własny protokół komunikacji, oparty na metodach komunikacji dostarczonych przez prototyp. Protokół ten przewiduje dwa typy wiadomości:

- „populate” – jeśli grafer nie jest pod kontrolą innego widżetu i użytkownik zezwolił na komunikację z innymi wtyczkami, wiadomość ta rysuje wykres używając kolekcji typu double, przekazywanej wraz z wiadomością jako współrzędne punktów wykresu. W przypadku, gdy użytkownik nie dał zezwolenia - widżet obraca się pokazując tylnią stronę i pokazuje część Bezpieczeństwo, prosząc tym samym użytkownika, aby zdefiniował czy widżet ma się komunikować z innymi;
- „populate_end” – wiadomość ta jest wysyłana przez widżet komunikujący się z Graferem w celu zakończenia połączenia.



Rysunek 33. Przód i tył widżetu Grafer

Źródło: Opracowanie własne

5.5.2 Monitor Systemu

Rolą tego widżetu jest monitorowanie poziomu zużycia zasobów systemowych. W przedniej części widżet wyświetla w formie tekstu bieżące zużycie zasobów (patrz Rysunek 34), takich jak: zużycie procesora, poziom zajętości pamięci fizycznej i wirtualnej, prędkość zapisu i odczytu danych z dysku oraz ilość danych wysyłanych i odczytywanych przez połączenie sieciowe. Wartości są odświeżane co jedną sekundę. Monitor Systemu nie posiada graficznej Interpretacji danych, które przetwarza - potrafi jednak za pomocą interfejsu dostarczonego przez widżet Grafer przejąć nad nim kontrolę i użyć go do wyświetlania zmian poziomu zużycia zasobów w czasie ostatnich 60 sekund. Widżet w swojej tylniej części zawiera panel pozwalający na skonfigurowanie komunikacji z widżetem Grafer. Po uruchomieniu widżetu używa on metody aplikacji hosta o nazwie getWidgetGuids w celu uzyskania wszystkich identyfikatorów uruchomionych instancji widżetu Grafer. Użytkownik następnie w panelu konfigurującym wybrać z listy widżet o danym GUID i użyć go do wyświetlania wykresu zmian zużycia zasobów. Dostępne są cztery opcje:

- „Nie integruj” – wyłącza integrację, Widżet Grafer wraca do swojego wyjściowego stanu;
- CPU – pokazuje w formie wykresu ostatnie 60 sekund procentowe użycie mocy obliczeniowej procesora;
- Net input – wyświetla w formie wykresu ilość danych odczytanych przez połączenie sieciowe w ciągu ostatniej minuty;
- Net output – wyświetla w formie wykresu ilość danych wysłanych przez połączenie sieciowe w ciągu ostatniej minuty.

Rysunek 34. Przód i tył widżetu Monitor Systemu

Źródło: Opracowanie własne

5.5.3 Zegar

Rolą tego widżetu jest wyświetlanie godziny w danej strefie czasowej. Godzina jest wyświetlana w postaci wskazówkowego, dwunastogodzinnego zegara (patrz Rysunek 35). Użytkownik może zmienić strefę czasową poprzez panel konfiguracji dostępny w tylniej części widżetu. Charakterystyczną cechą tego widżetu jest jego eliptyczny kształt.

Rysunek 35. Przód i tył widżetu zegar

Źródło: Opracowanie własne

6. Zalety i wady przyjętych rozwiązań

6.1. Airspace Problem

Przestrzeń Powietrzna „Airspace” jest abstrakcyjnym pojęciem opisującym jak dwie współdziałające (interoperational) aplikacje dzielą przestrzeń rysowania (render area) w ramach jednego okna [5]. Koncepcja Przestrzeni Powietrznej wpłynęła w zasadniczy sposób na projekt i architekturę interfejsu graficznego prototypu. W ramach top-level okna, każdy HWND (Window Handle), który używa jednej z technologii dostępnych w systemie Windows, służących do rysowania interfejsu graficznego posiada swoją własną Przestrzeń Powietrzną. Każdy piksel w ramach okna należy dokładnie do jednego HWND i składa się na jego przestrzeń powietrzną. Koncepcja Przestrzeni Powietrznej zakłada, że wszystkie warstwy lub okna próbujące rysować ponad tym pikselem muszą należeć do tej samej technologii rysowania. Próba rysowania pikseli WPF nad tymi używającymi technologii Win32 prowadzi do nieprzewidywalnych wyników.

