POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Kamil Gałek

Nr albumu 15029

Elastyczne zarządzanie oknami interfejsu użytkownika w systemie operacyjnym

Praca magisterska napisana

pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, wrzesień 2018

Streszczenie

Niniejsza praca jest opisem procesu tworzenia aplikacji zarządzającej układem okien programów w systemach rodziny GNU/Linux opartego o protokół Wayland. W pracy został zawarty opis różnic między serwerem X.org a protokołem Wayland oraz opis popularnych programów zarządzających przestrzenią pulpitu komputerowego.

Następnie opisane zostały trzy programy, opracowane na podstawie założeń przedstawionych w poprzednich rozdziałach. Zakończenie stanowi wnioski autora, przedstawia wady oraz zalety opracowanego rozwiązania oraz opis możliwości rozwoju prototypów.

Słowa kluczowe: Manager okien, GNU/Linux, X.org, Wayland, RPC

Spis treści

1 Wstęp

Już od samego początku istnienia komputerów osobistych najważniejszymi cechami programów były ich intuicyjność w użytkowaniu oraz użyteczność. W taki właśnie sposób programy w trybie tekstowym zaczynały zyskiwać na początku proste elementy graficzne wykonane za pomocą znaków alfabetu. Następnie do programów zaczęto dodawać bitmapy. Wraz z rozwojem technologii linię poleceń udało się zastąpić skuteczniejszymi oraz bardziej wygodnymi sposobami obsługi komputera. Podróż ta zatrzymała się na wprowadzeniu okien programów. Koncepcji, która w założeniu przewidywała, że każdy uruchomiony przez użytkownika program będzie wyświetlał swoją graficzną zawartość w ograniczonej ramce na ekranie. Pozwoliło to na bardzo prostą i intuicyjną pracę z wieloma programami jednocześnie na ograniczonym obszarze ówczesnych monitorów.

Od dłuższego czasu pomysł ten całkowicie dominuje na komputerach osobistych. Z biegiem lat jest ciągle usprawniany. Dodawane są nowe mechanizmy poprawiające komfort pracy z zainstalowanymi programami. System Windows firmy Microsoft, w chwili obecnej najpopularniejszy wśród użytkowników[1], zawdzięcza koncepcji okien nawet nazwę.

1.1 Cel pracy

Praca ma na celu stworzenie propozycji łatwo konfigurowalnego programu zarządzania oknami dla systemu GNU/Linux współpracującego z protokołem Wayland. Produkt ma udostępniać możliwość rozszerzania jego funkcji poprzez integrację z innymi programami. Celami pobocznymi pracy są:

- analiza i porównanie popularnych wśród użytkowników programów o podobnym przeznaczeniu, co umożliwi podjęcie decyzji odnośnie implementacji niektórych funkcji
- dostarczenie kodu źródłowego prototypu spełniającego najnowsze zalecenia odnośnie dobrych praktyk w programowaniu
- przedstawienie możliwości rozwoju produktu pracy
- wytworzenie działającego prototypu programu zarządzającego oknami
- wytworzenie działającego prototypu programu komunikującego się z programem zarządzającym oknami

1.2 Rozwiązania przyjęte w pracy

Implementacja prototypu została przeprowadzona z użyciem tzw. Modern C++, co oznacza, że użyte zostały praktyki zaproponowane w wersjach języka C++11 i wyższych. W celu zapewnienia możliwości integracji z innymi programami zaimplementowany został serwer RPC.

1.3 Rezultat pracy

Rezultatem pracy jest wyspecyfikowanie najważniejszych założeń programu zarządzającego oknami komunikującego się z użyciem protokołu Wayland. Następstwem tych wymagań jest wytworzony prototyp konfigurowalnego managera okien wraz z programem pozwalającym na zdalne sterowanie funkcji managera okien.

1.4 Organizacja pracy

Praca została podzielona na 7 rozdziałów. Drugi rozdział przedstawia stan sztuki, tak więc istniejące rozwiązania dostępne dla użytkowników. Opisy w nim zawarte dotyczą charakterystyki serwerów wyświetlania jak również popularnych wśród użytkowników managerów okien.

Trzeci rozdział pracy stanowi propozycję programu.

Następny rozdział skupia się wokół narzędzi i technologii użytych do implementacji działającego prototypu programu zaproponowanego w rozdziale trzecim.

Rozdział piąty jest projektem rozwiązania. Opisane zostały w nim tematy takie jak model okien czy realizacja serwera RPC. Rozdział ten zakończony jest przedstawieniem szczegółów implementacji.

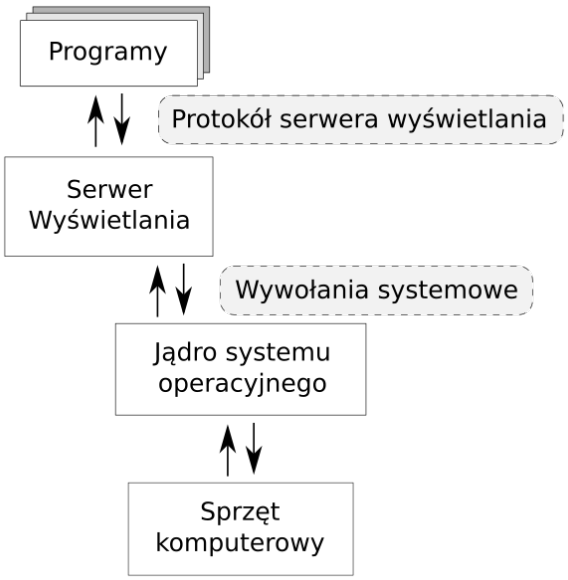
Przedostatni rozdział ma za zadanie przedstawić wyniki wybranych testów. Następujący po nim, ostatni rozdział, jest krótkim podsumowaniem pracy. Szczególna uwaga w tym rozdziale została skupiona wokół planów rozwoju projektu.

2 Istniejące rozwiązania

W tym rozdziale opisany został serwer wyświetlania X.org wraz z jego podstawowym protokołem oraz protokół Wayland. Między innymi poruszony został projekt architektury obu programów. Opis jest zakończony, krótkim porównaniem najważniejszych cech, zalet oraz wad. Następnie omówione zostały rodzaje programów zarządzających oknami oraz wybrane środowiska graficzne dla systemów rodziny GNU/Linux.

2.1 Serwery wyświetlania

Serwerem wyświetlania nazywa się programy wchodzące zazwyczaj w skład systemu operacyjnego. Ich głównym zadaniem jest pośredniczenie w komunikacji między innymi programami i jądrem systemu, który to kontaktuje się bezpośrednio ze sprzętem. Taki przepływ informacji przedstawiony jest na rysunku 1. Pośrednictwo odbywa się w uregulowany sposób, za pomocą odpowiedniego protokołu. Jest on niezależny od innych protokołów systemu operacyjnego.



Rysunek 1: Komunikacja między programami oraz serwerem wyświetlania i serwerem wyświetlania a jądrem systemu operacyjnego. Materiały własne.

Na chwilę obecną najbardziej popularny jest serwer X.org, aczkolwiek systematycznie protokół Wayland zwiększa swoje wpływy. Przez jakiś czas firma Canonical próbowała stworzyć też, konkurencyjny do Waylanda, serwer wyświetlania Mir. Miał być on przeznaczony dla popularnej

dystrybucji Ubuntu. Niestety zarząd firmy piątego kwietnia 2017 roku opublikował decyzję o porzuceniu tego projektu[5]. Wayland na chwilę obecną jest ciągle w fazie rozwoju i tylko nieliczne dystrybucje zdecydowały się na ustanowienie go domyślnym serwerem wyświetlania.

2.1.1 Serwer X.org

Pierwsza wersja serwera X.org została wydana w 2004 roku przez zespół z amerykańskiej uczelni MIT (ang. Massachusetts Institute of Technology) jako odgałęzienie projektu XFree86[6]. Co ciekawe wielu wiodących programistów pracujących nad pierwowzorem porzuciła ten projekt i przeszła do pracy nad jego odgałęzieniem jeszcze tego samego roku. Pierwsza stabilna wersja programu została opublikowana pod koniec 2005 roku.

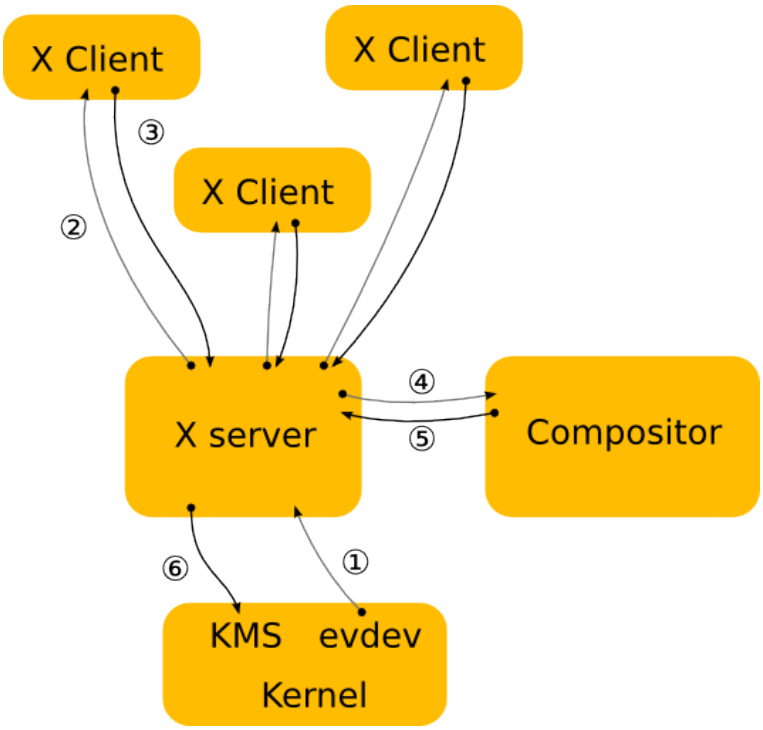
X.org jest gotowym rozwiązaniem, które dostarcza wcześniej opisane funkcje serwera wyświetlania. Jak każdy serwer pozwala on na komunikację z innymi programami, aczkolwiek nie powinny one ingerować w jego wewnętrzne mechanizmy. Jako największą wadę serwera X.org uznaje się jego wiek. Implementacja mimo wielkiemu nakładowi pracy ciągle posiada wiele przestarzałych konstrukcji utrudniających jego rozwój. Dodatkowo spora część modułów tego programu nie jest modularna, przez co utrzymanie jest wyjątkowo trudne.

Należy pamiętać, że w przypadku X.org, program zarządzający oknami jest osobnym bytem. Nie jest w żadnym wypadku częścią serwera wyświetlania, aczkolwiek potrafi się z nim komunikować. Na chwilę obecną takie rozwiązanie uważane jest za niepotrzebnie skomplikowane i przestarzałe.

Architektura klient-serwer

X.org zorganizowany jest w klasycznej architekturze klient-serwer. Oznacza to, że informacje pochodzące z jądra systemu przetworzone przez serwer mogą trafić do jednego z zarejestrowanych klientów. Taki przepływ informacji widać na rysunku 2. Klientami są programy, które niekoniecznie posiadają graficzny interfejs, aczkolwiek każdy z klientów może zarządzać od serwera o powołanie nowego okna.

Rysowaniem okna w X.org zazwyczaj zajmuje się specjalny moduł zwany kompozytorem. Jednocześnie jest też on odpowiedzialny za animacje oraz inne efekty graficzne widoczne na ekranie. Taka konstrukcja ma niestety spore wady. Przepływ informacji między klientem a mechanizmem rysowania nie jest bezpośredni. Z uwagi na to, że informacje o identyfikatorze i innych właściwościach okna przechowuje kompozytor serwer nie posiada informacji, do którego z klientów ma przesłać od-



Rysunek 2: Przykładowa obsługa informacji o wciśniętym klawiszu w serwerze X.org. Źródło: [7]

powiedź. Komunikacja między klientami gdy jeden z programów wysyła żądanie zmiany graficznego elementu drugiego programów jest przez to jeszcze bardziej skomplikowana.

Kolejną dużą wadą tego programu jest też fakt istnienia mechanizmu minimalnej wspieranej wersji protokołu. Każdy klient posiada tylko informację o minimalnej wersji serwera, z która może współpracować. Niestety późniejsze wersje mogą nie być kompatybilne z wersją, dla której klient był początkowo pisany. Zazwyczaj powoduje to przepływ nadmiarowych informacji między klientami a serwerem, co może być potencjalnie powodem błędów[8].

Podstawowy protokół

Podstawowa implementacja tego serwera wyświetlania używa bazowego protokołu X (oryginał ang. X Window System core protocol)[10]. Przewiduje on tylko cztery rodzaje informacji, których przesłanie jest wykonywane równolegle za pomocą asynchronicznych mechanizmów[9]. Te informacje to:

- Żądania (ang. Requests) - informacja przesyłana przez klienta do serwera z "prośbą" o wykonanie danej akcji. Wielkość pakietu informacji nie jest z góry określona i może się zmieniać.

- Odpowiedzi (ang. Replies) - informacje przesyłane przez serwer tylko i wyłącznie jako poprawny wynik działania akcji wywołanej w żądaniu. Podobnie jak w przypadku żądań, wielkość pakietu informacji nie jest z góry określona i może się zmieniać.
- Zdarzenia (ang. Events) - informacje o akcjach użytkownika przesyłane przez serwer do klienta. Zdarzenia mają stałą długość 32 bajtów.
- Błędy (ang. Errors) - informacje przesyłane przez serwer do klienta jeśli podczas wykonywania żądanej informacji wystąpią błędy. Podobnie jak zdarzenia, błędy mają stałą długość 32 bajtów.

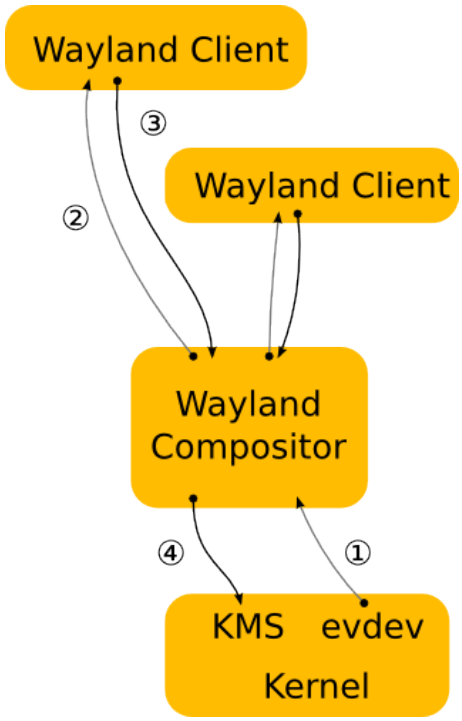
Protokół wymusza to, że nawiązanie połączenia jest obowiązkiem klienta, aczkolwiek, co ciekawe, nie opisuje sposobu w jaki klient jest uruchomiony. Do zdań klienta również należy przesłanie pierwszego pakietu informacji z wersją protokołu, której ma zamiar używać oraz z niezbędnymi informacjami pozwalającymi zidentyfikować danego klienta. W odpowiedzi na to żądanie serwer może zaakceptować podane informacje, co kończy procedurę nawiązywania połączenia. Pakiet może być również odrzucony, oznacza to odrzucenie próby połączenia z serwerem.

2.1.2 Wayland

Po wielu latach rozwoju X.org dostrzeżono potrzebę skonstruowania innego mechanizmu, który uprościłby proces wyświetlania elementów graficznych w systemach GNU/Linux. Potrzebę tą zaspokoił asynchroniczny, zorientowany obiektowo protokół Wayland. Rozwój tego protokołu jest ściśle związany z rozwojem biblioteki referencyjnej Weston w pełni implementującej jego założenia. Projekt rozpoczęto w 2008 roku w firmie RedHat. Jak łatwo zauważyć bardzo szybko, bo po niecałych 4 latach serwer X.org uznano za warty wymiany. Głównym założeniem tego projektu jest opracowanie systemu, który byłby pozbawiony jakichkolwiek wad. Sam autor upubliczniając swój projekt określił swój cel w następujący sposób: "Każda klatka będzie doskonała, przez co rozumiem, że aplikacje będą w stanie kontrolować renderowanie wystarczająco (dobrze), że nigdy nie zobaczymy przycinania się, lagów, przerysowywania, lub migotania"[11].

Wayland znacząco różni się od wcześniej opisanego serwera wyświetlania. Po pierwsze program, którego zadaniem jest zarządzanie oknami i przestrzenią pulpitu jest jednocześnie odpowiedzialny za komunikację z jądrem systemu i efekty graficzne. Jest więc tak naprawdę połączeniem kilku mechanizmów występujących w X.org jako osobne moduły czy nawet programy.

Architektura klient-kompozytor



Rysunek 3: Przepływ informacji między kompozytorem a klientami wg protokołu Wayland. Źródło: [12]

W przeciwieństwie do X.org, protokół Wayland nie zakłada istnienia osobnego serwera, będącego pośrednikiem między kompozytorem a programami klientów. W przypadku tego protokołu, każdy z klientów komunikuje się bezpośrednio z kompozytorem, który to zaś komunikuje się z jądrem systemu. Rysunek 3 przedstawia przepływ informacji w takiej architekturze. Można więc powiedzieć, że część funkcji serwera przejmuje kompozytor. Takie rozwiązanie pozwoliło wyeliminować problem braku informacji o parametrach okna występujący w X serwerze.

Kolejną różnicą jest to, że okno samo zleca rysowanie zmian na ekranie, kompozytor dostaje wyłącznie informację zwrotną odnośnie zmian[13]. Zbędny ruch informacji jest ograniczony do minimum dzięki eliminacji całego pośrednictwa występującego w X.org.

Z uwagi na to, że kompozytor komunikuje się z wieloma klientami, co do złudzenia przypomina architekturę klient-serwer, często się mówi o całym rozwiązaniu jako o "serwerze" Wayland. Społeczność skupiona wokół tego projektu nie uznaje tego sformułowania za błędne i zezwala na używanie go zamiennie z kompozytorem.

Kompozytor

Protokół określa kompozytor jako obiekt globalny zgodny ze wzorcem singleton. Oznacza to, że jądro i klienci przesyłają dane tylko do jednego kompozytora na raz. Gdyby próbować szukać analogii z serwerem X, kompozytor spełnia funkcje serwera wyświetlania, managera okien i programu kompozycji. Głównymi funkcjami tego programu są:

- zarządzanie właściwościami okien
- odbieranie zdarzeń systemowych z jądra systemu i przekazywanie ich do klientów
- komunikację z programami klientów
- pośredniczenie w rysowaniu okien

Protokół wyróżnia dokładnie trzy rodzaje kompozytorów: systemowy, sesji i osadzający. Kompozytor systemowy może być odpowiedzialny za zadania takie jak uruchamianie systemu, zarządzanie kompozytorami sesji czy przełączaniem między zalogowanymi użytkownikami. Kompozytory sesji dostarczają środowisko graficzne dla każdego z użytkowników. Ostatni z kompozytorów jest całkowicie odmienny od pozostałych. Jego zadaniem jest udostępnienie możliwości osadzania sesji X.org lub innych rozwiązań tego typu w sesji serwera zgodnego z protokołem Wayland. Wyodrębnienie takiego kompozytora pozwala na uruchomienie programów nieprzystosowanych do pracy z nowym protokołem, przez co migracja użytkowników powinna stanowić mniej problemów.

