Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa inżynierska

AMEBAE - WIELOOSOBOWY KOMUNIKATOR PROGRAMISTYCZNY

Joanna Daukszewicz, 71733
Tomasz Kaszkowiak, 71742
Paweł Martenka, 71746
Konrad Siek, 71747

Promotor
dr inż. Bartosz Walter

Poznań, 2008 r.

Spis treści

Rozdział 1

Wstęp

1.1 Rozwój komunikacji komputerowej

Obserwując historię komputerów od czasów rozwoju pierwszych maszyn obliczeniowych do osobistych komputerów powszechnie wykorzystywanych w teraźniejszości, można zauważyć, że cele w jakim są one wykorzystywane, ulegały stopniowym transformacjom od maszyn wspierających matematyków w obliczeniach do urządzeń, w których coraz większy nacisk kładziony jest na komunikację i współpracę [Mar99].

Nową erę zastosowań komputera rozpoczęła grupa inżynierów z ARPAnet w roku 1972 wysyłając pierwszą wiadomość poczty elektronicznej między komputerami połączonymi ze sobą przez sieć ARPA [Tom99]. Początkowo wśród swoich funkcji poczta elektroniczna posiadała jedynie możliwość przesłania zwykłej wiadomości tekstowej do jednego adresata, jednak wkrótce pojawiły się serwery pozwalające na rozsyłanie do listy użytkowników docelowych a także możliwość przesyłania treści graficznych czy też innego rodzaju plików binarnych. W kolejnych latach coraz większa liczba użytkowników przyzwyczajała się do korzystania z takiej metody komunikacji interpersonalnej, do tego stopnia, że w XXI stuleciu stała się ona najpopularniejszym zastosowaniem Internetu [JP02].

Rozwój technologii komputerowych oraz nowe potrzeby użytkowników doprowadziły do rozwoju nowych narzędzi mających na celu komunikację i współpracę komputerową grup osób. Pierwszym systemem tego typu jest rozwijany od roku 1972-ego Community Memory, który służył jako publiczna tablica ogłoszeń. Kolejnymi rozwijającymi tę ideę oraz zastosowanie narzędziami są powstałe później Computerized Bulletin Board System (CBBS) oraz Usenet oraz będące ich potomkami współczesne fora internetowe. Systemy te sprawiły, że możliwe stało się komunikowanie się między sobą grup specjalistów w celu współpracy nad projektami.

Osobną klasą programów komunikacyjnych są komunikatory internetowe pozwalające na rozmowy między użytkownikami w czasie rzeczywistym. Rozwijające się narzędzia takie jak IRC czy talk, oraz ich bezpośredni następcy: AIM, ICQ, czy polski system Gadu-Gadu zdają się być kolejnym ogniwem ewolucji komunikacji, dające zupełnie nowe możliwości współdziałania, w którym czas między poszczególnymi wypowiedziami jest niewielki. Wraz z popularyzacją komunikatorów na rynku pojawiły się rozwiązania oferujące więcej, niż tylko możliwość wysyłania tekstu - większość nowoczesnych programów tego typu pozwala na przesyłanie dowolnych plików poza rozmową i wstawianie wcześniej utworzonych obrazów do rozmowy. Inne programy, takie jak Skype, poza rozmowami tekstowymi umożliwiają rozmowy głosowe, czy też wideokonferencje.

1.2 Współpraca z wykorzystaniem komputerów

Rozwój komunikacji internetowej jest środkiem wspomagających pracę wszelkiego rodzaju specjalistów, w szczególności programistów pracujących nad projektami zespołowymi, w różnych fazach ich przygotowania, od projektowania do testowania.

Często pojawiającym się aspektem tego typu komunikacji jest dzielenie się, a nawet współtworzenie wszelkiego rodzaju schematów, diagramów oraz rysunków zawierających objaśnienia, gdy komunikacja słowna zawodzi. Podczas gdy istnieją rozwiązania pozwalające na wspólne rysowanie prostych rysunków (np. plugin Drawing do Jeti), to nie pozwalają one na sprawną edycje bardziej skomplikowanych obrazów, jak np. diagramy UML, co powoduje że zespół projektantów zmuszony jest edytować pliki programem lokalnym, zachowywać je i przesyłać do pozostałych osób zainteresowanych diagramem. Komunikacja taka staje się wówczas problematyczna, ponieważ oczekiwanie na odpowiedź się wydłuża, a użytkownicy są zmuszeni dbać o to, żeby oglądać najświeższe wersje diagramów.

Programiści często korzystają także z konsultacji dotyczących fragmentów napisanego kodu, w celu jego uzupełnienia, objaśnienia czy też wychwycenia błędów. Procedura takiego przeglądu staje się równie skomplikowana jak współdzielenie skomplikowanego diagramu, a dodatkowo każdy zmieniony fragment kodu winien jest zaznaczenia i objaśnienia. Istnieją narzędzia do wspólnej edycji dokumentów, takie jak Google Docs, nie będące częścią komunikatorów internetowych, jednak są one zaprojektowane przede wszystkim z nastawieniem na dokumenty biurowe, prezentacje i arkusze kalkulacyjne a nie dostarczają nawet najprostszej funkcjonalności przydatnej programistom, takiej jak kolorowanie składni czy też mechanizm wyszukiwania i zamiany fragmentów tekstu, oferują za to inne, niepotrzebne programistom funkcje.

1.3 Cel i zakres pracy

Celem pracy jest wyposażenie wybranego komunikatora internetowego w funkcje, które uczynią go bardziej przydatnym przy pracy wielu specjalistów nad projektem programistycznym - narzędzia zezwalające na wspólne tworzenie schematów i grafiki oraz wspólną edycję fragmentów kodu.

Zaproponowane rozszerzenia zostaną dołączone do komunikatora jako zestaw dwóch wtyczek, z których jedna, AmeBoard, odpowiedzialna będzie za umożliwienie użytkownikom korzystania z edytora grafiki wektorowej, który zezwoli, między innymi, na rysowanie figur i ścieżek oraz późniejszą ich manipulację.

Druga z wtyczek, AmeEdit, ma za zadanie udostępniać możliwość równoczesnej edycji jednego pliku z kodem źródłowym, przez użytkowników, wspomagając ich pracę poprzez kolorowanie składni oraz przez dostęp do narzędzi do szukania i wskazywania fragmentów tekstu.

Projektowane wtyczki, będąc częścią jednego pakietu, będą posiadać spójny i ergonomiczny interfejs użytkownika, AmeGUI, który pomoże użytkownikowi konfigurować wykorzystywane narzędzie oraz zapewni dostęp do informacji na temat użytkowników biorących aktualnie udział w edycji.

Powyższe moduły zostaną połączone ze sobą nawzajem oraz z komunikatorem za pomocą modułu AmePlug, który także będzie odpowiedzialny za nadzorowanie i bezpieczne dokonanie przesyłu danych do użytkowników zdalnych. Moduł ma także za zadanie wykrywanie sytuacji wyjątkowych związanych z utratą połączenia przez użytkowników oraz przedsięwzięcie odpowiednich akcji zaradczych.

1.4 Udział poszczególnych autorów pracy

Całość projektu została podzielona na mniejsze części - moduły, za zaprojektowanie, implementację i nadzór testów każdego z których odpowiedzialna była jedna z osób zajmujących się projektem. Przydział osób do modułów został określony w tablicy .

Moduł	Funkcjonalność	Autor
AmeBoard	edytor grafiki	Paweł Martenka
AmeEdit	edytor kodu	Konrad Siek
AmeGUI	graficzny interfejs użytkownika	Joanna Daukszewicz
AmePlug	łączenie modułów, komunikacja sieciowa	Tomasz Kaszkowiak

Tablica 1.1: Udział poszczególnych autorów pracy

Rozdział 2

Koncepcja rozwiązania problemu

2.1 Propozycja rozwiązania problemu

Projekt inżynierski Amebae ma na celu stworzenie narzędzi do ułatwienia współpracy i komunikacji między programistami pracującymi nad tworzeniem schematów i grafiki oraz wspólną edycję fragmentów kodu. Poprzez współpracę i komunikację rozumie się wymianę informacji przez minimum dwóch użytkowników, przy wykorzystaniu sieci Internet.

Protokołem komunikacyjnym jaki został wykorzystany podczas implementacji realizowanego projektu jest Jabber. Protokół ten jest znany jako *"the Linux of instant messaging"*[Fou05]. Jest bezpłatną alternatywą dla komercyjnych usług przesyłania natychmiastowych wiadomości jak AIM, ICQ, MSN, Yahoo i Gadu-Gadu. Pod nazwą Jabber kryją się strumieniowe protokoły XML oraz technologie umożliwiające wymianę natychmiastowych wiadomości, powiadomień o obecności (*presence*), oraz innych strukturalnych informacji, w czasie zbliżonym do czas rzeczywistego. Technologie Jabbera oferują kilka zalet:

Otwartość: protokoły Jabbera są bezpłatne, otwarte, publiczne i łatwe w zrozumieniu. W dodatku istnieje wiele implementacji klientów, serwerów, komponentów i bibliotek.

Standard: Internet Engineering Task Force (IETF), sformalizował rdzeń strumieniowego XML-owego protokołu jako technologie natychmiastowych wiadomości oraz powiadomień o obecności pod nazwą XMPP i opublikował specyfikacje XMPP jako RFC 3920 i RFC 3921.

Sprawdzony: pierwsze technologie Jabbera zostały rozwijane przez Jeremiego Millera w 1998 i w tej chwili są bardzo stabilne – setki programistów pracują nad technologią Jabber, istnieją dzisiaj tysiące działających serwerów Jabbera w internecie, a miliony osób używają Jabbera jako IM.

Decentralizacja: architektura sieci Jabber jest bardzo podobna do sieci e-mail, z czego wynika że każdy może posiadać swój własny serwer Jabbera.

Bezpieczeństwo: każdy serwer Jabbera może być odizolowany od publicznej sieci Jabbera i zabezpieczony metodami SASL i TLS które są wbudowane w protokół XMPP.

Rozszerzalność: używając XML-owych przestrzeni nazw, każdy może zbudować nową funkcjonalność.

Elastyczność: aplikacje, wykorzystujące protokół Jabber pod postacią komunikatorów internetowych, zawierają narzędzia do współpracy, gry, udostępniania plików, zdalnego monitorowania oraz zarządzania siecią.

Różnorodność: duża liczba firm oraz projektów *Open Source* używa protokołu Jabber do budowy aplikacji czasu rzeczywistego i usług [Fou05](tłum. własne).

Z założeniem, że tworzone narzędzia będą wykorzystywane przez użytkowników różnych systemów operacyjnych, wybór języka programowania w którym projekt miał zostać zaimplementowany, padł na język programowania Java. Kolejną decyzją którą należało podjąć był wybór komunikatora, do przesyłania informacji między użytkownikami.

Kryteria jakie zostały obrane podczas wyboru komunikatora:

1. obsługa protokołu Jabber,
2. wieloplatformowość,
3. mechanizm wtyczek oparty o język Java.

W sieci dostępnych jest kilka komunikatorów które spełniały powyższe wymagania, ale wybrane zostały JBother oraz Jeti.

Początkowo wybór padł na komunikator JBother lecz ostatecznie wybrano Jeti, który posiada aktywną społeczność, forum oraz jest wciąż rozwijany. Dodatkowym argumentem na rzecz Jeti jest wielość istniejących wtyczek. Zalety Jeti:

- przenośność – jedynym wymaganiem jest obsługa przez platformę języka Java w wersji 1.4,
- prosty interfejs użytkownika,
- mechanizm wtyczek – możliwość włączania tylko tych funkcji, które są potrzebne,
- podstawowa obsługa wiadomości indywidualnych oraz rozmów,
- obsługa połączeń SSL oraz Socks proxy,
- proste przesyłanie plików,
- podstawowa obsługa pokoi konferencyjnych,
- logowanie wiadomości,
- formatowanie wiadomości XHTML,
- emotikony, w tym animowane,
- obsługa rejestracji w transportach, logowanie/wylogowanie,
- podstawowe funkcje przeglądania usług,
- możliwość rejestracji nowego konta,
- konsola XML,
- 'metakontakty' (łączy pozycje o tej samej nazwie, pojedyncze kliknięcie na kontakt rozwija listę – również dla różnych zasobów).

Biorąc pod uwagę możliwości komunikatora Jeti oraz stawiane wymagania, cel projektu można określić jako zbudowanie użytecznego zestawu wtyczek dla projektantów-programistów, w oparciu o protokół Jabber i komunikator Jeti.

2.2 Ograniczenia

Podczas prowadzenia projektu programistycznego głównym ograniczeniem jest czas dostępny na realizację. Ograniczenie czasowe wymusza na programistach zastosowanie pewnych rozwiązań które mają wpływ na aplikację pod względem wydajnościowym, funkcjonalnym oraz bezpieczeństwa.

2.2.1 Ograniczenia wydajnościowe

- Podczas przesyłania dużej ilości danych (przy zachowaniu rozsądnego czasu transmisji), ograniczeniem jest przepustowość łącza, oraz wydajność serwerów Jabbera.
- Wydajność analizatora leksykalnego oraz klas `Document` i `JTextPane` ma wpływ na rozmiar przetwarzanego tekstu. Klasy te mają też wpływ na możliwości wyświetlania tekstu.
- Wydajność edytora grafiki AmeBoard zależy w dużej mierze od ilości narysowanych figur, szczególnie z wypełnieniem gradientowym.

2.2.2 Ograniczenia funkcjonalne

- Możliwość edycji zawartości tylko przez jednego użytkownika w danej chwili. Spowodowane jest to brakiem w architekturze jednostki centralnej (serwera), która by zarządzała wymianą informacji.
- Możliwość wymiany informacji w grupie jest wykonywalna pod warunkiem że każdy użytkownik "czatu" ma każdego innego użytkownika biorącego udział w "czacie" dodanego do listy kontaktów. Specyfikacja protokołu Jabber mówi o tym, że użytkownik A ma możliwość sprawdzenia statusu (dostępny, niedostępny) użytkownika B, jeśli został autoryzowany do tego przez użytkownika B.
- Możliwy jest odczyt tylko i wyłącznie poprawnych plików SVG, przy czym obsługiwane pliki są podzbiorem SVG – możliwe jest otworzenie pliku zgodnego z podzbiorem zdefiniowanym w projekcie, natomiast każde wykroczenie poza ten podzbiór nie pozwala na odczyt.
- Ograniczenie wynikające z architektury MVC stosowanej przez bibliotekę Swing. AmeEdit musi odzwierciedlać ten model, jeśli ma być kompatybilny z GUI, czyli musi mieć trzy warstwy które porozumiewają się ze sobą przez zdarzenia.
- Użycie komponentów ciężkich jak JFrame w GUI. Komponenty takie są realizowane poprzez użycie graficznych bibliotek GUI systemu operacyjnego i występują problemy z ich modyfikacją.

2.3 Specyfikacja wymagań

BUC1: Kreślenie figur

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Wykreślenie figury w obszarze rysowania

Zdarzenie:

Priorytet: Wysoki

Źródło: Joanna Daukszewicz

Pozyskano: 11.04.2007

Główny scenariusz:

1. Użytkownik wybiera typ kreślonej figury
2. Użytkownik określa atrybuty kreślenia
3. Użytkownik kreśli figurę po obszarze rysowania
4. System przekazuje figurę do użytkownika zdalnego

BUC2: Ustawienie atrybutów kreślonych figur

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Zmiana atrybutów kreślonych figur

Zdarzenie:

Priorytet: Wysoki

Źródło: Joanna Daukszewicz

Pozyskano: 11.04.2007

Zmodyfikowano: 12.04.2007 (Paweł Martenka)

Główny scenariusz:

1. Użytkownik wybiera grupę atrybutów do modyfikacji
2. Użytkownik zmienia pojedyncze atrybuty
3. System wykorzystuje te ustawienia do rysowania kolejnych figur

BUC3: Przesuwanie narysowanych figur

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Zmiana położenia wybranej figury

Zdarzenie:

Priorytet: Wysoki

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Główny scenariusz:

1. Użytkownik wybiera jedną figurę z obszaru rysowania
2. Użytkownik zmienia jej położenie
3. System zapamiętuje tak zmodyfikowaną figurę
4. System przekazuje zmianę do użytkownika zdalnego

BUC4: Zmiana atrybutów pojedynczej figury

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Zmodyfikowanie właściwości pojedynczej figury

Zdarzenie:

Priorytet: Niski

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Główny scenariusz:

1. Użytkownik wybiera figurę z obszaru rysowania
2. Użytkownik wybiera grupę atrybutów figury
3. Użytkownik modyfikuje wybrane atrybuty
4. System zapamiętuje tak zmodyfikowaną figurę
5. System przekazuje zmianę do użytkownika zdalnego

BUC5: Odczytanie rysunku

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Odczytanie rysunku uprzednio zapisanego do pliku

Zdarzenie:

Priorytet: Wysoki

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Główny scenariusz:

1. Użytkownik chce odczytać rysunek z pliku
2. Użytkownik wybiera odpowiedni plik
3. System wczytuje rysunek do obszaru rysowania
4. System przekazuje rysunek do użytkownika zdalnego

Rozszerzenia:

2.A Gdy system nie znajdzie pliku, informuje o tym użytkownika

3.A W przypadku niepoprawnego formatu pliku, system informuje o tym użytkownika

BUC6: Eksport figury do pliku

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Wyeksportowanie wybranej figury do pliku

Zdarzenie:

Priorytet: Wysoki

Źródło: Paweł Martenka

Pozyskano: : 11.04.2007

Główny scenariusz:

1. Użytkownik wybiera figurę z obszaru rysowania
2. Użytkownik pragnie zapisać wybraną figurę do pliku
3. Użytkownik wybiera nazwę pliku do zapisania
4. System zapisuje figurę do pliku

Rozszerzenia:

1.A Użytkownik może wybrać (po zgrupowaniu) kilka figur (BUC8)

3.A Gdy użytkownik poda już istniejącą nazwę to system o tym informuje

BUC7: Import figury z pliku

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Odczytanie pojedynczej figury

Zdarzenie:

Priorytet: Wysoki

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Główny scenariusz:

1. Użytkownik pragnie zaimportować wcześniej zapisaną figurę
2. Użytkownik wybiera plik do odczytu
3. System wczytuje figurę i umieszcza ją na rysunku
4. System przekazuje zmiany do użytkownika zdalnego

Rozszerzenia:

2.A Jeżeli plik nie istnieje, to system informuje o tym użytkownika

3.A Jeżeli format rysunku jest niepoprawny to system informuje o tym użytkownika

BUC8: Wpisywanie tekstu

Atrybuty:

Główny aktor: Użytkownik

Cel: Wypisanie dowolnego tekstu na obszarze rysowania.

Zdarzenie: Wypisanie tekstu w wybranym miejscu na obszarze rysowania

Priorytet: Wysoki

Źródło: Tomasz Kaszkowiak

Pozyskano: 11-04-2007

Główny scenariusz:

1. Użytkownik: Wybór miejsca wyświetlenia tekstu na obszarze rysowania.
2. Użytkownik: Wpisywanie tekstu w wybrany obszar.
3. System: System przekazuje zmianę do użytkownika zdalnego.

BUC9: : Wybór atrybutów tekstu

Atrybuty:

Główny aktor: Użytkownik

Cel: Dostosowanie wyglądu wyświetlanego tekstu do preferencji użytkownika

Zdarzenie: Modyfikacja atrybutów wyświetlanego tekstu.

Priorytet: Średni

Źródło: Tomasz Kaszkowiak

Pozyskano: 11-04-2007

Powiazanie: BUC8

Główny scenariusz:

1. Użytkownik: Wybór wcześniej wprowadzonego tekstu na obszarze rysowania.
2. Użytkownik: Zmiana atrybutów tekstu (czcionka, wielkość, grubość, kolor itp.).
3. System: Naniesienie zmian na wcześniej wprowadzonym tekście.
4. System: System przekazuje zmianę do użytkownika zdalnego

BUC10: : Skalowanie figur

Atrybuty:

Główny aktor: Użytkownik

Cel: Zmiana rozmiaru wykreślonej figury

Zdarzenie:

Priorytet: Wysoki

Źródło: Tomasz Kaszkowiak

Pozyskano: 11-04-2007

Główny scenariusz:

1. Użytkownik: Wybór wcześniej wykreślonej figury
2. Użytkownik: Zmniejszenie lub zwiększenie jej wymiarów
3. System : System przekazuje zmianę do użytkownika zdalnego

BUC11: Cofanie i powtarzanie polecenia (*undo/redo*)

Atrybuty:

Główny aktor: Użytkownik

Cel: Cofnąć lub powtórzyć zmianę w tekście

Zdarzenie: Zmiana zostanie cofnięta lub powtórzona

Priorytet: Wysoki

Źródło: Tomasz Kaszkowiak

Pozyskano: 11-04-2007

Główny scenariusz:

1. Użytkownik: Wydanie polecenia *undo/redo*.
2. System: Przywraca kod do wymaganej postaci.
3. System: System przekazuje zmianę do użytkownika zdalnego

BUC11: Wybór narzędzia do rysowania

Atrybuty:

Główny aktor: Użytkownik

Cel: Wybór narzędzia do rysowania przez użytkownika

Zdarzenie: System zapamięta parametry danego narzędzia

Priorytet: Wysoki

Źródło: Joanna Daukszewicz

Pozyskano: 11-04-2007

Główny scenariusz:

1. Użytkownik: Wybiera narzędzie do rysowania
2. System: Wyświetla w panelu roboczym atrybuty narzędzia
3. Użytkownik: Ustawia atrybuty narzędzia (BUC2)
4. System: Zapamiętuje parametry narzędzia

BUC13: Grupowanie figur

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Zgrupowanie kilku figur w jedną logiczną całość

Zdarzenie:

Priorytet: Średni

Źródło: Paweł Martenka, Konrad Siek

Pozyskano: 12.04.2007

Główny scenariusz:

1. Użytkownik zaznacza grupę figur
2. Użytkownik grupuje figury
3. System zapamiętuje tak powstały obiekt
4. System przekazuje zmiany do użytkownika zdalnego

Rozszerzenia:

2.A Zaznaczenie pojedynczej figury i zgrupowanie jej nie odnosi żadnych skutków

BUC14: Eksport rysunku

Atrybuty:

Główny aktor: Użytkownik

Cel: Zapisanie rysunku w postaci bitmapy

Zdarzenie:

Priorytet: Średni

Źródło: Paweł Martenka

Pozyskano: 12.04.2007

Główny scenariusz:

1. Użytkownik pragnie zapisać rysunek jako bitmapę
2. Użytkownik podaje nazwę pliku
3. Użytkownik wybiera format zapisu
4. System zapisuje rysunek do pliku

Rozszerzenia:

2.A Jeśli plik o podanej nazwie istnieje to system informuje o tym

BUC15: Wyświetlenie siatki

Atrybuty:

Główny aktor: Użytkownik, użytkownik zdalny

Cel: Wyświetlenie siatki linii pomocniczych na obszarze rysowania

Zdarzenie:

Priorytet: Niski

Źródło: Konrad Siek

Pozyskano: 12.04.2007

Zmodyfikowano:

Główny scenariusz:

1. Użytkownik pragnie wyświetlić siatkę pomocniczą
2. Użytkownik ustawia atrybuty siatki
3. System wyświetla siatkę
4. System przekazuje polecenie wyświetlenia siatki u użytkownika zdalnego

EUC1: Importowanie kodu

Atrybuty:

Główny aktor: Użytkownik, Użytkownik Zdalny

Cel: Załadować do edytora plik z kodem

Zdarzenie: Edytorowi zostanie dostarczony kod

Priorytet: Wysoki

Źródło: Konrad Siek

Pozyskano: 05-04-2007

Główny scenariusz:

1. Użytkownik: Wskazuje plik z kodem do załadowania
2. System: Koloruje składnie (EUC2)
3. System: Pokazuje kod Użytkownikowi
4. System: Pokazuje kod Użytkownikowi Zdalnemu

EUC2: Kolorowanie składni

Atrybuty:

Główny aktor: Użytkownik

Cel: Oznaczenie elementów styli elementów składni

Zdarzenie: Edytor przekształci kod wejściowy na kod z pokolorowana składnia

Priorytet: Wysoki

Źródło: Konrad Siek

Pozyskano: 05-04-2007

Główny scenariusz:

1. Użytkownik: Dostarcza nowy plik
2. System: Rozpoznaje język w jakim napisany jest plik
3. System: Oznacza elementy składni

Rozszerzenia:

2.A. System: Nie rozpoznał języka

2.A.1. Użytkownik: Narzuca język

2.A.1.A Użytkownik: Wyłącza kolorowanie składni

EUC3: Aktywny kursor (kursor online)

Atrybuty:

Główny aktor: Użytkownik, Użytkownik Zdalny

Cel: Wyświetlanie zmian oraz położenia kursora użytkownika aktywnego w oknie użytkownika pasywnego

Zdarzenie: System wyświetli zmiany położenia kursora użytkownika aktywnego w oknie użytkownika pasywnego

Priorytet: Wysoki

Źródło: Tomasz Kaszkowiak

Pozyskano: 07-04-2007

Zmodyfikowano: 11-04-2007, 12-04-2007 (Konrad Siek)

Główny scenariusz:

1. Użytkownik: Zmienia położenie kursora
2. System: Wyświetla w oknie użytkownika zdalnego

Rozszerzenia:

1.A Użytkownik zaznacza fragment tekstu

EUC4: Cofanie i powtarzanie polecenia (undo/redo)

Atrybuty:

Główny aktor: Użytkownik, Użytkownik Zdalny

Cel: Cofnąć lub powtórzyć zmianę w tekście

Zdarzenie: Zmiana zostanie cofnięta lub powtórzona

Priorytet: Wysoki

Źródło: Konrad Siek

Pozyskano: 11-04-2007

Główny scenariusz:

1. Użytkownik: Podaje instrukcje cofnięcia czynności, powtórzenia czynności.
2. System: Przywraca kod do wymaganej postaci.
3. System: Pokazuje kod Użytkownikowi Zdalnemu.

Rozszerzenia:

2.A Cofniecie lub powtórzenie nie może być wykonane.

2.A.1 System uaktywnia domyślny sygnał dźwiękowy u Użytkownika.

EUC5: Wycinanie, Wklejanie (*cut/paste*)

Atrybuty:

Główny aktor: Użytkownik, Użytkownik Zdalny

Cel: Wyciąć lub wkleić fragment tekstu

Zdarzenie: Fragment tekstu zostanie wycięty lub wklejony

Priorytet: Wysoki

Źródło: Konrad Siek

Pozyskano: 11-04-2007

Główny scenariusz:

1. Użytkownik: Podaje instrukcje wycięcia lub wklejenia fragmentu tekstu.
2. System: Wycina lub wkleja fragment tekstu.
3. System: Pokazuje kod Użytkownikowi Zdalnemu.

EUC6: Zwijanie i rozwijanie kodu

Atrybuty:

Główny aktor: Użytkownik, Użytkownik Zdalny

Cel: Rozwinąć luz zwinąć blok kodu

Zdarzenie: Fragment tekstu zostaje zwinięty do jednej linii lub rozwinięty

Priorytet: Niski

Źródło: Konrad Siek

Pozyskano: 11-04-2007

Zmodyfikowano:

Główny scenariusz:

1. Użytkownik: Wydaje instrukcje zwinięcia lub rozwinięcia fragmentu tekstu.
2. System: Zwija lub rozwija fragment tekstu Użytkownika.
3. System: Zwija lub rozwija fragment tekstu Użytkownika Zdalnego.

EUC7: Przywracanie wcześniejszej sesji

Atrybuty:

Główny aktor: Użytkownik, Użytkownik Zdalny

Cel: Załadować do edytora plik z kodem oraz jego historie zmian

Zdarzenie: Edytorowi zostanie dostarczony kod oraz historia

Priorytet: Średni

Źródło: Konrad Siek

Pozyskano: 11-04-2007

Zmodyfikowano:

Główny scenariusz:

1. Użytkownik: Wskazuje plik sesji do załadowania.
2. System: Koloruje składnie (EUC2).
3. System: Pokazuje kod Użytkownikowi.
4. System: Pokazuje kod Użytkownikowi Zdalnemu.

