POLITECHNIKA KOSZALIŃSKA
WYDZIAŁ ELEKTRONIKI I INFORMATYKI
KATEDRA INŻYNIERII KOMPUTEROWEJ

PRACA DYPLOMOWA INŻYNIERSKA

Dedykowany klient VoIP do wspomagania kontroli i organizowania pracy przedstawicieli handlowych (sprzedawców telefonicznych).

Daniel Dettlaff

Rodzaj studiów:
Dzienne, inżynierskie
Kierunek:
Informatyka
Specjalność:
Programowanie Komputerów i Sieci Informatyczne

Promotor:
dr inż. Walery SUSŁOW

Oświadczenie

Oświadczam, że przedstawiona praca inżynierska została przygotowana i wykonana samodzielnie.

Podpis

Streszczenie pracy.

Celem mojej pracy było stworzenie komunikatora głosowego, którego można by użyć do wspomagania pracy sprzedawców telefonicznych (tele-sprzedawców). W mojej pracy inżynierskiej, pokazuję w jaki sposób dochodziłem, od stworzenia projektu oprogramowania użytkowego, a następnie do jego realizacji w języku programowania. Przedstawiam w niej krok po kroku procesy, które doprowadziły do stworzenia komunikatora VoIP.

Spis treści

Podziękowania

1. Doktorowi Waleremu Susłowowi za cierpliwość i wiele dobrych rad.
2. Annie Śliwińskiej za ciągłe wsparcie.
3. Rodzicom za duchowe wsparcie w tworzeniu tego dokumentu.
4. Michałowi Heweltowi za pomoc w poprawianiu błędów gramatyki angielskiej.

1. Wstęp i uzasadnienie wyboru tematu

1.1 Wstęp

W wyborze mojej pracy inżynierskiej kierowałem się ambicją i chęcią zdobycia kolejnej praktyki. Chciałem napisać coś, co nauczyłoby mnie dobrze języka programowania, w którym kiedyś napisałem szkielet pewnej gry. Gdy kilka lat temu wpadłem na jej pomysł, nie miałem dość umiejętności by zdołać ją zaimplementować. Dziś dzięki doświadczeniu w pisaniu tej pracy będę w stanie to zrobić.

Aplikacja, którą napisałem, nazwałem „dSipCom” (od dmilith's SIP Communicator). Jest to multiplatformowy, wielowątkowy komunikator w 100% zgodny ze standardem SIP (RFC-3261). Charakteryzuje go czytelność interfejsu, prosta obsługa i niezależność od systemu operacyjnego. Dodatkowym atutem programu jest zaimplementowane w nim automatyczne dostosowywanie się do kodeka połączeń oraz automatycznego wyboru jakości rozmowy w zależności od przepustowości łącza sieciowego. Dzięki podwójnej licencji GPL/ LGPL, aplikacja może znaleźć zastosowanie w domowym użyciu, jak i stać się narzędziem obsługiwanym przez telesprzedawców w firmach telekomunikacyjnych.

1.2 Uzasadnienie wybóru tematu

Temat, który obrałem, nie był moim pomysłem. Miał być to projekt, który uzupełni funkcjonalność projektu Szymona Jeża o możliwość użycia dedykowanego klienta VoIP dla sprzedawców aktywnych (telesprzedawców). Wybrałem ten temat, ponieważ nie jestem zadowolony z aplikacji SIP dostępnych na rynku. Moim celem było napisanie aplikacji, która mogłaby przewyższyć funkcjonalnością komercyjnych konkurentów w tym najpopularniejszego – Skype przy zarówno mniejszym koszcie produkcji jak i programowym (obciążenie pamięci i CPU). Wyniki porównania dSipCom'a ze Skype znajdują się w rozdziale 2.

2. Analiza domeny i konkurencji

W tym rozdziale zawarłem krótką notatkę o branży VoIP, przedstawiam także przedstawienie wykorzystanej przeze mnie technologii. Następnie porównuję moją aplikację z innymi projektami Open Source.