2.1.3 Porównanie

Głównymi różnicami między dwoma przedstawionymi rozwiązaniami są ich budowa i założenia. Serwer X dostarcza kompleksową obsługę zdarzeń z jądra systemu jednocześnie wymuszając na programiście dostosowanie się do wewnętrznych mechanizmów programu. Jego monolityczna konstrukcja była uważana za przestarzałą już 4 lata po wydaniu pierwszej stabilnej wersji. Wtedy właśnie zaczęto pracę nad Waylandem.

Od samego początku projektowi Wayland stawiano bardzo ambitne cele. Miał być rozwiązaniem pozbawionym wad. Protokół obarcza programistę dużo większa odpowiedzialnością, co oczywiście skutkuje tym, że osoba pisząca kod dostaje o wiele większe możliwości podczas pracy. Wayland w założeniu pozwala na wymianę informacji okien bezpośrednio z kompozytorem, co bardzo upraszcza organizację przepływu komunikatów między oknami a pozostałymi komponentami.

Podsumowując, nowe podejście ma wiele zalet, które jednocześnie dla niektórych mogą być wadami. Bez względu na to, nieustanny rozwój Waylanda pozwala tworzyć to coraz lepsze rozwiązania dostępne dla coraz to szerszego grona użytkowników.

2.2 Zarządzanie oknami

Manager okien jest to program zarządzający rozmieszczeniem, zachowaniem oraz rysowaniem okien w środowisku graficznym. Jego zadaniem jest reagowanie na zdarzenia zmiany wielkości okna danego procesu czy zmiany położenia okna. Program takiego typu zazwyczaj jest częścią kompletnego środowiska graficznego, składających się zazwyczaj z wielu programów, których funkcje mają pozwolić na płynną pracę użytkownika.

Pomijając manager okien systemu Microsoft Windows, którego kod nie jest publiczny, wiele takich programów jest dostępnych publicznie. Dla systemów rodziny GNU/Linux dostępnych rozwiązań istnieje bardzo wiele. Często takie projekty prowadzone są przez jednego programistę i posiadają bardzo wąską grupę użytkowników. Stosunkowo dobrze rozwiniętą listę managerów okien można znaleźć między innymi na stronach dokumentacji dystrybucji ArchLinux [18].

W poniższym rozdziale przedstawiono zestawienie kilku najbardziej popularnych rozwiązań, takich jak Gnome 3, KDE, Openbox, AwesmeWM, i3wm oraz DWM. Wymienione programy były porównywane z uwzględnieniem takich cech jak typ zarządzania oknami, możliwości konfiguracji oraz rozszerzalnością. Została zwrócona uwaga też na specyficzne cechy każdego z rozwiązań. Ma to na celu pokazać różne sposoby podejścia do problemu zarządzania oknami.

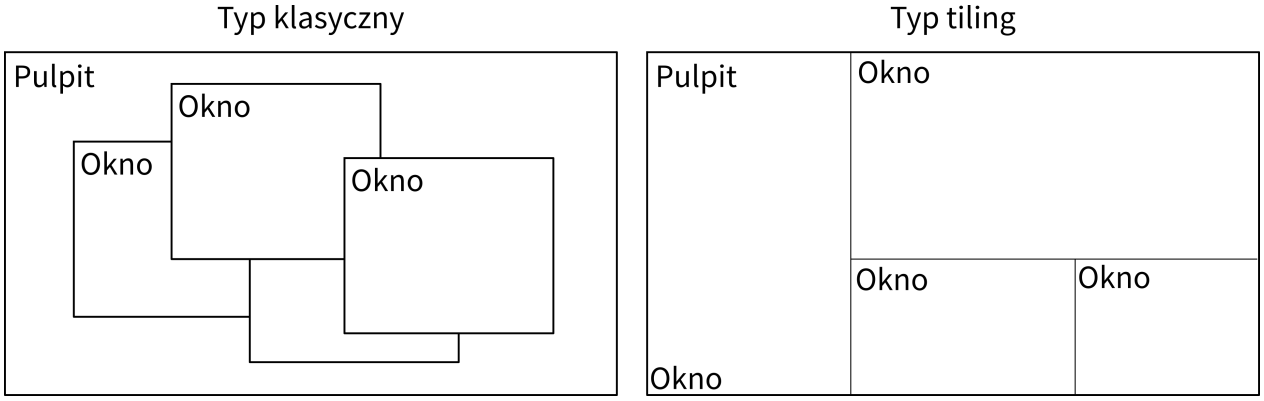
2.2.1 Opis dostępnych rodzajów managerów okien

Managery okien można podzielić wg typu na klasyczne oraz *tiling*. Klasyczne managery okien pozwalają użytkownikowi na przesuwanie okien za pomocą myszy. Rozmiar okna oraz jego umiejscowienie jest w pełni konfigurowalne przez osobę przed komputerem a dostępne okna nie muszą zajmować całej powierzchni pulpitu. W przypadku klasycznych managerów okien zmiana położenia lub rozmiaru okna zazwyczaj nie powoduje zmian geometrii[1] innych okien. Ważne jest to, że w takim podejściu okna mogą nachodzić na siebie (okno może znajdować się nad lub pod innym oknem). Managery okien klasycznego typu obecnie są najbardziej popularne między innymi dzięki ich implementacji w takich systemach jak Microsoft Windows oraz Apple MacOS.

Zarządzanie *tiling* charakteryzuje zupełnie inne podejście do problemu. W takim rozwiązaniu obszar całego pulpitu zawsze jest zapełniony oknami. Tworzą one pewnego rodzaju mozaikę, w której żadne okno nie może być nad lub pod innym oknem. Osoba przed komputerem ciągle może zmieniać rozmiar i położenie okna, ale każda zmiana powiązana jest ze zmiana geometrii pozostałych

[1] Geometrią okna nazywamy jego położenie oraz rozmiar

okien. Dzięki takim zmianom obszar ekranu jest zawsze w pełni zapełniony. Graficzne porównanie układu okien w obu typach przedstawiono na rysunku 4.



Rysunek 4: Graficzne przedstawienie managerów okien typy klasycznego oraz *tiling*. Źródło: materiały własne

Ograniczenie, jakim jest zawsze w pełni zapełniony obszar pulpitu powoduje to, że obsługa managerów typy *tiling* zazwyczaj jest trudniejsza i odbywa się głównie za pomocą klawiatury, a nie myszy, jak w przypadku klasycznego zarządzania oknami. Jednak dla osób takich jak programiści, których praca opiera się głównie na pisaniu na klawiaturze może być wygodniejsze i szybsze ponieważ eliminuje zbędne ruchy ku myszy komputerowej odrywające ręce od klawiatury.

Oba typy managerów można porównać do biurka, na którym leżą dokumenty. Podczas pracy pracownik musi korzystać z kilku dokumentów na przemian. W klasycznym podejściu dokumenty leżą w stercie na biurku. Odnalezienie tego potrzebnego w obecnej chwili wiąże się z przełożeniem innych dokumentów pod ten wskazany dokument. Gdy pracownik pracuje zgodnie z podejściem *tiling*, dokumenty leża porozkładane na całej powierzchni biurka. Odnalezienie odpowiedniego dokumentu w takim ułożeniu jest dużo prostsze. Ta analogia pozwala dostrzec kolejny problem podejścia *tiling*. Metaforyczne biurko ma określone, skończone rozmiary i przy dużej liczbie dokumentów nie wszystkie się na nim zmieszczą. Kolejnym problemem wynikającym z analogii jest trudność zarządzania dokumentami w drugim przypadku. Gdy pracownik otrzymuje nowy dokument do umieszczenia na biurku w klasycznym podejściu trafia on na stos i nie jest wykonywana żadna inna akcja. Niestety w drugim przypadku wstawienie nowego dokumentu wymaga przebudowy układu części biurka.

Problem przebudowy układu okien oczywiście w dzisiejszych czasach jest niewielki, ponieważ przeciętny sprzęt dostępny w obecnych czasach potrafi wykonać potrzebne obliczenia w niewielkich ułamkach sekundy. Niewystarczająca przestrzeń pulpitu również została rozwiązana z użyciem me-

chanizmu “pulpitów wirtualnych“. Jest to mechanizm pozwalający nadać wszystkim oknom pulpitu etykietkę z nazwą i wykonywanie operacji ukrycia okien na wszystkich oknach o danej etykiecie. Dzięki takiemu rozwiązaniu w łatwy sposób można zasymulować posiadanie wielu fizycznych ekranów. Z użyciem jednego skrótu klawiszowego możemy ukryć wszystkie okna z danego pulpitu wirtualnego i wyświetlić okna z innego, dokładnie tak jakby osoba pracująca przy komputerze przeniosła wzrok na drugi monitor podpięty do komputera. Analogicznie do biurka z dokumentami, rozwiązaniem problemu z niewystarczającym rozmiarem biurka jest dokupienie kolejnego biurka, na które zawsze można przenieść wzrok. Pulpity wirtualne nie są nowym pomysłem. Od wielu lat w managerach okien dla GNU/Linux znajdują się takie rozwiązania. W systemach Microsoft Windows pulpity wirtualne zostały wprowadzone dopiero w Windows 10 [19].

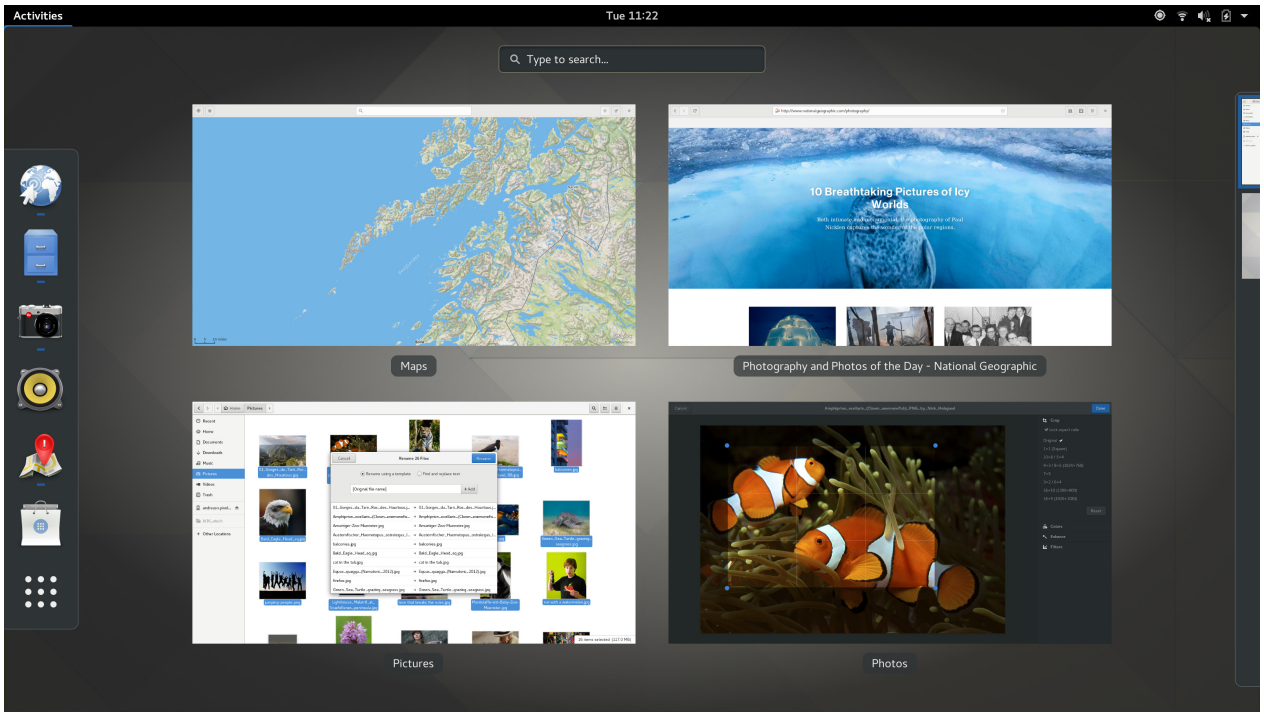
Mimo dominującej popularności managerów okien klasycznego typu, managery typu *tiling* mają sporą grupę zwolenników. Grupa dyskusyjna serwisu Reddit, skupiająca osoby zainteresowane rozwojem i użytecznością managerów okien skupia ponad 116 tysięcy użytkowników[4]. Ogromna część z nich to użytkownicy managerów *tiling*. Można też spojrzeć na statystyki repozytoriów poszczególnych projektów. Tysiące użytkowników serwisu GitHub przyznało "gwiazdkę" programom takie jak i3wm czy AwesomeWM[2][3]. Niestety wspomniane programy nie współpracują z Waylandem. Co więcej nie poczynione zostały żadne kroki w celu stworzenia odpowiednich integracji.

2.2.2 Gnome 3

Gnome 3 jest pełnym środowiskiem graficznym dla użytkowników systemów GNU/Linux. Jest to pakiet kilkudziesięciu programów, które powinny wystarczyć przeciętnemu użytkownikowi do codziennej pracy z komputerem. Podobnie zorganizowany jest również system Microsoft Windows, gdzie dostajemy w pakiecie programy takie jak przeglądarka plików, przeglądarka internetowa, czytnik plików pdf czy prosty edytor tekstowy.

Pierwsza stabilna wersja tego środowiska została wydana 6 kwietnia 2011[21]. Utrzymana jest w minimalistycznym i intuicyjnym stylu. Dość oryginalnym elementem tego pakietu jest tzw. tryb przeglądu (ang. overview), przedstawiony na rysunku 5, który pozwala w graficzny sposób nawigować miedzy oknami oraz pulpitami wirtualnymi. Dodatkowo ten tryb posiada listę zainstalowanych programów w postaci siatki ikon oraz pasek szybkiego uruchamiania.

Programem zarządzającym oknami w tym pakiecie jest Gnome Shell, który formalnie jest rozszerzeniem managera okien Mutter. Zastąpił on manager Metacity używany w poprzedniej wersji pakietu Gnome. Mutter początkowo obsługiwał wyłącznie serwer wyświetlania X.org, na chwilę



Rysunek 5: Tryb przeglądu środowiska Gnome. Widać na nim cztery uruchomione programy oraz pasek szybkiego uruchamiania (po lewej stronie). Źródło: [20]

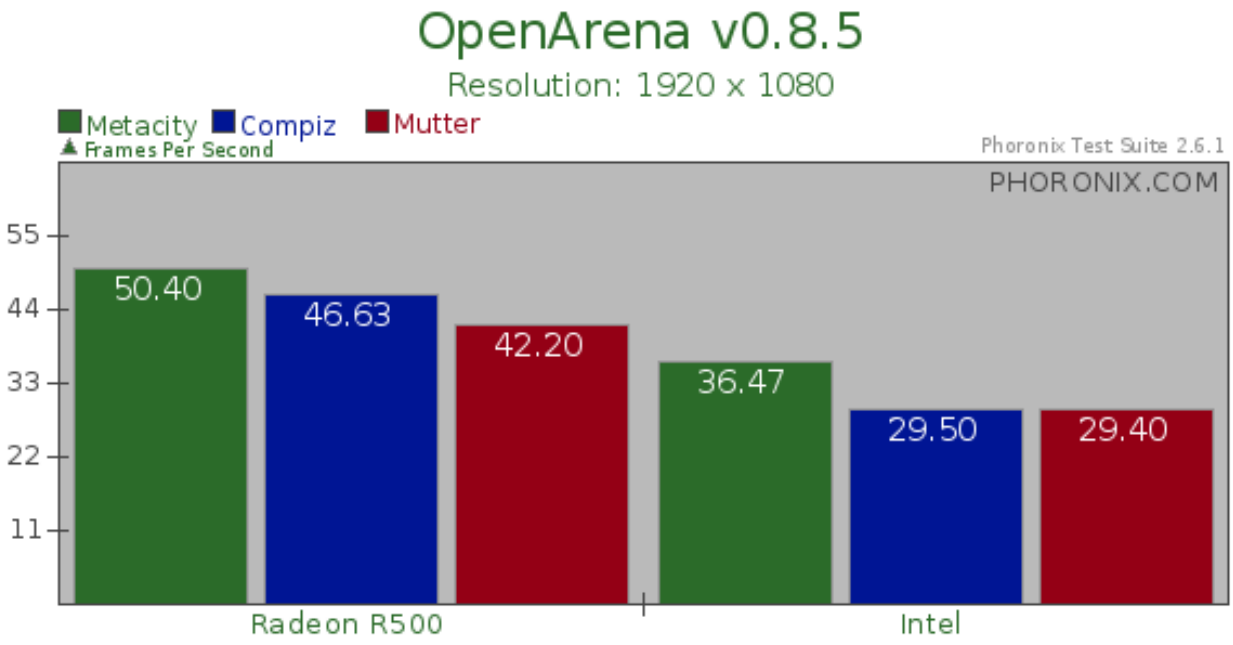
obecną używa do działania również Waylanda. Przejście z Metacity na Mutter umożliwiło między innymi używanie języka CSS, popularnego w technologiach internetowych, do określania niektórych właściwości elementów graficznych okien[22].

Niestety użyty w pakiecie Gnome 3 manager okien ma spore problemy z wydajnością, co jest szczególnie widoczne przy uruchomionych grach komputerowych. Portal Phoronix przeprowadził testy[23] z użyciem managerów Mutter, Metacity oraz Compiz, z których wynika, że spadek wydajności jest stosunkowo duży w porównaniu do poprzedniego managera okien. Wyniki testu przeprowadzonego przy włączonej grze OpenArena w wersji 0.8.5 przedstawiono na rysunku 6.

Konfiguracja Gnome Shell odbywa się z użyciem zewnętrznego programu Gnome Tweaks [24]. Pozwala ono na konfigurację czcionek wykorzystywanych przez środowisko, motywów okien, tapet oraz ikon. Pozwala ono też włączać różnego rodzaju rozszerzenia poprawiające wygląd lub zmieniające zachowanie managera okien[2].

Podsumowując, Gnome 3 podsiada duże możliwości rozszerzania środowiska w prosty sposób z użyciem gotowego repozytorium rozszerzeń. Sama konfiguracja jest nieco utrudniona, wymaga

[2]Pełną listę rozszerzeń można znaleźć pod adresem `https://extensions.gnome.org`



Rysunek 6: Porównanie wydajności managerów Metacity, Compiz oraz Mutter przy uruchomionej grze OpenArena. Jak zauważono Mutter wypada dużo słabiej od swojego poprzednika. Źródło: [23]

dodatkowego programu, który nie jest częścią pakietu. Wykorzystany manager okien Mutter ma problemy z wydajnością w niektórych zastosowaniach.

2.2.3 KDE

Podobnie jak Gnome 3, KDE to kompletne środowisko graficzne. Sama nazwa jest skrótem od "K Desktop Environment". KDE jest oparte głównie na bibliotekach Qt, co pozwoliło twórcom na szybkie tworzenie wieloplatformowych aplikacji, które działają również bez zainstalowanego KDE[25].

