

Maciej Przybylski, Jakub Możaryn

# Zasady budowy GUI

Materiały dydaktyczne, Ośrodek Kształcenia Na Odległość – OKNO

2009-08-14

## Zawartość

1 Zasady budowy GUI.....	3
1.1 Wprowadzenie do projektowania graficznego interfejsu użytkownika.....	3
1.1.1 Model użytkownika.....	3
1.1.2 Pomoc i komunikaty.....	6
1.1.3 Inteligentny ineterfejs użytkownika.....	8
1.1.4 Testy użyteczności.....	8
1.1.2 Wzorzec MVC.....	9
1.1.3 Dostosowanie interfejsu użytkownika ??.....	10
1.1.4 Estetyka i funkcjonalność.....	10
1.2 Podstawowe elementy GUI w środowisku Turbo Delphi.....	14
1.2.1 Menu (TMainMenu, TPopupMenu).....	14
1.2.2 Pasek stanu (TStatusBar).....	16
1.2.3 Elastyczne obszary interfejsu (TSplitter).....	17
1.3 Scentralizowane zarządzanie wywoływaniem i udostępnianiem funkcji.....	19
1.3.1 Lista akcji (TActionList).....	19

# 1 Zasady budowy GUI

## 1.1 Wprowadzenie do projektowania graficznego interfejsu użytkownika

Pierwszy graficzny interfejs użytkownika (ang. Graphical User Interface) został stworzony przez firmę Xerox w latach siedemdziesiątych. Wprowadzenie przez firmę Apple w roku 1984 pierwszego komputera osobistego z graficznym systemem operacyjnym zmieniło całkowicie podejście do tworzenia oprogramowania. Wraz z systemem Mac OS pojawiły się doskonale dziś znane pulpit, okno, folder, kosz czy ikony. Dzięki metaforom do przedmiotów z otoczenia system Mac OS był bardzo łatwy w nauce i obsłudze.

Od tamtej pory powstało wiele mniej lub bardziej użytecznych programów. Zazwyczaj o popularności oprogramowania decyduje łatwy w obsłudze i funkcjonalny interfejs użytkownika, dopiero w dalszej kolejności jego możliwości. Warto, więc stosować się do kilku zasad.

### 1.1.1 Model użytkownika

Zanim zaczniemy projektować GUI musimy zastanowić się jaki będzie przeciętny użytkownik naszego programu, czy będzie do specjalista od grafiki komputerowej czy ktoś kto dopiero poznaje środowisko graficzne i chciałby tylko napisać i wydrukować list. Musimy, więc stworzyć model użytkownika.

### *Interfejs powinien być bliski użytkownikowi*

We wstępie powiedzieliśmy już, że system operacyjny Mac OS zdobył swą dużą popularność dzięki zastosowanym metaforom pulpitu, folderów, kosza, etc. (rys. 1). Projektując interfejs poczynając od nazw i kolejności operacji, kończąc na ikonach powinniśmy odwoływać się do skojarzeń i doświadczeń potencjalnego użytkownika [1]. Oczywiście wcześniej powinniśmy odpowiedzieć sobie na pytanie dla kogo projektujemy aplikację. Pamiętajmy przy tym, że człowiek kieruje się najprostszym możliwym schematem. Należy także mieć świadomość faktu, że użytkownik ma ograniczone zdolności percepcji, m.in. słabą pamięć krótkotrwałą.

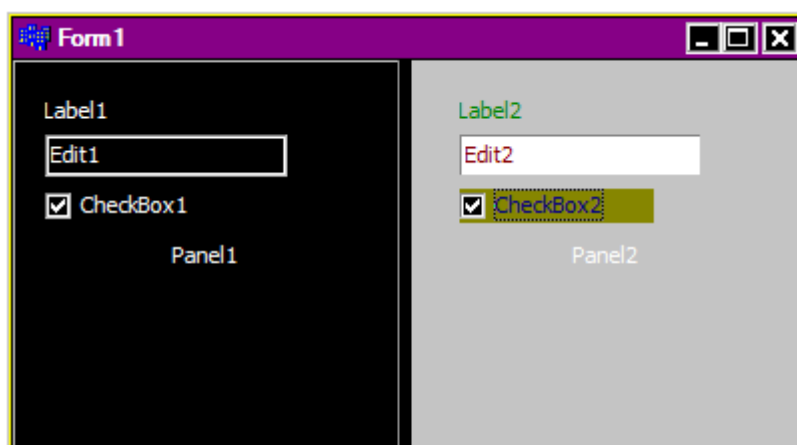


Rysunek 1: Pulpit systemu Mac OS

## ***Użytkownik nie lubi być zaskakiwany***

Wspólną cechą dla każdego użytkownika jest z pewnością fakt, że nie lubi on być zaskakiwany. Dlatego program powinien zachowywać się tak jak oczekuje tego użytkownik [2]. W praktyce oznacza to, że jeżeli piszemy edytor tekstu, to większość osób korzystających z systemu Windows będzie się spodziewać działania podobnego do programu Word. Oczywiście program Word nie jest narzędziem idealnym, jednak wiele mechanizmów wynika z wieloletniego doświadczenia producenta. Jeżeli nie masz tak rozbudowanego laboratorium zajmującego się badaniem użyteczności GUI, jak firma Microsoft, zastanów się dobrze zanim wprowadzisz własne udoskonalenia w interfejsie. Zasada ta dotyczy szczególnie wprowadzania własnych kontroltek.