EUC8: Zapisywanie kodu do pliku

Atrybuty:

Główny aktor: Użytkownik

Cel: Zapisać kod do pliku

Zdarzenie: Dane z edytora zostają zapisane

Priorytet: Wysoki

Źródło: Konrad Siek

Pozyskano: 11-04-2007

Zmodyfikowano:

Główny scenariusz:

1. Użytkownik: Wydaje polecenie zapisania kodu.
2. Użytkownik: Wprowadza atrybuty pliku do zapisu.
3. System: Zapisuje plik na dysku.

Rozszerzenia:

1.A Użytkownik wydaje polecenie zapisania sesji

1.B Użytkownik wydaje polecenie zapisania kodu jako plik HTML

EUC9: Importowanie definicji składni z pliku

Atrybuty:

Główny aktor: Użytkownik

Cel: Doda nową definicję składni do edytora

Zdarzenie: Definicja składni jest dodana do puli dostępnych składni edytora

Priorytet: Wysoki

Źródło: Tomasz Kaszkowiak

Pozyskano: 11-04-2007

Zmodyfikowano: 12-04-2007 (Konrad Siek)

Główny scenariusz:

1. Użytkownik: Wybiera plik z definicja składni.
2. System: Weryfikuje poprawność wczytanego pliku.
3. Użytkownik: Podaje nazwę wczytanej składni.
4. System: Dodaje nową pozycję do opcji wyboru składni.

Rozszerzenia:

2.A System uznaje plik za niepoprawny

2.A.1 Odrzuca plik.

2.A.2 Informuje Użytkownika o błędach.

EUC10: Dostosowanie formatu kolorowania składni

Atrybuty:

Główny aktor: Użytkownik

Cel: Dostosowanie kolorów wyświetlonego tekstu do preferencji użytkownika

Zdarzenie: Dane w edytorze zostają pokolorowane według ustalonych przez użytkownika kolorów

Priorytet: Średni

Źródło: Tomasz Kaszkowiak

Pozyskano: 11-04-2007

Zmodyfikowano: 12-04-2007 (Konrad Siek)

Główny scenariusz:

1. Użytkownik: Wybiera format (kolor, styl) zbiorów słów opisanych przez składnie.
2. System: Koloruje składnie (EUC2).

EUC11: Odświeżenie zawartości edytora @usunięte 12-04-2007 (Konrad Siek)

EUC12: Drukowanie pliku z kodem

Atrybuty:

Główny aktor: Użytkownik

Cel: Wykonanie wydruku kodu

Zdarzenie: Wykonany zostaje wydruk kodu

Priorytet: Niski

Źródło: Konrad Siek

Pozyskano: 12-04-2007

Główny scenariusz:

1. Użytkownik: Konfiguruje opcje wydruku.
2. Użytkownik: Wydaje polecenie drukowania.
3. System: Wysyła plik do druku drukarce.

Rozszerzenia:

3.A System nie jest w stanie przesłać danych do drukarki.

3.A.1 System informuje użytkownika o błędzie.

3.B Użytkownik nakazał drukowanie pokolorowanego pliku.

3.B.1 System wysyła do drukarki sformatowany plik (EUC2).

EUC13: Szukanie fragmentu tekstu

Atrybuty:

Główny aktor: Użytkownik, Użytkownik zdalny

Cel: Odnalezienie fragmentu w kodzie

Zdarzenie: Fragment zostaje odnaleziony i wskazany

Priorytet: Średni

Źródło: Konrad Siek

Pozyskano: 12-04-2007

Główny scenariusz:

1. Użytkownik: Definiuje fragment tekstu do odnalezienia.
2. System: Aplikuje podświetlenie dla wszystkich znalezionych elementów.
3. System: Aplikuje podświetlenie dla wszystkich znalezionych elementów u użytkownika zdalnego.
4. System: Odnajduje kolejne wystąpienie fragmentu w kodzie.
5. System: Odnajduje kolejne wystąpienie fragmentu w kodzie u użytkownika zdalnego.

Rozszerzenia:

1.A Użytkownik definiuje wyrażenie regularne.

1.A.1 System sprawdza poprawność wyrażenia regularnego.

1.A.1.A Wyrażenie regularne jest nieprawidłowe.

1.A.1.A.1 System informuje użytkownika o błędzie.

1.A.1.A.2 Użytkownik poprawia błąd lub rezygnuje z szukania.

EUC14: Zamiana fragmentu kodu na inny

Atrybuty:

Główny aktor: Użytkownik, Użytkownik zdalny

Cel: Odnalezienie fragmentu w kodzie

Zdarzenie: Fragment zostaje odnaleziony i wskazany

Priorytet: Średni

Źródło: Konrad Siek

Pozyskano: 12-04-2007

Główny scenariusz:

1. Użytkownik: Definiuje fragment tekstu oraz jego zamiennik.
2. System: Odnajduje i podświetla znalezione fragmenty u użytkownika (EUC13).
3. System: Odnajduje i podświetla znalezione fragmenty u użytkownika zdalnego (EUC13).
4. System: Zamienia fragmenty tekstu.
5. System: Koloruje składnie (EUC2).
6. Użytkownik: Nakazuje znaleźć i zamienić kolejny fragment.

Rozszerzenia:

6.A Użytkownik nakazał zamienić wszystkie wystąpienia.

6.A.1 System automatycznie zamienia wszystkie wystąpienia.

2.4 Wybór narzędzi

Eclipse : Eclipse to udostępniona na licencji *Open Source* platforma służąca do integracji różnego typu narzędzi programistycznych. Działają one w ramach jednego środowiska, co sprawia, że użytkownik przekonany jest, że ma do czynienia z zaawansowanym, ale pojedynczym produktem. Środowisko jest w pełni konfigurowalne i rozszerzalne.

Standardowo Eclipse wyposażony jest w zestaw narzędzi do tworzenia aplikacji w języku Java. Oprócz tego, że zapewnia duży komfort pracy i jest wyjątkowo elastyczne, bardzo

blisko integruje się z innymi modułami. Starannie przemyślane i odpowiednio zaprojektowane interfejsy udostępniane w ramach platformy stanowią wręcz zaproszenie dla każdego, kto chciałby sam dodać do Eclipse'a wtyczkę. Środowisko dostępne jest w wersjach dla wielu różnych systemów operacyjnych i (mimo że związane blisko z Javą) nie preferuje żadnego języka programowania. Dostępne są moduły do pisania w języku C, C++, HTML, a nawet w COBOL'u. A ponieważ jest to produkt *Open Source*, mamy dostęp do kodu źródłowego, i, co najważniejsze, z Eclipse korzystamy bezpłatnie [SS05].

System kontroli wersji (SVN) : Subversion jest potężnym systemem wersjonowania, zaprojektowanym od podstaw w celu zaspokojenia nowopowstałych potrzeb w dziedzinie wytwarzania oprogramowania. SVN czerpie z doświadczeń zdobytych z poprzedniego systemu, biorąc najlepsze elementy interfejsu dostarczonego przez bardzo popularnego CVS'a i integrując go z dobrze skonstruowanym, wysokiej klasy rdzeniem zbudowanym w celu zaspokojenia potrzeb nowej epoki rozwoju oprogramowania.[DB86](tłum. własne)

Mantis (Bug tracker): Mantis jest to darmowy, sieciowy system zgłaszania błędów (*bug tracker*). Został napisany w języku PHP oraz używa bazy danych (MySQL, PostgreSQL, MS SQL), a także serwera WWW. System obsługiwany jest z poziomu przeglądarki internetowej. Mantis oparty jest o licencję GNU General Public License (GPL) [Boc07] (tłum. własne).

Rozdział 3

Projekt i implementacja

3.1 Założenia projektowe

Główną ideą przyświecającą projektowi jest stworzenie narzędzia pomocnego w szczególności programistom i projektantom oprogramowania w pracy zespołowej, dając im możliwość prowadzenia komunikacji za pomocą nie tylko słów czy głosu, ale także rysując diagramy i schematy lub pokazując konkretne rozwiązania pisząc lub przeglądając części kodu źródłowego wspólnie. Istotnym założeniem jest wbudowanie projektu w komunikator internetowy, który stał się jednym z ważniejszych środków komunikacji internetowej.

W swojej pierwszej wersji zakłada się, że projekt będzie umożliwiał pracę z edytorem kodu oraz z edytorem grafiki wektorowej. Uszczegóławiając, edytor kodu, z założenia, będzie umożliwiał pisanie kodu źródłowego przez jednego z użytkowników i oglądanie go przez wszystkie podłączone osoby. Posiadał on będzie możliwość kolorowania kodu w sposób odpowiedni dla danego języka, używania zakreślacza do pokazania komentowanych fragmentów tekstu a także narzędzia pozwalającego na wyszukiwanie i zamianę fragmentów tekstu.

Założenia odnoszące się do drugiej z wtyczek, to przede wszystkim możliwość rysowania figur, ścieżek i tekstów, grupowania ich i zapisywania jako odrębne, wielokrotnie wykorzystywane schematy. Ponadto edytor grafiki wektorowej zapewnić powinien nanoszenie modyfikacji narysowanych już figur, takich jak skalowanie czy przemieszczenie. Zakładane jest także dostarczenie funkcji eksportowania danych do formatu SVG, co pozwoli na otwarcie lub modyfikację wytworzonych schematów w innych programach.

Więcej na temat zakładanych funkcji oprogramowania znajduje się w rozdziale koncepcja rozwiązania problemu, w podrozdziale specyfikacja wymagań.

Architektura oprogramowania została zaprojektowana w sposób, który umożliwi jak największe możliwe uniezależnienie jego modułów składowych od siebie nawzajem. Projekt Amebae zbudowany jest z trzech modułów, połączonych za pomocą modułu nadzorującego, AmePlug, wedle wzorca projektowego fasada [AS05]. Poszczególne moduły posiadają swoją odpowiedzialność, którą wykonują w sposób dopuszczający hermetyzacje. Komunikacja związanych ze sobą par modułów przebiegać powinna na dwa sposoby:

- poprzez obiekt konfiguracji (klasy `AmeBoardConfiguration` , `AmeEditConfiguration`), gdzie ustawiane są wartości na których należy działać.
- poprzez wysyłanie i nasłuch zdarzeń.

Zaprojektowanie w ten sposób współpracy modułów ma celu możliwość dodawania kolejnych wtyczek, które posiadałyby dodatkowe funkcje. Możliwość rozbudowy oprogramowania była jed-

nym z dodatkowych założeń projektu. Szczególnie przydatnym narzędziem, którego dodanie powinno zostać rozważone w wypadku rozszerzania pakietu Amebae byłyby człony do edycji równań.

Założone w projekcie rozdzielenie modułów także powoduje, że możliwa jest podmiana jednego z fragmentów oprogramowania w przyszłości, bez naruszania jego pozostałych części, czego zaletą miałoby być mniej pracochłonne utrzymanie kodu.

Rozszerzalność przyjmuje także postać możliwości dodawania nowych elementów związanych z treścią edytowanych obiektów. Edytor grafiki dawać będzie możliwość tworzenia nowych schematów kolorowania języków programowania, poprzez pliki XML. Podobnie jest z edytorem grafiki, który daje możliwość zapisywania nowych schematów obiektów w plikach SVG.

Istotną częścią projektu jest także możliwość uruchamiania programu w różnych środowiskach i na różnych komputerach, a więc zapewnienie przenośności. Osiągnięcie tego celu ma być możliwe dzięki wykorzystaniu języka Java, który poprzez mechanizm maszyny wirtualnej oddziela program od konkretnego komputera. Za istotne w projekcie także zostało przyjęte w dziedzinie przenośności zapewnienie założonego wyglądu interfejsu użytkownika niezależnie od czynników zewnętrznych.

3.2 Projekt szczegółowy

3.2.1 AmePlug

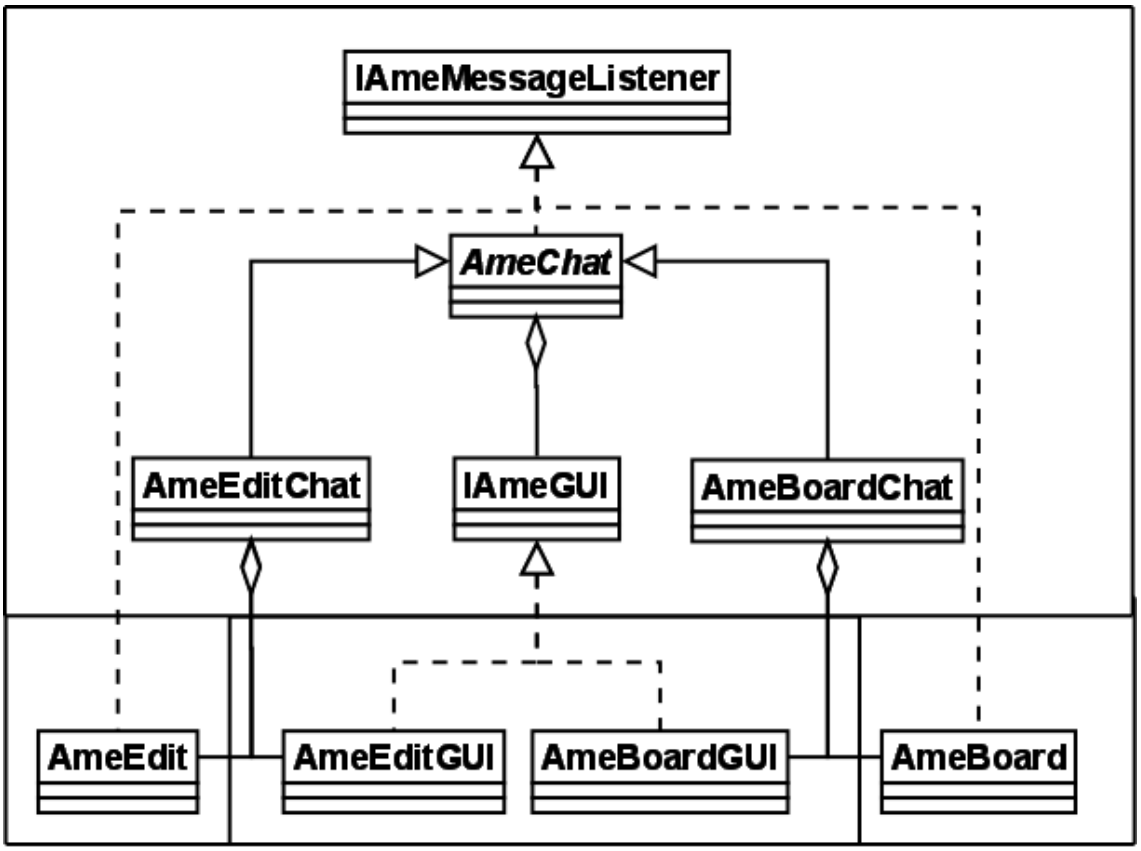
AmePlug jest modułem łączącym w jedną całość pozostałe moduły projektu. AmePlug można przedstawi jako pewnego rodzaju *chat*, a każde zestawienie instancji modułów (AmeGUI + AmeEdit oraz AmeGUI + AmeBoard) tworzą *chatroom*. Modułem ten nadzoruje *chatroom*-y, przechowuje informacje o użytkownikach, ma za zadanie zarządzania rolami użytkowników oraz całego ruchu komunikacyjnego, jaki między nimi zachodzi. Do jego głównych zadań należą:

- odbieranie zdarzeń (informacji) od modułów AmeBoard i AmeEdit,
- wysyłanie wiadomości przez komunikator Jeti do zdalnych modułów AmePlug,
- odbieranie wiadomości od zdalnych modułów AmePlug,
- przekazywanie zdarzeń (informacji) do modułów AmeBoard i AmeEdit,
- zarządzanie użytkownikami *czatu*,
- zarządzanie rolami użytkowników znajdujących się na czacie,
- synchronizacja informacji.

Komunikacja z modułami

Jednym z głównych zadań AmePlug jest komunikowanie się z modułami AmeBoard i AmeEdit, ma to na celu wymianę informacji między instancjami lokalnymi a zdalnymi tych modułów. Komunikacja miedzy AmePlug a AmeBoard oraz AmeEdit zachodzi za pomocą zdarzeń. Moduły te posiadają implementacje interfejsu `IAmeMessageListener` za pomocą którego zgłaszane są zdarzenia zarówno do AmePlug jak i do AmBoard oraz AmeEdit. Podczas komunikacji przekazywany jest obiekt typu `AmeMessageEvent` . Komunikację miedzy modułami przedstawiono na rysunku 3.2.

Moduł AmeGUI również musi wymieniać pewne informacje z modułem AmePlug, a mianowicie AmePlug wysyła do interfejsu listę dostępnych użytkowników, natomiast AmeGUI wysyła zdarzenia wywołujące odpowiednie metody pozwalające manipulować czatem. Wszystko odbywa się przez



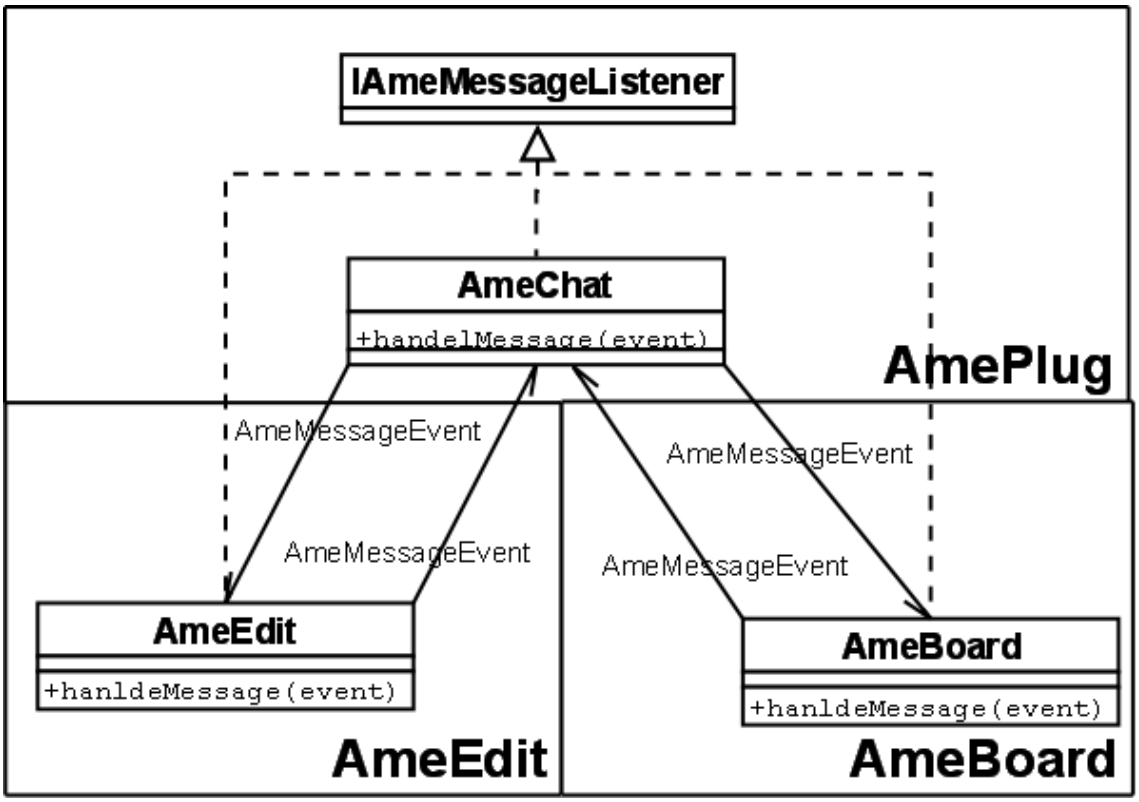
Rysunek 3.1: Diagram klas połączenia modułów

implementacje interfejsu w module AmeGUI, jakim jest `IAmeGUI` który udostępnia zestaw metod pozwalających oprogramować odpowiednie elementy interfejsu użytkownika.

Komunikacja z komunikatorem Jeti

Łącznikiem miedzy AmePlug a komunikatorem Jeti, jest obiekt typu `Backend` , który podczas inicjalizacji pluginu jest przekazywany jako parametr. Klasa `Backend` udostępnia szereg metod, w skład których wchodzi metoda `Send(Message)` , za pomocą której możliwe jest wysyłanie wiadomości do użytkowników sieci Jabber oraz metoda `addExtensionHandler (String,ExtensionHandler)` która jest odpowiedzialna za wywołanie handlera obsługującego wiadomości.

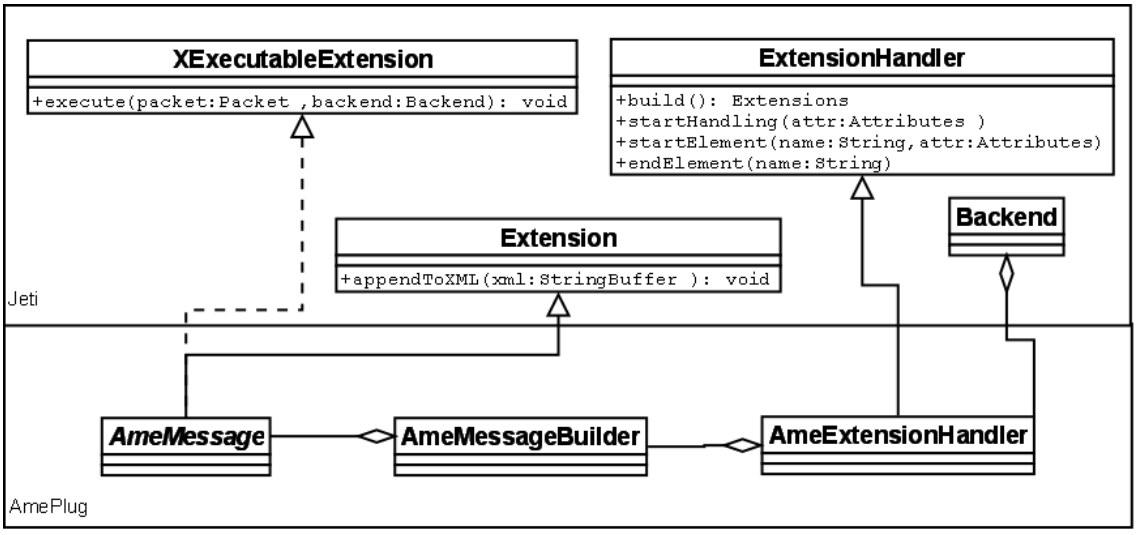
Aby wysłać wiadomość do użytkownika zdalnego konieczne jest wykorzystanie do tego metody send klasy `Backend` , która jako parametr przyjmuję obiekt klasy `Message` , do którego stworzenia potrzebne jest adres odbiorcy oraz obiekt klasy implementującej interfejs `XExecutableExtension` . `XExecutableExtension` udostępnia metodę `execute(Packet,Bacend)` która jest wywoływana przez komunikator Jeti użytkownika zdalnego. Implementacja tej metody powinna zawierać, instrukcje na temat tego ma się stać z wiadomością po jej odbiorze. Przed wysłaniem samej wiadomości potrzebne jest jej przygotowanie, a dokładnie przygotowanie obiektu typu `String` reprezentującego wiadomość w postaci XML. Bezpośrednio nie możemy przekazać do komunikatora Jeti wiadomością formatu XML, możemy to jedynie zrobić przez przeciążenie metody `appendToXML(StringBuffer)` udostępnionej przez klasę `Extension` . Metoda `appendToXML (StringBuffer)` przekazuje obiekty typu `StringBuffer` do którego dopisujemy naszą wiadomość w postaci XML. Jest to spowodowane tym że Jeti wyręcza nas w adresacji wiadomości oraz dopisuje z przodu i na końcu wiadomości znaczniki protokołu Jabber.



Rysunek 3.2: Odbieranie i wysyłanie komunikatów z modułami AmeBoard i AmeEdit

Jak wyżej zostało wspomniane, za odbiór wiadomości odpowiedzialna jest klasa dziedzicząca z klasy `ExtensionHandler` , która należy dodać do obiektu typu `Backend` podczas inicjalizacji pluginu. Klasa `ExtensionHandler` posiada cztery metody które należy przeciążyć do poprawnego rozpoznania, przetworzenia danych z odebranej wiadomości oraz utworzenia obiektu reprezentującego typ przesłanej wiadomości.

Rysunek 3.3 przedstawia diagram klas które mają za zadanie przygotowanie wiadomości do wysłania, odebranie, a następnie wykonanie na niej operacji.



Rysunek 3.3: Diagram klas połączenia modułu AmePlug z komunikatorem Jet

Role użytkowników

Z uwagi na założenie, że tylko jeden użytkownik ma możliwość edycji w modułach AmeEdit oraz AmeBoard, wyróżnia się dwa rodzaje użytkowników:

Editor Użytkownik edytujący

Viewer Użytkownik oglądający

Użytkownik z prawami edycji jest użytkownikiem uprzywilejowanym, i posiada więcej możliwości od użytkownika w roli oglądającego (*viewer*). W ich zakres wchodzą:

- Prawo o wprowadzania zmian w tekście(AmeEdit) oraz zmian w obszarze rysowania(AmeBoard)
- Możliwość dodawania nowych użytkowników do czatu
- Możliwość wyrzucenia danej osoby z czatu
- Oddawania prawa edycji użytkownikowi w roli oglądającego

Użytkownik *Viewer* ma tylko i wyłącznie możliwość śledzenia zmian.

Wyróżnia się dwa rodzaje zmiany roli, poprzez oddanie prawa edycji na rzecz innego użytkownika, oraz podczas przejścia edytora w stan *offline*, który może być wywołany przez zerwanie połączenia, zakończenie działania komunikatora lub na życzenie użytkownika. Podczas gdy użytkownika edytujący przechodzi w stan offline, zachodzi konieczność automatycznego przekazania roli edytora innemu użytkownikowi, znajdującemu się na czacie i będącemu online.

Synchronizacja

Synchronizacja czatu polega na uzyskaniu takiego samego stanu u wszystkich użytkowników znajdujących się na czacie. Ma na celu pobranie wszystkich danych takich jak:

- lista użytkowników
- aktualny licznik wiadomości
- treść czatu (schemat lub dokument)

aby każdy użytkownik czatu miał identyczną zawartość. Podczas synchronizacji zachodzi konieczność komunikacji z modułami AmeEdit oraz AmeBoard, po to aby pobrać oraz ustawić zawartość czatu. Odbywa się to przez implementacje interfejsu `IAmeContent` przez oba moduły. Interfejs udostępnia dwie metody, `setContent(List<Map<String,String>>)` oraz `getContent()` . Metoda `getContent` używana jest po stronie użytkownika od którego pobierane są dane i zwraca obiekty typu `List<Map<String,String>>` , natomiast `setContent` u użytkownika synchronizującego się. Wyróżnia się trzy sytuacje w których użytkownik poddawany jest procesowi synchronizacji.

- Podczas dołączenia się do czatu. Po zatwierdzeniu zaproszenia do czatu użytkownika zaczyna się synchronizować.
- Podczas gdy użytkownika znajdował się na czacie, ale przeszedł w stan *offline* a następnie w stan online.
- Podczas gdy użytkownik uczestniczył w czacie ale działanie programu Jeti zostało przerwane i użytkownik wznowił działanie programu.

Synchronizacja czatu jest funkcją bardzo przydatną ponieważ gwarantuje tą samą treść czatu wszystkim użytkownikom bez względu na czas przebywania na czacie.