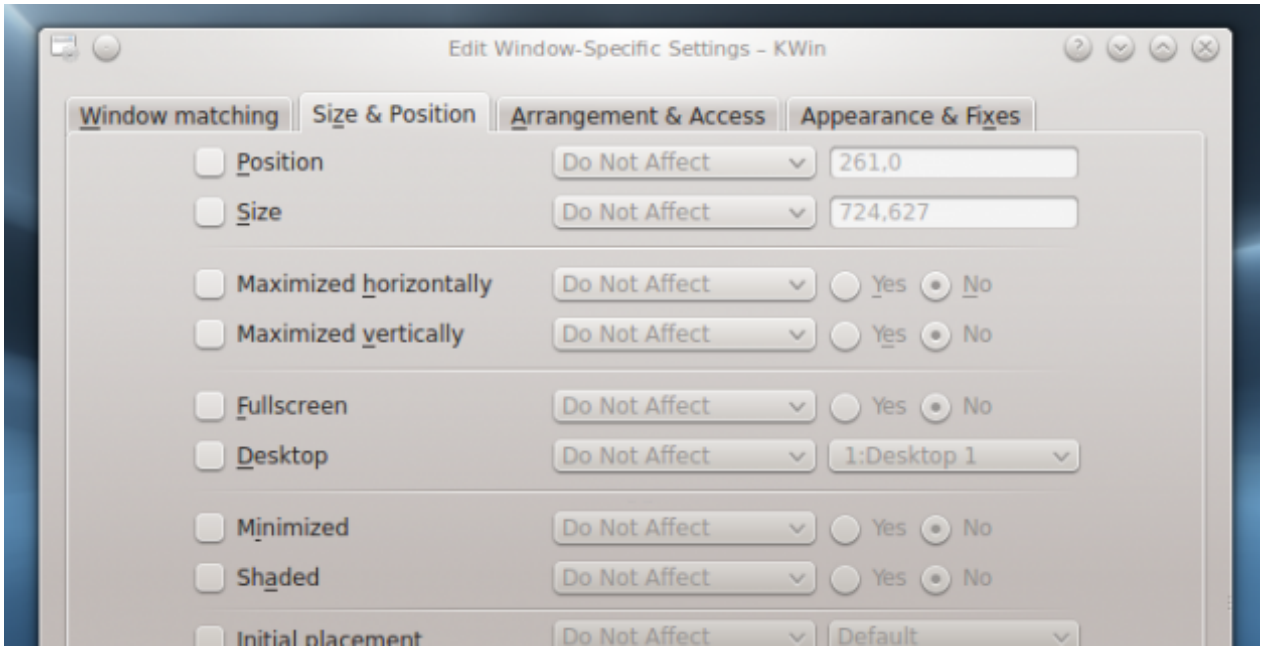
W przypadku KDE programem zarządzającym oknami jest Plazma (ang. Plasma). Elementem wyróżniającym Plazmę jest bardzo rozbudowany system Widgetów, które użytkownik może umieścić na swoim pulpicie. Widgety instaluje się przez specjalny instalator, którego obsługa jest stosunkowo prosta. Przez podobny program instaluje się również "motywy Plazmy", które zmieniają wygląd oraz kolorystykę pulpitu.

Konfiguracja środowiska odbywa się przez dostarczony w pakiecie konfigurator. Nie wymaga on dodatkowej instalacji programu jak w przypadku Gnome 3. Pozwala on skonfigurować wyżej wspomniane widgety i motywy jak również pobrać inny zestaw ikon czy zmienić czcionkę.

Obsługa KDE przypomina odrobinę obsługę systemu Microsoft Windows. Do dyspozycji jest

pulpit, na którym znajdować się mogą ikony programów. Pasek zadań umieszczony jest domyślnie na dole ekranu a menu programów przypomina "menu start". Interesująca częścią pakietu jest program KRunner, który pozwala szybko wyszukiwać zainstalowane programy oraz przeszukiwać posiadane pliki. Przy odpowiedniej konfiguracji potrafi nawet wyszukiwać informacje w internecie[26].

Managerem okien w tym pakiecie jest program KWin [27]. W pakiecie dostarczony jest również graficzny konfigurator tego managera. Pozwala on skonfigurować między innymi w jakim miejscu i jakiego rozmiaru mają być nowe okna. Rysunek 7 przedstawia GUI programu konfiguracyjnego KWin.



Rysunek 7: Panel konfiguracji managera okien KWin. Widoczna część ustawień pozwala zdefiniować reguły pozycji oraz geometrii okien niektórych programów. Źródło: [27]

Na chwilę obecną KWin obsługuje wyłącznie serwer X.org, prace nad obsługą Waylanda jeszcze trwają[28]. Podobnie jak Mutter, KWin powoduje spadki wydajności widoczne przy uruchomionych grach[29]. Jednak w przypadku KWin, spadki wydajności są mocno powiązane ze sterownikiem karty graficznej użytej podczas testów.

Tak więc, do zalet KDE z pewnością można zaliczyć duże możliwości konfiguracyjne. Istnieje też wiele rozszerzeń gotowych do zainstalowania przez dostarczone instalatory. Podobnie jak Mutter, manager okien KDE KWin miewa problemy z wydajnością w specyficznych sytuacjach.

2.2.4 Openbox

Openbox[30] jest managerem okien klasycznego typu. W odróżnieniu od KDE i Gnome 3 nie jest pakietem programów. Po instalacji i pierwszym uruchomieniu Openboxa otrzymuje się szary pulpit z kursorem na środku. Jeśli użytkownik chciałby mieć pasek zadań, musi go doinstalować osobno. Nawet graficzny konfigurator obconf[31] nie jest instalowany wraz z tym managerem. Konfiguracja tego środowiska znajduje się w kilku plikach XML, dzięki czemu ustawienia managera można zmieniać nawet za pomocą zwykłego edytora tekstowego. Openbox nie posiada żadnego systemu rozszerzeń.

Taka budowa programu jest celowa. Gnome 3 jak i KDE słabo sobie radzą z działaniem na starszych komputerach. Zajmują też dużo więcej miejsca na dysku przez to, że instalowany jest cały pakiet programów. Instalacja Openboxa zajmuje na dysku niecałe 2MiB[3]. Po uruchomieniu systemu z Openboxem zajęte jest 170Mb pamięci RAM, podczas gdy Gnome 3 wraz z systemem zajmuje ponad 700Mb pamięci komputera[4].

Dość ciekawym rozwiązaniem jest menu programów wyzwalane za pomocą kliknięcia drugim przyciskiem myszy na pulpicie. Menu nie jest dynamiczne, czyli po instalacji programu należy dodać odpowiednią pozycję w konfiguracji albo użyć zewnętrznego programu do wygenerowania odpowiedniego pliku konfiguracyjnego. Kolejną wada jest oczywiście sposób wyzwalania. Menu nie zostanie uruchomione jeśli którekolwiek z okien zajmuje całą powierzchnię pulpitu.

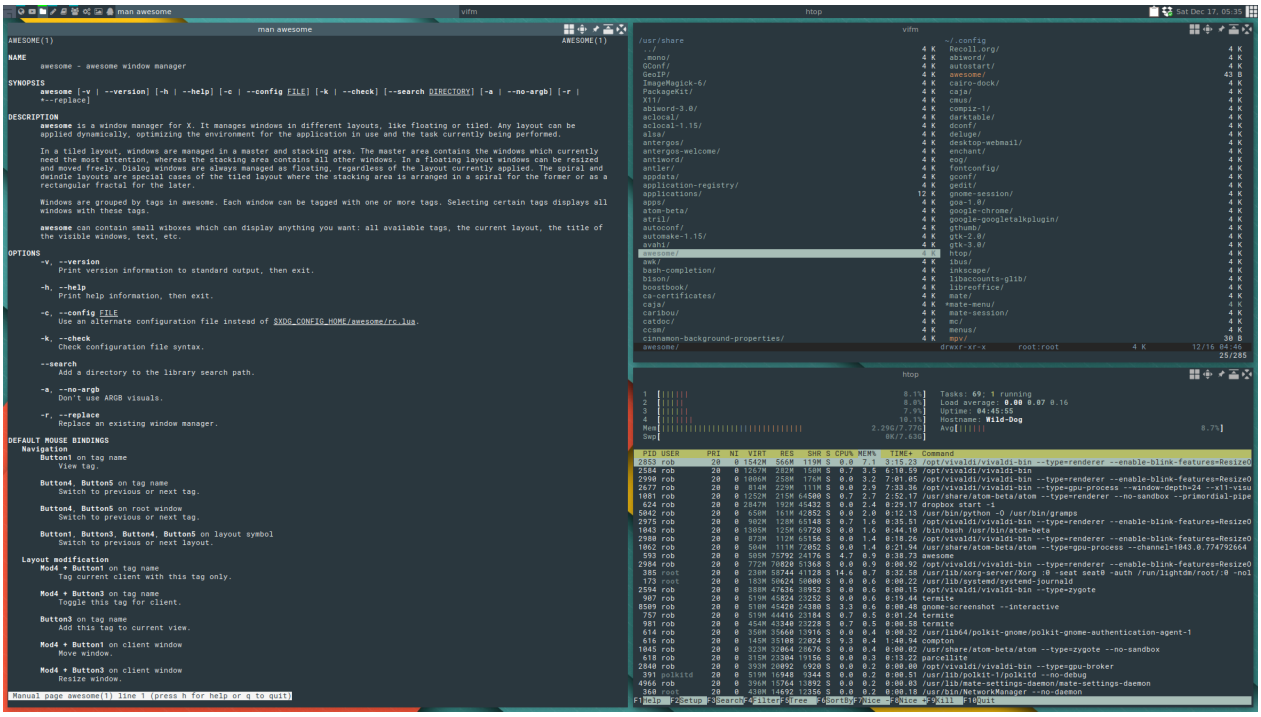
Openbox mimo braku systemu rozszerzeń ma swoje zalety w postaci niskich wymagań systemowych. Z tego powodu wymaga jednak więcej pracy podczas instalacji. Konfiguracja może być przeprowadzona z użyciem odpowiedniego programu lub przez edycję plików tekstowych.

2.2.5 AwesomeWM

Manager okien typu *tiling* AwesomeWM cechuje możliwość rozszerzania jego funkcji z użyciem języka Lua. W tym języku również przeprowadza się całą konfigurację. AwesomeWM jest tworzony od 2007 roku i jako pierwszy wykorzystywał bibliotekę XCB[32] pozwalającą na asynchroniczne zarządzanie serwerem wyświetlania X.org. Taki zabieg pozwolił na zwiększenie płynności operacji wykonywanych na ekranie, ponieważ żadna operacja nie blokuje działań użytkownika oraz innych procesów działających w tle. Pulpit AwesomeWM z trzema widocznymi oknami przedstawia rysunek 8.

[3]Instalacja dnia 27.01.2017 z repozytorium AUR

[4]Pomiary własne z użyciem Gnome 3 w wersji 3.26 oraz Openboxem pobranym z repozytorium AUR dnia 27.01.2017

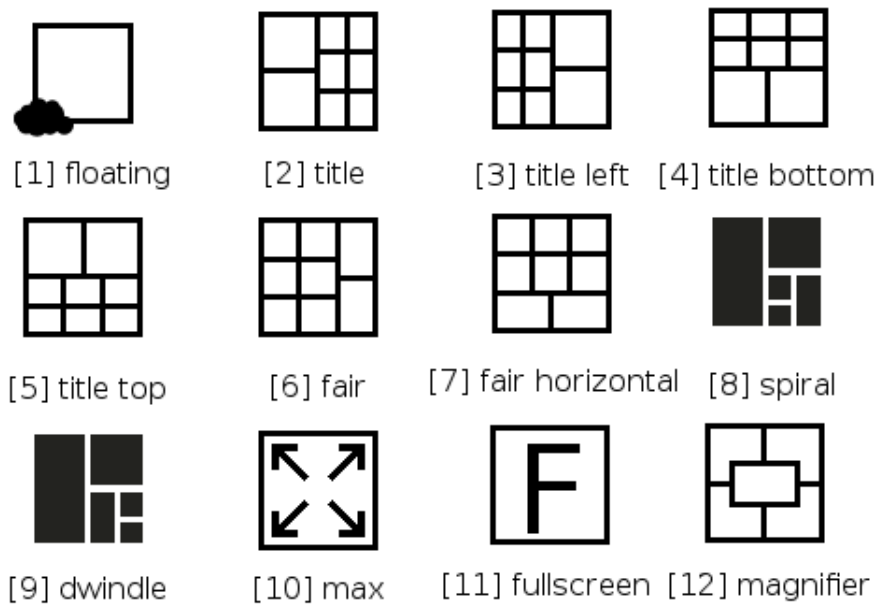


Rysunek 8: Pulpit managera okien AwesomeWM. Jak we wszystkich managerach typu *tiling* cała powierzchnia pulpitu jest zapełniona oknami oraz żadne z okien nie przesłania innego okna. Źródło: [32]

AwesomeWM nie posiada wcześniej opisanego mechanizmu pulpitów wirtualnych. Ta funkcja została zastąpiona etykietkami okien (ang. tags). Każde okno posiada przynajmniej jedną etykietę, użytkownik zamiast przełączać pulpity wirtualne przełącza wyświetlanie okien z danym tagiem. W przypadku gdy każde z okien ma tylko jedną etykietę działanie tego mechanizmu jest analogiczne do pulpitów wirtualnych. Gdy okno posiada więcej niż jedną etykietę, analogicznie to samo okno będzie wyświetlane na dwóch pulpitach wirtualnych.

Dzielenie ekranu na poszczególne okna przebiega z użyciem szablonów (ang. layouts). Szablon wyznacza dokładny algorytm operacji przy zdarzeniu pojawienia się nowego okna. Layout może być zmieniany dynamicznie z użyciem odpowiedniej kombinacji klawiszy, można też część domyślnych ustawień wyłączyć i korzystać tylko z jednego szablonu. Domyślne szablony wraz z ich ikonami przedstawia rysunek 9.

Twórcy AwesomeWM zdecydowali, że ich dzieło nie zostanie przystosowane do używania Way-



Rysunek 9: Domyślne szablony podziału ekranu w AwesomeWM. Każda z ikonek obrazuje sposób w jaki będą układane kolejne okna. Źródło: [33]

landa. Ich decyzja wynika między innymi ze sposobu implementacji programu[5].

Podsumowując, AwesomeWM posiada spore możliwości rozszerzania funkcji, aczkolwiek nie istnieje żadne repozytorium zawierające gotowe kawałki kodu Lua. Funkcje rozszerzania są raczej przeznaczone dla programistów, ponieważ wymagają napisanie odpowiedniego kodu. Konfiguracja odbywa się również z użyciem języka Lua, co jest sporym utrudnieniem dla zwykłych użytkowników.

2.2.6 i3wm

Ostatnio popularny manager okien i3wm to projekt rozpoczęty w 2009 roku mający na celu stworzenie programu zarządzającego oknami o wysokiej jakości kodu z dobrze udokumentowanymi mechanizmami wewnętrznymi. Do implementacji podobnie jak w przypadku AwesomeWM została wykorzystana biblioteka XCB, również z powodu zwiększenia płynności działania programu. O jego popularności świadczyć mogą statystyki repozytorium umieszczonego w portalu GitHub[2]. Posiada ponad 3400 gwiazdek, był forkowany ponad 380 razy.

[5]Pełne uzasadnienie decyzji można znaleźć w dyskusji pod adresem: `https://github.com/awesomeWM/awesome/issues/159`

Cechą wyróżniającą i3wm jest zorganizowanie modelu okien w strukturę drzewiastą. Liśćmi drzewa zawsze są obiekty poszczególnych okien. Każde okno znajduje się w elemencie drzewa zwanym kontenerem (ang. container). Kontener może zawierać w sobie również inne kontenery. Dzielenie ekranu zawsze odbywa się w ramach jednego kontenera, w efekcie czego po podziale w pierwotnym kontenerze znajdują się dwa mniejsze, każde z obiektem okna w środku[34]. Okna zawsze zajmują całą powierzchnię owlekającego je kontenera.

Kolejną dość rzadko spotykaną funkcją w managerach typu *tiling* jest tryb pływających okien (ang. floating mode). Pozwala on włączyć zarządzanie pojedynczymi oknami w sposób znany z klasycznych managerów okien. Okno w trybie pływającym znajduje się zawsze przed innymi oknami i nie podlega podziałom. Ta funkcja czyni z i3wm pewnego rodzaju hybrydę, która stara się łączyć oba typy managerów okien.

Konfiguracja przechowywana jest w postaci jednego pliku tekstowego, dzięki czemu jakiekolwiek zmiany można wykonać z użyciem dowolnego edytora tekstowego. Takie rozwiązanie niestety ma wadę, często pliki konfiguracyjne i3wm są długie, nieuporządkowane i trudne w analizie. Poza konfiguracja pulpitów wirtualnych, skrótów klawiszowych oraz programów startujących wraz z managerem, znajduje się też konfiguracja programu paska statusu i3bar, który jest domyślnie instalowany z managerem.

Możliwości rozszerzania programu zostały ograniczone do stworzenia interfejsu IPC, który pozwala na zdalne wykonywanie akcji managera oraz przechwytywanie zdarzeń. Tak więc każde rozszerzenie w rzeczywistości jest osobnym programem.

Podobne jak AwesomeWM, i3wm nie zostanie przystosowane do pracy z serwerem wyświetlania Wayland. Powstaje jednak inny manager okien, SwayWM[35], który w założeniach ma być zgodny z mechanizmami i konfiguracją opisaną w dokumentacji technicznej i3wm. Prace nad tym projektem są dość zaawansowane, niestety jednak SwayWM nie zachowuje wysokiej jakości kodu i dokumentacji zawartych w założeniach swojego pierwowzoru.

2.2.7 DWM

Ostatnim przedstawionym w tej pracy managerem okien jest DWM[36]. Jest to rozwiązanie typu *tiling*, którego cechą szczególną jest absolutny brak możliwości konfiguracji. Dostarczany jest bez żadnych dodatkowych narzędzi, użytkownik otrzymuje wyłącznie doskonałej jakości kod źródłowy programu managera. To rozwiązanie w założeniach jest skierowane do programistów. Twórcy zakładają, że wszystkie dodatkowe rzeczy, każdy z użytkowników sam sobie dopisze. Jest to możliwe z

uwagi na to, że cały kod programu znajduje się w jednym pliku o zawartości około 2000 linii kodu języka C. Takie podejście do dystrybucji wytworzyło wąską, wyspecjalizowaną grupę użytkowników, która ma w zwyczaju wymieniać się swoimi pomysłami na rozszerzanie otrzymanego kodu źródłowego.

DWM podobnie jak AwesomeWM nie posiada pulpitów wirtualnych, posiada za to system tagów. Działanie tego systemu różni się jednak, ponieważ każde okno posiada powiązanie również z ekranem na którym jest wyświetlane. Przy zmianie etykiety pokazują się wyłącznie okna z danego ekranu. Na tym jednak kończą się podobieństwa do AwesomeWM, ponieważ DWM nie udostępnia szablonów dzielenia ekranu.