O ile jest to możliwe korzystaj ze standardowych kontroltek WinAPI lub środowiska Borland. Może się okazać, że kontrolka napisana przez nas nie reaguje na ustawienia globalne systemu operacyjnego, na przykład na specjalne schematy kolorów o wysokim kontraście zaprojektowane dla osób ze słabszym wzrokiem (rys. 2), a przecież nasza aplikacja ma być użyteczna dla wszystkich. Podobne problemy mogą się pojawić na lokalnych wersjach systemu operacyjnego w innych krajach, gdzie nasza kontrolka może zachowywać się w sposób nieprzewidywalny.

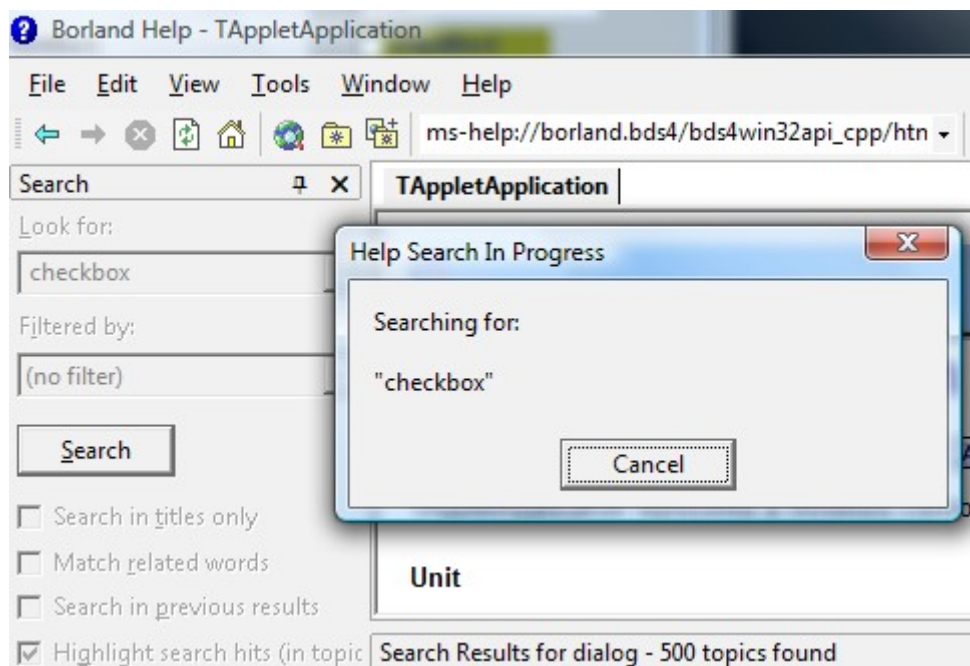


**Rysunek 2.** Komponenty, którym zmieniono domyślny kolor nie reagują na zmiany globalnych ustawień systemu operacyjnego, np. na wysoki kontrast.

Przy tej zasadzie można dodać, że interfejs powinien być spójny. Oznacza to, że podobne funkcje powinny przebiegać analogicznie tam gdzie to jest możliwe.

## ***Użytkownik nie lubi czekać***

Na ogólną ocenę działania programu znaczący wpływ ma czasu reakcji interfejsu. Istotne jest, aby interfejs reagował szybko na wywołanie operacji czy to przez zmianę kursora myszy na klepsydre, pojawienie się paska postępu, czy też wyświetlenie odpowiedniego komunikatu (rys. 3). Brak szybkiej informacji zwrotnej sprawi, że użytkownik ponownie uruchomi tę samą opcję sądząc, że wcześniej źle kliknął myszą, lub nie do końca wcisnął przycisk. Jeżeli nasz program nie jest odpowiednio zabezpieczony z pewnością doprowadzi to do nieprzewidzianej sytuacji.



**Rysunek 3:** Komunikat informujący o trwającej operacji wyszukiwania

Oczywiście ze względu na złożoność obliczeniową, niektóre operacje muszą trwać dłużej. Powinniśmy wtedy uprzedzić o tym użytkownika poprzez odpowiedni komunikat. Dobrym rozwiązaniem może być grupowanie kilku dłuższych procesów w jeden, oczywiście wyświetlając odpowiednią informację, i o ile to możliwe podając orientacyjny czas wykonania operacji. Takie podejście przyda się przy projektowaniu instalatorów. Zaleca się, aby wszystkie informacje dotyczące instalacji zostały zebrane przed jej rozpoczęciem. Tak, aby użytkownik mógł spokojnie pójść na kawę nie obawiając się, że po powrocie ujrzy instalator zatrzymany w połowie procesu i oczekujący na jego decyzję.

Warto sobie uświadomić fakt, że interfejsy graficzne obsługiwane za pomocą myszki nie zawsze są szybkie. Administratorzy sieci informatycznych, którzy na co dzień korzystają z wiersza poleceń w systemach Unix'owych z pewnością wykonają większość operacji znacznie szybciej niż użytkownik posługujący się myszką i programem z interfejsem graficznym. Oczywiście wymaga to znajomości kilkunastu lub więcej poleceń, natomiast w przypadku GUI nawet jeżeli nie znamy jakiegoś polecenia to znajdziemy je w menu głównym lub podręcznym. Interfejs graficzny i szybkość działania można połączyć wykorzystując skróty klawiszowe.