Pomimo, że w projektowanym prototypie pojedyncze widżety są rysowane z użyciem technologii WPF tak samo jak aplikacja hosta, to interfejs graficzny jest przesyłany z użyciem technologii Win32. Ograniczenie to jest wymuszone przez MAF. Kontrolka WPF przekazywana przez widżet jest opakowywana w obiekt proxy technologii Win32, co pozwala na przekazanie go przez kanał komunikacji i strefę izolacji. Technologia WPF na stan dzisiejszy nie oferuje narzędzi pozwalających na przekazywanie elementów GUI pomiędzy procesami bez wykorzystania technologii Win32. W trakcie tworzenia prototypu Autor pracy napotkał problemy z poprawnym wyświetlaniem widżetów. Ich interfejs graficzny zawsze znajdował się nad elementami aplikacji hosta, co w przypadku chęci użycia kontenera ScrollViewer pozwalającego na przewijanie treści, a tym samym ukrywającego jej części, powodowało niepożądane efekty. Rozwiązaniem okazało się użycie osobnych okien zawierających GUI widżetów i zmieniających swoje wymiary i położenie w zależności od położenia obiektu określającego jego pozycję w aplikacji hosta oraz od krawędzi kontenera ScrollViewer. Sposób implementacji tego rozwiązania opisano w Rozdziale 5.4.5.

6.2. Managed AddIn Framework

Pomimo tego, że MAF dostarcza rozwiązań wielu inżynieryjnych problemów, takich jak izolacja, bezpieczeństwo, komunikacja pomiędzy procesami, to krzywa czasu koniecznego na naukę (learning curve) tego rozwiązania jest wyjątkowo długa. Sam framework jest mało intuicyjny. Jego

architektura kanału komunikacji (Pipeline) sprawia trudności nawet doświadczonym programistom, a brak wyczerpującej dokumentacji znacznie zwiększa czas i koszty konieczne do stworzenia oprogramowania opartego o tą platformę. Zespół pracujący nad MAF w celu uproszczenia i zmniejszenia nakładu pracy programistów opracował rozwiązanie dystrybuowane poza platformą .NET w formie rozszerzenia Visual Studio. Rozszerzenie to o nazwie Pipeline Builder, pozwala na automatyzację tworzenia kanału komunikacji. Jego generyczność zmusza jednak autorów oprogramowania do nieingerowania w strukturę kanału komunikacji, co w przypadku opracowanego prototypu, z powodu złożonych oczekiwań co do Interfejsu Graficznego, nie było możliwe. Wadą stworzonej aplikacji jest niewątpliwie jej zachłanność co do pamięci operacyjnej systemu. Jest to jednak wynikiem założeń autora pracy co do stabilności i izolacji elementów systemu. Uruchamianie widżetów w osobnych procesach powoduje większe, niż w aplikacjach o mniejszym poziomie izolacji, użycie pamięci.

7. Proponowane plany rozwoju

7.1. Open Source

Autor pracy uważa, że udostępnienie źródeł na zasadach licencji GPL w wersji 3.0 (GNU General Public License) w znaczny sposób przyśpieszy jej rozwój oraz wprowadzanie poprawek dla ewentualnych usterek. Wybór licencji jest elementem strategii biznesowej tworzonego oprogramowania i w przypadku opracowanego prototypu intencją Autora pracy jest nadanie jej wymiaru edukacyjnego. Aplikacja ta może być prezentowana w czasie zajęć dydaktycznych jako przykład oprogramowania modułowego o rozbudowanej izolacji elementów lub jako studium tworzenia zaawansowanych interfejsów graficznych z wykorzystaniem WPF i .NET 3.5.

7.2. Użycie języków skryptowych

Wprowadzenie języków skryptowych (Javascript, LUA), jako narzędzi do definiowania interfejsu użytkownika, może znacznie rozszerzyć grono potencjalnych dostawców wtyczek. W tym przypadku konieczne byłoby stworzenie interpretera oraz wyspecjalizowanego API po stronie hosta, pozwalających na rysowanie widżetów. Wprowadzenie języków skryptowych pozwoliłoby na tworzenie widżetów nie tylko profesjonalnym programistom, ale także amatorom, zwykle posiadającym tylko podstawową wiedzę na temat technologii webowych. Autor pracy uważa, że zastosowanie Javascript w połączeniu z HTML'em jako narzędzi do definiowania GUI i interakcji z użytkownikiem byłoby najlepszym rozwiązaniem dla widżetów o prostych, ograniczonych funkcjonalnościach. Z istniejących rozwiązań takie podejście do projektowania logiki i interfejsu graficznego prezentuje Yahoo! Widget Engine.