Twórcy DWM nie zadeklarowali, że podejmą pracę nad przystosowaniem DWM do działania z serwerem wyświetlania Wayland. Z uwagi na cechy tego rozwiązania i niewielką społeczność użytkowników decyzja twórców wydaje się uzasadniona.

DWM jest więc rozwiązaniem, którego grupą docelową są wyłącznie programiści. Nie posiada możliwości konfiguracji i rozszerzania bez ingerencji w kod źródłowy. Jest stosunkowo trudny w instalacji oraz używaniu. Te cechy eliminują DWM jako rozwiązanie dla zwykłych użytkowników.

2.3 Podsumowanie

Przedstawione w tym rozdziale programy, których główną funkcją jest zarządzanie pozycją okien na pulpicie, są jedynie niewielkim podzbiorem programów tego typu. Opisane zostały rozwiązania najpopularniejsze wśród użytkowników. Jak łatwo zauważyć Gnome 3 oraz KDE, obecnie najczęściej wybierane, są managerami typu klasycznego tak jak w systemie Windows. Podobnie też dostarczane są wraz z pakietem podstawowych programów. Jest to ogromne uproszczenie dla użytkowników, którzy niekoniecznie chcą wnikać w budowę swojego systemu ponieważ po uruchomieniu zazwyczaj nie ma potrzeby pobierania z internetu dodatkowych aplikacji. Są proste w użyciu i łatwe w instalacji.

Tabela 1: Zestawienie najważniejszych cech przedstawionych programów.

Nazwa programu	Wsparcie dla Wayland	Typ zarządzania oknami	Sposób konfiguracji	Pulpity wirtualne
Gnome 3	Tak	Klasyczny	GUI	Tak
KDE	Tak	Klasyczny	GUI	Tak
Openbox	Nie	Klasyczny	Pliki tekstowe i GUI	Tak
AwesomeWM	Nie	Tiling	Pliki tekstowe	Nie, mechanizm tagowania
i3wm	Nie	Tiling	Pliki tekstowe	Tak, brak dynamicznego dodawania
DWM	Nie	Tiling	Kod źródłowy	Nie, mechanizm tagowania

Tabela 1 przedstawia zestawienie najważniejszych cech przedstawionych programów. Została zwrócona uwaga na to czy rozwiązanie jest zintegrowane z protokołem Wayland. Należy dodać, że w programach Openbox, AwesomeWM i3wm oraz DWM nie jest planowana implementacja takich mechanizmów.

Kolejną wyróżnioną cechą jest sposób konfiguracji. Tutaj w przypadku Gnome i KDE użytkownik otrzymuje podstawowy graficzny interfejs. W pozostałych przypadkach są to pliki tekstowe.

Ostatnia kolumna tabeli stanowi o sposobie implementacji pulpitów wirtualnych. Większość programów taki posiada, aczkolwiek w przypadku i3wm nowych pulpitów nie da się dodawać podczas działania programu.

3 Propozycja programu zarządzającego oknami

Poniższy rozdział opisuje założenia funkcyjne oraz pozafunkcjonalne managera okien oraz programu administracyjnego będącego klientem. Szczególna uwaga jest zwrócona na założenia wynikające ze specyficznej dziedziny proponowanego programu.

3.1 Rodzaj managera okien i model zarządzania przestrzenią roboczą

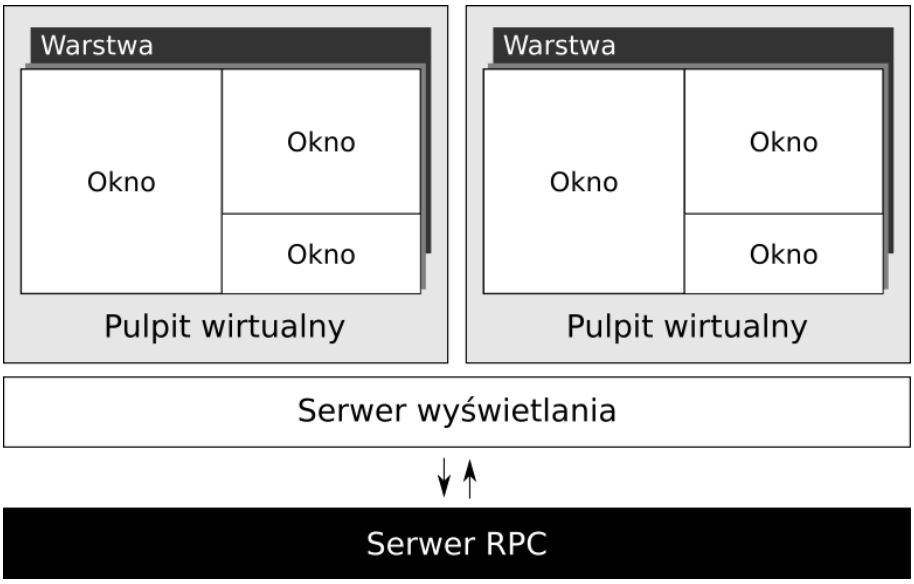
Program ma być managerem okien typu tiling. Projekt nie zakłada opisanego wcześniej trybu pływających okien. Ważnym założeniem jest wygodna obsługa środowiska bez potrzeby użycia myszy komputerowej, której wymaga wspomniany wyżej tryb. Z uwagi na dość skomplikowaną idee tagowania okien oraz wynikającego z niej założenia, że to samo okno może wstępować na różnych ekranach, został wybrany mechanizm wirtualnych pulpitów. Głównymi zaletami tego mechanizmu są jego prostota, intuicyjność oraz popularność w innych rozwiązaniach tego rodzaju.

Model okien zakłada istnienie dodatkowego poziomu warstw. Każde okno należy do jednej z warstw, które są na siebie nałożone w ramach jednego pulpitu wirtualnego. Użytkownik może zmieniać kolejność warstw aczkolwiek aktywne są okna tylko z warstwy na samym wierzchu. Mechanizm w działaniu wydaje się bardzo podobny do mechanizmu pulpitów wirtualnych. Różnicą jest to, że przy warstwach wszystkie okna są rysowane na ekranie. Dzięki temu użytkownik zyskuje możliwość wyświetlenia dowolnego programu w tle, co pozwala na bardzo generyczny sposób wyświetlania tapet lub nawet dynamicznych wizualizacji graficznych. Takie rozwiązanie wydaje się dość wygodne i eleganckie, zwłaszcza, że użytkownik w razie potrzeby może używać tylko jednej warstwy. W takim przypadku zarządzanie oknami będzie przebiegać dokładnie w taki sam sposób jakby tego mechanizmu nie było.

Rysunek 10 przedstawia w graficzny sposób model zarządzania przestrzenią roboczą. Podsumowując, składa się ona z okien umieszczonych na różnych warstwach. Warstwy zaś umieszczone są na pulpitach wirtualnych.

3.2 Marginesy ekranu oraz marginesy okien

Funkcją nie spotykaną w innych rozwiązaniach jest możliwość zdefiniowania marginesów na ekranie. Użytkownik powinien móc określić, jaka część ekranu ma zostać wyłączona z zarządzania przez program. Oczywiście należy udostępnić możliwość osadzenia dowolnego okna w wyłączonym obszarze, z tym wyjątkiem, że jego geometria będzie niezmienna.



Rysunek 10: Model okien wraz z przedstawieniem komunikacji między serwerem wyświetlania a serwerem RPC. Materiały własne.

Osoba przed ekranem powinna też, móc skonfigurować odstępy między oknami. Jedyną funkcją takiego zabiegu jest przyjemne dla oka i estetyczne oddzielenie okien na pulpicie. Jest to opcja spotykana w wielu managerach typu tiling.

3.3 Zdalne sterowanie

Dość przydatną opcją jest możliwość zdalnego sterowania ustawieniami managera okien. Założeniem więc jest to, że program musi pozwalać na kontrolę swoich mechanizmów zgodnie z popularnym protokołem do tego zaprojektowanym. Powodem uzasadniającym potrzebę użycia gotowego rozwiązania zamiast opracowania własnego jest ułatwienie implementacji późniejszych integracji z innymi programami. Użytkownik musi mieć w razie potrzeby możliwość wyłączenia tej opcji.

Komunikacja powinna odbywać się na zasadzie klient-serwer. Do podstawowych operacji dostępnych dla klienta należą:

- dodawanie, usuwanie oraz zmiana aktywnego pulpitu wirtualnego
- dodawanie, usuwanie oraz zmiana aktywnej warstwy
- przemieszczanie aktywnego okna w czterech kierunkach
- zmiana wysokości oraz szerokości aktywnego okna

- zmiana aktywnego okna
- ustawienie trybu wyświetlania okien
- zmiana konfiguracji programu

3.4 Pliki konfiguracyjne

Wszystkie pliki konfiguracyjne powinny być edytowalne z poziomu dowolnego edytora tekstowego. Jednocześnie muszą być zapisane w wygodnym dla człowieka formacie. Wynika to z popularności edytorów tekstowych takich jak nano czy vi działających wyłącznie z poziomu linii poleceń. Co więcej sytuacja wymagająca użycia takiego programu do edycji może być dość powszechna z uwagi na to, że przedmiotem konfiguracji jest program wyświetlający okna programów. Dane w postaci plików XML lub JSON mimo tego, że są bardzo praktyczne przy wczytywaniu danych przez program komputerowy, są mało czytelne dla osób nie zajmujących się programowaniem.

Konfiguracja w założeniu ma być podzielona na kilka plików dla wygody ich przechowywania. Każdy plik powinien zawierać konfigurację dotyczącą danej dziedziny. Listing 1 zawiera przykład proponowanej konfiguracji skrótów klawiszowych. Pierwsza wersja programu musi korzystać z konfiguracji dotyczącej:

- skrótów klawiszowych
- pulpitów wirtualnych wraz z ich warstwami
- startu procesów wraz z sesją serwera wyświetlania
- reguł okien
- wyglądu przestrzeni roboczej

Listing 1: Przykładowy kod konfiguracji skrótów klawiszowych.

```
emacs: logo+e emacs
pcmanfm: logo+m pcmanfm
terminal: ctrl+Return urxvt
```

Procedura wczytywania konfiguracji powinna być przetwarzana w taki sposób, żeby niepoprawne linie w pliku były ignorowane. Wystąpienie błędu w konfiguracji powinno być zgłoszone, aczkolwiek nie powinno blokować procedury inicjalizacji managera okien.

3.5 Logowanie informacji pochodzących z programu

Rozwiązanie musi przekazywać informacje o zdarzeniach w formie możliwej do zapisania. Ważne jest, żeby nie wszystkie błędy powodowały zatrzymanie działania programu. Informacje pochodzące z programu możemy podzielić więc na następujące kategorie:

- informacyjne - informacje nie będące błędem, których zadaniem jest wskazanie co obecnie dzieje się w mechanizmach managera okien
- ważne - kategoria podobna do informacyjnych z tą różnicą, że te informacje będą wyróżnione
- błędy - informacje o błędach, które nie są szkodliwe dla dalszego działania programu. Przesłanie takiej informacji nie powoduje zakończenia aplikacji
- krytyczne - informacje o błędach, których następstwem będzie zakończenie programu a w skutku powrót do trybu tekstowego

Każda z zapisanych informacji musi: zawierać dokładną datę wystąpienia zdarzenia, oznaczenie kategorii informacji oraz treść. W celu zapewnienia większej czytelności zapisanych danych, każda z kategorii powinna mieć przypisaną wartość kolorystyczną. Kolorowanie linii pozwoli to na wygodniejsze przeglądanie pliku.

Konfiguracja logowania powinna zawierać możliwość wskazania czy informacje maja być zapisywane do pliku czy przekazywane na standardowe wyjście. Dodatkowo powinno dać się skonfigurować, które kategorie informacji aplikacja ma pomijać.

3.6 Modularna implementacja

Ostatnim założeniem jest to, iż architektura programu ma pozwalać na łatwe dostosowanie mechanizmów managera okien do działania z serwerem wyświetlania X lub innym protokołem pozwalającym na zarządzanie oknami. To założenie jest bardzo ważne dla przyszłego rozwoju projektu. Odpowiednie odseparowanie logiki aplikacji od mechanizmów rysujących powinno rozwiązać problem modularnej implementacji, który przysporzył wielu problemów serwerowi X.

Implementacja powinna wykonywać wszystkie operacje najpierw w pamięci, następnie, gdy już wiadomo jak przestrzeń się zmieniła, powinno następować rysowanie zmian na ekranie. Takie podejście zapobiega zbędnym operacjom wejścia-wyjścia, których wykonanie zazwyczaj trwa dużo dłużej od operacji w pamięci. Niestety jest to odrobinę trudniejsze podejście, ponieważ zakłada niezbędny mechanizm detekcji zmian na poszczególnych elementach modelu okien.

3.7 Podsumowanie

Zaproponowane rozwiązanie ma dostarczyć wygodny sposób nawigacji między oknami za pomocą klawiatury. Wybrany typ managera okien pozwala na osiągnięcie tego celu. Pominięty na chwilę obecną został tzw. "tryb pływających okien" ponieważ nie wpływa on na zwiększenie użyteczności przestrzeni i łamie on założenia podejścia *tiling*.

Ciekawą cechą rozwiązania jest dodanie warstw do modelu okien. Pozwalają one na zorganizowaniu interaktywnego tła dla aktywnie używanych programów. Nie jest to cecha spotykana w innych rozwiązaniach. Unikalną funkcją są marginesy ekranu. Pozwalają one na pominięcie w zarządzaniu części ekranu. Konfiguracja powinna pozwolić na konfigurację marginesu każdej krawędzi ekranu z osobna.

Konfiguracja programu powinna być modularna i zapisywana w kilku plikach. Każdy z nich powinien zawierać pogrupowane wpisy. Ważnym aspektem jest sposób zapisu konfiguracji. Powinien on być w formacie wygodny dla człowieka, nawet w przypadku gdy będzie trudniejszy do wczytania przez komputer. Pozwoli to na bezproblemową edycję nawet dla mało zaawansowanych użytkowników.

4 Narzędzia i technologie zastosowane przy realizacji pracy

Dobór narzędzi został przeprowadzony z uwzględnieniem tego, że program jest dość niskopoziomowy oraz tego, że środowiskiem działania programu jest system rodziny GNU/Linux. Dobrym rozwiązaniem dla takich rozwiązań jest język C++, posiada on mechanizmy pozwalające operować bezpośrednio na pamięci a jednocześnie jest językiem obiektowym. Ogromną zaletą tego języka jest też to, że jego mechanizmy i składnia są unormowane przez standard ISO/IEC 14882:2017(E). Jest to rzadko spotykana cecha, która gwarantuje maksymalną kompatybilność wsteczną.

Kolejne narzędzia były dobrane na podstawie popularności wśród programistów oraz jakości dokumentacji. Procesem kompilacji steruje program *make*, którego plik konfiguracyjny jest wygenerowany przez rozwiązanie *CMake*.

W tym rozdziale szczegółowo zostaną omówione wykorzystane narzędzia takie jak język C++, CMake, Make, WLC oraz RPC. Opisane zostało również środowisko pracy jakim jest Emacs wraz z zastosowanymi pakietami pomagającymi pracować z kodem źródłowym C++.

4.1 git

Do wersjonowania kodu źródłowego został wykorzystany program git. Obecnie jest to najpopularniejsze wśród programistów rozwiązanie do decentralizowanego przechowywania kodu. Oryginalnym twórcą jest Linus Torvalds, który znany jest też z tego, że stworzył jądro systemu Linux [38]. Do jego zalet należą:

- praca bez dostępu do sieci
- duże wsparcie ze strony społeczności
- gwarancja tego, że program będzie darmowy
- jednoczesna praca wielu programistów jest wygodna nawet przez internet

4.2 CMake

Narzędzie CMake[14] pozwala zarządzać procesem kompilacji programu. Dzięki jego możliwościom budowanie programu jest niezależne od systemu operacyjnego, kompilatora oraz od sposobu zarządzania projektem. Ważne jest, że CMake sam nie przeprowadza kompilacji a jedynie tworzy potrzebne instrukcje dla danego środowiska.

CMake jest dość popularnym wyborem przy tworzeniu oprogramowania pod systemy oparte o GNU/Linux ponieważ zazwyczaj generuje on pliki Makefile. Dodatkową zaletą programu jest integracja z testami jednostkowymi.

4.3 Make

Make[15] to program stworzony w ramach projektu GNU, którego zadaniem jest sterowanie procesem tworzenia plików wykonywalnych na systemach rodziny GNU/Linux. Wykorzystuje on do budowy pliki zwane Makefile, które zawierają dokładną listę kroków do wykonania przez zewnętrzne programy takie jak kompilator czy linker.

To narzędzie jest niezwykle prostym i eleganckim rozwiązaniem i mimo swojego wieku jest to najpopularniejszy program tego typu[49]. W przypadku tej pracy, pliki Makefile są produktem działania wcześniej opisanego programu CMake.

4.4 C++ 17

Programy managerów okien, z uwagi na swoją niskopoziomowość, często implementowane są z użyciem języka C++. Jego niepodważalną zaletą jest możliwość bezpośrednich operacji na pamięci. Język C również posiada taką cechę, aczkolwiek C++ oferuje dodatkowo możliwość programowania obiektowego.

Wielu programistów uważa, że pisanie w tym języku powoduje poważne problemy z wyciekami pamięci. Poniekąd jest to prawda, ponieważ C++ jest pozbawiony mechanizmu zbieracza śmieci (ang. garbage collector), a za czyszczenie dynamicznie alokowanej pamięci odpowiedzialny jest programista. Opublikowany w 2017 roku standard C++ 17 posiada wiele mechanizmów wspomagających zarządzanie pamięcią [16] [50].

Obecnie panuje opinia, że należy możliwie unikać dynamicznej alokacji pamięci. W miejscach gdzie jest ona niezbędna zaleca się używanie inteligentnych wskaźników (ang. smart pointers), które same zwalniają pamięć, kiedy przestają być potrzebne. Ten mechanizm jest implementacją popularnego wzorca RAII[17] (ang. Resource Acquisition Is Initialization).

Warto wspomnieć, że począwszy wraz ze standardem C++11, język ten staje się coraz bardziej nowoczesny [39]. Z kolejnymi wydaniami zostały wprowadzone udogodnienia znane z innych języków, między innymi automatyczną dedukcję typów.

4.5 Google C++ Style Guide

C++ Style Guide[40] to dokument opublikowany przez firmę Google, w którym zawarte zostały propozycje zasad pisania czytelnego kodu w języku C++. Początkowo tekst był przeznaczony wyłącznie dla programistów firmy, został on jednak upubliczniony dla społeczności języka. Obecnie jest on wykorzystywany przez sporą rzeszę programistów jako zbiór dobrych praktyk. Niektóre edytory, jak między innymi GNU Emacs, wspierają nawet automatyczne formatowanie kodu zgodnie ze Style Guide[41].

4.6 WLC

WLC[42] (ang. Wayland Library Compositor) to biblioteka pomagająca w implementacji kompozytora protokołu Wayland. Znacząco upraszcza obsługiwanie zdarzeń serwera wyświetlania takich jak: utworzono nowe okno, zmiana pozycji kursora czy wciśnięcie klawisza na klawiaturze komputera.