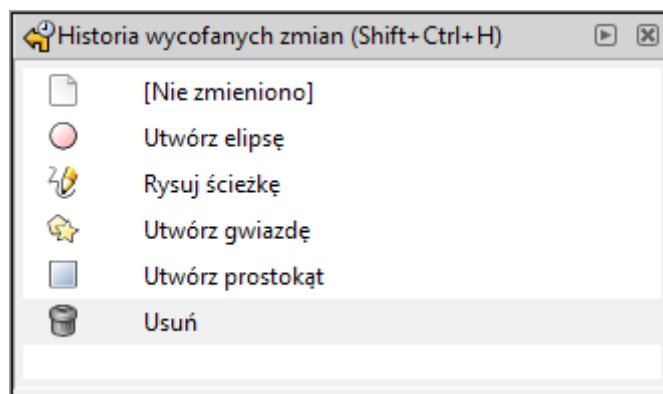
Jeżeli jakieś opcje w naszym programie powtarzane są często, warto przypisać im skróty klawiszowe, co z pewnością docenią zaawansowani użytkownicy pracujący z takim programem na co dzień. Wybierając skróty pamiętajmy, że wiele jest już zarezerwowanych dla typowych funkcji, wspólnych dla wielu aplikacji, takich jak zapisywanie (Ctrl+S) czy cofanie (Ctrl+Z). Spis najważniejszych skrótów klawiszowych systemu Windows można znaleźć pod adresami <http://www.ujk.edu.pl/bg/pliki/skrotyWindows.pdf> i <http://www.hongkiat.com/blog/100-keyboard-shortcuts-windows/> (język angielski).

Pamiętajmy również, że z komputerów korzystają także osoby o ograniczonych możliwościach ruchowych, którym precyzyjne operowanie myszką może sprawiać problem. Dla nich skróty klawiszowe mogą być znacznym ułatwieniem.

### ***Użytkownik popełnia błędy***

Każdy użyteczny program powinien być wyposażony w mechanizm historii zmian z

możliwością ich cofnięcia i powtórzenia. Może to być prosta liniowa historia, gdzie kilka ostatnich zmian zapamiętywanych jest na stosie i możliwe jest cofnięcie tylko tej zmiany, która jest na wierzchu stosu (rys. 4). Z kolei użytkownicy programu do obróbki zdjęć Photoshop mają do dyspozycji historię nielinową, gdzie z listy wykonanych operacji można cofnąć dowolną.



**Rysunek 4:** Liniowa historia zmian programu Inkscape

Jeżeli operacja jest nieodwracalna, jak na przykład zamknięcie programu bez zapisania zmian do pliku, należy koniecznie poinformować o tym użytkownika.

Dobrze napisana aplikacja powinna zachować się stabilnie nawet jeżeli użytkownik wywołał przypadkowe opcje w kolejności innej niż założył to twórca oprogramowania. Można temu zapobiec jednocześnie ułatwiając użytkownikowi pracę pozostawiając niedostępnymi opcje, które w danym momencie pracy programu są nieprzydatne.

### 1.1.2 Pomoc i komunikaty

Użytkownik pracując z programem musi podejmować wiele decyzji, a w sytuacjach wyjątkowych zmierzyć się z informacjami o błędach. To czy uda mu się wykonać zaplanowane zadanie zależy w dużej mierze od czytelności i ilości informacji wyświetlanych przez program. Oczywiście nadmiar komunikatów może być irytujący dla użytkownika, który już dobrze zna jakąś aplikację. Dlatego planując mechanizm komunikatów warto wziąć pod uwagę dwa aspekty.

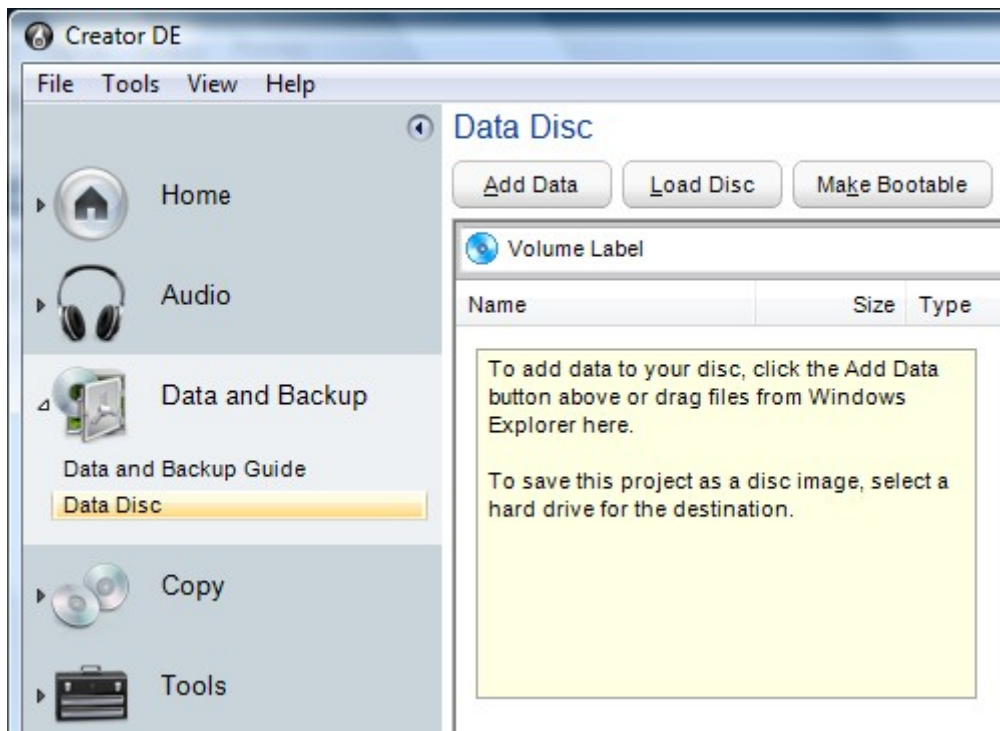
Po pierwsze użytkownicy różnią się między sobą znajomością danej aplikacji, systemu operacyjnego, czy terminologii informatycznej w ogóle. Zaawansowanym użytkownikom wystarczy krótka informacja, podczas gdy początkujący będą potrzebować dokładniejszych wyjaśnień. W innej sytuacji użytkownicy zaawansowani będą oczekiwali bardziej szczegółowych informacji, podczas gdy początkującym wystarczy informacja, że jakaś operacja nie może być wykonana. Dlatego warto stosować komunikaty z krótką informacją ogólną i opcjonalną informacją szczegółową, lub z odsyłaczem do odpowiedniej sekcji z pomocy.