Struktura wiadomości

Wiadomości przesyłane miedzy użytkownikiem lokalnym, a użytkownikiem zdalnym mają postać XML. Podstawowa wiadomość protokołu Jabber przedstawiona jest na rysunku 3.4. Zadaniem modułu AmePlug jest przesłać dane od jednego użytkownika do drugiego, za pomocą wiadomości Jabber. AmePlug za pomocą metoda udostępnionych przez komunikator Jeti umieszcza wszystkie niezbędne dane do przesłania w blok <data> o przestrzeni nazw ‘‘jeti:amebae’’ , następnie blok <data> wraz z danymi umieszczany jest między tagi <message> . Na rysunku 3.5 przedstawiono przykładową wiadomość utworzoną przez moduł AmePlug.

```
<message to="receiver@jabber.org/JETI$" from="sender@jabberorg/JETI">
</message>
```

Rysunek 3.4: Podstawowa wiadomość protokołu Jabber

```
<message to="receiver@jabber.org/JETI"  from="sender@jabberorg/JETI">
      <data xmlns="jeti:amebae" type="ActionMessage">
            <author>sender@jabber.org/JETI</author>
            <chatType>AmeTest</chatType>
            <counter>0</counter>
            <thread>22b0f0117a1e37a6f</thread>
      </data>
</message>
```

Rysunek 3.5: Wiadomość protokołu Jabber z rozszerzonymi danymi

3.2.2 AmeBoard

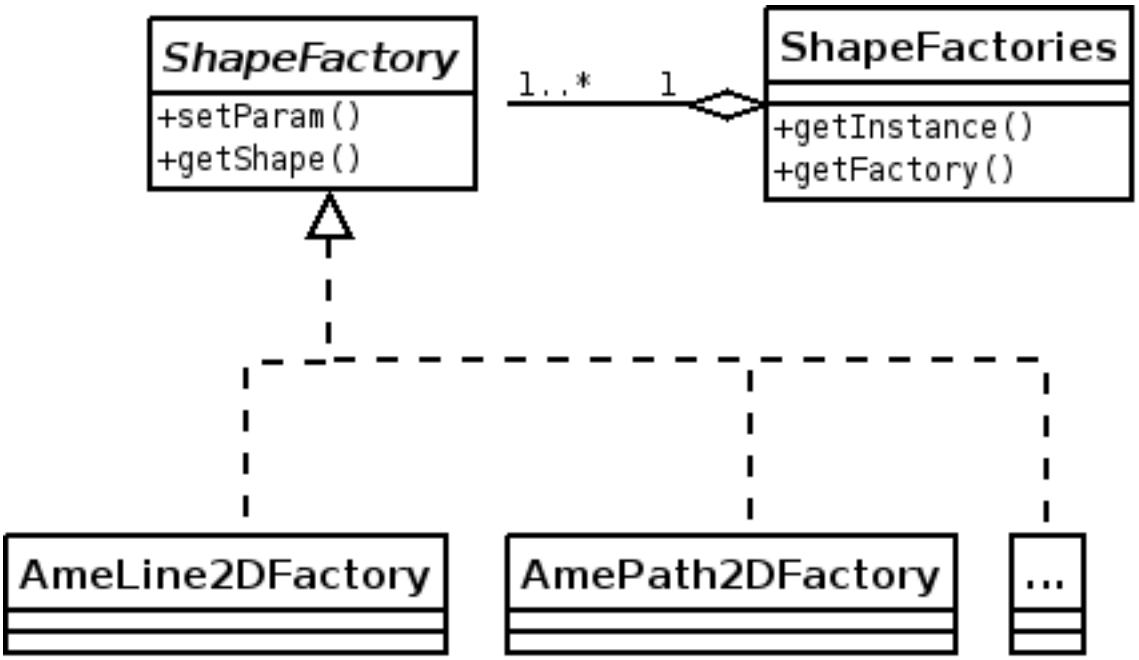
AmeBoard jest edytorem grafiki wektorowej, przy czym wszystkie operacje wykonane przez użytkownika są propagowane do odbiorców zdalnych. Edycja grafiki ma zapewniać podstawowe funkcje dostępne w popularnych edytorach: rysowanie prostych figur z atrybutami obrazu, dodawanie napisów, manipulacje.

Pierwowzorem dla AmeBoard były edytory grafiki wektorowej Inkscape i Corel Draw, edytor diagramów Dia oraz edytor grafiki rastrowej GIMP. Pierwsze dwa programy stanowiły podstawę do zaprojektowania głównych funkcji AmeBoard. Edytor Dia jest bardzo przydatnym narzędziem dla każdego projektanta i inżyniera, pozwala tworzyć większe struktury z mniejszych elementów, co stało się inspiracją do zaprojektowania modułu eksportu i importu klipartów. GIMP, a także Dia, jest podstawą do zaprojektowania interfejsu graficznego AmeBoard, który składa się z kilku mniejszych okien zamiast jednego.

Od strony sieciowej AmeBoard jest specyficzną odmianą komunikatora, w którym środkiem przekazu jest grafika. Komunikacją sieciową, a dokładniej komunikacją z modułem AmePlug, zajmuje się zestaw kilku modułów i funkcji, które przetwarzają obraz w wiadomości o ustalonym formacie.

Obszar rysowania

AmeBoard jest edytorem graficznym, więc kluczowym elementem aplikacji jest obszar rysowania. Pierwowzorem fizycznym dla obszaru rysowania jest tzw. ‘biała tablica’, po której pisze się i rysuje za pomocą pisaków suchościeralnych. Tablica jest bardzo pomocna przy przykazywaniu wiedzy innym osobom, także większym grupom, np. w trakcie wykładów akademickich.



Rysunek 3.6: Fabryki figur

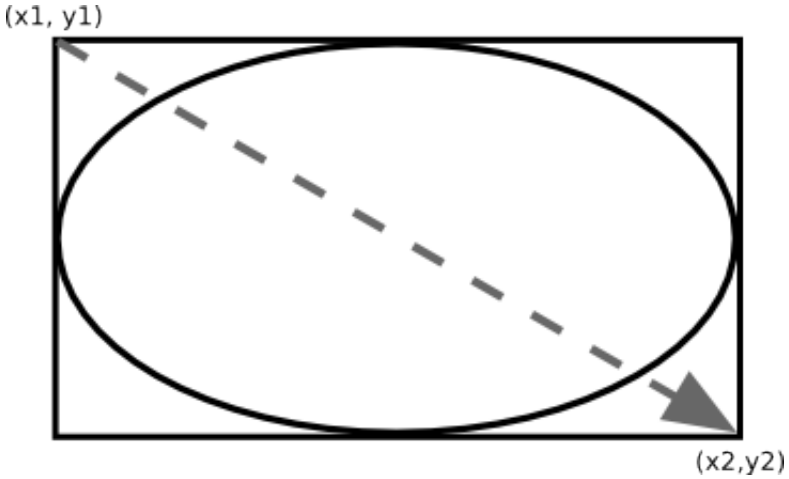
W aplikacji AmeBoard obszarem rysowania jest standardowy panel biblioteki graficznej języka Java - `javax.swing.JPanel` . W standardowych zastosowaniach panel służy za kontener dla innych komponentów, pomaga rozmieścić komponenty w dowolny sposób w interfejsie graficznym. Zaletą panelu, wykorzystaną w AmeBoard, jest łatwość rysowania na nim. `JPanel` pozwala na ustawienie koloru tła, a także na rysowanie dowolnych kształtów z różnymi właściwościami, np. kolorem, wypełnieniem gradientowym, stylem linii.

Fabryki figur

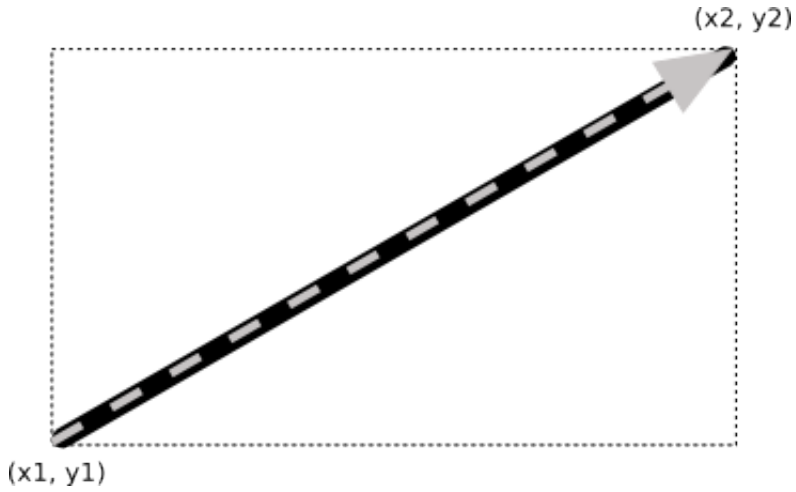
Fabryki figur służą tworzeniu obiektów figur na żądanie. Wybranie przez użytkownika dowolnego narzędzia do rysowania określonej figury, skutkuje zapamiętaniem odpowiedniej fabryki. Kiedy pojawia się potrzeba stworzenia figury, najpierw do fabryki zapisywane są parametry, a następnie dana figura jest produkowana. Wszystkie fabryki korzystają z jednego interfejsu, więc w chwili produkcji moduł, który żąda produkcji, nie wie, jaką figurę otrzyma. Dzięki temu moduł rysujący zajmuje się rysowaniem figur niezależnie od ich typu. Na rysunku 3.6 został przedstawiony diagram klas fabryk wraz z dodatkową klasą `ShapeFactories` przechowującą instancje wszystkich fabryk. Przy tworzeniu tej klasy został zastosowany wzorzec projektowy singletonu - w trakcie działania AmeBoard istnieje tylko jedna instancja kolekcji fabryk.

Moduł rysowania prostych figur

Dla potrzeb projektu przyjęto następującą definicję: przez proste figury należy rozumieć figury, które dają się określić jednoznacznie pewnym wektorem, przy czym wektor ten jest przekątną obrysu prostokątnego figury. Obrys prostokątny figury jest to prostokąt opisany na figurze, przy czym boki tego prostokąta są równoległe do osi OX i OY. Dzięki takiemu podejściu do problemu, każda figura ma punkt początkowy i końcowy wynikający z wektora. Nie jest to istotne dla prostokąta, ale ma znaczenie dla odcinka, szczególnie w trakcie manipulacji i pozwala to na ujednolicenie rozwiązania problemu. Do figur prostych należą prostokąt, prostokąt z zaokrąglonymi brzegami, odcinek, elipsa. Przykładowe figury proste z wektorem zostały przedstawione na ry-



Rysunek 3.7: Prostokąt i elipsa



Rysunek 3.8: Odcinek z obrysem

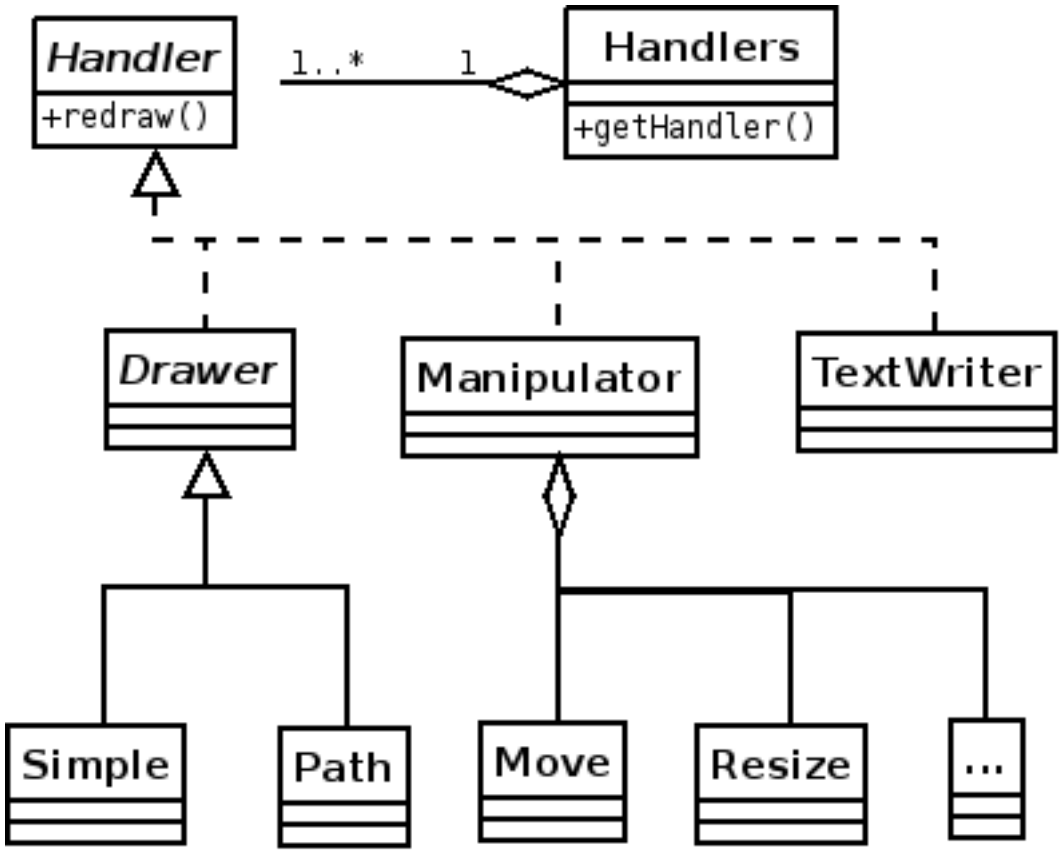
sunkach: 3.7 oraz 3.8. Moduł rysowania prostych figur otrzymuje zdarzenia myszy i odpowiednio na nie reaguje. Wciśnięcie klawisza myszy na obszarze rysowania rozpoczyna rysowanie, przesuwanie kursora rysuje figurę, wyciśnięcie klawisza powoduje zapamiętanie powstałej figury prostej. Figura jest dopasowana do obrysu prostokątnego powstałego z punktu wciśnięcia klawisza myszy i punktu wyciśnięcia.

Moduł rysowania ścieżek

Rysowanie ścieżek ma inny charakter niż rysowanie prostych figur. Dla potrzeb projektu przyjęto, że ścieżka to pewien zbiór punktów połączonych ze sobą odcinkami lub łukami.

Wciśnięcie lewego klawisza myszy na obszarze rysowania, przesuwanie kursora i wyciśnięcie go powoduje powstanie ścieżki, która odpowiada ruchowi kursora.

Wciśnięcie i wyciśnięcie lewego klawisza myszy na obszarze rysowania rozpoczyna rysowanie łamanej. Przesuwanie kursora pokazuje odcinek pomiędzy punktem rozpoczęcia rysowania a aktualną pozycją kursora. Kolejne wciśnięcie i wyciśnięcie lewego klawisza myszy dodaje do łamanej punkt i rozpoczyna rysowanie kolejnej części linii łamanej. Wciśnięcie prawego klawisza myszy kończy rysowanie. Efektem jest linia łamana, której punktami są kolejne miejsca wciśnięcia i wyciśnięcia klawisza myszy.



Rysunek 3.9: Moduły rysowania, manipulacji i wprowadzania tekstu

Moduł manipulacji

Moduł ten jest odpowiedzialny za obsługę operacji wyboru, przesunięcia, skalowania, grupowania, rozgrupowania, usunięcia, zmiany atrybutów, wycięcia i wstawienia wybranego obiektu z obszaru rysowania.

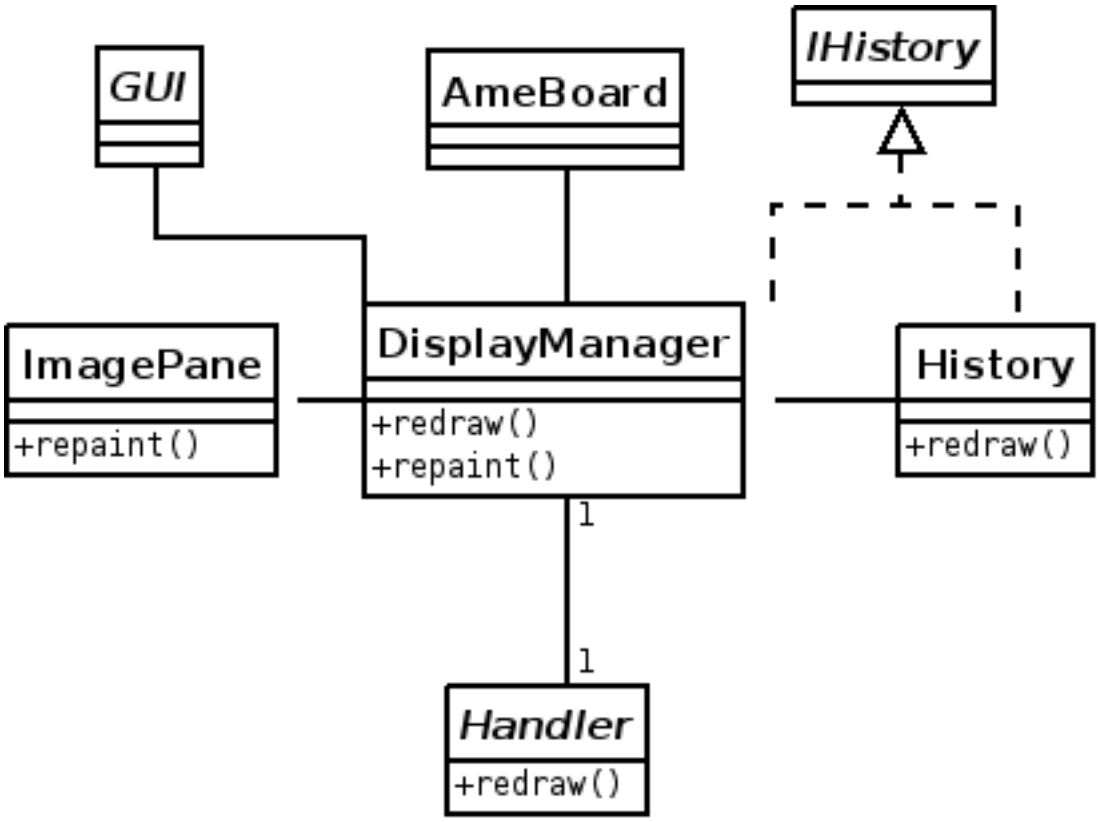
Operacje wyboru, przesunięcia i skalowania opierają się o zdarzenia myszy. Wszystkie operacje oprócz wyboru figury z obszaru rysowania muszą być poprzedzone wcześniejszym wyborem jakiegokolwiek obiektu z obszaru rysowania. Zachowanie poszczególnych operacji manipulacji jest zbliżone do większości programów do edycji grafiki wektorowej.

Wspólna struktura dla modułów manipulacji, rysowania i wprowadzania tekstu została przedstawiona na rysunku 3.9. Warto zauważyć, że instancje wszystkich modułów obsługi znajdują się w jednej klasie, która następnie jest wykorzystywana przez pozostałe elementy aplikacji.

Zarządca wyświetlania

Jednym z najważniejszych modułów AmeBoard jest zarządca wyświetlania. Zadania zarządcy to:

1. pośredniczenie między obszarem rysowania a modułami rysowania i manipulacji,
2. pośredniczenie między graficznym interfejsem użytkownika a modułami rysowania i manipulacji,
3. odbieranie od graficznego interfejsu użytkownika informacji o zmianie stanu aplikacji,
4. nadzorowanie dostępu do historii,



Rysunek 3.10: Pośrednik rysowania i otoczenie

5. pośredniczenie w odrysowywaniu.

Pierwsze trzy punkty mówią o przekazywaniu zdarzeń pomiędzy różnymi modułami oprogramowania, przy czym zdarzenia te mają charakter zdarzeń myszy, zdarzeń zmiany właściwości i zdarzeń akcji. Przez zdarzenia myszy należy rozumieć te pochodzące od urządzenia wskazującego typu mysz, przykładowo: wciśnięcie klawisza myszy, kliknięcie, ruch kursora. Przez zdarzenia zmiany właściwości należy rozumieć te, które wynikają ze zmian np. pewnych atrybutów obrazu. Przez zdarzenia akcji należy rozumieć te, które wynikają z wydania polecenia np. za pomocą wciśnięcia przycisku interfejsu użytkownika.

Wszystkie narysowane figury są umieszczane w historii, tak samo jak zdarzenia oznaczające manipulacje figurami. Zarządca wyświetlania jest widziany przez moduły rysowania i manipulacji jako historia. Dzięki temu można kontrolować operacje przeprowadzane na historii.

Zarządca wyświetlania pośredniczy w procesie odświeżania obrazu, ponieważ ma on połączenie z obszarem rysowania oraz z modułami, które mogą wymuszać to odświeżanie.

Wprowadzenie pośrednika w postaci zarządcy pozwala w łatwy sposób zaimplementować role użytkowników. W instancji AmeBoard, która nie posiada praw edycyjnych, zarządca blokuje zdarzenia i operacje pomiędzy modułami.

Na rysunku 3.10 został przedstawiony poglądowy diagram klas, obrazujący powiązania między pośrednikiem rysowania (`DisplayManager`), a historią (`History`), obszarem rysowania (`ImagePane`), dowolnym obiektem obsługi (interfejs `Handler`), graficznym interfejsem użytkownika (GUI) oraz głównym obiektem aplikacji (AmeBoard).

Konfiguracja

Konfiguracja jest modułem, który odbiera zdarzenia zmiany atrybutów obrazu od graficznego interfejsu użytkownika. Każde zdarzenie niesie ze sobą informację o nowej wartości określonego atrybutu. Tak skumulowane informacje są wykorzystywane przy tworzeniu obiektów graficznych, bez konieczności odwoływania się do interfejsu użytkownika. Można zbudować zupełnie inny interfejs użytkownika, który w łatwy sposób da się podłączyć do edytora AmeBoard.

Obiekty graficzne

Przez obiekt graficzny należy rozumieć pewien kształt (np. prostokąt) oraz atrybuty obrazu z nim związane. Przez atrybuty obrazu należy rozumieć kolor wnętrza figury, linii, styl linii, itp. Cechy obiektów graficznych:

1. zdolność do odrysowywania się na obiekcie java.awt.Graphics2D,
2. obiekt graficzny może zostać logicznie usunięty - będzie istniał w pamięci, ale nie będzie się odrysowywał,
3. obiekt graficzny może zostać zgrupowany z innymi obiektami,
4. obiekt graficzny podlega transformacjom graficznym: przesuwaniu i skalowaniu,
5. obiekt graficzny posiada unikalny identyfikator w obrębie sesji,
6. wszystkie atrybuty obiektu graficznego są modyfikowalne.

Do szczególnych obiektów graficznych należą grupy figur oraz napisy. Grupa jest to obiekt zawierający w sobie listę innych obiektów graficznych, powiązanych w logiczną całość. Obiekt grupy jest także obiektem graficznym, więc może zawierać w sobie inne grupy. Grupa zachowuje się jak jedna całość, tzn. podlega transformacjom jakby była jednym obiektem graficznym. Z drugiej strony niesie to pewne ograniczenia, a mianowicie grupa nie może posiadać jednolitych atrybutów obrazu.

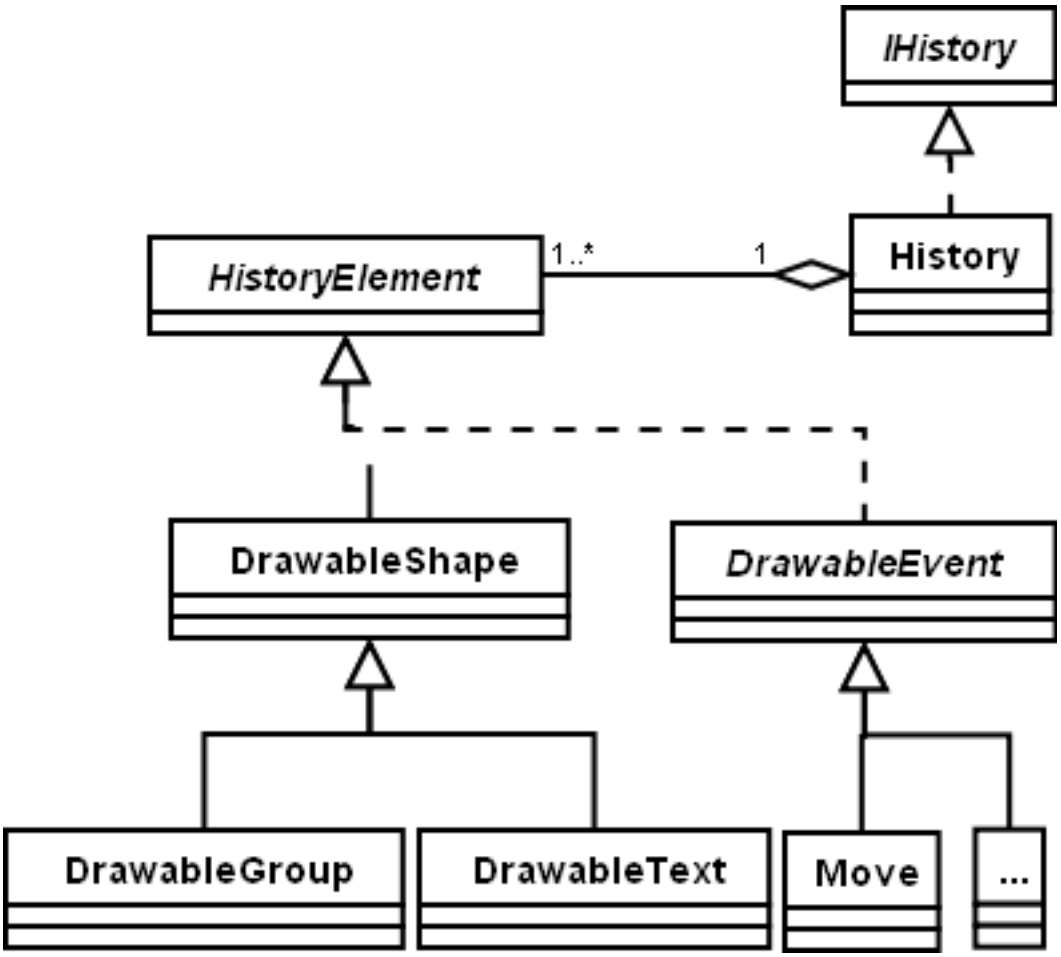
Napis w edytorze graficznym ma dwojaką naturę: jest kształtem (ścieżką) oraz ciągiem znaków. Reprezentacja napisu jako kształtu sprowadza zachowanie obiektu do zwykłego obiektu graficznego z atrybutami obrazu. Reprezentacja napisu jako ciągu znaków dodaje nowe funkcje (np. zmianę napisu) i nowe cechy (np. czcionkę).

Zdarzenia graficzne

Przez zdarzenia graficzne należy rozumieć specjalne obiekty służące do zapamiętania, przeprowadzonych na obiektach graficznych, operacji manipulacji (zgodnie z modułem manipulacji). Oprócz samej informacji o operacji, zdarzenia graficzne przechowują informację o obiekcie na którym manipulacja została wykonana oraz jego stary i nowy stan. Zdarzenia graficzne pozwalają na odtwarzanie starego lub nowego stanu.