Biblioteka posiada integrację z XWayland, który pozwala uruchamiać programy przystosowane wyłącznie do serwera wyświetlania X. Zaimplementowana jest w języku C, który bez problemów można używać wraz z wykorzystanym w tej pracy językiem C++. Istnieją również biblioteki pozwalające połączyć WLC z innymi językami jak Rust[43] czy OCalm[44].

4.7 rpclib

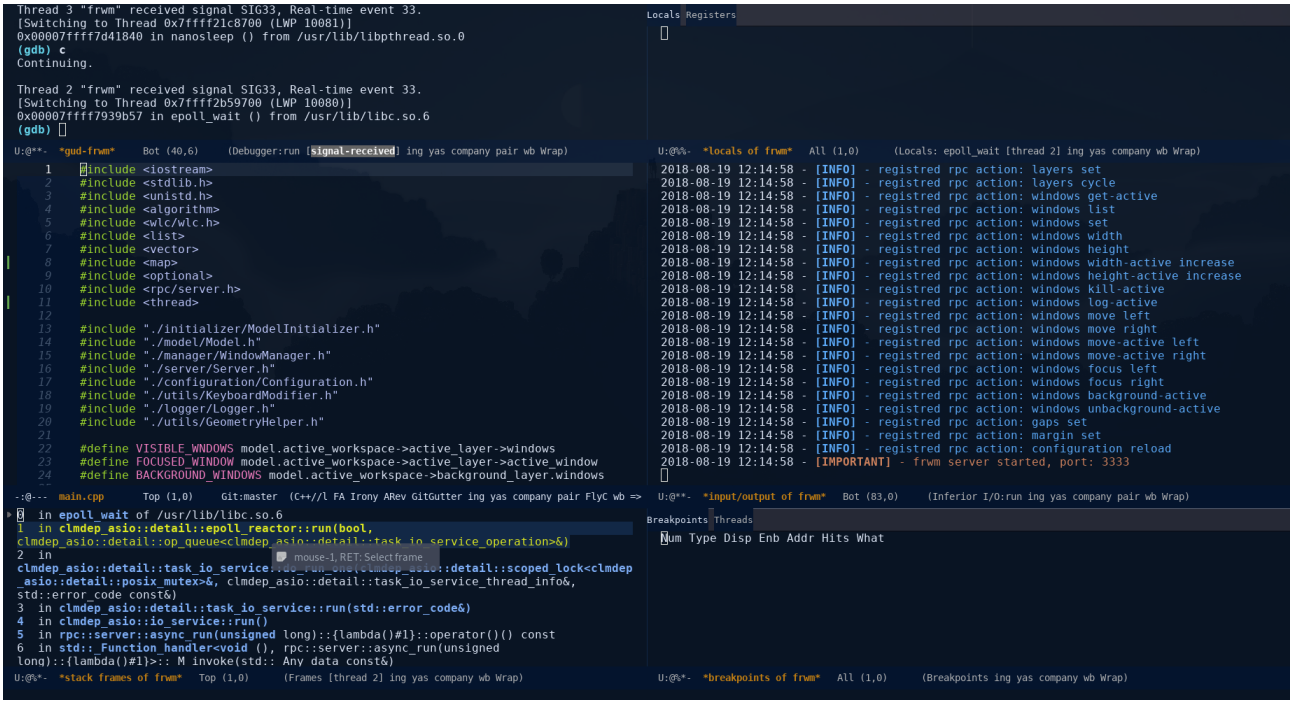
Do realizacji RPC została wykorzystana biblioteka rpclib[47]. Cechą ją wyróżniającą jest to, że jest napisana w pełni zgodnie ze standardem C++14, co pozwala na korzystanie z nowoczesnych mechanizmów tego języka. Dodatkowo, biblioteka gwarantuje, że nie będzie potrzeby generowania zbędnego kodu do obsługi, tak jak w przypadku bibliotek tego typu dla języka Java.

4.8 Emacs

Edytor tekstu oficjalnie wydawany przez Free Software Fundation od wielu lat jest bardzo popularny w środowisku programistów języka C++. Jego unikalną cechą jest system rozszerzeń, który pozwala dodać własny kod do kodu edytora. W jego przypadku pluginy, napisane w interpretowalnym języku Emacs List, po instalacji stają się częścią edytora, tak więc ich możliwości są nieograniczone żadnym API edytora [45] [46].

Emacs posiada zaawansowaną wbudowaną obsługę debuggera języka C++ GDB (ang. The GNU Project Debugger). Program debuggera obsługuje się zazwyczaj z poziomu linii poleceń, co nie

jest szczególnie wygodne. Polecenie edytora gdb-many-windows uruchamia specjalny bufor, którego zadaniem jest umożliwienie obsługi debuggera w graficzny sposób, co jest pokazane na rysunku **??**.



Rysunek 11: Emacs z włączonym buforem gdb-many-windows. Środowisko pozwala w wygodny sposób na debugowanie kodu C++. Źródło: materiały własne

Podczas implementacji projektu zostały wykorzystane następujące pakiety: irony-mode, flycheck-mode oraz google-c-style. Poza nimi użyte zostały również inne rozszerzenia, aczkolwiek rozszerzają one wygląd edytora lub sposób nawigacji po nim, tak więc nie wnoszą nic do sposobu obsługi kodu języka C++.

4.8.1 irony-mode

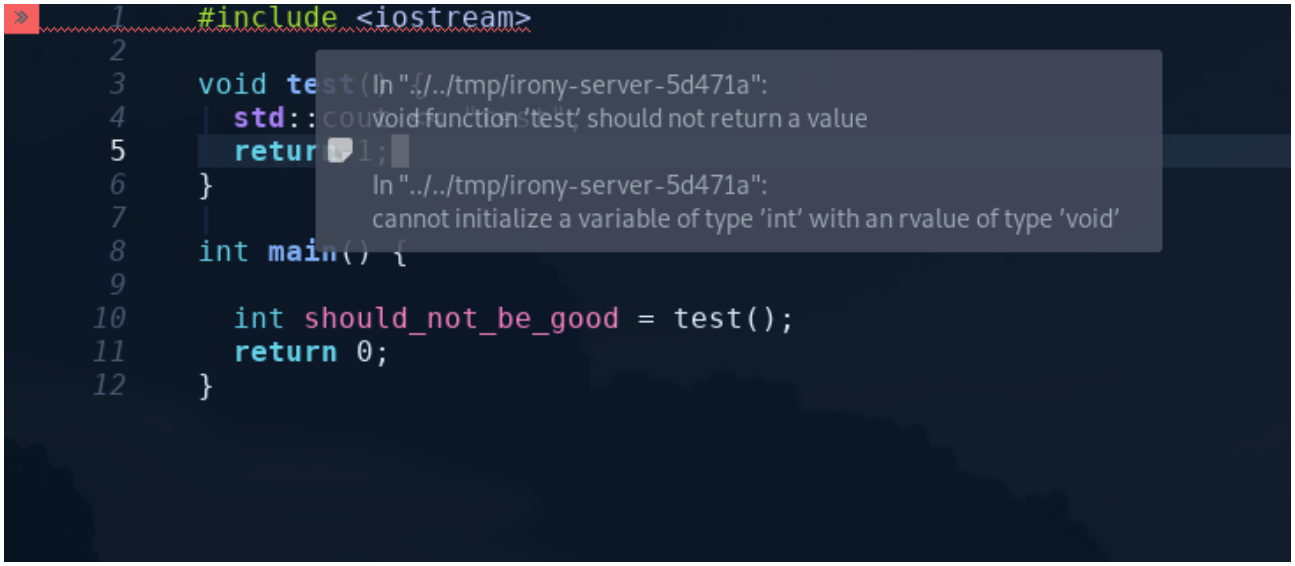
Rozszerzenie edytora dodające funkcje takie jak:

- mechanizm analizy i uzupełniania kodu C++
- analiza syntaktyczna kodu
- integracja z dokumentacją pisaną wraz z kodem źródłowym
- integracja z innymi rozszerzeniami, takimi jak na przykład silnik snippetów yasnippet

Dzięki użyciu irony-mode[48], Emacs otrzymuje funkcje IDE (ang. Integrated Development Environment). Do działania wykorzystuje bibliotekę libclang, która dostarcza bardzo zaawansowane metody przetwarzania kodu źródłowego języków rodziny C. Programista otrzymuje też szereg dodatkowych opcji jak możliwość budowania projektu z poziomu edytora. Dodatkowo błędy kompilacji zwrócone bezpośrednio do edytora są interaktywne i umożliwiają nawigację do odpowiednich linii kodu. Przydatną funkcją jest też uruchamianie testów jednostkowych wraz ze zapisywaniem zmian w plikach.

4.8.2 flycheck-mode

Wtyczka integrująca analizę syntaktyczną kodu dostarczoną przez irony-mode z mechanizmem graficznych powiadomień o błędach w składni kodu. Wszystkie błędy wyłapane przez to rozszerzenie przedstawiane są w czytelny graficzny sposób. Przykładowe zobrazowanie dwóch błędów jest pokazane na rysunku 12.



Rysunek 12: Emacs wraz z rozszerzeniem flycheck-mode podający informację o błędach w kodzie źródłowym C++. Źródło: materiały własne

4.8.3 google-c-style

Pakiet google-c-style dodaje predefiniwany styl formatowania kodu zgodny z Google C++ Style Guide. Dzięki temu, automatyczny format kodu zapewniony przez edytor jest zgodny z założenia-

mi projektu. Jest to znaczne uproszczenie dla programisty, ponieważ nie musi się on skupiać na brakujących białych znakach czy wcięciach.

5 Prototyp programu

Prototyp rozwiązania składa się z trzech programów. Pierwszy z nich to program zarządzający przestrzenią pulpitu i geometrią okien. Jest to najważniejsza część prototypu ponieważ to ona udostępnia możliwość zdalnego sterowania, które wykorzystuje drugi program. Implementuje on obsługę wszystkich dostępnych metod serwera RPC. Trzeci program jest praktycznym przykładem wykorzystania komunikacji miedzy dwoma pierwszymi programami. Pozwala on wyświetlić okno informujące użytkownika o stanie programu zarządzającego oknami.

5.1 Program managera okien - frwm

Implementacja programu zarządzającego okna wymagała dekompozycji jego części w taki sposób aby modyfikacja stanu pulpitu wykonywała się najpierw w pamięci komputera. Dopiero po wykonaniu wszystkich niezbędnych operacji należy nanieść wszystkie poprawki na ekran. Pozwala to oszczędzić czas na wykonywaniu zbędnych i bardzo kosztownych operacji. Program managera okien został podzielony na następujące elementy:

- Model okien
- Klasy zarządzające
- Serwer RPC
- Konfigurację

Każdy z wymienionych elementów spełnia określone zadanie, a w połączeniu całość pozwala na efektywne zarządzanie pulpitem użytkownika.

5.1.1 Sposób wyznaczania przestrzeni roboczej

Przestrzenią roboczą pulpitu można nazwać całą przestrzeń, na której są układane okna. Zazwyczaj środowiska pozwalają wykorzystać w tym celu całą dostępną przestrzeń. W managerach typu tiling jest to niestety problematyczne. Często się zdarza, że okna nachodzą na tzw. tackę systemową (ang. tray) przez co jest ona zasłonięta i niedostępna dla użytkownika.

Prototyp managera okien zawiera możliwość konfiguracji marginesów wewnętrznych ekranu. Każda krawędź ekranu monitora może zostać skonfigurowana osobno. Ta funkcja jest dość unikalna, żaden z programów tego typu przedstawiony przeglądzie sztuki jej nie zawierał. Możliwość konfiguracji marginesów, każdego z osobna, jest przydatna podczas dostosowywania środowiska.

5.1.2 Model okien

Model okien zgodnie z założeniami składa się nie tylko z samych okien. W takim przypadku, zarządzanie nimi byłoby bardzo trudne i nie pozwoliłoby na implementację pulpitów wirtualnych oraz warstw zaproponowanych w poprzednich rozdziałach. Model okien składa się z listy pulpitów wirtualnych oraz listy okien globalnych, czyli takich, które wyświetlane są na wszystkich ekranach. Każdy pulpit wirtualny składa się z listy warstw oraz jednej warstwy tła. Warstwy zawierają listę okien.

Pulpit wirtualny posiada unikalną nazwę, dzięki czemu, łatwo jest go rozróżnić i wskazać. Nazwa tak więc pełni też rolę identyfikatora. Prototyp nie posiada ograniczenia w postaci maksymalnej liczby pulpitów. Mogą być one dodawane i usuwane, w trakcie działania programu, zależnie od uznania użytkownika. Deklarację klasy pulpitu wirtualnego znajduje się w listingu 2.

Listing 2: Definicja klasy pulpitu wirtualnego.

```
class Workspace {
 public:
  Workspace();
  Workspace(string name);

  string name;

  list<Layer> layers;
  Layer *active_layer = nullptr;

  BackgroundLayer background_layer;
};
```

Warstwy w ramach każdego pulpitu wirtualnego również identyfikowane są po unikalnych nazwach. W odróżnieniu od pulpitów wirtualnych kolejność w liście jest ważna. Pierwsza warstwa zawsze jest tą aktywną i przykrywa inne warstwy. Zasadniczą różnica między pulpitem wirtualnym jest to, że wszystkie okna z warstw należących do danego pulpitu są widoczne. Okna z nieaktywnego pulpitu wirtualnego są ukryte.

Listing 3: Definicja warstwy.

```
class Layer {
 public:
  Layer();
  Layer(string name);
```

```
  string name;
  list<Window> windows;

  Window *active_window = nullptr;
  DISPLAY_MODE display_mode;
};
```

Warstwa tła ma za zadanie udostępnić możliwość ustawienia pełnoekranowych okien w tle. Marginesy przestrzeni pulpitu nie dotyczą okien z warstwy tła.

Każde okno posiada takie atrybuty jak określony liczbą naturalną identyfikator, typ, tytuł oraz klasę. Klasa okna używana jest do rozpoznawania okien o tych samych właściwościach. Jest to odpowiednik mechanizmu klas znanego z CSS.

Okna ze względu na typ można podzielić na:

- Normalne - standardowe okna programów
- Etykiety (ang. Tooltip) - okna, których zadaniem jest wyświetlanie podpowiedzi dla użytkownika
- Ekrany startowe (ang. Splash) - okna wyświetlane podczas uruchamiania programów, ich cechą szczególną jest to, że wyświetlane są idealnie wyśrodkowane
- Dialogowe (ang. Modal)
- Wyskakujące (ang. Popup)

Listing 4: Definicja okna.

```
class Window {
 public:
  Window();
  Window(uint id, WindowType type);
  Window(uint id, WindowType type, string title, string class_name);
  Window(uint id, WindowType type, string title, string class_name, Origin origin, Size size);

  uint id;
  string title;
  string class_name;

  Origin origin;
```

```
  Size size;

  WindowType type;
  bool hidden;
  bool fixed;

  Origin GetEndPosition();
  void Log();
  string ToString();
};
```

Okna układane są na dostępnej przestrzeni w ściśle określony sposób. Ważnym elementem algorytmu jest kolejność okien w liście. Okna znajdujące się bliżej początku listy są większe od swoich następników. Pierwsze okno zajmuje całą dostępną szerokość i wysokość przestrzeni roboczej. Za każdym razem gdy pojawia się nowe okno, rozmiar ostatniego z listy jest dzielony na pół a nowe okno wypełnia dostępną wolną przestrzeń. Doskonale algorytm ilustruje obraz 13, na którym widać uruchomiony prototyp z pięcioma uruchomionymi oknami.

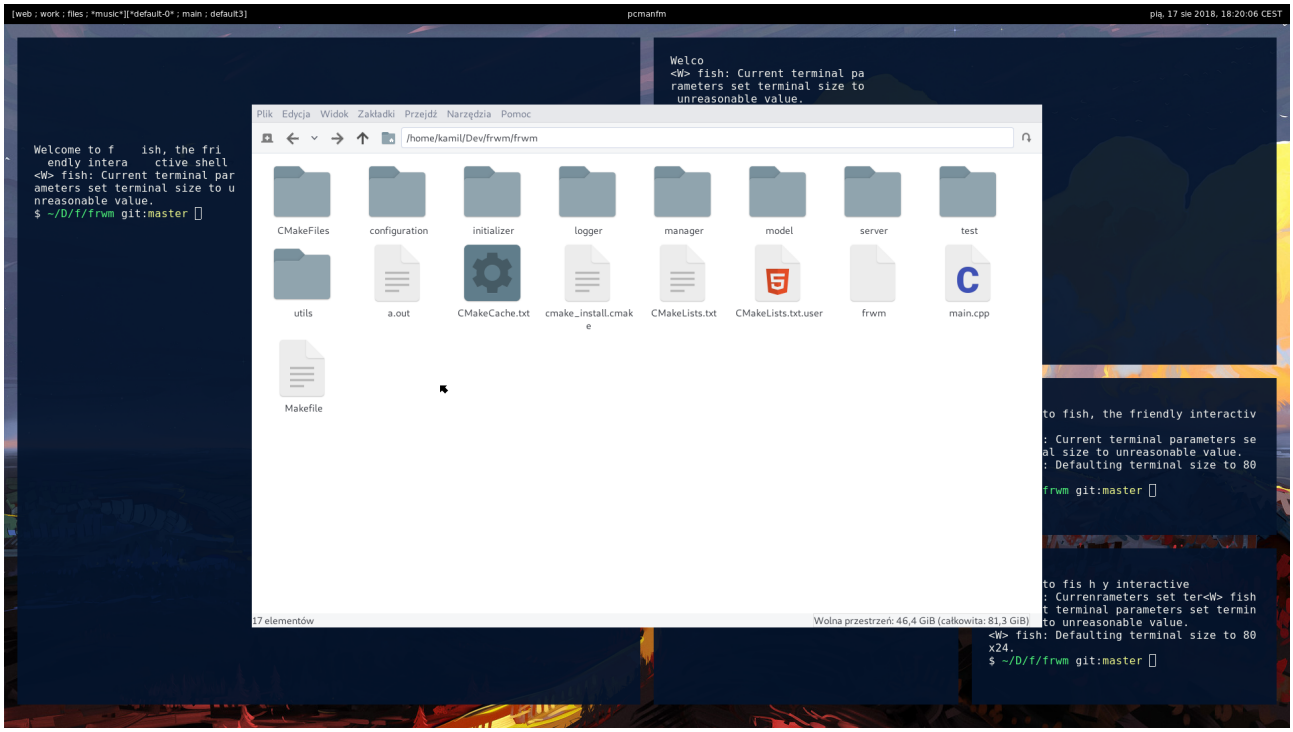
Rysunek 13: Pulpit z uruchomionymi pięcioma oknami emulatora terminala urxvt. Rozmiar okien jest stały, dzięki czemu użytkownik łatwo może przyzwyczaić się do działania managera okien. Materiały własne.

Przy tej metodzie układania okien należy wyróżnić dwie operacje, przesuwanie okna w lewo i w prawo. Jeśli listę okien zobrazuje się jako jednowymiarowa lista o skończonej długości, ułożonej w taki sposób, że początek listy znajduje się z prawej strony a koniec z lewej, operacja przesunięcia okna w lewo będzie zwiększać jego rozmiar. Analogicznie działa operacja przesunięcia w prawo, podczas, której okno zmniejsza swój rozmiar.