Wspólne dla wszystkich komunikatów powinna być zasada maksimum informacji, minimum tekstu. Należy unikać żartów, a także w przypadku komunikatów o błędach nie dać odczuć użytkownikowi, że to jego wina, informując raczej co można w takiej sytuacji zrobić.

Kolejna rzecz jaką musimy wziąć pod uwagę to odpowiedź na pytanie, jak często nasz program będzie wykorzystywany? Możemy spróbować tu podzielić programy na takie, z których korzystamy codziennie, okresowo lub okazjonalnie. W pierwszej kategorii możemy umieścić edytory tekstu, przeglądarki internetowe, programy pocztowe. Po kilku godzinach nauki użytkownik z łatwością porusza się wśród podstawowych funkcji i dodatkowe

informacje nie są potrzebne.

Do drugiej kategorii możemy zaliczyć na przykład dodatkowe funkcje arkusza kalkulacyjnego, z których korzystamy rzadko. W takim przypadku warto pola edycyjne wzbogacić o dodatkowe informacje (rys. 5). To samo dotyczy programów, z których korzystamy okazjonalnie, czyli na przykład instalatorów lub złożonych procesów, gdzie użytkownik musi podjąć wiele decyzji ze świadomością ich skutków. Tutaj regułą jest korzystanie z kreatorów, które prowadzą użytkownika krok po kroku służąc obszerną informacją, jednocześnie pozwalając na cofnięcie się do wcześniejszych etapów.



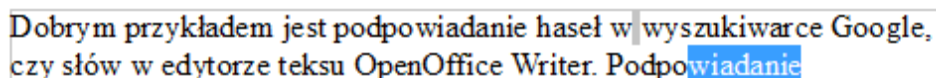
**Rysunek 5:** Dodatkowe informacje w obszarze roboczym programu do nagrywania płyt Roxio Creator

Bardziej rozbudowane programy, oprócz mechanizmu komunikatów, zawierają szczegółową pomoc. Tworząc pomoc musimy mieć na uwadze, że użytkownik szuka jej zazwyczaj w jednej z dwóch sytuacji: wystąpił jakiś błąd i potrzebna jest pilna informacja lub użytkownik potrzebuje dodatkowej wiedzy na temat jakiejś funkcji programu.



### 1.1.3 Inteligentny interfejs użytkownika

Wiele programów wykorzystuje w swoich interfejsach algorytmy heurystyczne, czyli takie, które podpowiadają najbardziej prawdopodobne rozwiązanie, co nie oznacza, że właściwe. Mogą one znacznie przyspieszyć i ułatwić pracę. Dobrym przykładem jest podpowiadanie haseł w wyszukiwarce Google, czy słów w edytorze tekstu OpenOffice Writer (rys. 6).



**Rysunek 6.** Algorytm heurystyczny - podpowiedzi programu OpenOffice Writer

Wprowadzaniu przez użytkownika danych zawsze będą towarzyszyć błędy przy czym nie zawsze jest konieczność wyświetlania komunikatu o błędzie. Część danych można łatwo poprawić np. kropka zamiast przecinka oddzielająca część całkowitą od dziesiętnej liczby. Inne mogą wymagać całych słowników, jak w przypadku edytorów tekstu.

Należy jednak uważać, aby algorytm heurystyczny zamiast pomagać nie zaczął irytować użytkownika. Przykładem może być nadgorliwość autokorekty w programie Word. Dobrze napisany algorytm heurystyczny powinien pozwalać na łatwe cofnięcie zmian.

### 1.1.4 Testy użyteczności

Nie można uznać za skończony program, który nie został przetestowany. O ile możliwe jest napisanie specjalnych testów sprawdzających zachowanie procedur i funkcji wywołanych z przypadkowymi wartościami, to przeprowadzenie testów użyteczności interfejsu użytkownika bez udziału testerów nie ma sensu.