Historia

Historia jest podstawową strukturą danych edytora AmeBoard, która przechowuje wszystkie obiekty oraz zdarzenia graficzne. Historia jest logicznym odzwierciedleniem stanu obszaru rysowania, stanowi podstawę do wykonania operacji cofania i powtarzania. Obiekt historii jest podstawą do przesyłania informacji przez sieć - każdy dodany obiekt, po przekształceniu, jest wysyłany



Rysunek 3.11: Historia i jej elementy

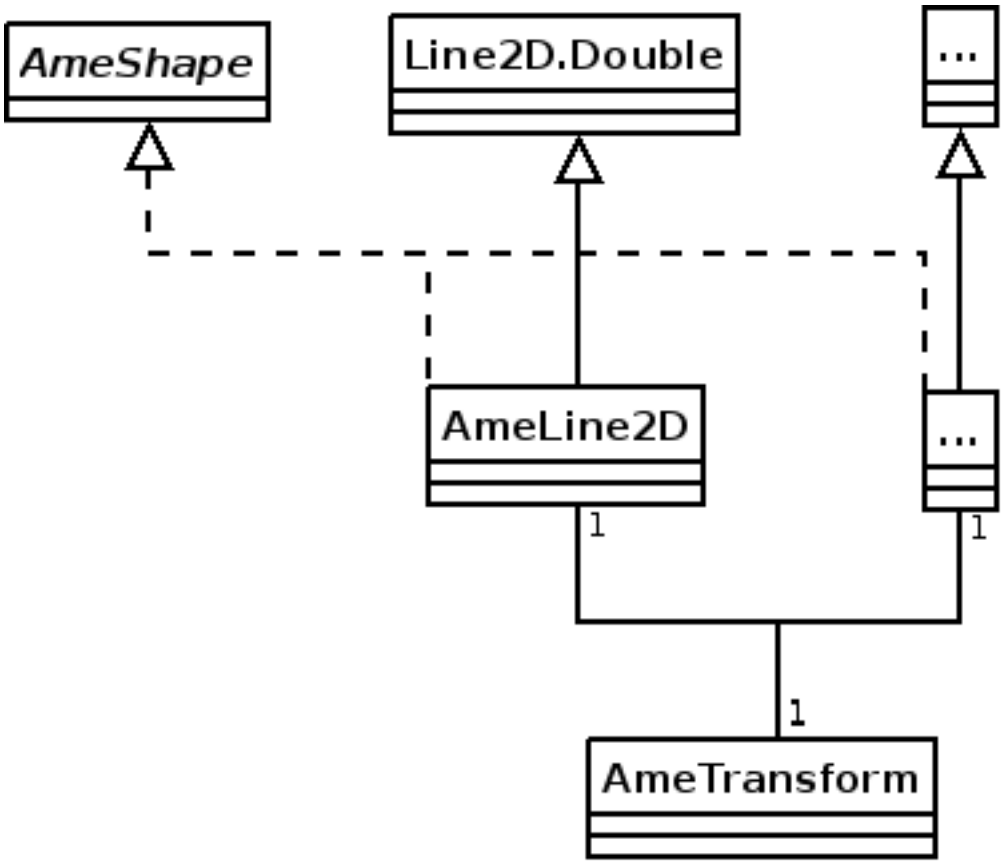
do modułu sieciowego AmePlug. Możliwe jest przeszukiwanie historii za pomocą punktu obrazu, identyfikatora figury, obszaru prostokątnego.

Na rysunku 3.11 została przedstawiona historia (`History`), oraz jej główne elementy: obiekty graficzne (`DrawableShape`) wraz ze specjalizacjami dla tekstu (`DrawableText`) i grupy (`DrawableGroup`) oraz zdarzenia graficzne (`DrawableEvent`) wraz z przykładowymi specjalizacjami zdarzeń.

Nowe definicje figur

Wszystkie obiekty graficzne mogą zostać poddane manipulacji (przesuwaniu, skalowaniu), co przekłada się na transformacje zawartych w nich kształtów. Wszystkie figury pakietu Java2D opierają się o interfejs `java.awt.Shape` , który nie udostępnia metod do ich transformacji (bez zmiany wewnętrznej struktury). W związku z tym konieczne jest wprowadzenie dodatkowego interfejsu oraz przedefiniowanie klas figur. Tak zmodyfikowane kształty mają spójny interfejs pozwalający w łatwy sposób dokonać ich transformacji, niezależnie od konkretnego typu figury.

Wprowadzone zmiany ilustruje rysunek 3.12, gdzie AmeShape reprezentuje nowy interfejs, `AmeTransform` jest dodatkową klasą transformacji. Na diagramie przedstawiono tylko jedną figurę, ale przeciążone zostały wszystkie używane w AmeBoard.



Rysunek 3.12: Przedefiniowane figury

Transformacje figur

Transformacje figur zostały oparte o macierz transformacji. Położenie punktu (x, y) na płaszczyźnie jest reprezentowane przez wektor P, analogicznie (x', y') jest reprezentowane przez wektor P'. Macierz transformacji została oznaczona jako M. Odpowiednie definicje macierzy i wektorów przedstawia zbiór równań 3.1. Zastosowane oznaczenia:

S_x, S_y – współczynniki skalowania,

H_x, H_y – współczynniki pochylenia,

T_x, T_y – współczynniki przesunięcia.

Jednym z możliwych przekształceń jest rotacja, która wymaga wprowadzenia dodatkowych wzorów na wyżej wymienione współczynniki.

$$M = \begin{bmatrix} S_x & H_x & T_x \\ H_y & S_y & T_y \\ 0 & 0 & 1 \end{bmatrix}, P = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}, P' = \begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} \tag{3.1}$$

$$P' = M \cdot P \tag{3.2}$$

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & H_x & T_x \\ H_y & S_y & T_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.3}$$

$$\begin{cases} x' = x \cdot S_x + y \cdot H_x + T_x \\ y' = y \cdot S_y + x \cdot H_y + T_y \end{cases} \tag{3.4}$$

Transformacje dowolnego punktu na płaszczyźnie za pomocą macierzy transformacji opisują wzory: 3.2 i 3.3. W zapisie algebraicznym transformacja ma postać układu równań 3.4. Jest to podstawowa metoda do przekształcania obrazów, rozszerzalna o kolejne wymiary przestrzeni.

$$\begin{cases} x'_1 = x_1 \cdot S_x + T_x \\ x'_2 = x_2 \cdot S_x + T_x \end{cases} \tag{3.5}$$

$$\begin{cases} S_x = \frac{x'_1 - x'_2}{x_1 - x_2} \\ T_x = x'_1 - S_x \cdot x_1 \end{cases} \tag{3.6}$$

Zastosowane oznaczenia x_1, x_2, x'_1, x'_2 są składowymi x dwóch punktów obrazu przed i po skalowaniu. Za pomocą układów równań 3.5 wyliczane są odpowiednie współczynniki transformacji 3.6, które następnie są stosowane do przekształcenia punktów opisujących figurę. Operacja pochylenia nie jest przewidziana w AmeBoard, więc współczynniki H_x i H_y domyślnie są zerowe.
Przeprowadzeniem właściwych przekształceń zajmuje się specjalna klasa `AmeTransform` , która wykorzystuje obiekt klasy `java.awt.geom.AffineTransform` do transformacji ścieżek oraz specjalizowane metody przedefiniowanych figur w pozostałych przypadkach.

Moduł wejścia/wyjścia

AmeBoard pozwala na zapis i odczyt całego obszaru rysowania do pliku, a także możliwy jest eksport i import pojedynczych obiektów graficznych. Dodatkową funkcją jest opcja zapisu obszaru rysowania do bitmapy. Domyślnym formatem zapisu jest SVG, co niesie za sobą następujące korzyści:

1. obraz zapisany jako SVG daje się wyświetlić w większości nowoczesnych przeglądarek internetowych (Mozilla Firefox 2, Microsoft Internet Explorer 7),
2. format ten jest standardem wielu programów graficznych do edycji grafiki wektorowej (Inkscape),
3. SVG jest ustandaryzowanym formatem, zwalnia projektanta od definiowania własnego formatu,
4. SVG jest formatem grafiki wektorowej, więc obraz łatwo poddaje się skalowaniu bez utraty jakości;
5. pliki z obrazem są z reguły niewielkie,
6. SVG jest niezależne od platformy.

Kliparty

AmeBoard może w każdej chwili stać się prostym edytorem diagramów. Każdy obiekt graficzny może zostać wyeksportowany do pliku SVG, więc można utworzyć tematyczną kolekcję takich obiektów – rysunków (np. elektronika: rezystory, kondensatory, tranzystory itp.) umieszczając je w katalogu. Tak spreparowane dane dają się odczytać w AmeBoard jako kolekcja klipartów z podglądem miniatur, a każdy z klipartów może zostać zaimportowany do obszaru rysowania. Dzięki takiemu podejściu AmeBoard może stać się użyteczny dla dowolnej grupy projektantów, wystarczy tylko przygotować zestaw klipartów, odpowiadający tematyce pracy.

Mapowanie obrazu

Historia, jako nośnik informacji, stanowi podstawę do przesyłania danych przez sieć oraz przy zapisie do pliku. Każdy element dodany do historii jest przetwarzany na pewną strukturę, która posiada format akceptowany przez AmePlug. Ta struktura jest przekazywana do modułu AmePlug, który zajmuje się dalszą propagacją wiadomości. Z drugiej strony, wiadomość otrzymana od AmePlug jest przetwarzana na obiekt, który jest dodawany do historii. Przetwarzaniem wiadomości i obiektów historii dla celów sieciowych zajmuje się specjalna klasa `AmeBoardMessageMapper`.

3.2.3 AmeEdit

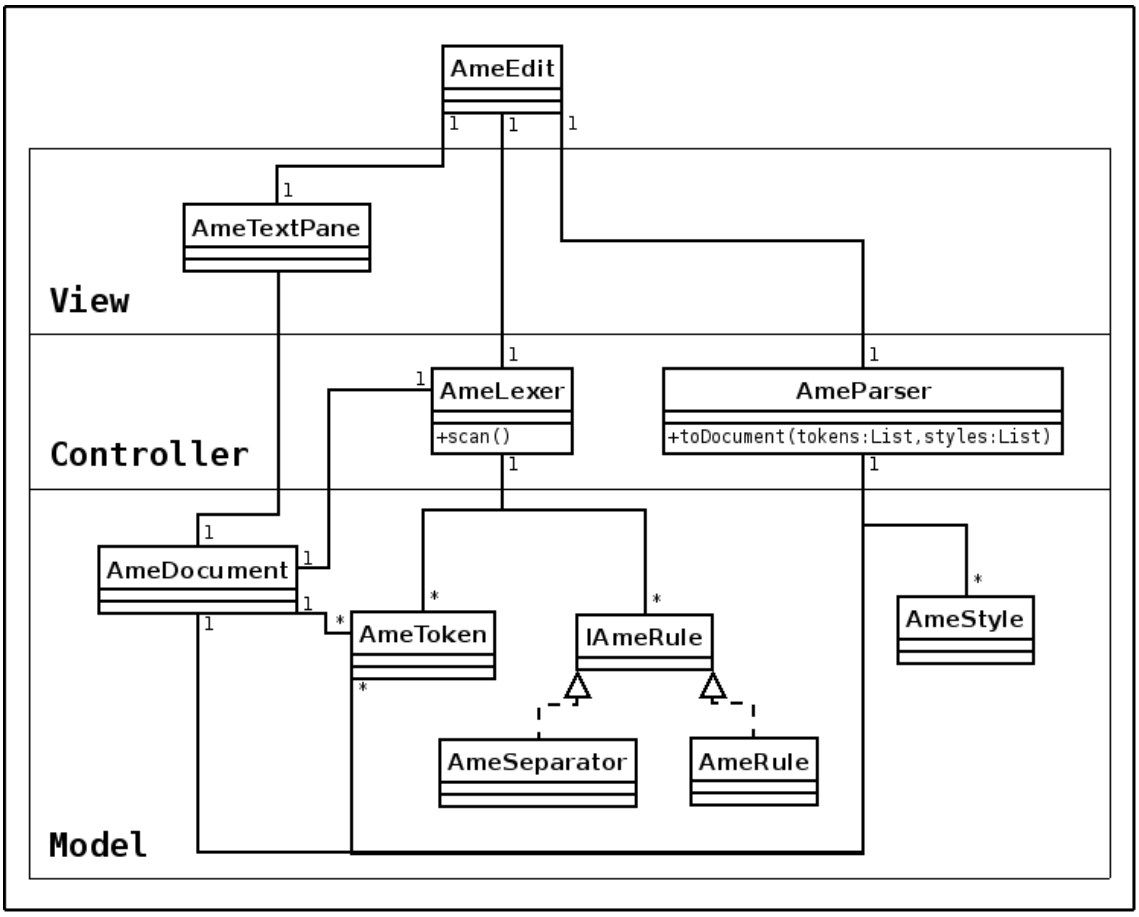
Moduł AmeEdit jest edytorem kodu, w zakres odpowiedzialności którego wchodzą zarówno zdefiniowanie i utrzymywanie modelu dokumentu i zapewnienie możliwości jego manipulacji, ale także dostarczenie elementu wyświetlającego zawartość dokumentu użytkownikowi, który współpracowałby z biblioteką graficzną użytą przy tworzeniu interfejsu użytkownika. Moduł AmeEdit związany jest więc ściśle z trzema warstwami architektury biblioteki Swing [Fow03], której warstwa modelu (*Model*) jest wzorem dla struktury dokumentu AmeEdit, warstwa reprezentacji wizualnej (*Viewer*) stanowi podstawę dla komponentu włączonego w całość oprawy wizualnej wtyczki, a warstwa kontroli (*Control*), która zajmuje odwzorowaniem działań operatora na model. Dokładne przedstawienie odwzorowania warstw jest pokazane na rysunku 3.13.

Dodatkowo edytor musi także mieć możliwość współpracy sieciowej, a więc wysyłania wiadomości o dokonanych zmianach do użytkowników zdalnych oraz odbierania i wykonywania zmian, przy współdziałaniu z modułem AmePlug. Poszczególne modyfikacje odnoszące się do tych działań względem zwykłego edytora (to znaczy, dla jednego użytkownika) są uwzględnione są umiejscowione w warstwie kontroli.

Programami stanowiącymi wzór dla AmeEdit były przede wszystkim dwa edytory kodu, a mianowicie edytor języka Java zawarty w programie Eclipse SDK oraz program gedit, będący częścią środowiska graficznego Gnome. Dostarczyły one inspiracji na temat funkcji jakie powinien spełniać edytor, jak powinien reagować na pojawiające się trudne do rozwiązania sytuacje przy kolorowaniu, oraz jak powinno zachowywać się wyszukiwanie fraz w pliku. Żaden z tych programów nie posiada jednak możliwości edycji pliku wspólnie, wraz ze współautorami komunikującymi się z użytkownikiem lokalnym przez Internet lub innego rodzaju sieć. AmeEdit więc ma na celu złączenie możliwości komunikatora internetowego z edytorem kodu, żeby umożliwić zdalną współpracę między programistami.

Definicja dokumentu

Model dokumentu edytora AmeEdit jest oparty o zbiór obiektów, które odpowiadają za przechowywanie informacji o tokenach oraz o obiekt klasy rozszerzającej klasę AbstractDocument z



Rysunek 3.13: Warstwy AmeEdit w odniesieniu do edycji dokumentu.

biblioteki Swing. Rozszerzenia polegają na udostępnieniu dokumentowi możliwości działania na tokenach i konwersji ich do postaci oczekiwanej przez komponenty zawarte w interfejsie użytkownika. Na stan każdego z tokenów natomiast składają się ich nazwy, przechowywane wartości - napisy oraz indeks zajmowany w tekście. Tak pełny zestaw danych pozwala na manipulacje tokenami i porównywanie ich ze sobą wzajemnie przez analizator leksykalny.

Dodatkowe informacje o dokumencie to także definicje elementów składni języka programowania oraz definicja stylów związanych z tymi elementami. Elementy te są definiowane przez użytkownika w plikach XML i mają wpływ na dzielenie tekstu na tokeny i wyświetlanie dokumentu. Zbiory XML opatrzone także są schematami XML Schema, które umożliwiają weryfikację ich poprawności w momencie wczytania ich przez edytor. Po otwarciu przez edytor pliki XML zamieniane zostają na listy odpowiednich obiektów (`IAmeRule` i `AmeStyle` , odpowiednio) które następnie mogą być aplikowane do tekstu przez obiekt klasy `AmeParser` , a także konfigurowane przez użytkownika w preferencjach.

Dokument jest także definiowany przez historię operacji na nim wykonanych, co jest rozszerzeniem mechanizmów dostarczanych przez bibliotekę Swing.

Po otrzymaniu poszczególnych elementów składowych stanowiących o istocie dokumentu jego synteza zostaje dokonana w kolejnej warstwie, odpowiedzialnej za manipulacje tekstem.

Edycja tekstu

Problem edycji tekstu zaczyna się w warstwie najbliższej użytkownika, w miejscu, w momencie kiedy obszar edycyjny otrzymuje polecenie do wykonania. Polecenie to jest przesłane do wykonania do kolejnej warstwy, gdzie dane zostają poddane podziałowi na tokeny według zdefiniowanych reguł, zaaplikowany im zostaje styl i zostają przekazane do modelu, który z kolei wywołuje zmiany w części oglądanej przez użytkownika.

Jedną z najważniejszych części tego cyklu jest proces tworzenia tokenów, czyli analiza leksykalna. Analizator leksykalny, będąc uprzednio zaopatrzony w definicje separatorów przegląda edytowaną treść, izolując poszczególne słowa i przekazuje je na listę. Lista używanych separatorów zawiera obiekty, których stan rozróżnia je na dwie podgrupy traktowane inaczej:

- separatory - zdefiniowane jako zbiory tokenów,
- 'enkapsulacje' - zdefiniowane jako najkrótszy obszar zawarty między otwierającą i zamykającą sekwencją znaków, z uwzględnieniem sekwencji ucieczek.

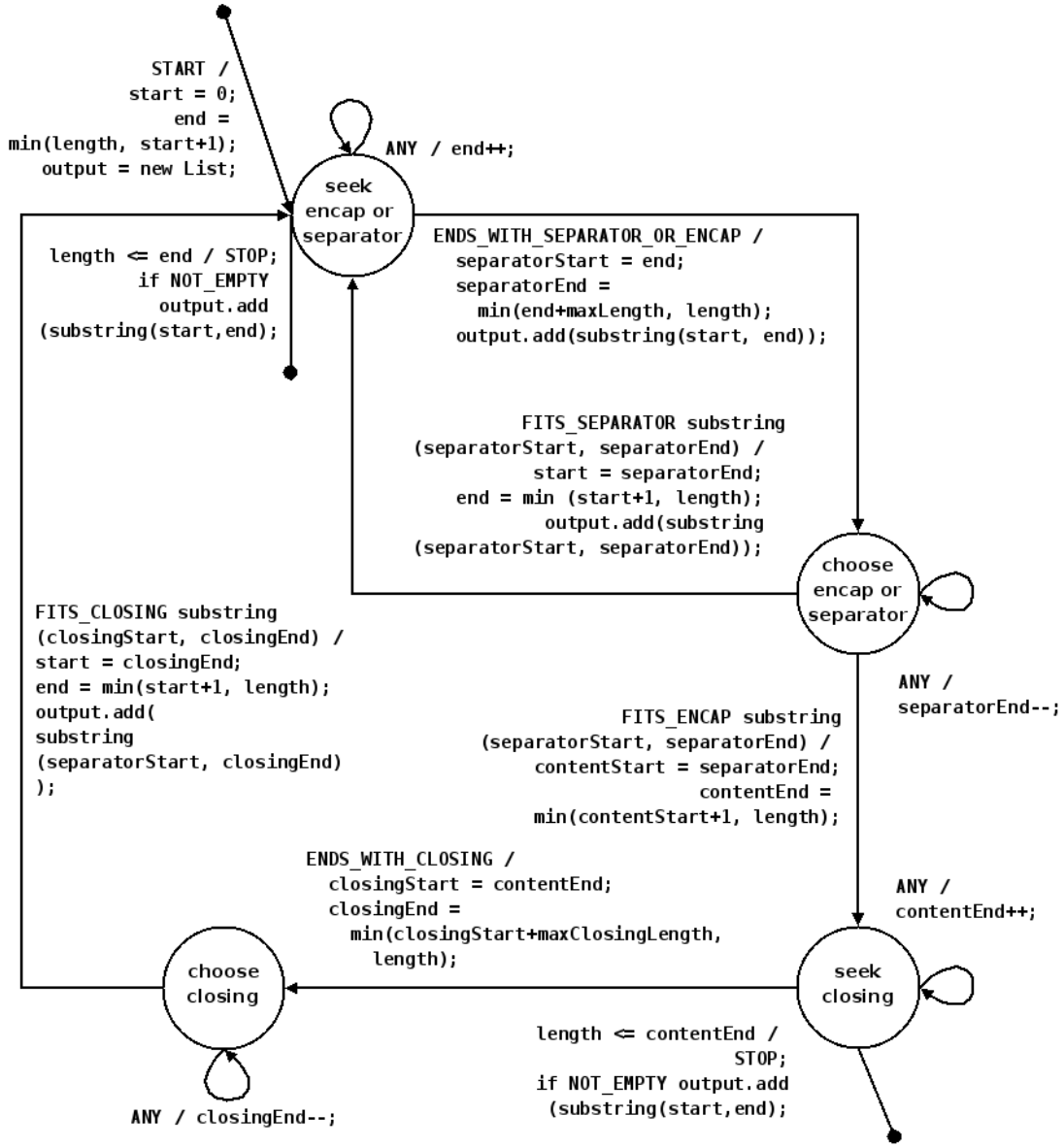
Uwzględnienie w przetwarzaniu drugiej z wyżej wymienionych grup ma na celu zapewnienie obsługi elementów, takich jak literały ciągów znaków (sekwencje pomiędzy parą znaków ' “'), które w ogólności mogłyby zostać potraktowane jak zbiór tokenów, podczas gdy faktycznie pokolorowane powinny być tak jak jeden. Ten proces skanowania tekstu jest przedstawiony na schemacie automatu stanów na rysunku 3.14.

Otrzymana z analizatora leksykalnego lista tokenów jest następnie przyrównywana do aktualnie wyświetlonej. Algorytm porównujący tokeny ma za zadanie wytworzyć dwie listy: jedną, która zawiera tokeny do usunięcia z bieżącej wersji oraz drugą, która zawiera tokeny, które należy w niej umieścić, żeby stara i nowa wersja były tożsame. Algorytm najpierw dokonuje tymczasowych alteracji znanej tokenom pozycji. Następnie przyjmowane są założenia, mówiące o tym, że z bieżącego ciągu tokenów nie należy usunąć wszystko i wstawić wszystkie tokeny ponownie. Powtarzające się w tokeny w liście do usunięcia są składowane w osobnym zbiorze. W kolejnym kroku dokonywane są dwie operacje odejmowania na zbiorach - od zbioru tokenów do usunięcia odejmowany jest nowa lista tokenów, od zbioru do dodania odejmowana jest lista istniejących tokenów. Do zbioru usuwanego dodawane są też tokeny, które wcześniej podlegały powtórzeniom, ze względu na zmiany pozycji. Ostatecznie ze zbiorów tokenów tworzone są listy, uporządkowane według pozycji tokenów, a zaaplikowane przesunięcia zostają zniwelowane. Algorytm zapisany w pseudokodzie znajduje się na rysunku 3.15.

Zmiany dokonane przez komparator list tokenów są następnie nanoszone na komponent widziany przez użytkownika, po uprzednim przypisaniu im zdefiniowanych stylów na podstawie typów tokenów.

Komunikacja z użytkownikiem zdalnym

Użytkownik lokalny, który posiada możliwość edycji kodu przy każdej zmianie, ponadto, że uzyskuje nową treść w swoim własnym oknie edycyjnym, wysyła także wiadomości do wszystkich nasłuchujących użytkowników zdalnych, u których zmiana ta także odświeża obszar edycyjny. Zmiany przesyłane są przez moduł AmePlug. Przesłane zostają jedynie elementy związane z modelem - rodzaj zmiany, wstawiony tekst, pozycja jego wstawienia, długość usuniętego ciągu znaków - natomiast kolorowanie składni, a więc cała analiza leksykalna, przyrównywanie list tokenów, przypisywanie stylów, jest wykonywane u każdego z użytkowników lokalnie, wedle zdefiniowanych przez niego języków i stylów.



Rysunek 3.14: Automat stanowy analizatora leksykalnego.

Podczas gdy wysyłanie komunikatów to kwestia informowania o zachodzących zmianach, nasłuch wymaga wytworzenia mechanizmu, który symulowałby działanie użytkownika na podstawie otrzymywanych wiadomości. Każda przychodząca wiadomość jest więc rozpoznawana pod względem typu operacji, jaką powinien przeprowadzić symulację edytora, a następnie przekazywana odpowiedniemu obiektowi, który wykonuje na odpowiedniej części (przykładowo, operacje modyfikacji dokumentu na modelu, operacje zakreślania na obszarze edycyjnym) modułu AmeEdit odpowiednią operację na podstawie uzyskanych danych.

Komunikacja z modułem sieciowym, AmePlug, odbywa się za pomocą przesyłania komunikatów do obiektów instancji klas implementujących interfejs `IAmeMessageListener` .

Obsługa i wysyłanie komunikatów przez moduł zostały przedstawione na rysunku 3.16.

```
for all token in oldTokens do
  if token.position ≥ endOfChangeRegion then
    token.position ← token.position + lengthOfChange
  end if
end for

removeSet ← oldTokens
replaceSet ← newtokens
redundacySet ← oldTokens ∩ removeSet

removeSet ← (removeSet \ oldTokens) ∪ redundancySet
replaceSet ← replaceSet \ newTokens

replaceList ← sort (replaceSet)
removeList ← sort (removeSet)

if changeLength > endOfChangeRegion then
  for all token in removeList do
    if token.position - lengthOfChange ≥ endOfChangeRegion then
      token.position ← token.position - lengthOfChange
    end if
  end for
end if

return replaceList, removeList
```

Rysunek 3.15: Algorytm porównywania list tokenów.

Obszar edycyjny

Obszar edycyjny edytora jest oparty całkowicie o komponent `JTextPane` zawarty w bibliotece Swing. Zapewnia on połączenie modelu dokumentu oraz części kontrolnej do części połączonej z graficznym interfejsem użytkownika za pomocą modelu zdarzeniowego. Komponent nasłuchuje zdarzeń, które wysyłane są przy zajściu zmian w modelu, na bazie których widok zostaje uaktualniony do nowego stanu.

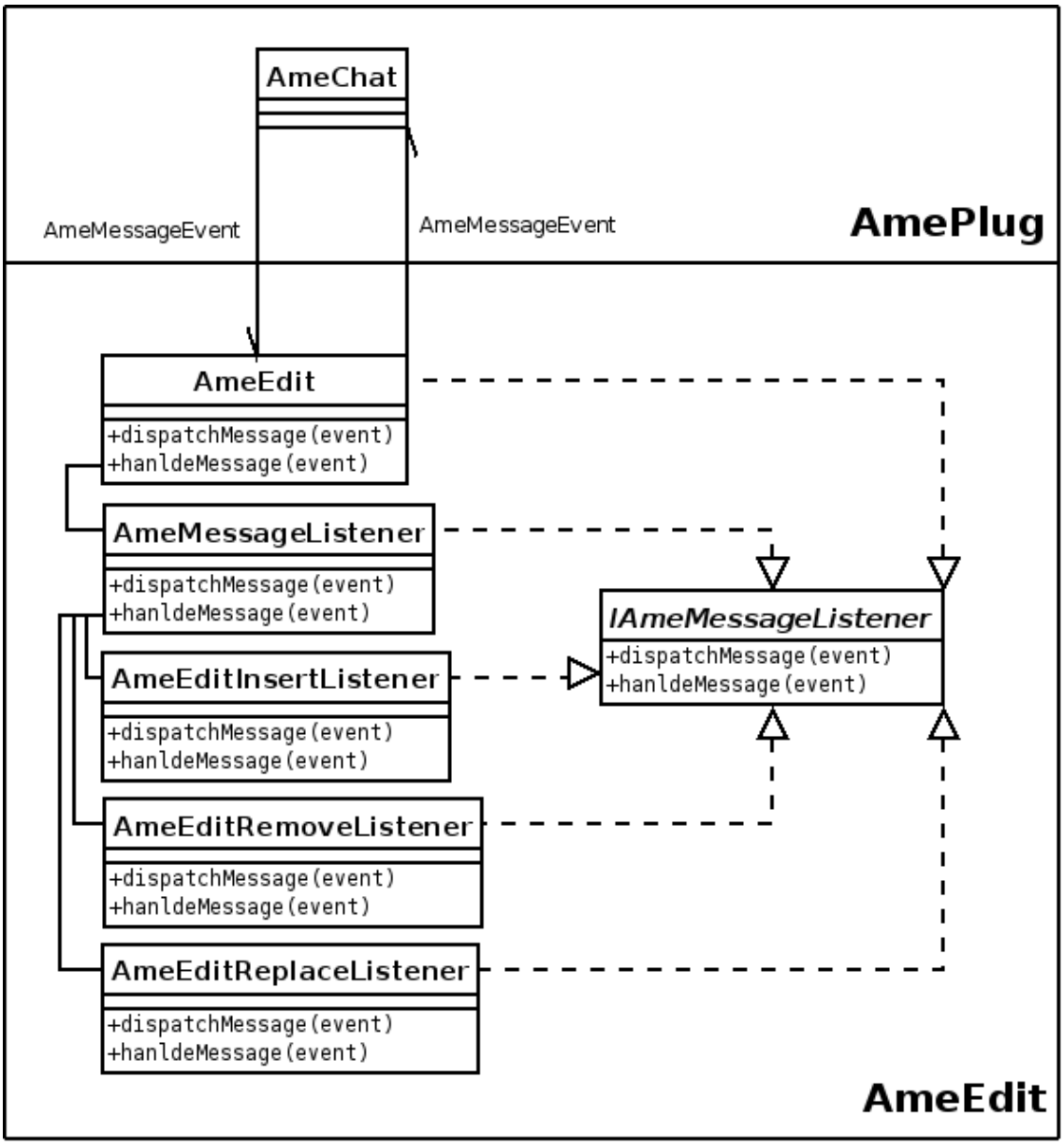
Obszar edycyjny, będąc rozszerzonym nieco komponentem może być łatwo wpasowany w interfejs napisany za pomocą biblioteki graficznej Swing.