5.1.3 Tryby wyświetlania

Prototyp oferuje trzy tryby wyświetlania. Poza normalnym wyświetlaniem mozaiki okien dostępny jest tryb pełnoekranowy oraz "tryb skupienia". Okno w trybie pełnoekranowym analogicznie do znanych rozwiązań wyświetla okno rozciągnięte na całą powierzchnię ekranu. Marginesy w tym trybie są ignorowane.

"Tryb skupienia" pozwala wyświetlić okno na samym środku ekranu, przed pozostałymi oknami ułożonymi w standardowy sposób. Rozmiary okna w tym trybie są w pełni konfigurowalne. To rozwiązanie dostarcza wygodny sposób powiększania małych okien w celu dokładniejszego zbadania ich zawartości. Rysunek 14 pokazuje pulpit z aktywowanym "trybem skupienia".



Rysunek 14: Pulpit z aktywowanym "trybem skupienia. Materiały własne.

5.1.4 Klasy zarządzające

Modyfikacja opisanego modelu przeprowadzana jest za pomocą klas zarządzających. Każda z nich ma za zadanie przeprowadzać operacje na oknach w całkowicie kontrolowany sposób. Wyodrębnione zostały trzy klasy, których zadaniem jest zarządzanie poszczególnymi częściami modelu.

Klasa WorkspaceManager zawiera metody pozwalają na modyfikację listy pulpitów wirtualnych. Dostępne w niej są trzy operacje:

- Dodawanie nowych pulpitów wirtualnych
- Usuwanie już istniejących pulpitów wirtualnych
- Ustawianie aktywnego pulpitu wirtualnego

Na chwilę obecną prototyp nie pozwala na zmianę nazwy pulpitu oraz na scalanie dwóch pulpitów w jeden. Są to rzadko spotykane opcje i niekoniecznie wykorzystywane.

Zarządzanie warstwami jest dość analogiczne do zarządzania pulpitami wirtualnymi. Różnicą są funkcje związane z kolejnością w liście. Klasa zarządzająca warstwami (LayerManager) w ramach aktywnego pulpitu wirtualnego zawiera następujące metody:

- Dodawanie nowych warstw do danego pulpitu wirtualnego
- Usuwanie istniejących warstw
- Ustawianie aktywnej warstwy
- Przejście do kolejnej warstwy

Przy przejściu do kolejnej warstwy pierwszy element listy jest dopisywany do ogona listy a drugi element staje się nową głową. Jak łatwo zauważyć przy odpowiedniej liczbie przejść, równej liczbie elementów w liście, użytkownik powróci do warstwy, od której zaczynał. Aby uniknąć zbędnych operacji, użytkownik ma też możliwość przełączenia się od razu do wskazanej warstwy.

Najbardziej rozbudowany mechanizm zawarty jest w klasie zarządzającej oknami WindowManager. Pozwala on na operacje:

- Dodanie nowego okna
- Usunięcie istniejącego okna

- Ustawienie aktywnego okna
- Zamianę miejscami dwóch okien
- Przesunięcie okna w lewo, jednoznaczne ze zwiększeniem rozmiaru okna
- Przesunięcie okna w prawo, jednoznaczne ze zmniejszeniem rozmiaru okna
- Przeniesienie okna do innej warstwy
- Przeniesienie okna na warstwę tła
- Przeniesienie okna z warstwy tła

5.1.5 Reguły okien

Dla zwiększenia wygody użytkownika powstała metoda wymuszania rozmiaru i pozycji wskazanych okien. Przykładowo, gdyby osoba przed komputerem miała potrzebę "przypiąć" zegarek lub kalendarz do pulpitu, w taki sposób, żeby jego okno znajdowało się zawsze w tym samym miejscu, może skonfigurować tzw. "regułę okna".

Do zdefiniowania reguły użytkownik musi wskazać klasę okna, którego reguła dotyczy, oraz żądane parametry. Co ważne, parametry ignorują narzucone marginesy. Tak więc istnieje możliwość "wyłączenia" części ekrany dla normalnych okien i użycie jej jako panelu widgetów. Kod źródłowy definicji reguły okna znajduje się w listingu 5.

Listing 5: Definicja reguły okna.

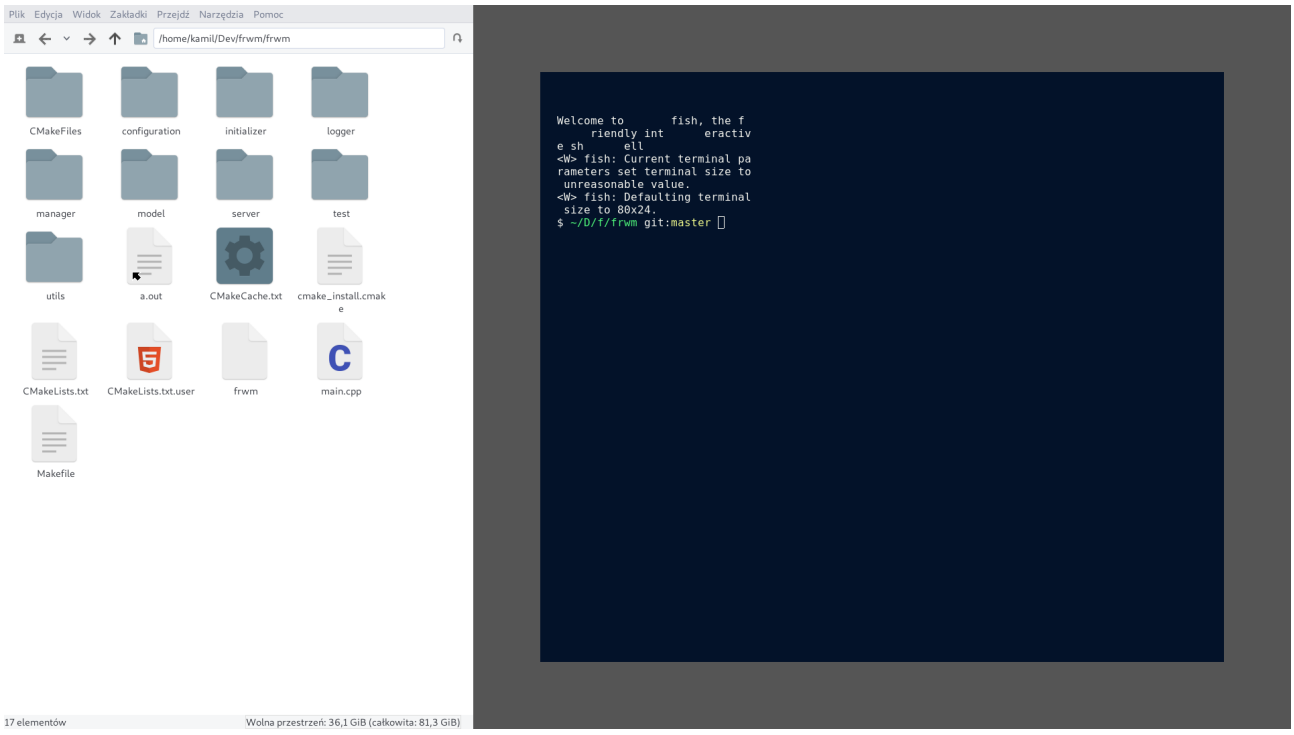
```
class WindowRule {
 public:
  string window_class;
  optional<Origin> origin;
  optional<Size> size;
  bool visible_on_all_screens = false;
  bool background = false;

  static WindowRule FromString(string str);

 private:
  static string SanitizePart(string str);
};
```

Rozmiar i położenie lewego górnego rogu są opcjonalne. Dodatkowo możliwa jest konfiguracja tego, że okno widoczne jest na wszystkich pulpitach wirtualnych lub jest umiejscowione na warstwie tła. Przykładowy wpis w konfiguracji przedstawiony jest w listingu 6.

Listing 6: Przykładowy wpis w konfiguracji reguł okien.

```
Pcmanfm: size: 700x1080 origin: 0x0 all
```

Mechanizm ten pozwala na umieszczenie okna o określonych rozmiarach w dowolnym miejscu na ekranie. Niezależnie od tego czy zostanie przesłonięte inne okno. Taką sytuację ilustruje rysunek 15, na który widoczne jest okno pogramu Pcmanfm skonfigurowanego zgodnie z wpisem z listingu 6. Przedstawia on "przypięte" do lewej krawędzi okno przeglądarki plików, dzięki czemu jest ona zawsze dostępna. Z prawej strony widać wyznaczoną przestrzeń roboczą. Reguły pozwalają na wyświetlanie wskazanego okna na każdym pulpicie wirtualnym. Takie rozwiązanie pozwala na wygodną konfigurację wyświetlania programów takich jak paski statusu czy tacka systemowa.



Rysunek 15: Zastosowanie reguł okien. Materiały własne.

5.1.6 Pliki konfiguracyjne

Zgodnie z założeniem, konfiguracja została podzielona na kilka plików, których format jest wygodny dla użytkownika. Konfiguracja programu została podzielona na następujące pliki:

- autostart - konfiguracja programów uruchamianych na etapie inicjalizacji sesji
- rules - reguły okien
- shortcuts - konfiguracja skrótów klawiszowych
- wm - ogólne ustawienia managera okien
- workspaces - konfiguracja pulpitów wirtualnych

W pliku z ogólnymi ustawieniami programu zarządzającego oknami znajdują się ustawienia takie jak szerokość marginesów, port serwera RPC czy też rozdzielczość ekranu. Dodatkowo można skonfigurować domyślną nazwę pulpitu wirtualnego i warstwy.

5.1.7 Serwer RPC

Zdalne sterowanie realizowane jest za pomocą serwera RPC, które działanie jest w pełni opcjonalne. Metody jakie udostępnia to:

- Wymuszenie ponownego rysowania okien
- Ustawienie trybu wyświetlania okien
- Akcje klasy zarządzającej pulpitami wirtualnymi
- Akcje klasy zarządzającej warstwami
- Akcje klasy zarządzającej oknami
- Ustawienie szerokości przerw między oknami
- Ustawienie szerokości marginesu
- Przeładowanie konfiguracji

5.1.8 Logger

Zapis zdarzeń realizowany jest w przyjazny dla użytkownika sposób. Przykładowy log widoczny jest na ilustracji 16.

Informacja o zdarzeniu składa się z trzech części. Pierwszą z nich jest data wystąpienia. Następną jest rodzaj zdarzenia a ostatnią tekst wiadomości.

```
$ ~/D/f/frwm git:master ./frwm
2018-08-15 12:26:39 - [INFO] - loading wm configuration
2018-08-15 12:26:39 - [INFO] - loading line: margin top: 40
2018-08-15 12:26:39 - [INFO] - loading line: margin right: 400
2018-08-15 12:26:39 - [INFO] - loading line: margin bottom: 25
2018-08-15 12:26:39 - [INFO] - loading line: margin left: 10
2018-08-15 12:26:39 - [INFO] - loading line: gaps width: 50
2018-08-15 12:26:39 - [INFO] - wm configuration loaded
2018-08-15 12:26:39 - [INFO] - loading shortcuts configuration
2018-08-15 12:26:39 - [INFO] - loaded shortcut: emacs: ctrl+e emacs -q
2018-08-15 12:26:39 - [INFO] - loaded shortcut: term: ctrl+x urxvt
2018-08-15 12:26:39 - [INFO] - loaded shortcut: height: logo+Up frwmctl windows height-active increase 2
2018-08-15 12:26:39 - [INFO] - loaded shortcut: height: logo+Down frwmctl windows height-active increase -2
2018-08-15 12:26:39 - [INFO] - loaded shortcut: width: logo+Right frwmctl windows width-active increase 2
2018-08-15 12:26:39 - [INFO] - loaded shortcut: width: logo+Left frwmctl windows width-active increase -2
2018-08-15 12:26:39 - [INFO] - loaded shortcut: move-left: ctrl+Left frwmctl windows move-active left
2018-08-15 12:26:39 - [INFO] - loaded shortcut: move-right: ctrl+Right frwmctl windows move-active right
2018-08-15 12:26:39 - [INFO] - loaded shortcut: fullscreen: ctrl+f frwmctl fullscreen mode toggle
2018-08-15 12:26:39 - [INFO] - loaded shortcut: focus: logo+f frwmctl focus mode toggle
2018-08-15 12:26:39 - [INFO] - loaded shortcut: dmenu: logo+r dmenu_run -b
2018-08-15 12:26:39 - [INFO] - loaded shortcut: kill: ctrl+q frwmctl windows kill-active
2018-08-15 12:26:39 - [INFO] - loaded shortcut: info: logo+l frwmctl windows log-active
2018-08-15 12:26:39 - [INFO] - shortcuts configuration loaded
2018-08-15 12:26:39 - [INFO] - loading workspaces cofiguration
2018-08-15 12:26:39 - [INFO] - loaded workspace: web
2018-08-15 12:26:39 - [INFO] - loaded workspace: work
2018-08-15 12:26:39 - [INFO] - loaded workspace: files
2018-08-15 12:26:39 - [INFO] - loaded workspace: music
2018-08-15 12:26:39 - [INFO] - workspaces configuration loaded
2018-08-15 12:26:39 - [INFO] - loading autostart configuration
2018-08-15 12:26:39 - [INFO] - loaded: urxvt
2018-08-15 12:26:39 - [INFO] - loaded: python /home/kamil/Dev/frwm/example_tools/topbar.py
2018-08-15 12:26:39 - [INFO] - loaded: feh /home/kamil/Dev/dotfiles/I3/wallpaper11.jpg
2018-08-15 12:26:39 - [INFO] - loading rules configuration
2018-08-15 12:26:39 - [INFO] - loaded rule: Frwm_panel: size: 1920x30 origin: 0x0 all
2018-08-15 12:26:39 - [INFO] - loaded rule: topbar_qt.py: size: 1920x40 origin: 0x0 all
2018-08-15 12:26:39 - [INFO] - loaded rule: Orage: size: 380x200 origin: 1530x50
2018-08-15 12:26:39 - [INFO] - loaded rule: feh: size: 1920x1080 origin: 0x0 all
2018-08-15 12:26:39 - [IMPORTANT] - configuration loaded
2018-08-15 12:26:39 - [INFO] - initializing workspaces
2018-08-15 12:26:39 - [INFO] - initialized workspace: web
2018-08-15 12:26:39 - [INFO] - initialized workspace: work
2018-08-15 12:26:39 - [INFO] - initialized workspace: files
2018-08-15 12:26:39 - [INFO] - initialized workspace: music
2018-08-15 12:26:39 - [IMPORTANT] - model initialized
2018-08-15 12:26:39 - [INFO] - registred rpc action: render
2018-08-15 12:26:39 - [INFO] - registred rpc action: display mode set
2018-08-15 12:26:39 - [INFO] - registred rpc action: fullscreen mode toggle
```

Rysunek 16: Zapis zdarzeń podczas uruchamiania prototypu programu. Dla wygody użytkowania odpowiednie rodzaje zdarzeń są wyróżnione kolorami. Materiały własne.

Mechanizm pozwala na zapis różnego rodzaju zdarzeń. Każde z nich jest wyróżnione innym kolorem. Dostępne typy wyróżnienia to: informacyjne, błędy oraz ważne.

Domyślnie informacje przekierowane są na standardowy strumień wyjściowy programu, dzięki

czemu użytkownik bez problemu, z użyciem dowolnych narzędzi konsolowych, może dane przekierować do pliku lub na strumień wejściowy innego programu.

5.2 Program narzędzia administracyjnego - frwmctl

Przykładem wykorzystania wystawionego przez program zarządzający oknami serwera RPC jest napisany program administracyjny. Pozwala on na wywołanie wszystkich zdefiniowanych metod serwera z linii poleceń systemu GNU/Linux. Dodatkowo może działać w trybie interaktywnej pętli nieskończonej, tzw. REPL.

Program może być wykorzystywany bezpośrednio przez użytkownika, lub do napisania prostych narzędzi zintegrowanych ze środowiskiem lub je automatyzujących.

5.2.1 REPL

Tryb interaktywnej pętli nieskończonej narzędzia administracyjnego pozwala na zdalne zarządzanie zorganizowane w stylu powłoki systemowej. Pozwala on na nawiązanie połączenia z serwerem RPC managera okien a następnie bez wyłączania programu na synchroniczne wykonywanie wielu metod zdalnych.

Po wywołaniu akcji jej rezultat jest wypisywany na ekran. Program przechodzi do następnej linii, wyświetla znak zachęty i oczekuje na wprowadzenie przez użytkownika kolejnego polecenia. Dzięki czemu jeśli jakaś operacja wymaga więcej niż jedną akcję, użytkownik ma poczucie ciągłości.

Różnicą między używaniem programu w trybie REPL a wywołaniem polecenia z linii komend jest taka, że REPL nawiązuje połączenie z serwerem RPC tylko raz. Standardowe wywołanie za każdym poleceniem musi nawiązać nowe połączenie.

5.2.2 Przykład integracji z narzędziem administracyjnym

Jako przykład integracji z wykorzystaniem opisanego narzędzia administracyjnego został wykonany program wyświetlający informacje o obecnym stanie środowiska. Kod źródłowy został napisany z użyciem języka Python. Jest on przedstawiony w listingu 7.

Listing 7: Kod źródłowy programu zintegrowanego ze środowiskiem za pomocą narzędzia administracyjnego.

```
from tkinter import *
import time
import os
```

```
root = Tk(className='Frwm_panel')
root.title('Frwm_panel')
root.columnconfigure(0,weight=1, uniform='third')
root.columnconfigure(1,weight=1, uniform='third')
root.columnconfigure(2,weight=1, uniform='third')
root.configure(background='black')

left_label = Label(root, text='***', bg='black', fg='white').grid(row=0, column=0, sticky="wn", padx=10, pady
    =5)
center_label = Label(root, text='frwm panel - api example', bg='black', fg='white').grid(row=0, column=1,
    sticky="n", padx=10, pady=5)
right_label = Label(root, text='***', bg='black', fg='white').grid(row=0, column=2, sticky="en", padx=10, pady
    =5)

def join(str_list):
    return '[' + ' ; '.join(str_list) + ']'

def get_data():
    date = os.popen("date").readlines()[0].replace('\n', '')
    workspaces = os.popen("frwmctl workspaces list").readlines()
    active_workspace = os.popen("frwmctl workspaces get-active").readlines()[0]

    layers = os.popen("frwmctl layers list").readlines()
    active_layer = os.popen("frwmctl layers get-active").readlines()[0]

    workspaces = list(map(lambda x: x.replace('\n', ''), workspaces))
    workspaces = list(map(lambda x: x if x != active_workspace else '*' + x + '*', workspaces))

    layers = list(map(lambda x: x.replace('\n', ''), layers))
    layers = list(map(lambda x: x if x != active_layer else '*' + x + '*', layers))

    window = os.popen("frwmctl windows get-active").readlines()[0].split(';')[2]

    left = join(workspaces) + join(layers)
    center = window
    right = date

    labels = root.winfo_children()

    labels[0]['text'] = left
    labels[1]['text'] = center
    labels[2]['text'] = right

    root.after(1000, get_data)
```

```
root.after(1000, get_data)
root.mainloop()
```

Pogram wykorzystuje pięć poleceń narzędzia administracyjnego. Co sekundę pobiera listę pulpitów wirtualnych, listę warstw, nazwę aktualnie wybranego pulpitu wirtualnego, nazwę aktywnej warstwy oraz dane aktywnego okna. Po przetworzeniu tych danych, we wskazanym z pomocą reguł okien miejscu, wyświetlane jest odpowiednie okno. Wycinek pulpitu z uruchomionym programem przedstawiony jest na obrazie 17.