7.3. Baza widżetów online

Popularność każdego z istniejących systemów widżetów bezpośrednio jest dyktowana przez dostępność i różnorodność wtyczek dla nich tworzonych. Także w przypadku opracowanego prototypu konieczne, zdaniem Autora, jest stworzenie centralnej bazy widżetów. Baza taka powinna dawać możliwość zewnętrznym dostawcom na dodawanie wtyczek i zarządzanie ich wersjami. Dodatkowo baza może dostarczać funkcjonalności śledzenia błędów, dzięki której użytkownicy widżetów będą mogli zgłaszać usterki w działaniu widżetów, a także wprowadzać propozycje dotyczące ich rozwoju.

8. Podsumowanie

Systemy zarządzające generycznymi widżetami są coraz bardziej popularne. Stają się wręcz niezbędne dla osób często korzystających z komputerów. Uzupełniają braki funkcjonalne systemów operacyjnych oraz oprogramowania. Zakres funkcjonalności jest niezwykle szeroki, od aplikacji rozrywkowych, po te zarządzające bazami danych. Wiele lat doświadczeń producentów systemów oraz samych widżetów dało podstawy autorowi pracy do zdefiniowania szeregu standardów oraz ogólnie przyjętych norm co do jakości i funkcjonalności dostarczanych przez produkty tego typu. Dotyczą one zaawansowanych rozwiązań architektonicznych w obszarze warstw logiki biznesowej, dostępu do danych oraz prezentacji. Autor zwraca uwagę na złożoność architektury oraz na problemy z którymi zmagają się twórcy systemów widżetów.

Opracowany w ramach pracy prototyp realizuje założenia biznesowe tego typu aplikacji. Implementuje je używając nowoczesnych i nowatorskich technologii. Z ich użyciem rozwiązuje problemy tworzenia widżetów, wersjonowania, bezpieczeństwa, izolacji oraz interfejsu graficznego. W ramach prototypu została stworzona aplikacja hosta, służąca do zarządzania widżetami wyświetlanymi na pulpicie użytkownika. Ponadto zostały opracowane trzy widżety prezentujące możliwości oferowane przez system w zakresie ich interfejsu graficznego, czy też komunikacji. Pomostem łączącym wtyczki oraz aplikację hosta jest kanał komunikacji (Pipeline). Został on zbudowany w oparciu o technologię dostarczoną przez firmę Microsoft wraz z platformą .NET 3.5. Kanał ten składa się z wielu segmentów gwarantujących możliwość wersjonowania i izolacji. Każdy z segmentów został utworzony jako osobny projekt, aby ułatwić analizę użytych rozwiązań i ułatwić, w przyszłośći, wprowadzanie zmian.

Oprogramowanie opracowane w ramach pracy może zostać wykorzystane na potrzeby dydaktyczne, jako studium przypadku tworzenia aplikacji modułowych, w których architektura i kod aplikacji hosta jest ukryty przed dostawcą modułu. Jakość stworzonej aplikacji pozwala na wykorzystanie jej do rozszerzania i dodawania nowych funkcjonalności w ramach środowiska pracy użytkownika.

9. Bibliografia

[1] Sad Ben Kunz, Why Widgets Don't Work
http://www.businessweek.com/print/technology/content/feb2008/tc20080229_131531.htm

[2] Thor Kottelin, Widget inflicts malware—Facebook ignores advisory?
http://blog.anta.net/2008/01/05/widget-inflicts-malware-facebook-ignores-advisory/

[3] Christoph Eicke, Apple Dashboard Widget Insecurity
http://www.geisterstunde.org/drupal/?q=node/67

[4] Butler W. Lampson, Protection,
źródło: ACM SIGOPS Operating Systems Review, Volume 8, Issue 1 (January 1974)
ISSN:0163-5980
http://portal.acm.org/citation.cfm?id=775268&dl=GUIDE&coll=GUIDE&CFID=51122237&CFTOKEN=75946605

[5] Dokumentacja MSDN .NET Framework 3.5, WPF Interoperation: "Airspace" and Window Regions Overview
http://msdn.microsoft.com/en-us/library/aa970688.aspx

[6] Andrew Troelsen, Pro C# 2008 and the .NET 3.5 Platform Fourth Edition, Apress 2007, ISBN10: 1-59059-884-9

10. Spis rysunków

11. Spis przykładów