3.2.4 AmeGUI

Tworzenie aplikacji wyposażonych w graficzny interfejs użytkownika w języku Java nie musi opierać się tylko na wykorzystaniu standardowych komponentów wizualnej interakcji programu z użytkownikiem. Java oferuje zestaw klas, dzięki którym można wykreślić dowolną grafikę i przekształcić komponenty lekkie w niezależny od platformy systemowej wygląd.

Programowanie GUI pozwala na kreślenie komponentów za pomocą kodu Java przy wykorzystaniu pakietu javax.swing oraz java.graphics2D, przez co wygląd aplikacji jest uzależniony tylko od ustaleń programisty.

AmeGUI to interfejs użytkownika dla AmeBoard oraz AmeEdit. Obydwa *pluginy* posiadają ujednolicony interfejs użytkownika. Poniżej zostaną opisane poszczególne elementy z których składając się interfejsy.



Rysunek 3.16: Odbiór i wysyłanie komunikatów.

AmeBoard

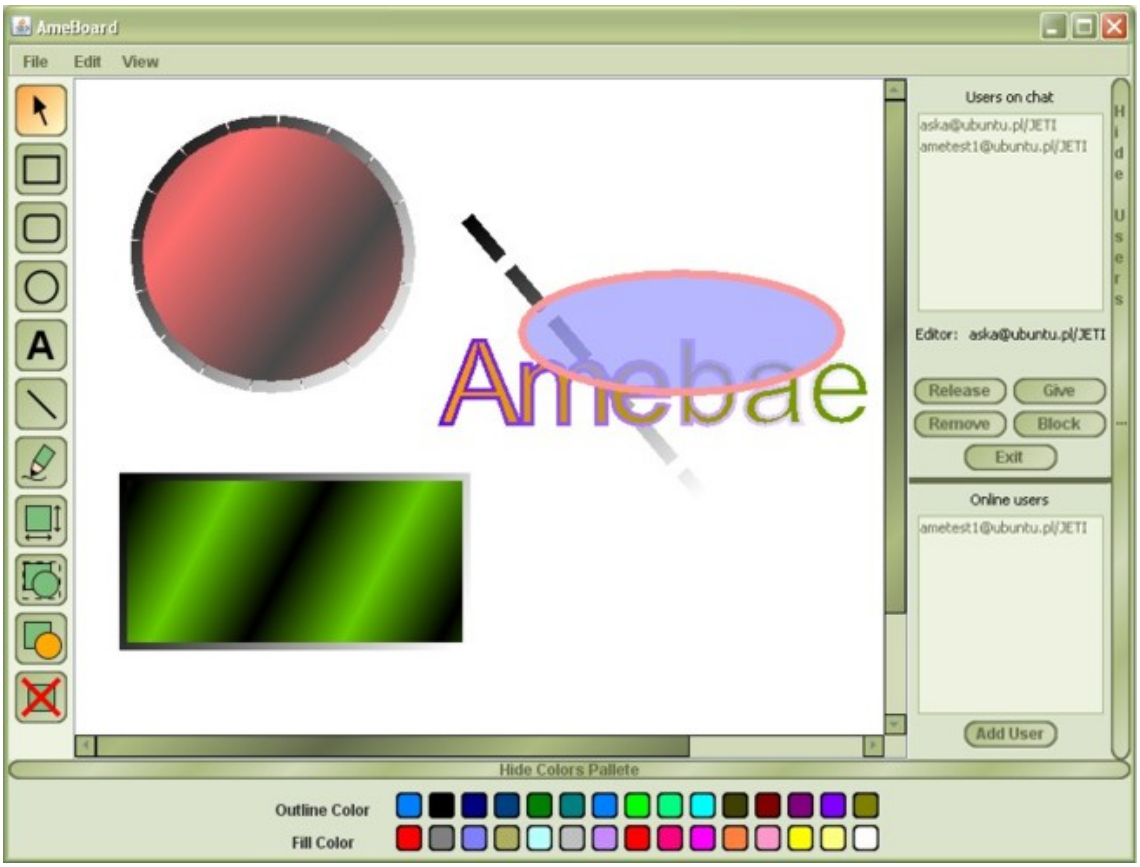
Właściwości rysowanej linii

Moduł ten pozwala ustawić właściwości rysowanej linii – grubość, kształty zgięcia, zakończenia, sposób łączenia.

Wykreślana linia może mieć trzy rodzaje kształtów zgięcia – spłaszczone, zaokrąglone lub zaostrzone. Ponadto możliwe są trzy rodzaje zakończeń – ścięte, półkoliste lub kwadratowe.

Można określić parametry odpowiedzialne za długość wykreślanych linii i odstępów. Dzięki temu możemy wykreślać linie przerywane, ciągłe, linie składające się z kropek, oraz różne ich kombinacje.

Dodatkowo można ustawić wartość maksymalnej długości dla zaostrzenia, po przekroczeniu, której połączenie zaostrzone zostanie zastąpione połączeniem spłaszczonym. Przydatne jest to wtedy, gdy linie nachylone są do siebie pod małym kątem, wówczas połączenie może okazać się



Rysunek 3.17: AmeBoard - Interfejs użytkownika

zbyt długie.

Wypełnianie obszarów

Moduł ten umożliwia wypełnienie obrysów oraz obszarów obiektów kolorem jednolitym lub gradientem.

Wypełnienie gradientem umożliwia wypełnienie danego obszaru płynnym przejściem tonalnym pomiędzy dwoma kolorami.

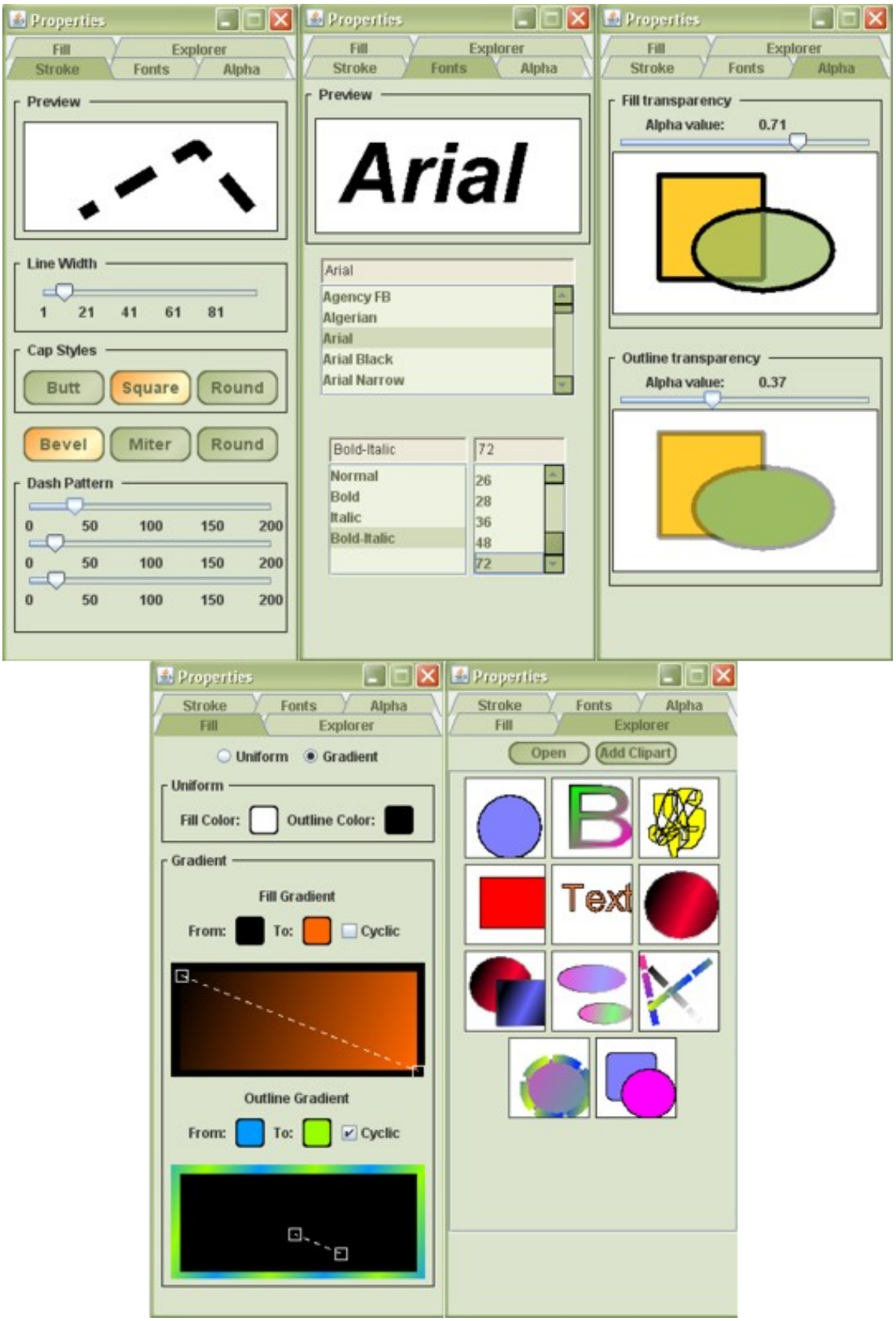
Właściwości gradientu określamy poprzez ustawienie takich właściwości jak:

- punkt początkowy (P_1)
- punkt końcowy (P_2)
- kolor początkowy (C_1)
- kolor końcowy (C_2).
- cykliczność gradientu

Po ustawieniu wyżej wymienionych parametrów, zostaje utworzona gama kolorów.

Kolory poszczególnych punktów, leżących na odcinku łączącym punkty P_1 i P_2 zmieniają się tonalnie od koloru C_1 do C_2.

Dodatkowo można określić cykliczność gradientu. W przypadku gradientu acyklicznego punkty leżące za P_1 mają kolor C_1, a punkty leżące za P_2 mają kolor C_2. Natomiast, gdy gradient jest



Rysunek 3.18: Panel właściwości

ustawiony na cykliczny, to punkty leżące poza odcinkiem, mają kolory zmieniające się cyklicznie tak jak wewnątrz odcinka.

Przezroczystość obiektów

Moduł ten umożliwia ustawienie stopnia przezroczystości na poziomie obiektu, dla którego właściwość ta pozwala na odrysowanie bytów graficznych znajdujących się poniżej.

Parametr *Alpha* zmienia się od 0 do 1. Wartość 0 oznacza obiekt całkowicie przezroczysty, natomiast wartość 1 powoduje, że obiekt jest wykreślany bez zastosowania przezroczystości.

Właściwości czcionki

Moduł ten umożliwia ustawienie właściwości czcionki, poprzez określenie takich parametrów jak:

- nazwa,
- styl,
- rozmiar.

Powyżej opisane moduły działają analogicznie. Podczas zmiany któregokolwiek parametru, zostaje wysłane odpowiednie zdarzenie, które jest odpowiedzialne za ustawienie parametrów dla rysowanych obiektów czy też obrysów. Ponadto wysyłane jest zdarzenie do okna podglądu, dzięki temu na bieżąco widoczne są zmiany dokonane przez użytkownika. Bieżące wartości poszczególnych parametrów są zapisywane w konfiguracji.

Przeglądarka plików SVG

Moduł ten umożliwia wybranie dowolnego katalogu i wczytanie obrazów w formacie SVG, które następnie zostają wyświetlone w postaci miniatur. Po wybraniu katalogu zostaje pobrana lista plików w formacie SVG a następnie zostaje wysłane zdarzenie i następuje przetworzenie obrazów SVG z pomocą odpowiedniego parsera do miniaturowej postaci, które są następnie wyświetlone w oknie dialogowym przeglądarki plików. Dzięki temu użytkownik może dodać wybrany obiekt do tworzonego schematu.

Wprowadzanie/Edycja tekstu

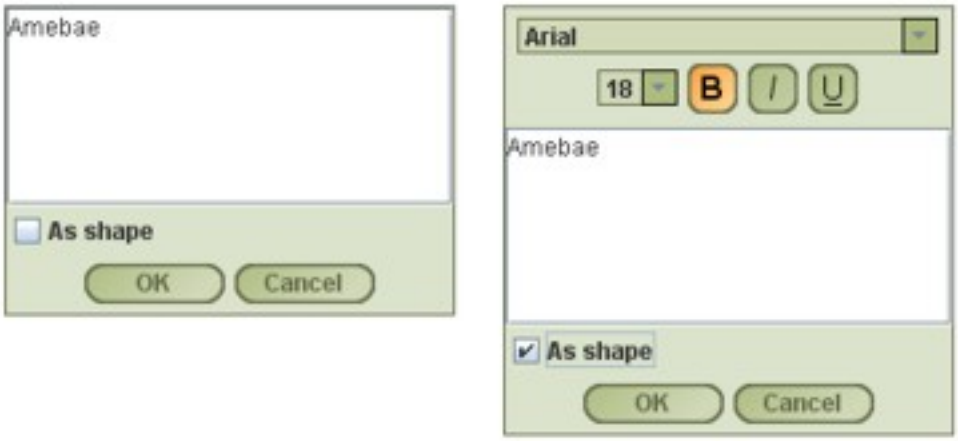
Wprowadzanie tekstu możliwe jest po wybraniu odpowiedniego narzędzia z palety oraz kliknięciu przyciskiem myszy w dowolnym miejscu obszaru do rysowana. Ponadto możliwa jest modyfikacja wcześniej wprowadzonego tekstu. Okno dialogowe do edycji tekstu umożliwia zmianę takich parametrów jak rodzaj czcionki, styl czy rozmiar. Dodatkowo możliwa jest zmiana koloru wypełnienia, stylu i koloru obrysu, czy też przezroczystości, za pomocą okna dialogowego zmiana właściwości obiektu.

Okno dialogowe - Zmiana właściwości obiektu

Okno dialogowe właściwości figury umożliwia zmianę właściwości wykreślonego obiektu takich jak styl obrysu, kolor wypełnienia czy przeźroczystość. Każdą figurę, bądź tekst możemy dowolnie modyfikować i zmieniać jej właściwości poprzez nowo dobrane parametry.

Pasek narzędzi

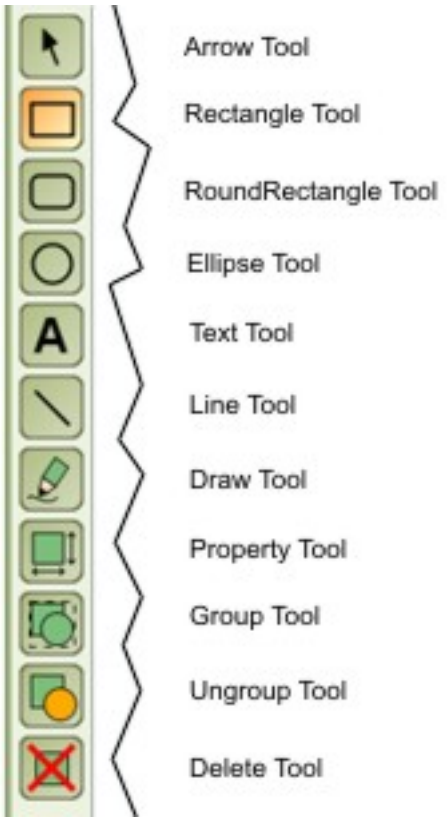
Pasek narzędzi, tak jak w standardowych programach graficznych, jest wyświetlany po lewej stronie aplikacji. Zawiera podstawowe narzędzia służące do tworzenia prostej grafiki wektorowej i przeprowadzania operacji na obiektach.



Rysunek 3.19: Okno dialogowe - Wprowadzanie i edycja testu

Podczas wybrania konkretnego narzędzia do kreślenia obiektów, wysyłane jest zdarzenie do fabryki figur, która jest odpowiedzialna za wykreślanie odpowiednich figur.

Ponadto dostępne są narzędzia odpowiedzialne za grupowanie, rozgrupowanie, usuwanie obiektów oraz narzędzie za pomocą, którego możemy uzyskać dostęp do właściwości konkretnego obiektu. Aby skorzystać z tego typu narzędzi należy najpierw zaznaczyć wybrany obiekt.



Rysunek 3.20: Pasek z dostępnymi narzędziami

Arrow Tool **-**– Zaznaczanie obiektów

Rectangle Tool **-**– Wykreślanie prostokątów

***RoundRectangle* -**– Wykreślanie prostokątów z zaokrąglonymi narożnikami

***Ellipse Tool* -**– Wykreślanie elipsy

***Text Tool* -**– Wpisywanie tekstu

***Line Tool* -**– Wykreślanie linii

***Draw Tool* -**– Rysowanie

***Properties Tool* -**– Uzyskanie właściwości zaznaczonego obiektu

***Group Tool* -**– Grupowanie zaznaczonych obiektów

Ungroup Tool – Rozgrupowanie zaznaczonego obiektu

***Delete Tool* -**– Usuniecie zaznaczonego obiektu

Paleta kolorów

Aplikacja zawiera dodatkowy panel z podręczną wysuwaną paletą kolorów, która jest używana do wypełniania obiektów jednolitym kolorem - po wybraniu koloru dla wypełnienia bądź obrysu wysyłane jest zdarzenie informujące o tym oraz następuje zapisanie odpowiednich danych w konfiguracji.

AmeEdit

Okno dialogowe - Właściwości

Okno dialogowe właściwości umożliwia zdefiniowanie stylów kolorowania składni poszczególnych elementów czy słów kluczowych wybranego języka programowania. Można określić nazwę, typ, rozmiar oraz kolor czcionki użyty do kolorowania plików z kodem. Po zmianie języka i zatwierdzeniu zmian wysyłane jest zdarzenie, które powoduje zmianę zaznaczenia danego języka w menu głównego okna aplikacji oraz odświeżenie obszaru edycyjnego.

Okno dialogowe - Find/Replace

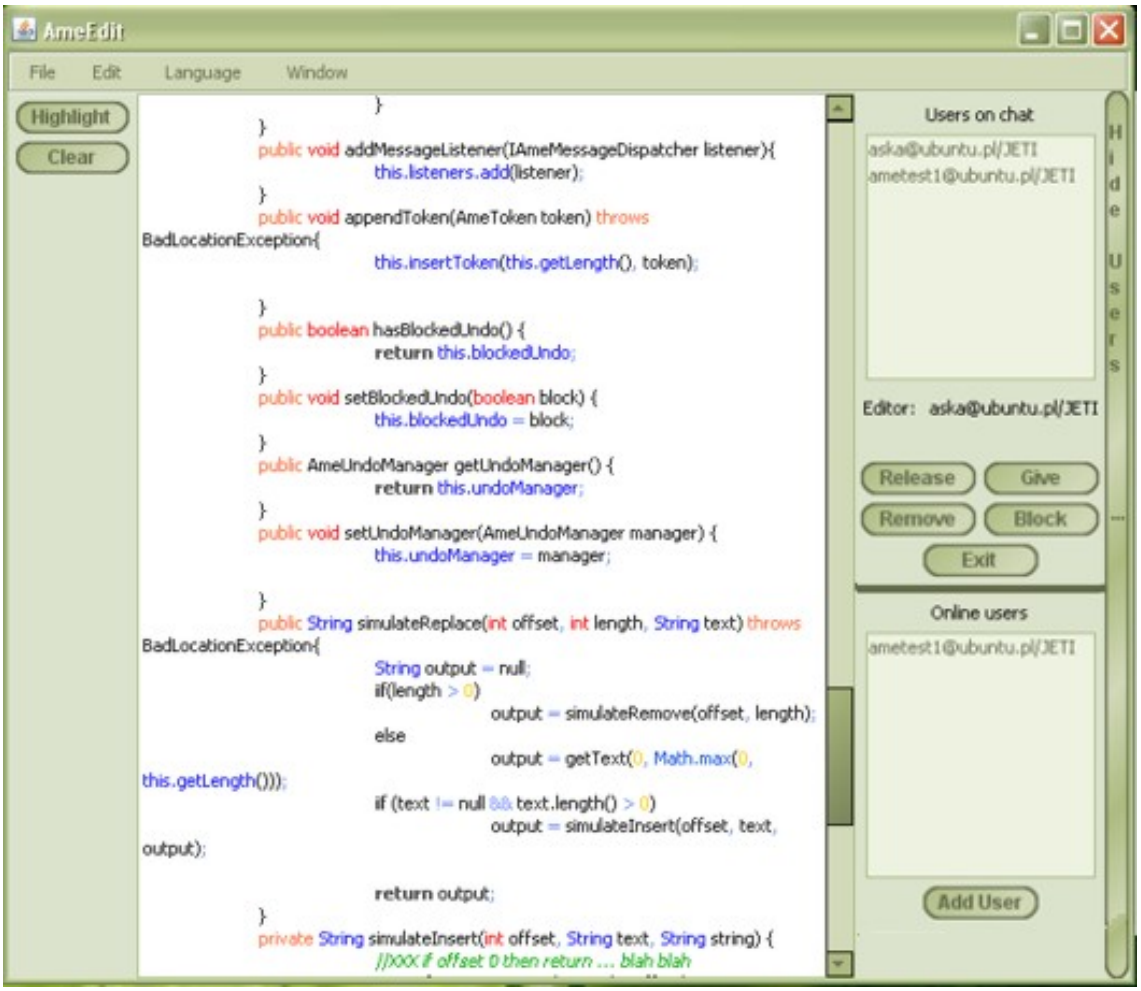
Okno dialogowe *Find/Replace* umożliwia wyszukiwanie lub zmianę słowa, frazy w kodzie programu. Pozwala na określenie kryteriów wyszukiwania takich jak rozróżnianie wielkości liter, znajdowanie całych wyrazów, wyrażeń regularnych, przeszukiwanie w przód, w tył, czy też przeszukiwanie całego dokumentu. Po wybraniu interesujących użytkownika opcji oraz określeniu rodzaju wykonanej operacji (przeszukanie, przeszukanie z zamianą czy też zamiana frazy) zostaje wysyłane zdarzenie i następuje jego obsługa.

Panel z użytkownikami

Ponadto oby dwa *pluginy* z prawej strony posiadają wysuwany panel z użytkownikami przyłączonymi do czatu, oraz wszystkimi użytkownikami dostępnymi w czasie rzeczywistym. Użytkownik posiadający prawo do edytowania zawartości okna, (którym początkowo jest założyciel czatu) ma do dyspozycji takie opcje jak:

Give – przekazanie praw edytora innemu użytkownikowi,

Remove – usunięcie danego użytkownika z rozmowy,



Rysunek 3.21: AmeEdit - interfejs użytkownika

Block – blokowanie przychodzących wiadomości,

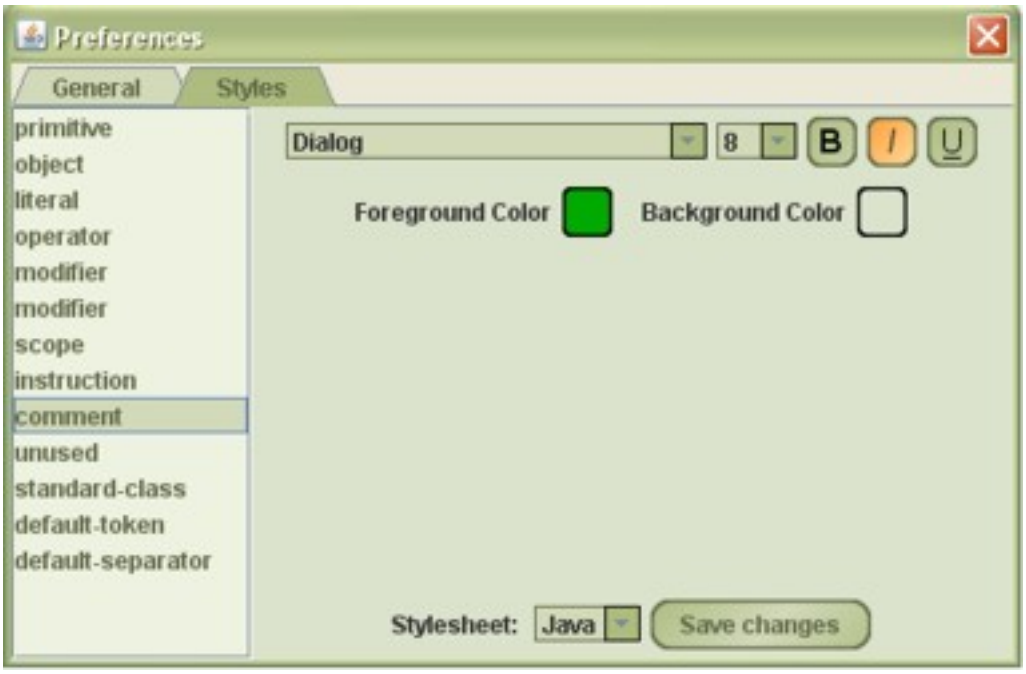
Exit – opuszczenie rozmowy.

Prawo do kreślenia obiektów z wykorzystaniem paska narzędzi, palety kolorów oraz panelu właściwości ma tylko użytkownik posiadający prawo edytora.

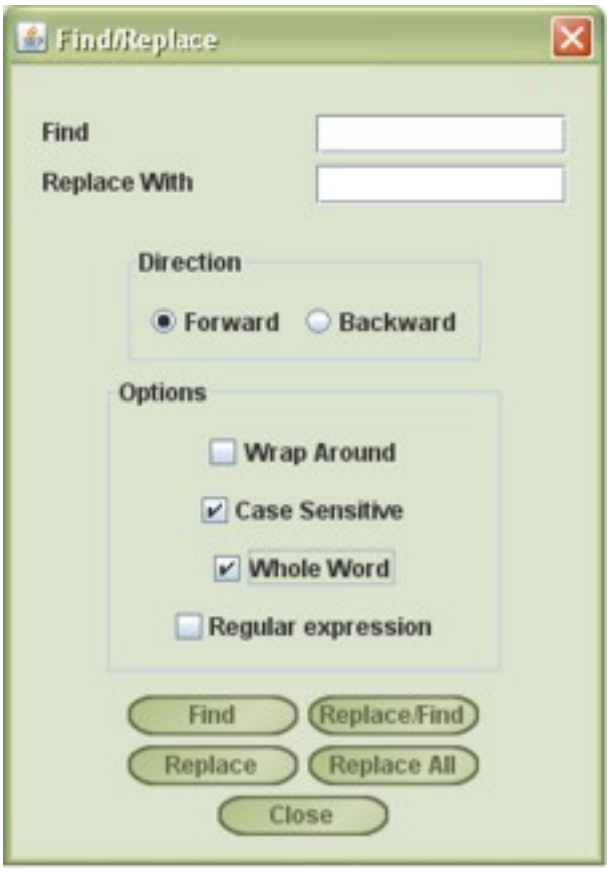
3.3 Implementacja

3.3.1 AmePlug

Podczas fazy implementacji modułu AmePlug, pierwszym głównym problemem jaki został napotkany to brak dokumentacji komunikatora Jeti. Początkowe tworzenie modułu AmePlug podłączającego się pod komunikator Jeti odbywało się na podstawie prób i błędów. Brak dokumentacji nadrabiany był aktywnym forum dla programistów komunikatora oraz głównie duża liczba pluginów napisanych dla Jeti. To właśnie gotowe pluginy podczas implementacji były pomocne w rozwiązywaniu problemów korzystania z API Jeti.