Rysunek 17: Wycinek pulpitu z uruchomionym programem wskazującym obecny stan programu zarządzającego oknami. Materiały własne.

6 Opis wybranych przypadków użycia

W poniższym rozdziale opisane zostały przypadki testowe wybranych funkcji prototypu. Przedstawiony został sposób wykonywania poszczególnych testów oraz ich rezultat. Rozdział zakończony jest krótkim podsumowaniem, w którym zestawione zostały początkowe założenia projektu oraz uzyskany rezultat.

Podczas testów w plikach konfiguracyjnych programu znajdują się tylko i wyłącznie niezbędne do testów wpisy. Z tego powodu większość rysunków w tym rozdziale może wydawać się surowa.

6.1 Wyznaczanie przestrzeni roboczej - marginesy ekranu

W celu sprawdzenia czy wyznaczona przestrzeń robocza jest odpowiedniej powierzchni, należało skonfigurować program w taki sposób aby każdy z czterech marginesów był różnej szerokości. Następnie należy uruchomić dowolne okno i za pomocą odpowiedniego oprogramowania linijki ekranowej zmierzyć marginesy.

Manager okien został skonfigurowany w sposób przedstawiony w listingu 8. Jednostką podanych wartości są piksele.

Listing 8: Konfiguracja marginesów ekranu w pliku konfiguracyjnym wm.

```
margin top: 100
margin right: 200
margin bottom: 300
margin left: 500
```

Rysunek 18 przedstawia ekran wyświetlony po uruchomieniu programu oraz jednego okna emulatora terminala. Marginesy mają odpowiednie szerokości.

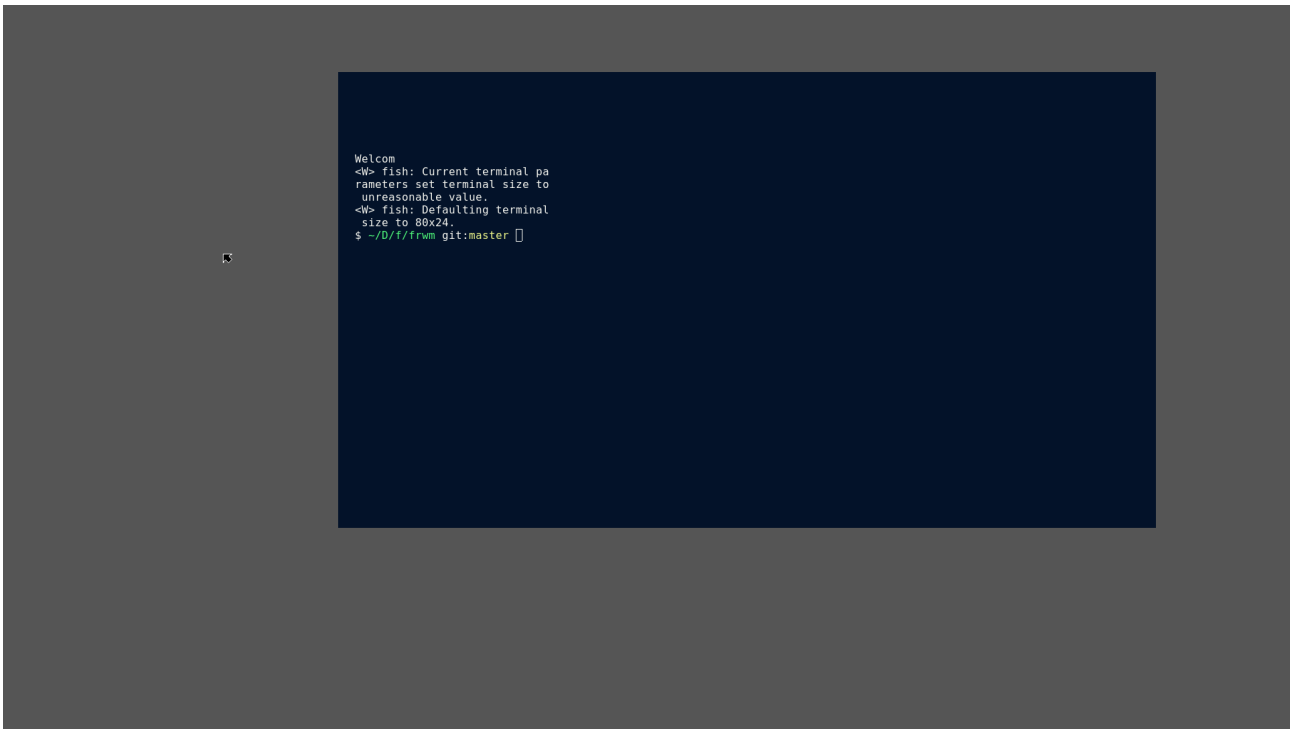
6.2 Marginesy okien

Sprawdzenie marginesów okien podobnie jak marginesów ekranu polega na wprowadzeniu odpowiedniej konfiguracji i zmierzeniu odległości na ekranie. W tym przypadku aby poprawnie zbadać działanie funkcji należy uruchomić przynajmniej dwa okna.

Do pliku konfiguracyjnego wm został wprowadzony tekst widoczny w listingu 9.

Listing 9: Konfiguracja marginesów okien w pliku konfiguracyjnym wm.

```
gaps width: 20
```



Rysunek 18: Rezultat testów wyznaczania przestrzeni roboczej. Materiały własne.

Na rysunku 19 widać uruchomione dwa okna. Można zaobserwować, że przestrzeń bezpośrednio między nimi jest dwukrotnie większa od przestrzeni dzielącej krawędź ekranu od dowolnego z okien. Jest to pożądane działanie ponieważ każde z okien ma własny margines.

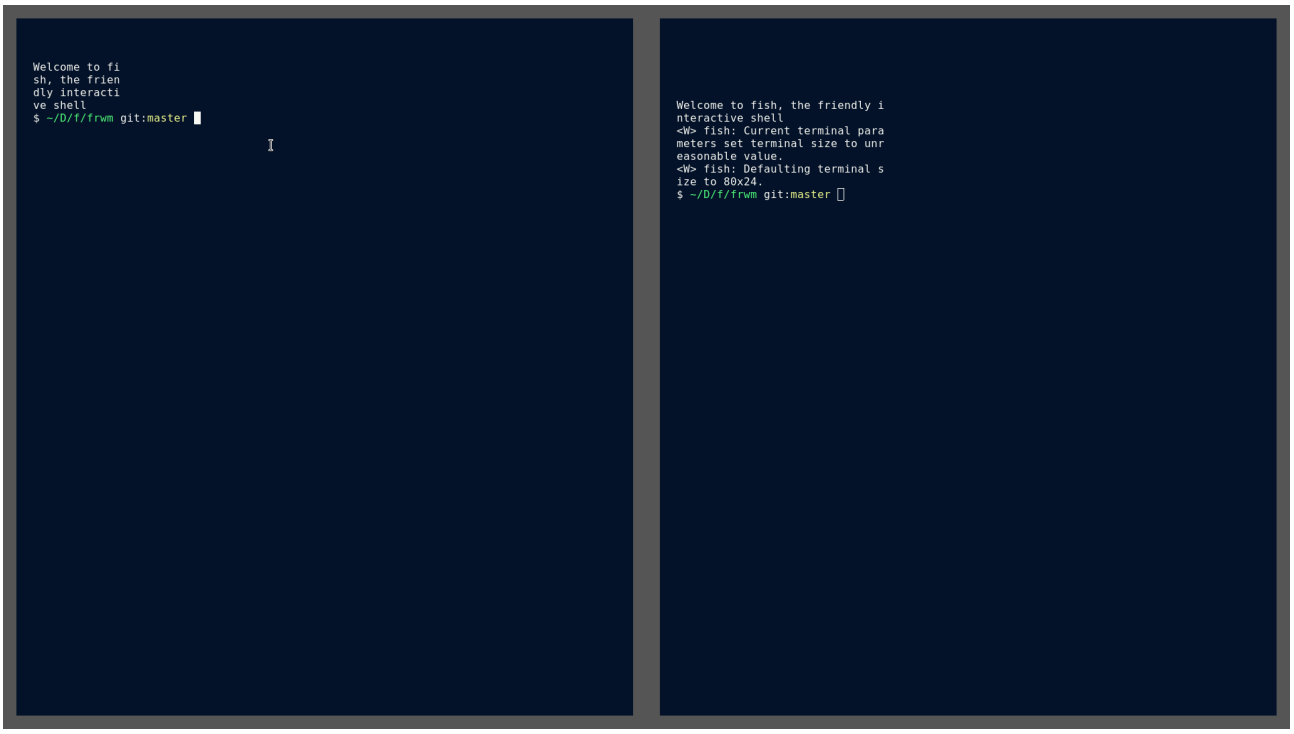
Drugi test ten funkcji polegał na ustawieniu wartości zerowej parametru marginesu okien. W takim przypadku okna powinny przylegać bezpośrednio do siebie. Rezultat tego testu jest przedstawiony na rysunku 20

6.3 Algorytm podziału przestrzeni ekranu

Sprawdzanie algorytmu podziału wykonano poprzez uruchamianie kolejnych okien i obserwację czy przestrzeń jest dzielona zgodnie z założeniami. Do celu testów ustawiono marginesy okien, aby łatwiej można było zidentyfikować poszczególne okna (widoczne przerwy między nimi).

Rysunki 21 i 22 przedstawiają kolejno przestrzeń roboczą z jednym i czterema oknami.

Jak można zauważyć, przestrzeń jest dzielona zgodnie z założeniami.



Rysunek 19: Rezultat testów konfiguracji marginesów okien. Materiały własne.

6.4 Wczytywanie konfiguracji

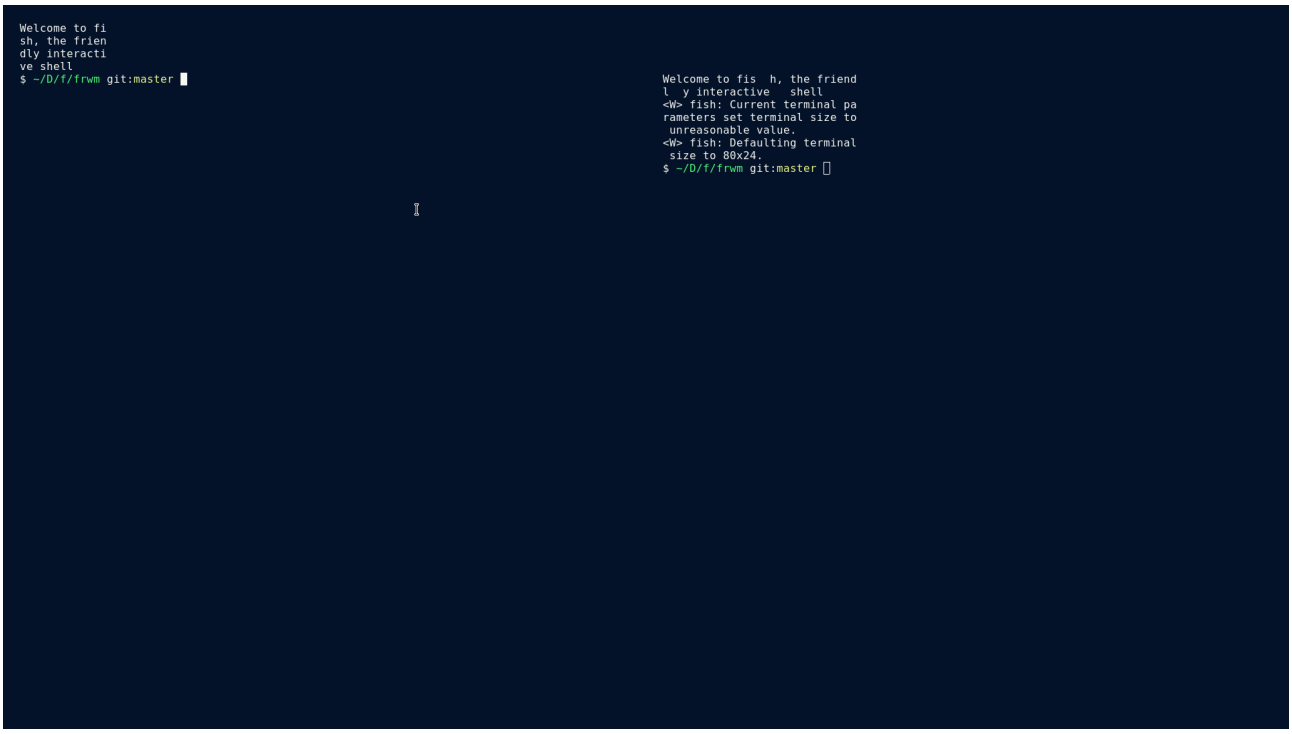
Zgodnie z założeniami niepoprawne linie konfiguracji mają być pomijane. Żeby sprawdzić funkcjonowanie tego modułu najłatwiej jest wstawić niepoprawne linie do konfiguracji i następnie obserwować zapis zdarzeń z programu. Program powinien się uruchomić pomimo błędnej konfiguracji.

Listing 10 zawiera zawartość pliku konfiguracyjnego wm. Połowa wpisanych linii jest niepoprawna, dla utrudnienia są one ułożone na przemian.

Listing 10: Konfiguracja programu z błędnymi liniami.

```
margin bottom: 0
niepoprawna: 100
margin right: 0
bardzo niepoprawna: 0
margin left: 0
margines: 0
margin top: 0
magin lop: 0
gaps width: 5
```

Podczas ładowania program na standardowy strumień wyjściowy zwrócił tekst znajdujący się



Rysunek 20: Rezultat testów konfiguracji marginesów okien w przypadku gdy ustawiona jest wartość zerowa. Materiały własne.

w listingu 11. Każda z niepoprawnych linii została zwrócona jako błąd, program został jednak uruchomiony mimo błędów w konfiguracji.

Listing 11: Wycinek tekstu zwróconego przez program.

```
2018-08-20 20:23:01 - [INFO] - loading wm configuration
2018-08-20 20:23:01 - [INFO] - loading line: margin bottom: 0
2018-08-20 20:23:01 - [INFO] - loading line: niepoprawna: 100
2018-08-20 20:23:01 - [ERROR] - cannot load wm configuration, invalid line: niepoprawna: 100
2018-08-20 20:23:01 - [INFO] - loading line: margin right: 0
2018-08-20 20:23:01 - [INFO] - loading line: bardzo niepoprawna: 0
2018-08-20 20:23:01 - [ERROR] - cannot load wm configuration, invalid line: bardzo niepoprawna: 0
2018-08-20 20:23:01 - [INFO] - loading line: margin left: 0
2018-08-20 20:23:01 - [INFO] - loading line: margines: 0
2018-08-20 20:23:01 - [ERROR] - cannot load wm configuration, invalid line: margines: 0
2018-08-20 20:23:01 - [INFO] - loading line: margin top: 0
2018-08-20 20:23:01 - [INFO] - loading line: magin lop: 0
2018-08-20 20:23:01 - [ERROR] - cannot load wm configuration, invalid line: magin lop: 0
2018-08-20 20:23:01 - [INFO] - loading line: gaps width: 5
2018-08-20 20:23:01 - [INFO] - wm configuration loaded
```

Rysunek 21: Przestrzeń robocza z jednym oknem. Materiały własne.

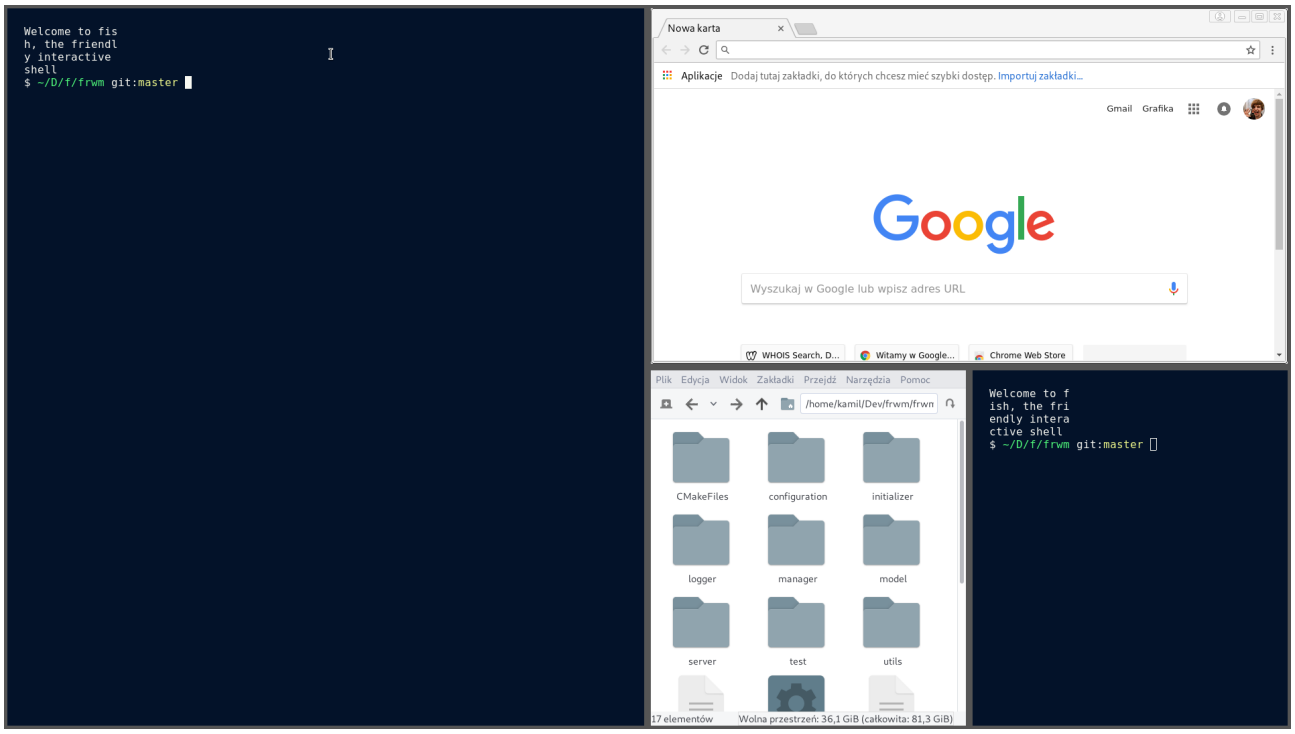
6.5 Reguły okien

Reguły okien standardowo są konfigurowane w specjalnym pliku konfiguracyjnym. Test polega na zdefiniowaniu reguły umieszczającej okno w miejscu poza przestrzenią roboczą. Tak więc w pierwszym kroku należy skonfigurować marginesy, dla celów testu wystarczy ustawienie marginesu lewego z wartością 800 pikseli i pozostałych z wartością 100 pikseli. Następnie skonfigurowana została reguła dla programu Pcmanfm umieszczająca okno o rozmiarach 700 pikseli szerokości i 1080 pikseli wysokości w lewym górnym rogu. Odpowiedni wpis konfiguracyjny jest widoczny w listingu 12.