Wbrew pozorom do przetestowania interfejsu użytkownika nie potrzeba wielu testerów. Wystarczy kilka, 5-6 osób. Przy takiej liczbie testerów wyniki statystycznie nie mają znaczenia. Jednak w praktyce okazuje się, że zwiększenie tej liczby nie wnosi nowych informacji.

Ważny natomiast jest sposób przeprowadzenia badań. Osoby wybrane do testu nie mogą mieć wcześniejszego kontaktu z badanym oprogramowaniem. Testerowi towarzyszy obserwator, któremu nie wolno w żaden sposób sugerować rozwiązania. Jego zadaniem jest natomiast notowanie zachowań i uwag testera. Dobrze jeżeli tester zastanawia się na głos i na bieżąco komentuje wykonywane czynności.

Ocena użyteczności nie jest prostym zadaniem, można jednak wyróżnić kilka atrybutów, które podlegają ocenie [1]. Po pierwsze jest to łatwość przyswajania. Oczywiście w przypadku oprogramowania specjalistycznego nie możemy liczyć, że księgowy z łatwością przyswoi sobie działanie programu typu CAD.

Drugi atrybut to szybkość działania. Nie mamy tu na myśli czasu liczonego w sekundach, a raczej szybkość zgodna z przyzwyczajeniami i oczekiwaniami użytkownika. Specjalista od efektów specjalnych będzie przyzwyczajony do tego, że renderowanie może zajmować dużo czasu, natomiast recepcjonistka w przychodni obsługująca dziesiątki pacjentów dziennie będzie oczekiwała natychmiastowego dostępu do bazy danych.

Kolejny aspekt oceny użyteczności, czyli niezawodność jest oczywisty. Mimo nietypowych zachowań użytkownika program nie ma prawa się zawiesić, ani zniszczyć efektów jego ciężkiej pracy. Zbliżonym atrybutem jest możliwość i łatwość cofania zmian oraz wycofania



się z przypadkowo wywołanych opcji.

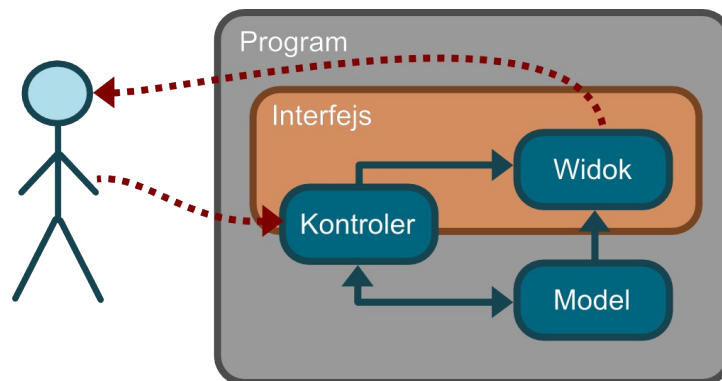
Ostatni atrybut dotyczy zdolności do adaptacji, czyli czy działanie programu ograniczone jest do jednego modelu pracy, czy możliwe są różne scenariusze.

### 1.1.2 Wzorzec MVC

W inżynierii oprogramowania istnieje pojęcie wzorców projektowych. Wzorce projektowe mogą opisywać strukturę programu, sposób tworzenia nowych obiektów lub współpracę między obiektami. Spośród wielu gotowych wzorców opracowanych przez doświadczonych twórców oprogramowania można znaleźć jeden konkretny, który będzie bardzo dobrze pasował do postawionego problemu. W tym rozdziale chcemy jednak omówić bardziej ogólny wzorzec, który dotyczy architektury programu jako całości, bez rozważania konkretnych zastosowań, i może być wykorzystany w dowolnym oprogramowaniu. Jest to wzorzec model-widok-kontroler (ang. model-view-controller MVC).

Najważniejszą cechą wzorca MVC jest wyraźne oddzielenie interfejsu od logiki programu. Warto zaznaczyć, że niezależnie od tego czy nasz program jest dokładną realizacją wzorca MVC czy nie, to powinniśmy przestrzegać separacji interfejsu od głównej funkcjonalności.

Sama nazwa model-widok-kontroler zawiera istotę tego wzorca. Zależności między elementami pokazane zostały na rysunku 7. Model to zbiór klas, które realizują główną funkcjonalność programu. Inaczej mówiąc są implementacją postawionego problemu np. zjawiska fizycznego, czy zbioru danych i relacji zachodzących między nimi. Model nie wie nic na temat widoku, a w prostym przypadku nie wie również o istnieniu kontrolera.



Rysunek 7: Wzorzec MVC

Podstawowym zadaniem kontrolera jest obsługa zdarzeń. W głównej mierze będą to zdarzenia pochodzące od użytkownika, ale mogą to być również zdarzenia wywołane przez model. Łatwo możemy wyobrazić sobie sytuację, w której zaszły jakieś zmiany w modelu wymagające odświeżenia widoku. Mamy więc do czynienia z przypadkiem, w którym model musi przewidywać istnienie jakiegoś zewnętrznego obiektu odbierającego zdarzenia. Rola widoku jest łatwa do przewidzenia. Istotny jest tylko fakt, że widok wie o istnieniu modelu, ale nie wie nic o kontrolerze. Kontroler natomiast musi posiadać wiedzę o modelu i widoku.