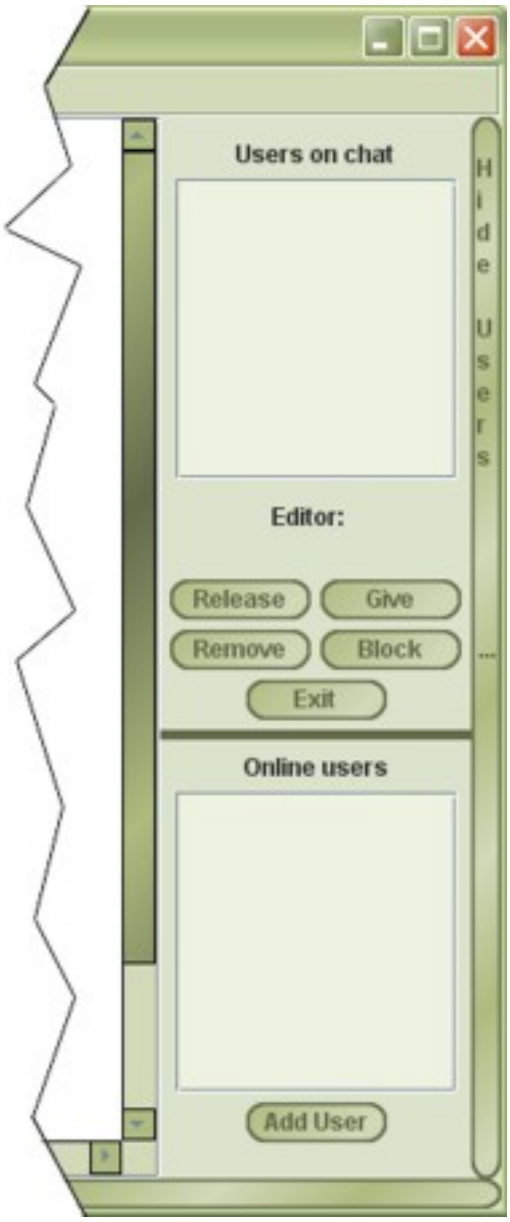
Rysunek 3.22: Okno dialogowe - Właściwości



Rysunek 3.23: Okno dialogowe - Znajdź/Zamień

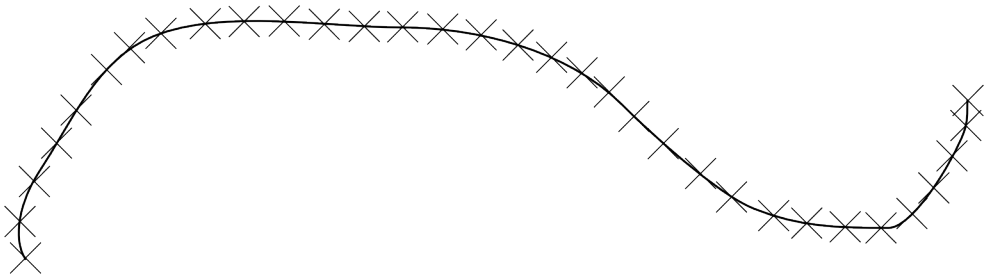
Odczytywanie statusu użytkowników

Kolejny problem pojawił się kiedy potrzebne okazało się odczytywanie statusu użytkowników (dostępny, niedostępny). Protokołu Jabber wymaga aby możliwy był odczyt statusu użytkownika



Rysunek 3.24: Paleta kolorów

przez innego użytkownika, konieczna do tego jest autoryzacja osoby której status chcemy znać. Odczyt informacji na temat statusu użytkownika głównie wiązał się z osobą edytującą czat w daje chwili, ponieważ czat implementowany w module AmePlug musi posiadać zawsze jednego użytkownika edytującego do poprawnego działania, zaszła konieczność uodpornienia aplikacji na rozłączenie się użytkownika edytującego. W tym celu zastosowano mechanizm który nasłuchuje zmianę statusu użytkownika edytującego i jeśli nadejdzie informacja o przejściu edytora czatu w stan offline, aplikacja wybiera pierwszego użytkownika z listy użytkowników znajdujących się na czacie który jest online i nadaje mu prawa edytora. W tym momencie pojawia się problem sprawdzania statusu użytkowników. W przypadku gdy na czacie znajdują się użytkownicy A, B oraz C i gdy użytkownik A jest edytorem czatu oraz występuje między nim a innymi użytkownikami obustronna autoryzacja, ale między użytkownikami B i C nie zachodzi obustronna autoryzacja występuje problem sprawdzenia statusu użytkowników po zerwaniu połączenia przez edytora czatu który jest użytkownika A. Ponieważ aplikacja wybiera pierwszego użytkownika będącego on-



Rysunek 3.25: Powolny ruch kursora

line z listy, użytkownicy B i C nie mogliby sprawdzić nawzajem statusów, wynikiem tego było by wybranie siebie jako edytora czatu, co by było z niegodne z założeniami które mówią że może występować tylko jeden edytor czatu Rozwiązaniem tego problemu jest konieczność autoryzacji miedzy użytkownikami czatu. Każdy użytkownik dołączający do czatu jest proszony o autoryzacje innych użytkowników znajdujących się na czacie, z którymi nie ma zachodzi obustronna autoryzacja.

Kolejność wiadomości

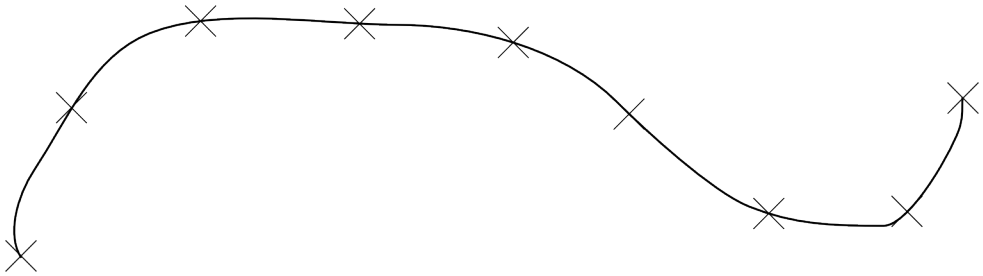
Podczas prowadzenia testów zaszła sytuacja której wynikiem było odebranie wiadomości u użytkownika zdalnego w kolejności nie zgodnej z kolejnością wysyłanych wiadomość. Ukazanie tego problemu wymusiło zaimplementowanie mechanizmu gwarantującego dostarczenie danych do modułów AmeBoard i AmeEdit w odpowiedniej kolejności. Każdy użytkownik posiada tzn. licznik wiadomości którego wartość jest jednakowa dla wszystkich użytkowników czatu. Jako że na czacie występuje tylko jedna osoba edytująca, za zmianę licznika i umieszczenie jego aktualnej wartości w wiadomości odpowiedzialny jest moduł AmePlug użytkownika edytującego. Moduł AmePlug użytkowników typu viewer odbiera wiadomości a następnie sprawdza czy aktualny licznik jest równy numerowi wiadomości, jeśli tak to inkrementuje licznik i umieszcza dane w modułach AmeBoard i AmeEdit, jeśli nie to czeka na przyjście wiadomości z odpowiednim numerem.

3.3.2 AmeBoard

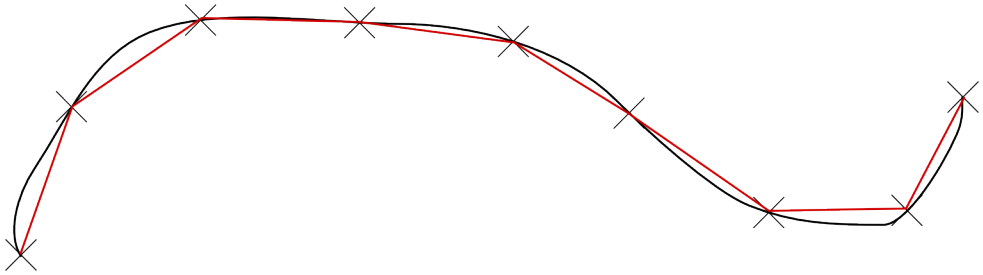
Rysowanie ścieżek

Rysowanie ścieżki, która ma odpowiadać ruchowi kursora, jest silnie zależne od zdarzeń pochodzących od myszy. Każdy ruch urządzeniem wskazującym generuje zdarzenia niosące informacje o położeniu kursora. Zależnie od szybkości poruszania kursorem myszy, różna jest ilość zdarzeń przypadająca na drogę kursora. Z innej perspektywy szybki ruch wygeneruje w przybliżeniu podobną ilość zdarzeń co ruch wolny, zakładając że oba ruchy trwały tyle samo czasu. W praktyce oznacza to, że pewna krzywa będąca odzwierciedleniem ruchu kursora, w ruchu szybkim będzie opisana przez mniej zdarzeń niż w dokładnym ruchu powolnym. Rozkład zdarzeń przypadający na krzywą ruchu kursora przedstawiają rysunki 3.25 – dla ruchu powolnego oraz 3.26 – dla ruchu szybkiego. Znak ‘x’ oznacza zarejestrowane zdarzenie myszy. Najprostszym sposobem rysowania ścieżki jest łączenie odcinkami kolejnych par punktów, które wynikają ze zdarzeń myszy. W warunkach idealnego ruchu powolnego, ścieżka będzie dokładnym odzwierciedleniem ruchu kursora i będzie miała postać zbioru wszystkich punktów leżących na krzywej. W rzeczywistych warunkach pracy programu wynikiem będzie linia łamana, co demonstruje rysunek 3.25.

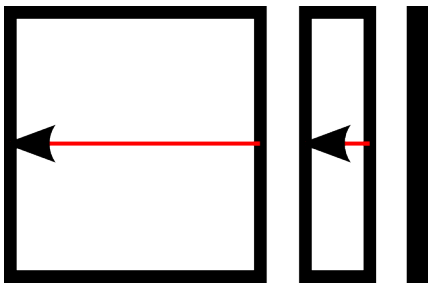
Rozwiązaniem tego problemu jest aproksymowanie ruchu kursora za pomocą krzywych parametrycznych. Krzywa musi przechodzić przez punkty pochodzące od zdarzeń myszy. Zadaniem al-



Rysunek 3.26: Szybki ruch kursora



Rysunek 3.27: Wynik połączenia punktów



Rysunek 3.28: Kolejne etapy skalowania

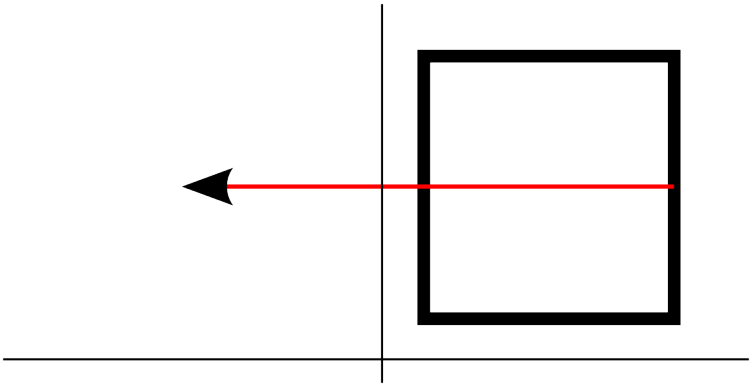
gorytmu aproksymacji jest wyznaczenie punktów kontrolnych krzywej na podstawie pewnej liczby kolejnych punktów pochodzących od myszy.

AmeBoard, w założeniu, jest prostym edytorem grafiki wektorowej, w którym nacisk położono na wspólną pracę zdalną nad jednym rysunkiem. W związku z tym rysowanie ścieżek zostało zaimplementowane w najprostszy sposób. W przyszłości warto wprowadzić algorytm aproksymacji, aby poprawić jakość rysowanych ścieżek.

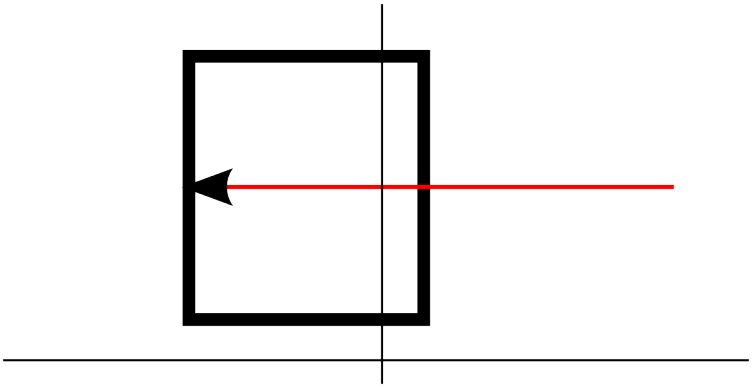
Transformacje figur

W AmeBoard transformacje figur mają charakter interaktywny, tzn. zależą bezpośrednio od poleceń wydawanych cały czas przez użytkownika za pomocą urządzenia wskazującego. Szczególnie ciekawy problem może wystąpić przy skalowaniu figur, który dla potrzeb pracy został nazwany problemem przeskalowania przez zero. Na rysunku 3.28 pokazano proces skalowania, który zmniejsza figurę w kolejnych krokach. Zmniejszając figurę coraz bardziej, program dojdzie do momentu kiedy któryś z wymiarów (szerokość lub wysokość) otrzyma wartość zero, co spowoduje, że współczynnik skalowania może przyjąć nieoczekiwaną wartość, co z kolei spowoduje nieprawidłowe przeskalowanie figury. Mając dane składowe z dwóch punktów obrazu, przy czym z_i oznacza składową przed transformacją, a z'_i po transformacji, wymiar w zdefiniujemy jako:

$$w = |z_1 - z_2|$$



Rysunek 3.29: Przed skalowaniem



Rysunek 3.30: Po skalowaniu

Wymiar w przyjmie wartość zero, kiedy $z_1 = z_2$ co w konsekwencji spowoduje:

$$S_x = \frac{z'_1 - z'_2}{z_1 - z_2} = \frac{z'_1 - z'_2}{0}$$

W AmeBoard taka sytuacja nastąpi, kiedy użytkownik będzie zmniejszał prostokąt 3.28 aż prawy bok pokryje się dokładnie z lewym. Również taka sytuacja nastąpi, kiedy użytkownik będzie chciał przeskalować figurę na przeciwną stronę któregoś z boków – użytkownik przenosi prawy bok figury na lewo od boku lewego, tak jak to pokazano na rysunku 3.29 oraz 3.30. Rozwiązaniem tego problemu, zastosowanym w AmeBoard, jest sprawdzenie czy przeskalowanie nie spowoduje zmniejszenia któregoś z wymiarów prawie do zera. Określenie 'prawie do zera' należy rozumieć jako pewien próg dokładności.

Problem ze skalowaniem pojawia się także w takich programach jak Inkscape – udało się doprowadzić do sytuacji, kiedy w efekcie ciągłego zmniejszania grupy figur, jedna z figur przestała istnieć (prawdopodobnie w efekcie przeskalowania któregoś z wymiarów do zera).

Na problem skalowania szczególnie podatne są grupy figur. Ważne jest poprawne umiejscowienie figur wewnątrz grupy, którego skalowanie zaburzać nie może. Jeśli wewnątrz grupy znajduje się odpowiednio mała figura, to skalowanie całej grupy może doprowadzić do sytuacji jak w programie Inkscape.

Skalowanie z uwzględnieniem przeskalowania przez zero było na tyle trudne, że musiało powstać kilka wersji oprogramowania nim algorytm transformacji zaczął działać poprawnie dla wszystkich założonych instancji problemu. W AmeBoard był to niewątpliwie najtrudniejszy fragment do wykonania.

Inny problem przekształcania figur, wynika z samej konstrukcji obiektów języka Java. Kształty

Rysunek 3.31: Napis przekształcony

typu prostokąt i elipsa są opisane za pomocą punktu początkowego (lewy górny narożnik) oraz wysokości i szerokości. Takie podejście skutecznie utrudnia pochylanie oraz obroty figur. Po przekształceniu figury do ścieżki problem znika, ale utracona zostaje informacja o typie figury, co może mieć znaczenie np. dla prostokąta z zaokrąglonymi brzegami.

Historia

Historia w AmeBoard jest strukturą danych, która przechowuje wszystkie elementy obrazu, tj. obiekty i zdarzenia graficzne. Takie podejście jest wygodne, ponieważ agreguje w jednym miejscu wszystkie potrzebne dane, ale z drugiej strony może być nieelastyczne. Lepszym rozwiązaniem byłoby rozdzielenie historii na dwie osobne struktury, odpowiednio dla obiektów i zdarzeń.

Obiekty graficzne

Obiekty graficzne napisów i grup wymagają szczególnego potraktowania. Grupa musi zachowywać się jak pojedynczy obiekt graficzny - jej transformacja musi propagować się na wszystkie składowe grupy. Istotnym elementem każdego obiektu graficznego jest obrys prostokątny, więc w przypadku grupy konieczne było wprowadzenie metod do jego wyliczenia na podstawie zawartości grupy.

Obiekty, które są dodawane do grupy, muszą wcześniej istnieć w historii. Kiedy grupa jest tworzona to tak naprawdę w historii pojawia się zdarzenie, a nie konkretny obiekt grupy. Wynika to z faktu, że każda składowa w historii odrysowuje się w obszarze rysowania, więc grupa nie musi się już odrysowywać. Obiekt grupy zawarty w zdarzeniu tworzy logiczne powiązanie między składowymi. Same składowe posiadają odwołanie do grupy, które jest pomocne przy przeszukiwaniu historii. W przypadku obiektów tekstowych problematyczna jest ich dualność. Napis reprezentowany jako figura podlega normalnym transformacjom graficznym. Zmiana ciągu znaków we wprowadzonym tekście nie może wpłynąć na kształt, więc nowy napis musi zostać przekształcony tak aby pasował do starego kształtu. Na rysunku 3.31 został pokazany przetransformowany napis wraz z opisującym go wektorem, który w razie zmian ciągu znaków pozwoli zachować pierwotny kształt.

Przetwarzanie SVG

Zapis i odczyt obiektów graficznych i obszaru rysowania do SVG, jest wykonywany przez klasy: `SVGReaderWriter` i `SVGMapper` . Pierwsza z nich zajmuje się parsowaniem plików SVG oraz zapisem dokumentu do pliku, zaś druga zajmuje się przetwarzaniem dokumentu i elementów historii. Przez dokument należy rozumieć obiekt typu `org.w3c.dom.Document` , który jest oparty o DOM. Wykorzystanie tego modelu bardzo ułatwia manipulowanie danymi, odczyt i zapis poszczególnych elementów dokumentu SVG. Parsowaniem plików XML (SVG to też XML) zajmuje się klasa `javax.xml.parsers.DocumentBuilder` . Jej główna zaletą jest generowanie obiektu klasy `org.w3c.dom.Document` , a główną wadą powolność przetwarzania.

AmeBoard jest prostym edytorem grafiki wektorowej, więc obsługuje tylko pewien podzbiór języka SVG. Każdy plik stworzony i zapisany przez AmeBoard jest poprawnym obrazem SVG,

ale dowolny plik SVG (nie stworzony w AmeBoard) nie musi zostać dobrze zinterpretowany przez aplikację.

3.3.3 AmeEdit

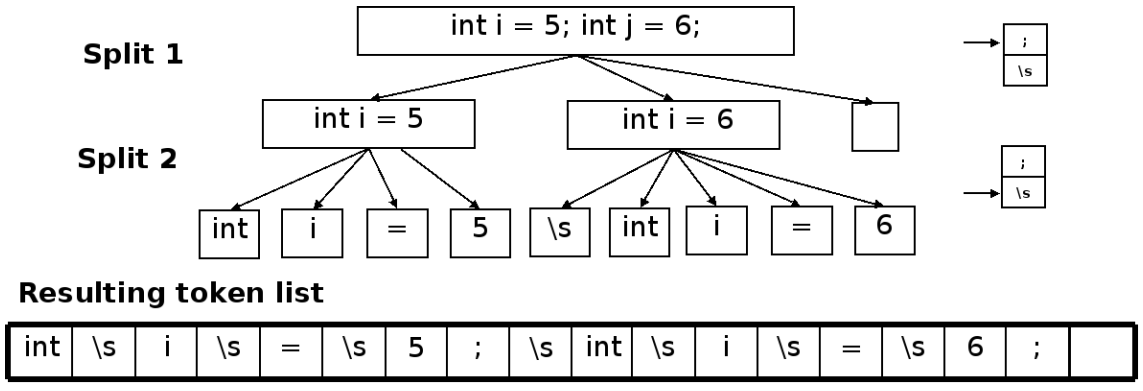
Analiza leksykalna

Implementacja funkcji związanych z edycją kodu rozpoczęta została od zbudowania mechanizmu pozwalającego na definicję języka oraz związanych z nim stylów, które następnie użyte miały być przez analizator leksykalny. Stworzone zostały schematy XML, na podstawie których następnie wygenerowane zostały pliki XML opisujące język Java, w oparciu o materiały udostępnione przez Sun w dokumentacji tego języka [JG96].

Kolejnym krokiem implementacji było zdefiniowanie struktur przechowujących informacje o dokumencie wewnątrz programu. Stworzone zostały klasy do przechowywania elementów uzyskanych z plików XML. Co ważniejsze w implementacji parserów do plików XML opisywanych przez odmienne schematy użyto parsera skonstruowanego w taki sposób, że poprzez podmianę jednej funkcji, która zajmuje się tworzeniem listy obiektów wynikowych oraz ustawienie pól odnoszących się do nazw elementów i schematu udało się uspójnić proces czytania plików i otworzyć go na możliwość rozbudowy, tak, że w później pojawiających się koniecznościach skorzystania z tego mechanizmu ponownie nie powstawały trudności w jego specjalizacji.

Otrzymane elementy, w połączeniu z prostym testowym GUI oraz standardowymi elementami zawartymi w bibliotece Swing pozwoliły na próbę zbudowania pierwszego analizatora leksykalnego, który umożliwiłby kolorowanie kodu. Wyprodukowany analizator był prototypem do testów i za cel miał umożliwienie wykrycie wszystkich potencjalnych problematycznych przestrzeni, rozważenie których dało by możliwość zaprojektowania bardziej złożonego i dokładnego analizatora leksykalnego - takiego, jak przedstawiono w podrozdziale projekt.

Pierwszy analizator działał na zasadzie rekurencyjnego podziału ciągu znaków według separatorów ustawionych na stosie, a następnie łączenie efektu podziału, wraz ze znakami używanymi do podziału, w listę tokenów. Działanie prototypu zostało zobrazowane na rysunku 3.32.

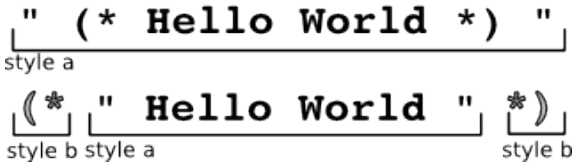


Rysunek 3.32: Działanie prototypu analizatora leksykalnego.

Po wykonaniu testów okazało się niezbędnym wyróżnienie części separatorów, które, okalając fragment tekstu, powodowały, że nie powinien on zostać wewnętrznie podzielony na dalsze tokeny, a jedynie traktowane jako jednolita całość. Separatory te, które kopertują tekst, są więc wyszczególnione w opisie języka i podział z ich pomocą odbywa się przed innymi separatorami, w taki sposób, że znajdowany jest początkowy i kończący separator, który określany jest jako jeden token, a wszelki tekst znajdujący się przed i po tym obszarze poddane zostają kolejnym rekuren-

cyjnym podziałom. Na podstawie tego mechanizmu zbudowany został drugi prototyp analizatora leksykalnego.

Drugi analizator wykazał, że niezbędna będzie implementacja procesu analizy w odmienny sposób. Wykonywanie rekurencyjnych operacji dzielenia ciągu znaków było podatne na błędy związane z kolejnością definicji separatorów otaczających. Szczególnie zauważalny było niewłaściwe kolorowanie jednego typu struktur okalających wewnątrz drugich, gdzie w zależności od ustawienia ich kolejności względem siebie w pliku konfiguracyjnym można było zaobserwować, że jeśli kolejnościowo druga struktura okala tą pierwszą, to kolorowanie będzie wykonane niewłaściwie, tak jak zostało to pokazane na rysunku 3.33, gdzie zarówno para tokenów '(*', '*)' jak i token '"' są takimi strukturami. Chociaż ten błąd można znaleźć w części oprogramowania dostępnego na rynku (przykładowo w programie gedit w wersji 2.20.4 z włączonym kolorowaniem dla języka OCaml), to zdecydowano się dążyć do jego wyeliminowania w końcowej wersji analizatora.



Rysunek 3.33: Błąd wstępnych wersji analizatora leksykalnego.

Na podstawie wyniesionych nauk i wniosków z dwóch prototypów rozpoczęto implementację właściwego analizatora, opartego na przeglądaniu ciągu znaków z możliwością cofania, wykonującego operacje na podciągach znaków w zależności od swojego stanu. Wszelkie kolejne modyfikacje wytworzonego analizatora miały na celu poprawę efektywności jego działania.

Interakcja między warstwami

Implementacja warstw związanych z modelu MVC zastosowanego w bibliotece graficznej Swing skupia się przede wszystkim na rozszerzeniu klas dostarczanych przez tą bibliotekę o funkcje potrzebne do przetwarzania tekstu na tokeny i dostosowania mechanizmu analizatora leksykalnego do współpracy ze zdarzeniami wysyłanymi przez warstwy.

Głównym problemem związanym tą częścią implementacji było zminimalizowanie komunikacji między obiektami z poszczególnych warstw, które powodowały niedopuszczalnie duże opóźnienia przy edycji przez użytkownika (rzędu kilku sekund lub nawet do minuty w przypadku dużego pliku). Wynikało to z faktu, że przy każdej edycji, która w wielu wypadkach sprowadza się do wprowadzenia jednego znaku do tekstu, cały tekst zostaje przeanalizowany i stary tekst zostaje zastąpionym nowym. W połączeniu ze sposobem wprowadzania zmian do dokumentu powodowało to, że przy każdej próbie edycji z warstwy modelu do warstwy widoku została wysyłana nieproporcjonalnie duża do wprowadzonej zmiany liczba zdarzeń. Rozwiązaniem tego problemu było ograniczanie zmian faktycznie wykonywanych na dokumencie poprzez zastosowanie algorytmu wykrywania zmian w listach tokenów, opisanego w projekcie.

Rozwiązanie to nie jest jeszcze rozwiązaniem idealnym, ze względu na fakt, że analizator leksykalny za każdym razem musi skanować całość tekstu, jednakże próba znalezienia uniwersalnego algorytmu, który byłby w stanie wyróżnić minimalny podciąg, na którym powinna być zaaplikowany analizator spełzła na niczym, ze względu na dodatkowy koszt w postaci wykonywanych przy każdym działaniu obliczeń i działań na ciągach znaków. Przy założeniu, że edytowane pliki rzadko przekraczałyby rozmiar kilkuset linii kodu wprowadzenie kolejnego algorytmu na wejściu filtra powodowałoby z najwyższym prawdopodobieństwem jedynie straty efektywności.