Listing 12: Konfiguracja reguły okna dla programu Pcmanfm.

```
Pcmanfm: size: 700x1080 origin: 0x0
```

Dla zobrazowania ustawień przestrzeni roboczej, poza oknem programu Pcmanfm uruchomione zostało też inne okno bez przypisanej reguły. Rysunek 23 potwierdza działanie wprowadzonej konfiguracji. Okno zostało umiejscowione w odpowiednim miejscu, ignorując ustawienia marginesów.



Rysunek 22: Przestrzeń robocza z czterema oknami. Materiały własne.

6.6 Komunikacja z serwerem RPC

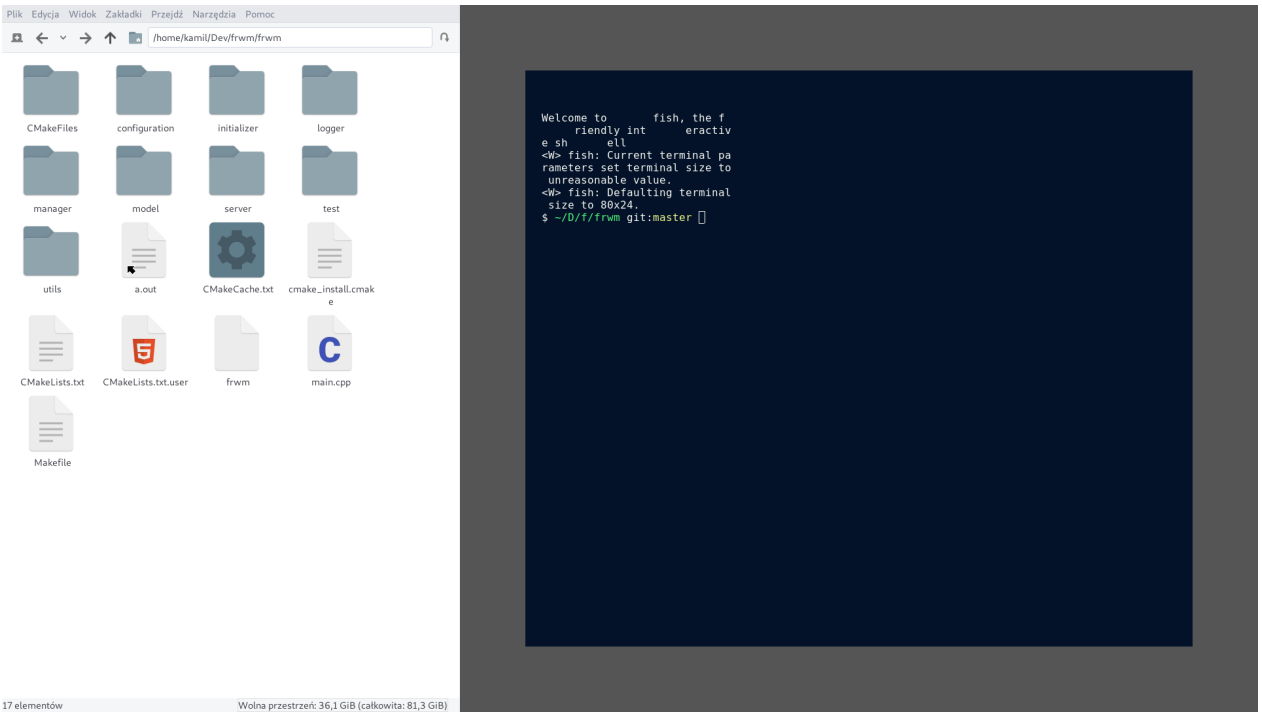
Przykładem użycia serwera RPC może być dodanie nowego pulpitu wirtualnego. Do sprawdzenia działania został użyty program administracyjny, którego prototyp został opisany w poprzednim rozdziale. Z linii poleceń należy wykonać trzy polecenia:

- Pobranie listy pulpitów wirtualnych
- Dodanie nowego pulpitu wirtualnego
- Ponowne pobranie listy pulpitów wirtualnych

Przebieg całej operacji dokumentuje listing 13. Przed stworzeniem nowego pulpitu wirtualnego o nazwie *very-new-workspace* istniały cztery zdefiniowane pulpitu. Po dodaniu lista została rozszerzona o nowy pulpit.

6.7 Podsumowanie

Przedstawione przypadki użycia i testy funkcji związanych z nimi potwierdzają, że działanie prototypu jest zgodne z założeniami. Połączenie funkcji marginesów ekranu wraz z regułami okien,



Rysunek 23: Przestrzeń robocza z dwoma oknami, do jednego z nich została zdefiniowana reguła. Materiały własne.

pozwala użytkownikowi na zagospodarowanie ekranu na wiele różnych sposobów. Dzięki dokładnemu logowaniu zdarzeń programu, użytkownik może w trakcie pracy bez problemów poprawić błędy powstałe w trakcie ładowania konfiguracji.

Udekorowaniem całej pracy jest udostępnienie serwera RPC, który pozwala na sterowanie środowiskiem przez inne programy. Pozwala to na automatyzacje pracy użytkownika i integrację z różnego rodzaju usługami.

Listing 13: Wynik operacji dodania nowego pulpitu wirtualnego.

```
> ~/D/f/frwm git:master frwmctl workspaces list
web
work
files
music
> ~/D/f/frwm git:master frwmctl workspaces add very-new-workspace
OK!
> ~/D/f/frwm git:master frwmctl workspaces list
web
work
files
music
very-new-workspace
> ~/D/f/frwm git:master
```

7 Podsumowanie oraz plany rozwoju projektu

Ostatni rozdział tej pracy opisuje zalety i wady programu wytworzonego podczas pisania pracy. Przedstawione również są wnioski bazujące na przemyśleniach autora. Rozdział zakończony jest krótkim opisem możliwości dalszego rozwoju prototypu z uwzględnieniem wciąż rozwijającej się społeczności użytkowników protokołu Wayland.

7.1 Zalety i wady rozwiązania

W przypadku tej pracy ciężko do zalet zaliczyć ładnie wykonany interfejs użytkownika czy wieloplatformowość. Prototyp realizuje planowane funkcje i dostarcza użytkownikowi wygodny sposób zarządzania oknami. Jednocześnie zaletą i wadą jest typ managera okien. Nie każdy użytkownik doceni fakt, że okna układają się w mozaikę, dzięki czemu ma zawsze dostęp do wszystkich widocznych okien. Zapewne wiele osób obsługujących komputer będzie preferowała klasyczne podejście do tego zagadnienia. Zaletą tego rozwiązania jest to, że nawigacja między oknami, zmiana rozmiaru i inne operacje wykonywane są z pominięciem myszy komputerowej.

Sposób wyznaczania przestrzeni roboczej, mianowicie konfiguracja każdego z marginesów z osobna, jest ogromną zaletą programu. Jest to unikalna funkcja i nie występuje w żadnym z programów przedstawionych w przeglądzie sztuki. Połączenie marginesów i reguł okien tworzy swojego rodzaju synergię funkcji, ponieważ reguły okien pozwalają na zagospodarowanie przestrzeni zwolnionej przy konfiguracji. Można w ten sposób osiągnąć przypinanie różnych okien jak widgetów.

Największą wadą prototypu jest brak trybu pływających okien. Jest on spotykany w wielu innych programach i mógłby być opcjonalną częścią programu. Dobrym pomysłem byłoby też dostarczenie więcej niż jednego algorytmu wyznaczania wielkości okien. Namiastką tych funkcji są opisane wcześniej tryby wyświetlania, aczkolwiek dla zaawansowanych użytkowników mogą być one niewystarczające.

Kolejną zaletą prototypu jest serwer RPC. Użycie popularnego protokołu do realizacji zdalnego sterowania upraszcza przyszłe integracje między programami. Do prostych operacji można użyć również narzędzia administracyjnego, tak jak zostało to wykonane w programie wyświetlającym status programu.

7.2 Możliwości rozwoju

Wartym uwagi aspektem prototypu jest rozszerzenie jego funkcji o dodatkowe algorytmy podziału przestrzeni ekranu. Dobrym sposobem realizacji tej funkcji byłoby dodanie nowych funkcji do zdalnego sterowania. Serwer mógłby przesyłać odpowiednie dane do klienta, tak aby ten mógł zadecydować gdzie i jakiej wielkości powinno być nowe okno. Powinna być to funkcja całkowicie opcjonalna i niezalecana, ponieważ przejmuje ona sporą odpowiedzialność z programu managera i nie gwarantuje spójności w zagospodarowaniu pulpitu. Jako sposób dużo łatwiejszy do wykonania byłoby zaimplementowanie różnych strategii tak jak w jest to wykonane w programie AwesomeWM.

Implementacja trybu pływających okien jest również pożądanym kierunkiem rozwoju. Wielu użytkowników spiera się w temacie jego użyteczności, aczkolwiek gdyby był konfigurowalny i byłaby możliwości jego wyłączenia, nie powinien być problematyczny dla osób, które z niego nie korzystają. Zagadnieniem związanym z tym trybem jest również uproszczenie nawigacji za pomocą myszy. Przemyślany system gestów mógłby usprawnić pracę z pulpitem. Przykładem operacji, która mogłaby być wywoływana z użyciem gestów myszy są przesunięcia okien w lewo oraz prawo.

Ciekawą i niespotykaną opcją byłoby też stworzenie programu z graficznym interfejsem, który pozwalałby na zmianę konfiguracji programu. Takie zmiany mogłyby być wprowadzane w czasie rzeczywistym. Dodatkową funkcją mógłby być też import i eksport plików konfiguracyjnych. Pozwoliłoby to na udostępnianie swojej konfiguracji pulpitu dla innych użytkowników. Na chwilę obecną użytkownicy dzielą się konfiguracją po przez przechowywanie jej w repozytorium i udostępnianie go po przez serwis internetowy github. Takie działanie ma jednak ogromną wadę w tym, że konfiguracja nie jest przechowywana w usystematyzowany sposób co praktycznie uniemożliwia zautomatyzowany import.

Uwieńczeniem rozwoju rozwiązania byłoby udostępnienie jego kodu dla społeczności na zasadach open source. Wśród użytkowników systemu GNU/Linux bardzo liczebną grupa są programiści, których pomysły i pomoc mogłaby rozwinąć program z dużo większym tempem. Upublicznienie pozwoliłoby też na pozyskanie większej liczby użytkowników.

7.3 Wnioski

Duża dynamika prac związanych z rozwiązaniem Wayland skutkuje pojawianiem się coraz nowszych koncepcji wykorzystania tego co oferuje. Mimo dość dużego zaawansowania tego rozwiązania nie istnieją jeszcze popularne managery typu *tiling* współpracujące z tym protokołem. Wielu użytkowników jest przektycznie bez szansy na migrację swojego pulpitu. Opisana sytuacja stwarza niewielką

niszę, dla programów realizujących zarządzanie oknami zgodnie z zasadami *tiling*.

Problem zarządzania okien nie ma najlepszego rozwiązania. Managery okien typu klasycznego pozwalają na bardzo wygodną nawigację za pomocą myszy komputerowej. W przypadku gdy osoba przed komputerem bardziej ceni sobie wygodę pracy wyłącznie z klawiaturą lepszym rozwiązaniem są programy typu "tiling". Wyłącznie od preferencji użytkownika zależy w takim przypadku, który program powinien wybrać. Mnogość dostępnych rozwiązań nie pomaga w wyborze. Wypróbowanie wszystkich może pochłonąć bardzo dużo czasu, co w skutku może być powodem irytacji sporej części użytkowników.

Wytworzony prototyp przedstawia jedynie odrobinę całej złożoności problemu jakim jest ergonomia w komunikacji człowiek-komputer. Świetnie realizuje on zarządzanie oknami i konfigurację pulpitu jednocześnie pozwalając na rozszerzanie jego opcji z pomocą zdalnego sterowania. Dla wielu użytkowników, zwłaszcza osób związanych z programowaniem, wykorzystanie programu do codziennej pracy może być ciekawą opcją i wprowadzeniem do świata mniej klasycznych managerów okien.

8 Literatura

[1] *Operating System Market Share Worldwide*
http://gs.statcounter.com/os-market-share, dotęp 15.06.2018r.

[2] *i3: A tiling window manager*
https://github.com/i3/i3, dotęp 25.08.2018r.

[3] *AwesomeWM*
https://github.com/awesomeWM/awesome, dotęp 25.08.2018r.

[4] *r/unixporn*
https://www.reddit.com/r/unixporn/, dotęp 25.08.2018r.

[5] *Growing Ubuntu for cloud and IoT, rather than phone and convergence*
https://blog.ubuntu.com/2017/04/05/
growing-ubuntu-for-cloud-and-iot-rather-than-phone-and-convergence, dostęp 01.06.2018r.

[6] *X Window System protocols and architecture*
https://en.wikipedia.org/wiki/X_Window_System_protocols_and_architecture, dostęp 01.06.2018r.

[7] *Wayland Architecture*
https://wayland.freedesktop.org/architecture.html, dostęp 01.06.2018r

[8] *The Wayland Situation: Facts About X vs. Wayland*
https://www.phoronix.com/scan.php?page=article&item=x_wayland_situation&num=1, dostęp 01.06.2018r.

[9] *X Window System core protocol*
https://en.wikipedia.org/wiki/X_Window_System_core_protocol, dostęp 01.06.2018r.

[10] *Architecture*
https://wiki.ubuntu.com/X/Architecture, dostęp 01.06.2018r.

[11] *Wayland: A New X Server For Linux*
https://www.phoronix.com/scan.php?page=article&item=xorg_wayland&num=1, dostęp 02.06.2018r.

[12] *Wayland Architecture*

`https://wayland.freedesktop.org/architecture.html`, dostęp 02.06.2018r.

[13] *X vs. Wayland Architecture*

`https://wayland.freedesktop.org/docs/html/ch03.html`, dostęp 03.06.2018r.

[14] *CMake*

`https://cmake.org/`, dostęp 10.06.2018r.

[15] *Make*

`https://linux.die.net/man/1/make`, dostęp 10.06.2018r.

[16] *C++ 17 Features*

`http://www.bfilipek.com/2017/01/cpp17features.html`, dostęp 06.06.2018r.

[17] *RAII*

`http://en.cppreference.com/w/cpp/language/raii`, dostęp 06.06.2018r.

[18] *Window manager*

`https://wiki.archlinux.org/index.php/window_manager`, dostęp 10.05.2018r.

[19] *Virtual Desktops in Windows 10 – The Power of Windows... Multiplied*

`https://blogs.windows.com/windowsexperience/2015/04/16/virtual-desktops-in-windows-10-the-power-of-windowsmultiplied/`, dostęp 10.05.2018r.

[20] *Gnome 3*

`https://www.gnome.org/gnome-3/`, dostęp 10.05.2018r.

[21] *Informacje o wydaniu GNOME 3.0*

`https://help.gnome.org/misc/release-notes/3.0/`, dostęp 10.05.2018r.

[22] *Mutter: a window manager for GNOME 3*

`https://lwn.net/Articles/344734/`, dostęp 10.05.2018r.

[23] *Mutter Can Cause A Gaming/OpenGL Performance Hit Too*

`https://www.phoronix.com/scan.php?page=article&item=mutter_composite_hit`, dostęp 10.05.2018r.

[24] *GNOME Tweaks*

`https://gitlab.gnome.org/GNOME/gnome-tweaks`, dostęp 10.05.2018r.

[25] *The Qt Issue*

`https://www.kde.org/community/history/qtissue.php`, dostęp 12.05.2018r.

[26] *Krunner*

`https://userbase.kde.org/Plasma/Krunner`, dostęp 12.05.2018r.

[27] *KWin*

`https://userbase.kde.org/KWin`, dostęp 12.05.2018r.

[28] *KWin/Wayland*

`https://community.kde.org/KWin/Wayland`, dostęp 12.05.2018r.

[29] *KWin Can Cause A Performance Hit Too, But It's Different From Compiz*

`https://www.phoronix.com/scan.php?page=article&item=kwin_speed_test&num=1`, dostęp 12.05.2018r.

[30] *Openbox*

`http://openbox.org/wiki/Main_Page`, dostęp 13.05.2018r.

[31] *ObConf*

`https://github.com/danakj/obconf`, dostęp 13.05.2018r.

[32] *AwesomeWM*

`https://awesomewm.org/`, dostęp 14.05.2018r.

[33] *Awesome WM III. – konfiguračný súbor I. časť*

`https://linuxos.sk/clanok/awesome-wm-iii-konfiguracny-subor-i-cast/`, dostęp 14.05.2018r.

[34] *i3 User's Guide*

`https://i3wm.org/docs/userguide.html`, dostęp 14.05.2018r.

[35] *Sway*

`https://github.com/swaywm/sway`, dostęp 14.05.2018r.

[36] *DWM*

`https://dwm.suckless.org/`, dostęp 15.05.2018r.

[37] *RFC1057*

`https://tools.ietf.org/html/rfc1057`, dostęp 20.06.2018r.

[38] Jon Loeliger, Matthew McCullough. *Kontrola wersji z systemem Git. Narzędzia i techniki programistów.*,

Helion, ISBN: 9788324681761

[39] Stephan Roth. *Czysty kod w C++17. Oprogramowanie łatwe w utrzymaniu*,

Helion, ISBN: 9788328343405

[40] *Google C++ Style Guide*

`https://google.github.io/styleguide/cppguide.html`, dostęp 20.06.2018r.

[41] *Google's C/C++ style for c-mode*

`https://raw.githubusercontent.com/google/styleguide/gh-pages/google-c-style.el`, dostęp 20.06.2018r.

[42] *WLC*

`https://github.com/Cloudef/wlc`, dostęp 20.06.2018r.

[43] *Rust bindings for wlc, the Wayland compositor library*

`https://github.com/way-cooler/rust-wlc`, dostęp 20.06.2018r.

[44] *Experimental OCaml binding for Wlc*

`https://github.com/Armael/ocaml-wlc`, dostęp 20.06.2018r.

[45] Debra Cameron, James Elliott, Marc Loy. *Learning GNU Emacs*

O'Reilly Media, ISBN: 9780596552374

[46] Bob Glickstein. *Writing GNU Emacs Extensions. Editor Customizations and Creations with Lisp*,

O'Reilly Media, ISBN: 9781449399733

[47] *RPC library for C++*

`https://github.com/rpclib/rpclib`, dostęp 21.06.2018r.

[48] *A C/C++ minor mode powered by libclang*

`https://github.com/Sarcasm/irony-mode`, dostęp 21.06.2018r.

[49] Robert Mecklenburg. *Managing Projects with GNU Make*,
O'Reilly Media, ISBN: 9780596552541

[50] Scott Meyers. *Skuteczny nowoczesny C++. 42 sposoby lepszego posługiwania się językami C++11 I C++14*,
Promise, ISBN: 9788328345010

Spis rysunków

Spis listingów

Spis tabel