Kontroler może, ale nie musi być elementem interfejsu. W środowisku Borland każdy komponent wizualny możemy traktować jako zintegrowany kontroler i widok, gdyż pełnią one podwójną rolę. Wyświetlają informacje i są jednocześnie wyposażone w obsługę zdarzeń.

Przedstawiona tu została tylko ogólna idea wzorca MVC, który jest bardzo popularny i doczekał się wielu modyfikacji. Warto jednak jeszcze zaznaczyć, że często dzieli się model na dwa elementy; model podstawowy i model aplikacji. W tym ujęciu model podstawowy można zdefiniować podobnie jak już to zrobiliśmy wcześniej. Natomiast model aplikacji będzie już

uwzględniał istnienie interfejsu, jednak nie jest on w żaden sposób związany z konkretną jego implementacją. W modelu aplikacji będzie zaszyta logika, która zadba, aby po zajściu jakiegoś zdarzenia wywołane zostały odpowiednie metody modelu podstawowego, a widok został odświeżony.

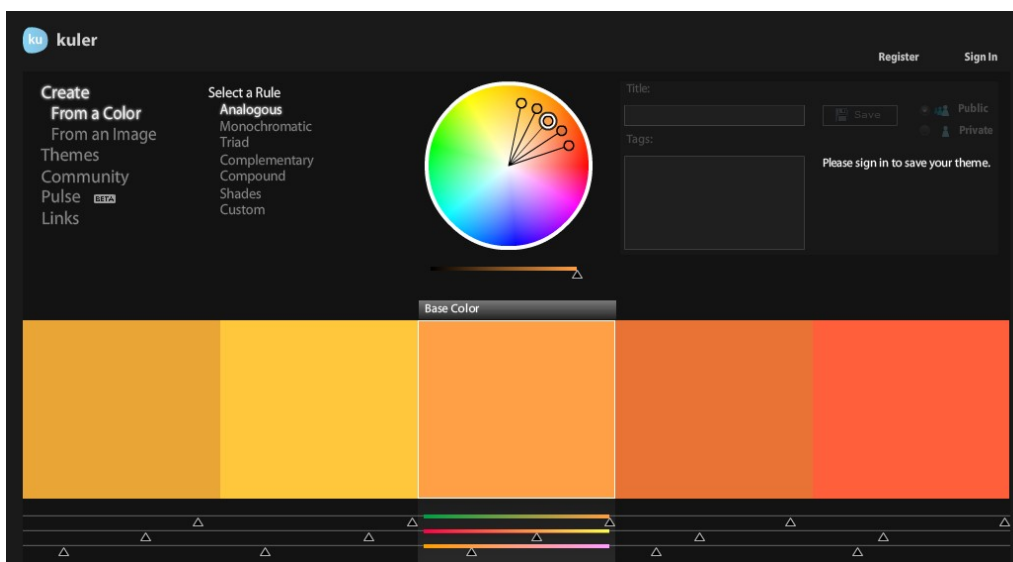
Stosowanie wzorca MVC nakłada pewne ograniczenia na twórców oprogramowania i z pewnością nie zmniejsza nakładu pracy. Jakie są zatem zalety tego wzorca? Najważniejszy jest fakt, że możemy wymienić dowolny z elementów bez modyfikacji pozostałych. Inną zaletą jest możliwość stworzenia dowolnej ilości niezależnych instancji kontrolerów czy widoków.

### 1.1.4 Estetyka i funkcjonalność

#### *Kolory i czcionki*

We wstępie mówiliśmy, że powinno stosować się standardowe komponenty i kolory, gdyż mamy pewność, że zachowają się odpowiednio przy zmianach ustawień globalnych systemu operacyjnego. Czasami jednak potrzeba uatrakcyjnienia interfejsu jest silniejsza.

Podstawową zasadą jest używanie jak najmniejszej liczby kolorów. W zupełności wystarczą 2-3 kolory podstawowe. Przy wyborze tych kolorów przydatne będą aplikacje www umieszczone na stronach <http://kuler.adobe.com> (rys. 8) i <http://websitetips.com/colortools/sitepro/>. W obu przypadkach możemy wybrać gotowy schemat kolorów m.in. monochromatyczny, kolory zbliżone czy kolory dopełniające się. Przedstawiając kolory na kole barw kolory zbliżone będą znajdować się blisko siebie, natomiast kolory dopełniające się znajdą się po przeciwnych stronach koła. Schematy kolorów dopełniających się stosowane są w prezentacjach ze względu na duży kontrast. W aplikacjach i stronach www wykorzystuje się częściej schematy monochromatyczne lub kolory zbliżone, które mniej męczą wzrok.



**Rysunek 8:** Aplikacja do generowania schematów kolorów - <http://kuler.adobe.com>

Jeżeli potrzebujemy więcej kolorów można wykorzystać odcienie lub przejścia między kolorami podstawowymi. Do wygenerowania przejść między kolorami można skorzystać z aplikacji umieszczonej na stronie <http://websitetips.com/colortools/colorblender/>.