Operacje zdalne

Kolejną klasą zagadnień są te związane z wprowadzeniem komunikacji z użytkownikiem zdalnym. Trzonem tych zagadnień są sposoby zabezpieczania przed niepowołaną edycją przez użytkowników oraz ich konsekwencje w warstwach modelu i kontroli modułu. W ogólności użytkownicy mogą posiadać możliwość edycji lub mieć ją zablokowaną, jednak u użytkownika który tylko oczekuje na nadejście wiadomości musi istnieć sposób na modyfikację wyświetlanej treści (poprzez modyfikację modelu) przez zdarzenia otrzymywane od użytkownika zdalnego. Początkowo założony więc binarny system uprawnień został w implementacji rozszerzony do trójstanowego systemu, który poza możliwością edycji i jej braku ma także stan *update*, który służy do synchronizacji wersji obecnej z tą, którą opisują przychodzące wiadomości. Wprowadzenie tego stanu spowodowało, że zdarzenia wywoływać mogą metody w sposób jaki są one wywoływane przez interfejs użytkownika po przestawieniu tymczasowo stanu dokumentu i związanych z nim obiektów.

Samo przesyłanie i odbieranie zdarzeń zostało zaimplementowane w sposób opisany w projekcie, przy użyciu implementujących wspólny interfejs klas. Każdy z obiektów typu tych klas posiada funkcje statyczną, za pomocą której dokonywana jest jego inicjacja, oraz zestaw stałych, określających pola, jakie się do niego stosują. Po utworzeniu tego typu obiektu wiadomości zostaje on wysłany do warstwy nadrzędnej. Użytkownik zdalny odebrawszy wiadomość rozpoznaje jej typ, a następnie wysyła do obsługi przez dedykowane obiekty nasłuchujące, które przeprowadzają wymagane operacje na dokumencie.

Zachowanie kursora i zakreślacza

Wykonanie obsługi edycji na modelu dokumentu wymaga dokonywania obliczeń pozycji kursora, który, w zależności od miejsca i typu wykonanej operacji może zostać przemieszczony w dowolnym kierunku, pozostać w miejscu oraz zmienić swój kształt. W implementacji (oraz przy testowaniu) jest to trudny problem z uwagi na fakt istnienia rozległej przestrzeni warunków nadzwyczajnych, z których część łatwo pominąć podczas programowania. Przemieszczanie kursora jest w konsekwencji tego najmniej stabilnym elementem w module.

Od tego rodzaju problemów wolna też nie jest warstwa widoku, w której element zakreślacza boryka się z podobnymi problemami, rozszerzonymi w dodatku o aspekty związane z przesuwaniem się tekstu. Zakreślacz musi być w stanie modyfikować kształt i pozycję istniejących już zakreśleń, a nawet je usuwać wraz ze zmianami dokonywanymi w modelu.

3.3.4 AmeGUI

Programowanie graficznego interfejsu użytkownika polega na pisaniu kodu odpowiedzialnego za wygląd aplikacji. Java dostarcza zestawy gotowych klas komponentów, które mogą być dowolnie rozbudowane przez programistę.

Graficzny interfejs użytkownika oprócz samego wyświetlania komponentów reaguje na zdarzenia pochodzące od użytkownika. Java dzięki zastosowanej koncepcji delegacyjnego modelu zdarzeń, umożliwia przekazanie obsługi zdarzenia od źródła zdarzenia do innego obiektu – słuchacza zdarzenia. Dzięki temu możliwe było połączenie ze sobą poszczególnych modułów i ich wzajemne powiązanie z graficznym interfejsem użytkownika. Odrębne klasy słuchaczy pozwoliły na odseparowanie obsługi zdarzeń od GUI.

Wszystkie komponenty lekkie, takie jak przyciski, listy czy menu, wywodzą się ze wspólnej klasy, `JComponent` , która udostępnia wiele użytecznych właściwości i metod, między innymi metodę `paintComponent()` . Poprzez odpowiednią implementację metody tej metody, z wyko-

rzystaniem pakietu java.graphics2D, można wykreślić dowolną grafikę na powierzchni wybranego komponentu. Taka metoda sprawdza się dla mniej złożonych komponentów takich jak przyciski, panele czy etykiety.

Niektóre komponenty wymagają implementacji kilku szczegółowych metod odpowiedzialnych za ich wygląd, dlatego też powyżej opisana metoda nie zawsze jest najlepszym rozwiązaniem. W niektórych przypadkach należy skorzystać z odpowiednich klas z pakietu javax.swing.plaf.basic. Pakiet ten definiuje abstrakcyjne klasy - delegaty interfejsu użytkownika (UI) dla komponentów swinga. Każdy komponent lekki posiada swój odpowiednik w klasach z pakietu javax.swing.plaf.basic (np. `JComboBox` – `BasicComboBoxUI`). Klasy te udostępniają wiele metod odpowiedzialnych za wygląd bardziej szczegółowych elementów danego komponentu (np. wygląd listy podczas rozwinięcia, zakładek czy też suwaków). Dzięki temu poprzez stworzenie nowych klas dziedziczących z komponentów lekkich, można w najdrobniejszych szczegółach zaprogramować wygląd dowolnego komponentu. Utworzone w ten sposób poszczególne komponenty były dalej podstawą do zaprogramowania interfejsu użytkownika [Fla99].

Komponentom lekkim można nadawać dowolne kształty czy przezroczystość, jednak nie można zastosować tej metody kreślenia elementów dla komponentów ciężkich takich jak okna główne czy okna dialogowe `JFrame` , `JDialog` itd.), gdyż za ich wygląd odpowiadają graficzne biblioteki GUI systemu operacyjnego. Zmiana wyglądu komponentu ciężkiego np. samo stworzenie własnego paska tytułu do komponentu `JFrame` wiązałaby się z obsługą zdarzeń odpowiedzialnych za przenoszenie oraz rozciąganie okna, co nie jest na pewno trywialnym zadaniem. W rezultacie uzyskałoby mało płynne przejścia przy przeciąganiu czy przesuwaniu okna, co dałoby mało estetyczny wygląd [Rog03].

Rozdział 4

Testowanie

Testowanie oprogramowania jest bardzo istotnym elementem procesu tworzenia oprogramowania, pomaga zapewnić odpowiednią jakość końcowego produktu. Testowanie aplikacji opiera się przede wszystkim na testach jednostkowych wykonywanych indywidualnie, a także integracyjnych. Dodatkowo przez programistów wykonywane są wzajemne inspekcje fragmentów kodu, które pomagają sprawdzić poprawność lub też rozwiązać trudny problem.

Proces testowania oprogramowania Amebae wspomagany jest narzędziem do śledzenia błędów Mantis Bug Tracker. System ten pozwala na rejestrowanie wykrytych nieprawidłowości, błędów i wyjątków oraz zarządzanie nimi. Każdy z uczestników projektu testując swój fragment oprogramowania w ramach uruchomionej aplikacji ma szansę wykryć i często wykrywa błędy w pozostałych częściach kodu. Dzięki temu wiele modułów projektu jest testowanych wielokrotnie przez różne osoby, a, że każdy z uczestników wie niewiele o pozostałych częściach, testy te mają charakter wykonanych przez potencjalnego użytkownika końcowego lub klienta.

Kolejnym narzędziem wspomagającym proces testowania jest mechanizm zadań w Eclipse (Tasks). Dzięki zastosowaniu znaczników `TODO` i `FIXME` w komentarzach pomiędzy liniami kodu, można oznaczać miejsca w których wykryto błędy lub potencjalnie niebezpieczne miejsca, a także punkty do zrobienia i rozwinięcia. Platforma Eclipse zapewnia widok Tasks, który pozwala przeglądać wprowadzone zadania `TODO` i `FIXME` wraz z odnośnikami do miejsc w kodzie.

4.1 Plany testów

4.1.1 Wprowadzenie

Większość testów systemu Amebae została opracowana na podstawie przypadków użycia. Testy dla konkretnego przypadku użycia są z reguły jego rozwinięciem, co wynika z potrzeby możliwie wyczerpującego przetestowania danego przypadku.

Za przeprowadzenie testów odpowiedzialni są poszczególni wykonawcy modułów oprogramowania, przy czym dodatkowo wszystkie testy wykonują także pozostali członkowie zespołu.

Identyfikatory przypadków testowych są tworzone według schematu XTCD, gdzie:

X litera oznaczająca moduł, G – testy wspólne dla wszystkich modułów, B – testy AmeBoard, E – testy AmeEdit

TC (ang. *test case*) przypadek testowy

D kolejny numer przypadku testowego

4.1.2 Środowiska testowe

Wszystkie testy zostaną wykonane na następujących maszynach (procesor (GHz); pamięć RAM (MB), systemy operacyjne):

1. Intel Pentium D 2,8; 2048; Microsoft Windows Vista Business, openSUSE 10.3
2. Intel Celeron M 1,3; 512; Microsoft Windows XP Professional Edition SP2, openSUSE 10.3
3. Intel Celeron M 1,4; 1024; Microsoft Windows XP Home Edition SP2, openSUSE 10.3
4. Intel Pentium 4M 2.00; 1024; Debian Release testing (lenny) Kernel Linux 2.6.22-3-686 GNOME 2.20.2
5. Intel Core 2 Duo 1,8; 2048; Microsoft Windows XP Professional Edition SP2

Niektóre testy zostaną wykonane na słabszym komputerze, w celu sprawdzenia zapotrzebowania na zasoby przez oprogramowanie Amebae. Parametry komputera (według powyższego schematu): Pentium II 0,366; 192; Fluxbuntu 7.10

Do testów sieciowych wymagane jest połączenie sieciowe oraz więcej niż jedna instancja programu Amebae. Wymagana wersja wirtualnej maszyny: Java™ SE Runtime Environment 6 (build 1.6.0_03-b05).

4.1.3 Testy wspólne

GTC1: Próba otwarcia nieistniejącego pliku

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BTC10, BTC11, BTC13, BTC14

Stan systemu przed testem: Aplikacja jest włączona.

Dane wejściowe lub warunki testu:

1. Użytkownik wpisuje nazwę pliku do otwarcia,
2. Użytkownik zatwierdza otwarcie pliku.

Wynik oczekiwany: System wyświetla komunikat o braku pliku i zmusza użytkownika do wpisania innej nazwy lub przerwania operacji.

GTC2: Próba nadpisania pliku

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BTC12, BTC20, BTC21

Stan systemu przed testem: W systemie plików systemu operacyjnego użytkownika istnieje plik o pewnej nazwie.

Dane wejściowe lub warunki testu:

1. Użytkownik wpisuje nazwę pliku taką samą jak innego pliku w systemie plików,
2. Użytkownik zatwierdza zapis.

Wynik oczekiwany: System wyświetla pytanie o nadpisanie pliku z możliwością przerwania lub kontynuacji.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 05.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania:

Stan systemu przed testem: Stan odpowiedni dla danego testu. Użytkownik ma prawa edycyjne.

Dane wejściowe lub warunki testu: Użytkownik wykonuje test: BTC1, BTC2, BTC3, BTC5, BTC6, BTC7, BTC10, BTC13, BTC16, BTC17, BTC18, BTC19, BTC22, BTC24, GTC4, GTC5, ETC1, ETC4, ETC5, ETC11, ETC12, ETC13.

Wynik oczekiwany: Wynik oczekiwany zgodny z wybranym testem, ale sprawdzony na komputerze lokalnym oraz zdalnym.

GTC4: Cofanie operacji – *undo*

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 05.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania:

Stan systemu przed testem: Aplikacja jest uruchomiona.

Dane wejściowe lub warunki testu: Użytkownik wykonuje polecenie *undo*.

Wynik oczekiwany: Zostaje cofnięta ostatnio wykonana operacja, jeśli żadna nie została jeszcze wykonana to system nie reaguje.

GTC5: Powtarzanie cofniętej operacji – *redo*

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 05.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania:

Stan systemu przed testem: Aplikacja jest uruchomiona.

Dane wejściowe lub warunki testu: Użytkownik wykonuje polecenie *redo*.

Wynik oczekiwany: Uprzednio cofnięta operacja jest wykonywana, jeśli żadna operacja nie była cofnięta to system nie reaguje.

4.1.4 Testy AmeBoard

BTC1: Kreślenie prostych figur

Atrybuty:

Źródło: Joanna Daukszewicz

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC1

Stan systemu przed testem: Wybrano narzędzie do wyrysowania figury (prostokąt, prostokąt z zaokrąglonymi brzegami, elipsa, linia prosta), kursor myszy umieszczony w obszarze rysowania.

Dane wejściowe lub warunki testu:

1. Użytkownik wciska lewy klawisz myszy,
2. Użytkownik przeciąga kursorem po obszarze rysowania,
3. Użytkownik wyciska lewy klawisz myszy w miejscu innym niż punkt wciśnięcia.

Wynik oczekiwany: Pojawienie się na obszarze rysowania figury, umiejscowionej zgodnie z punktami wciśnięcia i wyciśnięcia klawisza myszy.

BTC2: Kreślenie dowolnych ścieżek (kredka)

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.07

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC1

Stan systemu przed testem: Wybrano narzędzie do rysowania – kredkę, kursor myszy umieszczony w obszarze rysowania.

Dane wejściowe lub warunki testu:

1. Użytkownik wciska lewy klawisz myszy,
2. Użytkownik przeciąga kursorem po obszarze rysowania kreśląc dowolną figurę,
3. Użytkownik wyciska lewy klawisz myszy w dowolnym miejscu obszaru rysowania.

Wynik oczekiwany: Pojawienie się na obszarze rysowania figury odpowiadającej drodze po której poruszał się kursor.

BTC3: Kreślenie łamanej

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.07

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC1

Stan systemu przed testem: Wybrano narzędzie do rysowania – kredkę, kursor myszy umieszczony w obszarze rysowania.

Dane wejściowe lub warunki testu:

1. Użytkownik klika myszą,
2. Użytkownik przesuwa kursor,
3. Użytkownik ponownie klika myszą,
4. Użytkownik powtarza czynności z punktów 2. i 3. aż do momentu wciśnięcia prawego klawisza myszy.

Wynik oczekiwany: Pojawienie się odcinków pomiędzy punktami kliknięcia, co daje w efekcie linię łamaną.

BTC4: Ustawienie atrybutów kreślonych figur

Atrybuty:

Źródło: Joanna Daukszewicz

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC2

Stan systemu przed testem: Włączono panel z atrybutami kreślonych figur (osobne okno dla wielu własności, panel z podstawowymi kolorami).

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera grupę atrybutów,
2. Użytkownik zmienia wybrane ustawienia,
3. Użytkownik rysuje wybraną figurę, która wykorzystuje zmienione atrybuty.

Wynik oczekiwany: Narysowana figura powinna posiadać widocznie zmienione atrybuty.

BTC5: Przesuwanie narysowanych figur – ze wstępnym zaznaczeniem

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC3

Stan systemu przed testem: W obszarze rysowania znajdują się narysowane obiekty. Użytkownik klika na wybraną przez siebie figurę, a system ją zaznacza. Kursor myszy znajduje się nad zaznaczoną figurą.

Dane wejściowe lub warunki testu:

1. Użytkownik wciska przycisk myszy,
2. Użytkownik przesuwa figurę,
3. Użytkownik wyciska przycisk myszy.

Wynik oczekiwany: Figura ulega przesunięciu zgodnie z ruchem kursora myszy.

BTC6: Przesuwanie narysowanych figur – bez wstępnego zaznaczenia

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC3

Stan systemu przed testem: W obszarze rysowania znajdują się narysowane obiekty. Kursor myszy znajduje się nad wybraną figurą.

Dane wejściowe lub warunki testu:

1. Użytkownik wciska przycisk myszy,
2. Użytkownik przesuwa figurę,
3. Użytkownik wyciska przycisk myszy.

Wynik oczekiwany: Figura ulega przesunięciu zgodnie z ruchem kursora myszy.

BTC7: Zmiana atrybutów wybranej figury

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Niski

Powiązania: BUC4

Stan systemu przed testem: W obszarze rysowania znajdują się figury. Dokładnie jedna z nich, nie będąca grupą, jest wybrana.

Dane wejściowe lub warunki testu:

1. Użytkownik zmienia atrybuty figury,
2. Użytkownik zatwierdza zmiany.

Wynik oczekiwany: Figura zmieniła swoje właściwości zgodnie z wprowadzoną zmianą.

BTC8: Próba zmiany atrybutów grupy

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 27.12.2007

Zmodyfikowano:

Priorytet: Niski

Powiązania: BUC4

Stan systemu przed testem: W obszarze rysowania znajduje się przynajmniej jedna grupa, dokładnie jedna z nich jest wybrana.

Dane wejściowe lub warunki testu: Użytkownik próbuje zmienić atrybuty graficzne.

Wynik oczekiwany: System nie pozwala na zmianę.

BTC9: Próba zmiany atrybutów kilku zaznaczonych figur

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 27.12.2007

Zmodyfikowano:

Priorytet: Niski

Powiązania: BUC4

Stan systemu przed testem: W obszarze rysowania znajdują się narysowane figury, kilka z nich zostało zaznaczonych.

Dane wejściowe lub warunki testu: Użytkownik próbuje zmienić jakikolwiek atrybut zaznaczonych figur.

Wynik oczekiwany: System nie pozwala na zmianę.

BTC10: Odczytanie prawidłowego pliku z rysunkiem

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC5

Stan systemu przed testem: Aplikacja jest włączona, istnieje przynajmniej jeden plik z rysunkiem w obsługiwanym formacie.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera plik z rysunkiem w obsługiwanym formacie,
2. Użytkownik zatwierdza otwarcie pliku,
3. System odczytuje plik.

Wynik oczekiwany: Odczytanie rysunku i umieszczenie go w obszarze rysowania.

BTC11: Próba odczytania nieprawidłowego pliku z rysunkiem

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC5

Stan systemu przed testem: Aplikacja jest włączona, istnieje przynajmniej jeden plik z rysunkiem w nieobsługiwanym formacie lub o niepoprawnej zawartości.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera plik z rysunkiem w nieobsługiwanym formacie lub o niepoprawnej zawartości,
2. Użytkownik zatwierdza otwarcie pliku,

3. System odczytuje plik.

Wynik oczekiwany: Pojawienie się komunikatu o nieobsługiwanym formacie lub niepoprawnej zawartości pliku, brak zmian w obszarze rysowania.

BTC12: Eksport figury do pliku

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC6

Stan systemu przed testem: W obszarze rysowania znajduje się przynajmniej jedna figura, dokładnie jedna jest zaznaczona.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera opcję eksportu figury do pliku,
2. Użytkownik wybiera nazwę pliku,
3. System zapisuje figurę do pliku.

Wynik oczekiwany: Pojawienie się w systemie plików systemu operacyjnego użytkownika pliku o podanej wcześniej nazwie. Plik posiada prawidłowy format i zawartość adekwatną do wybranej figury.

BTC13: Import figury z prawidłowego pliku

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC7

Stan systemu przed testem: Aplikacja jest włączona, istnieje przynajmniej jeden plik w systemie plików systemu operacyjnego użytkownika z figurą w obsługiwanym formacie.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera poprawny plik z figurą,
2. Użytkownik zatwierdza otwarcie pliku,
3. System odczytuje plik.

Wynik oczekiwany: Pojawienie się w obszarze rysowania figury zgodnej z zapisem w pliku.

BTC14: Próba zaimportowania nieprawidłowego pliku z rysunkiem

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC7

Stan systemu przed testem: Aplikacja jest włączona, istnieje przynajmniej jeden plik w systemie plików systemu operacyjnego użytkownika z figurą w nieobsługiwanym formacie lub o niepoprawnej zawartości.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera niepoprawny plik z figurą,
2. Użytkownik zatwierdza otwarcie pliku,
3. System odczytuje plik.

Wynik oczekiwany: Pojawienie się komunikatu o nieobsługiwanym formacie lub o niepoprawnej zawartości, brak zmian w obszarze rysowania.

BTC15: Weryfikacja pliku z wyeksportowaną figurą

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BTC12, BUC6, BTC13, BTC14

Stan systemu przed testem: W systemie plików systemu operacyjnego użytkownika istnieje plik z wyeksportowaną wcześniej figurą – eksport został wykonany poprawnie, według testu BTC12.

Dane wejściowe lub warunki testu: Użytkownik wykonuje test BTC13 lub BTC14 dla eksportowanego pliku.

Wynik oczekiwany: Zgodny z wynikiem testu BTC13 lub BTC14.

BTC16: Wpisywanie tekstu w obszarze rysowania

Atrybuty:

Źródło: Tomasz Kaszkowiak

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC8

Stan systemu przed testem: Wybrano narzędzie do wpisywania tekstu, kursor znajduje się w obszarze rysowania.

Dane wejściowe lub warunki testu:

1. Użytkownik klika w obszarze rysowania,
2. System pokazuje okno do wpisania tekstu,
3. Użytkownik wpisuje tekst,
4. Użytkownik zatwierdza wpisany tekst.

Wynik oczekiwany: Pojawienie się napisu na rysunku.

BTC17: Modyfikacja tekstu wpisanego

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC17

Stan systemu przed testem: W obszarze rysowania znajduje się przynajmniej jeden napis, wybrano narzędzie do wpisywania tekstu.

Dane wejściowe lub warunki testu:

1. Użytkownik klika w wybrany napis z obszaru rysowania,
2. System wyświetla okno modyfikacji wybranego tekstu,
3. Użytkownik modyfikuje tekst,
4. Użytkownik zatwierdza zmiany.

Wynik oczekiwany: Pojawienie się w obszarze rysowania zmodyfikowanej wersji napisu.

BTC18: Modyfikacja atrybutów tekstu wpisanego

Atrybuty:

Źródło: Tomasz Kaszkowiak

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Średni

Powiązania: BUC9

Stan systemu przed testem: W obszarze rysowania znajduje się przynajmniej jeden napis, wybrano narzędzie do wpisywania tekstu.

Dane wejściowe lub warunki testu:

1. Użytkownik klika w wybrany napis z obszaru rysowania,
2. System wyświetla okno modyfikacji wybranego tekstu,
3. Użytkownik modyfikuje atrybuty tekstu,
4. Użytkownik zatwierdza zmiany.

Wynik oczekiwany: Widoczna zmiana atrybutów modyfikowanego napisu.

BTC19: Skalowanie figur

Atrybuty:

Źródło: Tomasz Kaszkowiak

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC10

Stan systemu przed testem: W obszarze rysowania znajduje się przynajmniej jedna figura, dokładnie jedna jest wybrana. Kursor znajduje się nad jednym z dziewięciu punktów manipulacji figurą.

Dane wejściowe lub warunki testu:

1. Użytkownik wciska klawisz myszy,
2. Użytkownik przeciąga kursor przesuwając przy tym punkt manipulacji figurą,
3. Użytkownik wyciska klawisz myszy.

Wynik oczekiwany: Figura uzyskuje kształt zgodny ze zmianą jej obrysu.

BTC20: Zapis rysunku do pliku

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC16

Stan systemu przed testem: Aplikacja jest uruchomiona.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera opcję zapisu do pliku,
2. Użytkownik wpisuje nazwę pliku,
3. Użytkownik zatwierdza zapis.

Wynik oczekiwany: Pojawienie się w systemie plików systemu operacyjnego użytkownika pliku z obrazem rysunku (w poprawnym formacie).

BTC21: Eksport rysunku do pliku bitmapy

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet: Średni

Powiązania: BUC14

Stan systemu przed testem: Aplikacja jest uruchomiona.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera opcję eksportu rysunku do pliku bitmapy,
2. Użytkownik wpisuje nazwę pliku,
3. Użytkownik zatwierdza zapis.

Wynik oczekiwany: Pojawienie się w systemie plików systemu operacyjnego użytkownika pliku z obrazem rysunku w postaci bitmapy.

BTC22: Grupowanie figur

Atrybuty:

Źródło: Paweł Martenka, Konrad Siek

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: BUC13

Stan systemu przed testem: W obszarze rysowania znajdują się conajmniej dwa obiekty graficzne, conajmniej dwa są zaznaczone.

Dane wejściowe lub warunki testu:

1. Użytkownik wydaje polecenie zgrupowania,
2. Użytkownik wykonuje operację aplikowalną dla grupy (np. przesuwanie), próbując jej na pojedynczym elemencie grupy.

Wynik oczekiwany: Zaznaczone figury stają się logiczną całością, wykonanie operacji przesunięcia na składowej grupy przesunie wszystkie elementy grupy.

BTC24: Rozgrupowanie grupy

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet:

Powiązania: BUC18, BTC5, BTC6

Stan systemu przed testem: W obszarze rysowania znajduje się conajmniej jedna grupa, conajmniej jedna jest wybrana.

Dane wejściowe lub warunki testu:

1. Użytkownik wydaje polecenie rozgrupowania,
2. Użytkownik próbuje przesunąć jedną z figur składowych - wykonuje test BTC5 lub BTC6.

Wynik oczekiwany: Zostaje przesunięta tylko jedna z figur składowych grupy.

BTC25: Próba rozgrupowania figury nie będącej grupą

Atrybuty:

Źródło: Paweł Martenka

Pozyskano: 26.12.2007

Zmodyfikowano:

Priorytet: Niski

Powiązania: BTC24

Stan systemu przed testem: W obszarze rysowania znajduje się przynajmniej jedna figura nie będąca grupą, dokładnie jedna jest wybrana.

Dane wejściowe lub warunki testu: Użytkownik wydaje polecenie rozgrupowania.

Wynik oczekiwany: System nie reaguje.

4.1.5 Testy AmeEdit

ETC1: Importowanie kodu z pliku

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 05.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: EUC1, ETC1

Stan systemu przed testem: W systemie plików systemu operacyjnego użytkownika istnieje przynajmniej jeden plik z kodem.

Dane wejściowe lub warunki testu:

1. Użytkownik wskazuje plik do otwarcia,
2. Użytkownik zatwierdza otwarcie,
3. System importuje plik.

Wynik oczekiwany: W edytorze pojawia się kod z pliku, kod jest pokolorowany zgodnie z zasadami zdefiniowanymi w AmeEdit.

ETC2: Kolorowanie składni

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 05.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: EUC2

Stan systemu przed testem: Aplikacja jest uruchomiona.

Dane wejściowe lub warunki testu: Użytkownik dostarcza kod do edytora.

Wynik oczekiwany: Kod jest kolorowany zgodnie z zasadami zdefiniowanymi w AmeEdit.

ETC3: Aktywny kursor

Atrybuty:

Źródło: Tomasz Kaszkowiak

Pozyskano: 07.04.2007

Zmodyfikowano: 11.04.2007, 12.04.2007 Konrad Siek

Priorytet: Wysoki

Powiązania: EUC3

Stan systemu przed testem: W edytorze znajduje się więcej niż jeden znak. Użytkownik ma prawa do edycji zawartości pliku.

Dane wejściowe lub warunki testu: Użytkownik zmienia położenie kursora.

Wynik oczekiwany: Kursor zmienia położenie na komputerze zdalnym zgodnie ze zmianą na komputerze lokalnym.

ETC4: Wycinanie i wklejanie tekstu

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: EUC5

Stan systemu przed testem: W edytorze znajduje się kod.

Dane wejściowe lub warunki testu:

1. Użytkownik zaznacza fragment tekstu,
2. Użytkownik wkleja wycięty wcześniej tekst.

Wynik oczekiwany: Wycięty tekst znika z edytora, jest zapamiętywany, a po wklejeniu pojawia się w wybranym przez użytkownika miejscu. Wycięty i wklejony fragment tekstu zachowuje kolejność znaków.

ETC5: Przywracanie wcześniejszej sesji

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Średni

Powiązania: EUC7

Stan systemu przed testem: W systemie plików systemu operacyjnego użytkownika istnieje plik z kodem i plik sesji z historią jego modyfikacji.

Dane wejściowe lub warunki testu:

1. Użytkownik wskazuje plik z kodem i plik sesji,
2. Użytkownik zatwierdza otwarcie.

Wynik oczekiwany: System jest przywracany do stanu zgodnego ze stanem przed zapisaniem systemu do pliku sesji.

ETC6: Zapisywanie kodu do pliku

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 11.04.2007

Zmodyfikowano:

Priorytet: Wysoki

Powiązania: EUC8

Stan systemu przed testem: W edytorze znajduje się kod do zapisania.

Dane wejściowe lub warunki testu:

1. Użytkownik określa nazwę pliku,
2. Użytkownik zatwierdza zapis.

Wynik oczekiwany: W systemie plików systemu operacyjnego użytkownika pojawia się plik o podanej wcześniej nazwie, jego zawartość jest odzwierciedleniem zawartości edytora.

ETC7: Importowanie poprawnej definicji składni z pliku

Atrybuty:

Źródło: Tomasz Kaszkowiak

Pozyskano: 11.04.2007

Zmodyfikowano: 12.04.2007 Konrad Siek

Priorytet: Wysoki

Powiązania: EUC9

Stan systemu przed testem: W systemie plików systemu operacyjnego użytkownika istnieje przynajmniej jeden plik z poprawną definicją składni języka.

Dane wejściowe lub warunki testu: System importuje nową składnię.

Wynik oczekiwany: Do AmeEdit zostaje dodana nowa definicja składni języka, której można później użyć.

ETC8: Próba importowania niepoprawnej definicji składni z pliku

Atrybuty:

Źródło: Tomasz Kaszkowiak

Pozyskano: 11.04.2007

Zmodyfikowano: 12.04.2007 Konrad Siek

Priorytet: Wysoki

Powiązania: EUC9

Stan systemu przed testem: W systemie plików systemu operacyjnego użytkownika istnieje przynajmniej jeden plik z niepoprawną definicją składni języka.

Dane wejściowe lub warunki testu: Użytkownik wybiera niepoprawną definicję składni.

Wynik oczekiwany: System odrzuca składnię i informuje o błędnej definicji.

ETC9: Dostosowanie formatu kolorowania składni

Atrybuty:

Źródło: Tomasz Kaszkowiak

Pozyskano: 11.04.2007

Zmodyfikowano: 12.04.2007 Konrad Siek

Priorytet: Średni

Powiązania: EUC10, ETC2

Stan systemu przed testem: Aplikacja jest uruchomiona.

Dane wejściowe lub warunki testu: Użytkownik wybiera format (kolor, styl) zbiorów słów opisanych przez składnię.

Wynik oczekiwany: System koloruje składnię zgodnie z nowymi zasadami.

ETC10: Drukowanie pliku z kodem

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet: Niski

Powiązania: EUC12

Stan systemu przed testem: W edytorze znajduje się kod. W systemie istnieje poprawnie skonfigurowana, działająca drukarka. Istnieje połączenie z drukarką.

Dane wejściowe lub warunki testu:

1. Użytkownik wybiera opcje drukowania,
2. Użytkownik wydaje polecenie drukowania.

Wynik oczekiwany: Kod zostaje wydrukowany na określonej drukarce.

ETC11: Szukanie fragmentu tekstu – za pomocą słowa

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet: Średni

Powiązania: EUC13

Stan systemu przed testem: W edytorze istnieje słowo, które ma zostać wyszukane.

Dane wejściowe lub warunki testu:

1. Użytkownik wpisuje słowo, które chce wyszukać,
2. Użytkownik wydaje polecenie wyszukania.

Wynik oczekiwany: System zaznacza podświetlenie dla wszystkich znalezionych elementów, znalezione elementy są identyczne z szukanym słowem.

ETC12: Szukanie fragmentu tekstu – za pomocą wyrażenia regularnego

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet: Średni

Powiązania: EUC13

Stan systemu przed testem: W edytorze istnieje słowo, które ma zostać wyszukane.

Dane wejściowe lub warunki testu:

1. Użytkownik wpisuje wyrażenie regularne, które pasuje do danego słowa,
2. Użytkownik wydaje polecenie wyszukania.

Wynik oczekiwany: System zaznacza podświetlenie dla wszystkich znalezionych elementów, wśród znalezionych elementów znajduje się przygotowane wcześniej słowo.

ETC13: Zamiana fragmentu kodu na inny

Atrybuty:

Źródło: Konrad Siek

Pozyskano: 12.04.2007

Zmodyfikowano:

Priorytet: Średni

Powiązania: EUC14

Stan systemu przed testem: W edytorze istnieje słowo, które ma być zamienione.

Dane wejściowe lub warunki testu:

1. Użytkownik definiuje fragment tekstu oraz jego zamiennik,
2. Użytkownik wydaje polecenie zamiany.

Wynik oczekiwany: System znajduje zdefiniowany fragment i zamienia go na podany zamiennik.

4.2 Wyniki testów

4.2.1 Wprowadzenie

Każdy wynik testu posiada identyfikator, który jednoznacznie przypisuje go do określonego testu. W przypadku testów, które wymagają wykonania innych przypadków testowych, w powiązaniach znajdą się odpowiednie odwołania ujednoznaczniające.

4.2.2 Wyniki testów ogólnych

GTC1: Próba otwarcia nieistniejącego pliku

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Test zakończony wyświetleniem odpowiedniego komunikatu.

GTC2: Próba nadpisania pliku

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Test zakończony wyświetleniem odpowiedniego okna dialogowego, wybór opcji zgodnie z oczekiwaniami.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC1

Wynik uzyskany: Wszystkie proste figury są poprawnie przesyłane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC2

Wynik uzyskany: Ścieżka jest poprawnie przesyłana do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC3

Wynik uzyskany: Łamana jest przekazywana niepoprawnie - ostatni odcinek łamanej nie jest przekazywany do użytkownika zdalnego.

Decyzja: Konieczna poprawa modułu mapującego obiekty na struktury sieciowe lub modułu do rysowania ścieżek.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC3

Wynik uzyskany: Łamana jest przekazywana poprawnie do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC5, BTC6

Wynik uzyskany: Przesunięcia figur są przesyłane poprawnie.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 30.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC7

Wynik uzyskany: Zmiany wybranych atrybutów zostały poprawnie przepropagowane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 30.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC10

Wynik uzyskany: Rysunek został poprawnie przesłany do użytkownika zdalnego i umieszczony w obszarze rysowania.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 30.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC13

Wynik uzyskany: Figura została poprawnie przesłana do użytkownika zdalnego i umieszczona w obszarze rysowania.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 30.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC16

Wynik uzyskany: Tekst został poprawnie przesłany do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 30.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC17

Wynik uzyskany: Zmiana tekstu została poprawnie przesłana do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 30.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC18

Wynik uzyskany: Modyfikacja atrybutów tekstu została poprawnie przepropagowana do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC19

Wynik uzyskany: Skalowanie dowolnej figury jest poprawnie przekazywane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC22

Wynik uzyskany: Grupowanie figur jest poprawnie przesyłane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: BTC24

Wynik uzyskany: Rozgrupowanie istniejącej grupy zostało poprawnie przekazane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: GTC4, AmeBoard

Wynik uzyskany: Cofanie operacji zostało poprawnie przesłane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: GTC4, AmeEdit

Wynik uzyskany: Cofanie operacji zostało poprawnie przesłane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: GTC5, AmeBoard

Wynik uzyskany: Operacja powtórzenia cofniętej operacji została poprawnie przesłana do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: GTC5, AmeEdit

Wynik uzyskany: Operacja powtórzenia cofniętej operacji została poprawnie przesłana do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: ETC1

Wynik uzyskany: Kod został poprawnie przesłany do użytkownika zdalnego i pokolorowany według jego ustawień.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: ETC4

Wynik uzyskany: Operacje wycinania i wklejania tekstu są poprawnie przesyłane do użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: ETC5

Wynik uzyskany: System zdalny jest przywracany do stanu zgodnego z plikiem sesji.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: ETC11

Wynik uzyskany: Znalezione fragmenty kodu są poprawnie zaznaczane po stronie użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: ETC12

Wynik uzyskany: Znalezione fragmenty kodu są poprawnie zaznaczane po stronie użytkownika zdalnego.

GTC3: Przesyłanie zdarzeń do użytkownika zdalnego

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Konrad Siek

Powiązania: ETC13

Wynik uzyskany: Zamiany kodu są poprawnie przesyłane do użytkownika zdalnego.

GTC4: Cofanie operacji – *undo*

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: AmeBoard

Wynik uzyskany: Operacja uległa cofnięciu, brak odświeżenia obrazu.

Decyzja: Dodanie odpowiednich instrukcji wymuszających odświeżenie obrazu.

GTC4: Cofanie operacji – *undo*

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: AmeEdit

Wynik uzyskany: Operacja uległa cofnięciu w sposób poprawny.

GTC5: Powtarzanie cofniętej operacji – *redo*

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: AmeBoard

Wynik uzyskany: Cofnięta operacja uległa powtórzeniu, brak odświeżenia obrazu.

Decyzja: Dodanie odpowiednich instrukcji wymuszających odświeżenie obrazu.

GTC5: Powtarzanie cofniętej operacji – *redo*

Atrybuty:

Wykonano: 25.01.2008

Wykonawca: Paweł Martenka

Powiązania: AmeEdit

Wynik uzyskany: Cofnięta operacja uległa powtórzeniu.

4.2.3 Wyniki testów AmeBoard

BTC1: Kreślenie prostych figur

Atrybuty:

Wykonano: 30.11.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Dla wszystkich dostępnych figur wynik rysowania jest poprawny.

BTC2: Kreślenie dowolnych ścieżek (kredka)

Atrybuty:

Wykonano: 23.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Ścieżka narysowana zgodnie z oczekiwaniami.

BTC3: Kreślenie łamanej

Atrybuty:

Wykonano: 23.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Łamana narysowana zgodnie z oczekiwaniami.

BTC4: Ustawienie atrybutów kreślonych figur

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Atrybuty zostały ustawione zgodnie z oczekiwaniami.

BTC5: Przesuwanie narysowanych figur – ze wstępnym zaznaczeniem

Atrybuty:

Wykonano: 18.11.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Przesunięta linia uległa deformacji.

Decyzja: Algorytm transformacji wymaga poprawy.

BTC5: Przesuwanie narysowanych figur – ze wstępnym zaznaczeniem

Atrybuty:

Wykonano: 16.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Przesunięta linia uległa deformacji.

Decyzja: Algorytm transformacji wymaga kolejnej poprawy.

BTC5: Przesuwanie narysowanych figur – ze wstępnym zaznaczeniem

Atrybuty:

Wykonano: 23.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Przesunięcie nastąpiło zgodnie z oczekiwaniami.

BTC6: Przesuwanie narysowanych figur – bez wstępnego zaznaczenia

Atrybuty:

Wykonano: 23.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Przesunięcie nastąpiło zgodnie z oczekiwaniami.

BTC7: Zmiana atrybutów wybranej figury

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Atrybuty zostały zmienione zgodnie z oczekiwaniami.

BTC8: Próba zmiany atrybutów grupy

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: System zgodnie z oczekiwaniami zablokował próbę.

BTC9: Próba zmiany atrybutów kilku zaznaczonych figur

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: System zgodnie z oczekiwaniami zablokował próbę.

BTC10: Odczytanie prawidłowego pliku z rysunkiem

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Plik wczytał się prawidłowo, obraz pojawił się w obszarze rysowania.

BTC11: Próba odczytania nieprawidłowego pliku z rysunkiem

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: System wyświetlił komunikat zgodny z oczekiwanym.

BTC12: Eksport figury do pliku

Atrybuty:

Wykonano: 04.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Figura została wyeksportowana, ale jest nieprawidłowo pozycjonowana w pliku.

Decyzja: Konieczne jest przetransformowanie figury przed zapisem.

BTC12: Eksport figury do pliku

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Figura została prawidłowo wyeksportowana.

BTC13: Import figury z prawidłowego pliku

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Figura została poprawnie zaimportowana, pojawiła się w obszarze rysowania.

BTC14: Próba zaimportowania nieprawidłowego pliku z rysunkiem

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: System wyświetlił komunikat zgodny z oczekiwanym.

BTC15: Weryfikacja pliku z wyeksportowaną figurą

Atrybuty:

Wykonano: 04.01.08

Wykonawca: Paweł Martenka

Powiązania: BTC13, BTC14

Wynik uzyskany: Wyeksportowana figura poprawnie się zaimportowała.

BTC16: Wpisywanie tekstu w obszarze rysowania

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Tekst pojawił się w obszarze rysowania.

BTC17: Modyfikacja tekstu wpisanego

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Tekst został zmodyfikowany zgodnie z oczekiwaniami.

BTC18: Modyfikacja atrybutów tekstu wpisanego

Atrybuty:

Wykonano: 30.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Atrybuty tekstu zostały zmodyfikowane zgodnie z oczekiwaniami.

BTC19: Skalowanie figur

Atrybuty:

Wykonano: 17.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Próba przeskalowania grupy figur zakończona niepowodzeniem, zupełnie niezrozumiały wynik skalowania.

Decyzja: Algorytm transformacji wymaga poprawy.

BTC19: Skalowanie figur

Atrybuty:

Wykonano: 23.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Figura przeskalowana poprawnie, także grupa figur.

BTC20: Zapis rysunku do pliku

Atrybuty:

Wykonano: 04.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Obszar rysowania poprawnie zapisany do pliku.

BTC21: Eksport rysunku do pliku bitmapy

Atrybuty:

Wykonano: 17.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Obszar rysowania wyeksportowany, ale obraz ma niepoprawne wymiary - niektóre elementy zostały obcięte.

Decyzja: Moduł eksportu wymaga poprawy algorytmu wyliczania wymiarów obrazu.

BTC21: Eksport rysunku do pliku bitmapy

Atrybuty:

Wykonano: 05.01.08

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Obszar rysowania wyeksportowany poprawnie.

BTC22: Grupowanie figur

Atrybuty:

Wykonano: 05.01.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Figury zostały zgrupowane poprawnie.

BTC24: Rozgrupowanie grupy

Atrybuty:

Wykonano: 17.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Wcześniej utworzona grupa została rozgrupowana prawidłowo.

BTC25: Próba rozgrupowania figury nie będącej grupą

Atrybuty:

Wykonano: 17.12.07

Wykonawca: Paweł Martenka

Powiązania:

Wynik uzyskany: Brak reakcji systemu na próbę rozgrupowania.

4.2.4 Wyniki testów AmeEdit

ETC1: Importowanie kodu z pliku

Atrybuty:

Wykonano: 22.11.07

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Złe kolorowanie literałów dla typu `long` , np. `0L` .

Decyzja: Potrzeba rozszerzenia odpowiedniego wyrażenia regularnego.

ETC1: Importowanie kodu z pliku

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Plik jest zaimportowany i pokolorowany poprawnie.

ETC2: Kolorowanie składni

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Składnia koloruje się właściwie, odpowiednio do preferencji i języka

ETC3: Aktywny kursor

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Kursor zmienia położenie odpowiednio.

ETC4: Wycinanie i wklejanie tekstu

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Mechanizm wycinania i wklejania działa poprawnie.

ETC5: Przywracanie wcześniejszej sesji

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: System został poprawnie przywrócony.

ETC6: Zapisywanie kodu do pliku

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Kod został poprawnie zapisany do pliku.

ETC7: Importowanie poprawnej definicji składni z pliku

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Nowa definicja została poprawnie zaimportowana.

ETC8: Próba importowania niepoprawnej definicji składni z pliku

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Niepoprawna definicja została odrzucona.

ETC9: Dostosowanie formatu kolorowania składni

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Składnia została pokolorowana zgodnie z nowymi zasadami.

ETC10: Drukowanie pliku z kodem

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: Drukowanie działa na systemach Microsoft Windows, na systemie Debian nie działa poprawnie.

Decyzja: Jest to problem systemu operacyjnego Linux i nie można go rozwiązać na tym etapie.

ETC11: Szukanie fragmentu tekstu – za pomocą słowa

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: System znajduje podane słowa i je zaznacza.

ETC12: Szukanie fragmentu tekstu – za pomocą wyrażenia regularnego

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: System znajduje pasujące słowa i je zaznacza.

ETC13: Zamiana fragmentu kodu na inny

Atrybuty:

Wykonano: 25.01.08

Wykonawca: Konrad Siek

Powiązania:

Wynik uzyskany: System znajduje i zamienia fragmenty kodu zgodnie z definicją.

Rozdział 5

Podsumowanie

5.1 Zrealizowane zadania

W ramach pracy inżynierskiej udało się stworzyć narzędzia usprawniające pracę grupową nad wspólnym tworzeniem schematów i grafiki oraz wspólną edycją fragmentów kodu.

Zrealizowany projekt jest dodatkiem do jednego z dostępnych komunikatorów internetowych umożliwiającym współtworzenie schematów, rysunków, diagramów oraz kodów programów w czasie rzeczywistym.

Narzędzie AmeEdit wyposażone zostało w funkcjonalność, taką jak kolorowanie składni czy też mechanizm wyszukiwania i zamiany fragmentów tekstu, która znacznie ułatwi współpracę grupy osób nad projektem programistycznym.

AmeBoard posiada funkcje usprawniające pracę na obiektach graficznych (prostokąty, elipsy, itp.), tekstowych i grupowych. Zapewnia współtworzenie wszelkiego rodzaju prostych schematów, diagramów oraz rysunków zawierających objaśnienia z możliwością przesyłania ich do zainteresowanych osób.

5.2 Napotkane problemy

W trakcie realizacji projektu napotkano na problemy, których z różnych przyczyn nie udało się rozwiązać. Niemniej jednak większość założeń i wymagań o wysokim priorytecie zostało zrealizowanych.

Wtyczka AmeEdit posiada tylko analizę leksykalną kodu, natomiast nie udało się zrealizować analizy leksykalno-składniowej, ponieważ wymagałoby to wystosowania gramatyki dla każdego języka, a tym samym stworzenia osobnego analizatora dla każdego języka. Wiązałoby się to z uzyskaniem, bądź użyciem gotowego narzędzia typu YACC. Z uwagi na to, że nie udało się uzyskać takiego analizatora, niemożliwe było wykonanie zwijania funkcji do nagłówka.

We wtyczce AmeBoard nie udało się uzyskać wyświetlenia siatki, jednakże ta funkcja miała niski priorytet. Ponadto stworzony parser oparty o DOM, jest mało kompatybilny z SVG, jego działanie jest bardzo powolne. Ze względu na skomplikowanie problemu nie udało się stworzyć parsera zdarzeniowego, który zapewniłby lepszą wydajność.

Rysowanie dowolnej krzywej za pomocą kredki jest niedokładne, ponieważ nie zastosowano żadnego algorytmu aproksymacji (np. za pomocą krzywych parametrycznych) z uwagi na dużą złożoność problemu.

Ponadto nie zostały zrealizowane funkcje odpowiedzialne za kopiowanie i wklejanie obiektów, brakuje krzywych parametrycznych oraz możliwości modyfikacji każdej figury jako krzywej. Ponadto brakuje innych operacji na figurach takich jak rotacja czy lustro, a operacje zaimplementowane

mają często uproszczony charakter. Wymienione funkcje nie zostały zrealizowane z uwagi na ich niski priorytet, chociaż pewne przygotowania zostały poczynione – istnieją szkielety klas, które w przyszłości można rozbudować.

AmePlug nie umożliwia prowadzenia rozmowy przy jednoczesnej edycji zawartości przez każdego użytkownika konwersacji jednocześnie. Wiąże się to z trudnościami w zarządzaniu treścią przez wtyczki AmeBoard i AmeEdit. Jednoczesna edycja zawartości okna przez każdego użytkownika, mogłaby spowodować błędne przesyłanie zawartości tworzonego schematu bądź kodu.

Każdy komponent w AmeGUI został rozszerzony i wykreślony za pomocą kodu Java, jednakże nie udało się zaprojektować unikalnego paska tytułu okna. Problemem było to, że pasek tytułu jest elementem kontenera najwyższego poziomu - komponentu ciężkiego `JFrame` . Komponenty takie są realizowane poprzez użycie graficznych bibliotek GUI systemu operacyjnego. Oczywiście można utworzyć nieudekorowane okno i stworzyć pasek tytułu, jednak problemem będzie obsługa zdarzeń myszy reagujących na zmianę rozmiaru okna. Przejścia takie będą mało płynne, co w rezultacie daje niezbyt estetyczny wygląd [Rog03].

5.3 Zdobyte doświadczenia

Realizacja projektu pozwoliła na sprawdzenie się w pracy zespołowej. Odpowiedni przydział obowiązków, poszczególnym członkom zespołu pozwolił na efektywne wykonywanie zadań. Współpraca, wzajemna motywacja, dobra komunikacja i duże zaangażowanie każdego z członków zespołu zaowocowało osiągnięciem celu, który został na początku jasno sprecyzowany.

W efekcie pracy nad projektem członkowie grupy mieli szansę lepiej poznać środowisko programistyczne Eclipse, które oferuje bogatą funkcjonalność i, co najważniejsze, jest narzędziem elastycznym dzięki możliwości rozszerzenia o dodatkowe wtyczki. Podczas realizacji projektu skorzystano między innymi z takich wtyczek jak Metrics, TeXlipse czy Subclipse.

Wtyczka Metrics umożliwiła na bieżąco szacować rozmiar i złożoność kodu, a TeXlipse to wtyczka wspierająca LaTeXw Eclipse, która umożliwiła napisanie niniejszej pracy.

Dzięki wtyczce subclipse zapoznano się z systemem kontroli wersji oprogramowania (SVN), który jest bardzo przydatny w pracy zespołowej. Usprawniło to współpracę, na bieżąco można było śledzić zmiany w kodzie źródłowym, oraz łączyć czy wprowadzać modyfikacje do projektu.

5.4 Propozycje rozwoju

Projekt Amebae, z uwagi na konstrukcję wtyczkową, łatwo rozbudować o kolejne moduły np. edytor równań. Wykorzystanie jednolitego GUI dla wszystkich modułów projektu stanowi przesłankę do stworzenia całego pakietu narzędzi do pracy zespołowej.

Proponowanym kierunkiem rozwoju dla edytora AmeEdit jest poszerzenie jego możliwości o pracę nad całymi projektami programistycznymi, a więc udostępnienie funkcji edycji wielu plików jednocześnie. W ten sposób z edytora tekstowego AmeEdit mógłby stać się prostym wieloosobowym środowiskiem programistycznym. AmeBoard można rozwinąć w bardziej rozbudowany, rozproszony edytor diagramów, o możliwościach na poziomie programu Dia oraz ArgoUML.

Jednym z bardziej zaawanowanych rozwiązań, o jakie możnaby wzbogacić istniejące rozwiązanie, to próba rozszerzenia wachlarzu obsługiwanych protokołów sieciowych, czego konsekwencją potencjalnie mogłoby być umożliwienie współpracy Amebae'y z innym oprogramowaniem działającym na podobnych zasadach, które ma szansę rozwinąć się w przyszłości.

Dodatek A

Słownik Pojęć

analizator leksykalny – program wykonujący konwersję tekstu z postaci ciągu znaków na listę tokenów;

analizator składniowy – program wnioskujący syntaktyczną strukturę na podstawie listy tokenów;

atrybut lub właściwość obrazu – cecha nadawana figurze lub posiadana przez figurę, przykładowo atrybutem jest kolor linii lub wypełnienia;

atrybut znaczący figury – cecha mająca wpływ na figurę, przykładowo dla linii znaczący będzie kolor linii, ale nieistotny będzie kolor wypełnienia;

DOM – (ang. *Document Object Model*) neutralny pod względem języka i platformy interfejs pozwalający programom i skryptom na dynamiczny dostęp do zawartości, struktur i stylów dokumentu;

eksport figury – zapisanie pojedynczej figury do pliku grafiki wektorowej;

eksport rysunku – zapisanie całej zawartości obszaru rysowania do pliku w postaci bitmapy;

figura – dowolna figura prosta (prostokąt, elipsa itp.), ścieżka,łamana, tekst, grupa figur;

figura prosta – prostokąt, prostokąt z zaokrąglonymi brzegami, elipsa,linia prosta; figura którą można opisać za pomocą dwóch punktów;

grupa – logicznie powiązana lista figur, grupa jest figurą;

interfejs użytkownika lub GUI – zestaw środków graficznych używanych przez komputer w komunikacji z użytkownikiem;

import figury – odczyt pojedynczej figury z pliku grafiki wektorowej;

kliknięcie – wciśnięcie i wyciśnięcie klawisza myszy w tym samym punkcie obrazu;

manipulator – narzędzie AmeBoard służące do przesuwania, zaznaczania i skalowania figur;

obszar rysowania – obszar w oknie AmeBoard służący do rysowania i manipulacji figurami nań narysowanymi;

plik – dowolny plik dyskowy, dla AmeBoard jest to plik grafiki wektorowej SVG, dla AmeEdit plik tekstowy;

punkt manipulacji figurą – niewielki kwadrat znajdujący się w narożnikach obrysu figury lub też na środkach jego boków, wykorzystywany jako wizualizacja "uchwytów" do manipulacji figurami; wciśnięcie klawisza myszy,gdy kursor jest nad takim punktem i przeciągnięcie tego kursora w inne miejsce powoduje określoną zmianę obrazu;

sekwencja ucieczki (ang. *escape sequence*) – ciąg znaków traktowany w sposób specjalny;

singleton – wzorzec projektowy, ograniczający możliwość tworzenia obiektów klasy do jednej instancji;

SVG – (ang. *Scalable Vector Graphics*) format grafiki wektorowej oparty o język XML; dokument SVG jest dokumentem XML;

tekst – napis posiadający cechy i zachowanie figury;

token – fragment tekstu lub ciąg znaków z przydzieloną kategorią;

użytkownik – w tym wypadku jest to osoba testująca;

użytkownik lokalny – użytkownik aplikacji obsługujący system lokalny;

użytkownik zdalny – użytkownik aplikacji obsługujący system zdalny;

wtyczka lub plugin – (ang. *plugin*) program działający rozszerzający program nadrzędny, dostarczając nowych funkcji na żądanie;

XML – (ang. *Extensible Markup Language*) rozszerzalny (definiowany przez autora) język znacznikowy służący do ustrukturalizowanego przechowywania danych oraz informacji je opisujących;

zaznaczenie lub wybranie figur – efekt działania manipulatora –kliknięcie pojedynczej figury lub otoczenie kilku figur obrysem prostokątnym;

Literatura

[AS05] James R. Trott Alan Shalloway. *Projektowanie zorientowane obiektowo. Wzorce projektowe.* Helion, 2005.

[Boc07] Victor Boctor. Mantis. [on-line] `http://http://www.mantisbt.org/`, 2007.

[DB86] Garrett Rooney Daniel Berlin. *Practical Subversion, Second Edition.* Apress, 1986.

[Fla99] David Flanagan. *Java Foundation Classes: A Desktop Quick Reference.* O'Reilly, Cambridge, MA, 1999.

[Fou05] Jabber Foundation. What is jabber®? [on-line] `http://www.jabber.org/about/overview.shtml`, 2005.

[Fow03] Amy Fowler. A swing architecture overview. [on-line] `http://java.sun.com/products/jfc/tsc/articles/architecture/`, 2003.

[JG96] Guy Steele James Gosling, Bill Joy. Java language specification - second edition. [on-line] `http://java.sun.com/docs/books/jls/second_edition/html/syntax.doc.html`, 1996.

[JP02] H. Sharp J. Preece, Y. Rogers. *Interaction Design: Beyond Human-Computer Interaction.* John Wiley and Sons, New York, 2002.

[Mar99] John Markoff. An internet pioneer ponders the next revolution. [on-line] `http://partners.nytimes.com/library/tech/99/12/biztech/articles/122099outlook-bobb.html?Partner=Snap`, 1999.

[Rog03] Paweł Rogalinski. Programowanie graficznych interfejsów użytkownika. [on-line] `http://pawel.rogalinski.staff.iiar.pwr.wroc.pl/dydaktyka/INE2018L_JP3_Java/Programowanie%20GUI.pdf`, 2003.

[SS05] Scott Fairbrother Dan Kehn John Kellerman Pat McCarthy Sherry Shavor, Jim D'Anjou. *Java Developer's Guide to Eclipse.* Helion, 2005.

[Tom99] Ray Tomlinson. The first network email. [on-line] `http://openmap.bbn.com/~tomlinso/ray/firstemailframe.html`, 1999.