Kolorów należy używać ostrożnie, a przede wszystkim konsekwentnie. Najlepiej jest najpierw zaprojektować układ interfejsu a dopiero na koniec wprowadzić kolory, pamiętając o

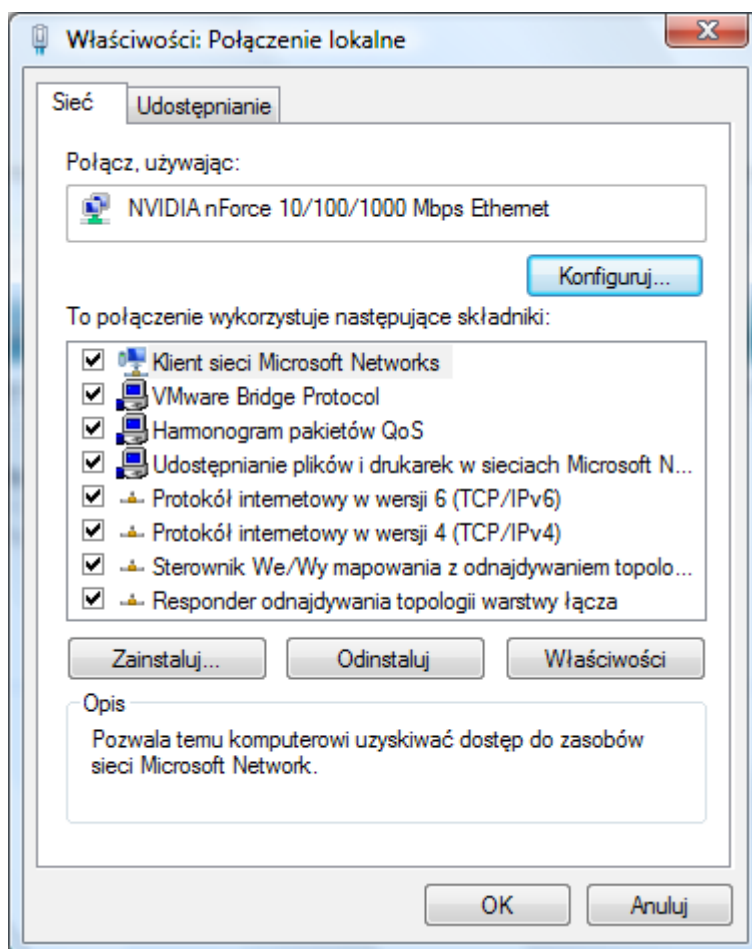
tym, że ich głównym celem jest podkreślenie wagi pewnych elementów interfejsu.

W interfejsie zaleca się stosowanie czcionek bezszeryfowych (np. Arial w przeciwieństwie do Times New Roman), ponieważ są one bardziej czytelne. Trudno wprowadzić jakieś innowacje do czcionek proponowanych przez system. Warto jednak pamiętać, że szczególnie w przypadku czcionek bezszeryfowych litery duże „I” i małe „l” są nieodróżnialne, podobnie jak cyfra „1” i litera mała „l” dla czcionek szeryfowych. Warto także zastanowić się jakiej czcionki użyć we wszelkiego rodzaju formularzach proszących użytkownika o dane. Dobrym rozwiązaniem może być wykorzystanie czcionki o stałym odstępach (ang. monospace) np. Courier. Poprawiają one czytelność i ułatwiają zaznaczanie oraz poprawianie fragmentów tekstu, w odróżnieniu od czcionek o zmiennej szerokości znaków, gdzie zaznaczenie litery „i” wymaga precyzyjnego operowania myszką.

## ***Układ komponentów***

### **Okno dialogowe**

Rozkład komponentów w oknie dialogowym nie musi być symetryczny. Powinno się jednak zwrócić uwagę na to, żeby komponenty rozkładały się równomiernie wzdłuż głównej przekątnej okna (od lewego-górnego do prawego-dolnego rogu). Dzięki temu wzrok w naturalny sposób podąża za najważniejszymi elementami okna, co ma związek z przyzwyczajeniem do czytania od lewej do prawej i z góry na dół (rys. 9).

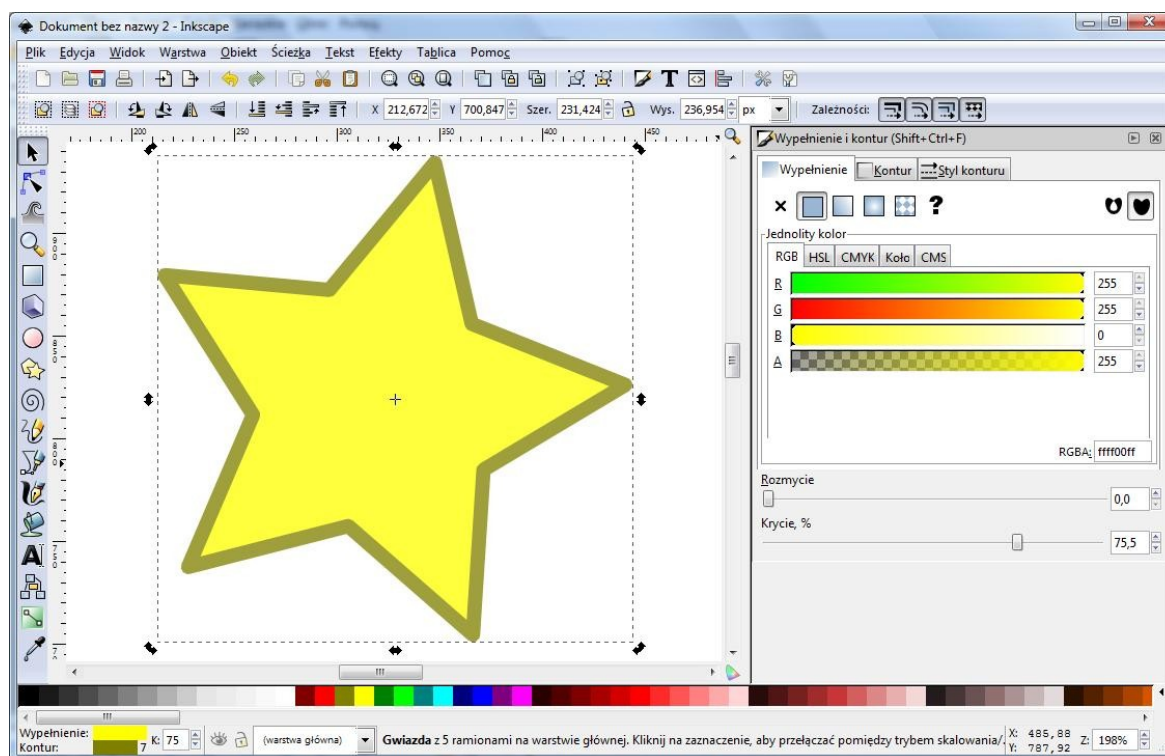


**Rysunek 9:** Usytuowanie komponentów wzdłuż głównej przekątnej okna

Projektując okna dialogowe, a w szczególności okna konfiguracji czy kreatorów, pamiętajmy, że użytkownik może skupić się tylko na kilku informacjach naraz. Dlatego opcje należy podzielić tematycznie korzystając na przykład z zakładek, lub w przypadku kreatorów tworząc kolejne kroki.

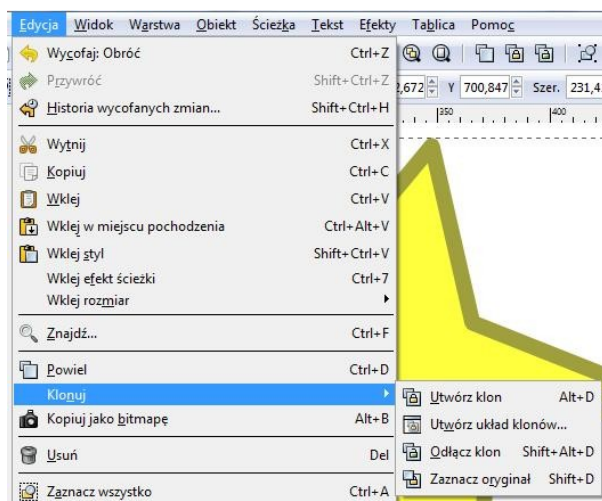
### Główne okno programu

Okno aplikacji powinno zostać zaprojektowane tak, aby użytkownik nie miał wątpliwości co do tego, gdzie znajduje się główny obszar roboczy. W większości przypadków zajmie on centralną pozycję i większą część obszaru okna. Dla obszaru roboczego często stosowany jest inny kolor, co ma na celu wyróżnienie go spośród pozostałych elementów interfejsu.



**Rysunek 10:** Typowy układ głównego okna aplikacji – program do grafiki wektorowej Inkscape

Klasycznymi elementami każdej bardziej rozbudowanej aplikacji są menu główne i menu podręczne. W menu głównym powinny się znaleźć wszystkie funkcje programu. Ich nazwy powinny być krótkie, ale znaczące, co pozwoli użytkownikowi dotrzeć do szukanej opcji bez czytania podręcznika. Menu nie powinno składać się z więcej niż dwóch, trzech poziomów, a opcje które w danym momencie nie mogą być uruchomione powinny być nieaktywne.



**Rysunek 11:** Menu główne powinno mieć maksymalnie 2-3 poziomy

Równie typowy jak menu główne jest pasek narzędziowy. Na pasku narzędziowym umieszczamy funkcje, które są najczęściej stosowane. Nietrawnym zadaniem jest zaprojektowanie odpowiednich ikon. O ile stosunkowo łatwo jest zaprojektować ikonę przedstawiającą rzecz, o tyle trudniejsze będzie zaprojektowanie ikony, która ma opisywać jakąś czynność. Do zaprojektowania ikony można wykorzystać jeden z darmowych programów np. Icon Suite. Można też poszukać odpowiednich darmowych ikon na specjalnych serwisach np. <http://www.freeiconsweb.com/>, których jednak w większości nie można wykorzystać do celów komercyjnych.

Paski narzędziowe umieszczane są też często po bokach obszaru roboczego, co jest popularne w programach graficznych. Z boku obszaru roboczego zostaje zazwyczaj wystarczająco miejsca na umieszczenie dodatkowego panelu zawierającego na przykład właściwości zaznaczonego obiektu lub opcje aktualnie wywołanej funkcji.

Typowe okno aplikacji zazwyczaj zwieńczone jest u dołu paskiem stanu. Czas poświęcony na zaprojektowanie tego słabo widocznego elementu GUI z pewnością docenią zaawansowani użytkownicy, dobrze zaznajomieni z aplikacją, którzy oczekują szybkiej informacji o stanie programu. Pasek stanu jest dobrym miejscem do informowania użytkownika o procesach zachodzących w tle na przykład automatycznym zapisywaniu do pliku.