2.1 Informacje o branży i krótka historia

Komunikacja w dzisiejszych czasach to priorytet zarówno w biznesie jak i codziennym życiu. Rozmowy głosowe to najszybszy i najłatwiejszy sposób przekazywania informacji. Od czasu popularyzacji Internetu na świecie, każdy może za darmo zadzwonić, korzystając z aplikacji przesyłających rozmowy przez sieć. Po latach stworzono standardy protokołów. Jednym z nich jest SIP (Session Initiation Protocol).

Session Initiation Protocol to protokół zaproponowany przez IETF (http://ietf.org/), opisany jako standard RFC-3261 (http://www.ietf.org/rfc/rfc3261.txt). Jest protokołem mającym dostarczyć zestaw funkcji obsługi połączeń i innych cech obecnych w publicznej sieci telefonicznej (w tym: wybieranie numeru, dzwonek w telefonie, sygnały zajętości itp.), przy czym ich implementacja i terminologia jest odmienna. SIP jest protokołem peer-po-peer. Oznacza to, że do połączenia nie jest potrzebny udział pośredniego serwera. Protokół SIP współgra z innymi protokołami. Jest tylko nośnikiem dla SDP (Session Description Protocol), który z kolei opisuje transportowane multimedia w sieci (np. kodek audio/ wideo, port, itp.). SIP jest protokołem tekstowym, podobnym do http. Cechą wspólną są np. podobne kody błędów w tym słynny błąd 404. Przewagą SIP nad innymi protokołami jest fakt, że można go zastosować zarówno do transportu audio jak i wideo.

2.3 Przedstawienie technologii

Sieć SIP składa się z wielu elementów. Podstawowym elementem są terminale końcowe (nodes), które mogą być zarówno aplikacjami komputerowymi (softphones) jak i

korzystać z tradycyjnej sieci telefonicznej (hardphones), poprzez bramki dostawców takich usług. Obecnie wielu producentów na całym świecie oferuje możliwość odpłatnego założenia konta na serwerze (proxy), który udostępnia użytkownikowi możliwość dzwonienia z komputera na telefony stacjonarne i na odwrót. Terminalem może być urządzenie z wyglądu przypominające zwykły telefon stacjonarny, podłączane do gniazdek sieciowych i używające protokołu SIP do nawiązywania połączeń.

Kolejnym elementem sieci SIP jest rejestrator SIP (registrar), który jest specjalną bazą danych, która komunikuje się z węzłami SIP w celu zbierania i archiwizacji informacji na temat użytkowników SIP. Dane te wykorzystuje się w momencie nawiązywania połączenia do odnajdywania w sieci węzłów docelowych. Po zarejestrowaniu, każdy węzeł w momencie nawiązywania połączenia przekazuje rejestratorowi informacje o adresie IP i porcie węzła na którym ma zostać nawiązane połączenie. Rejestrator pełni tutaj rolę routera na poziomie protokołu SIP, umożliwiając węzłowi inicjującemu połączenie odnalezienie węzła docelowego.

Połączenia SIP, które są podłączane do różnych rejestratorów SIP, muszą być przekazywane przez specjalne serwery pośredniczące (SIP proxy). Serwer proxy ma za zadanie przekazanie do odpowiedniej domeny żądania połączenia. Proxy analizuje polecenie INVITE protokołu SIP (wysyłane podczas żądania nawiązania połączenia) i na podstawie zawartego w nim adresu kieruje je do innego węzła w sieci. Ten sposób wykonywania połączeń pozwala ominąć problemy związane z translacją adresów (NAT) w sieciach z nieroutowalnymi (lokalnymi) adresami IP. Innym zastosowaniem serwerów SIP proxy może być kontrola planów taryfowych, rejestracja rozmów i prowadzenie rejestrów billingowych.

2.4 Porównanie na tle konkurencji.

W porównaniu mojej aplikacji z konkurencją posłużyłem się następującymi narzędziami:

- htop – Monitor dający wgląd w listę procesów systemu wraz ze

szczegółami (http://htop.sourceforge.net/).

- monitor systemu KDE4.1 – Monitor mający podobną funkcjonalność co htop (http://kde.org/).

Porównałem swoją aplikację z aplikacją Skype (http://skype.com/) – jedną z najpopularniejszych tego typu aplikacji na swiecie. Sprzęt na którym przeprowadziłem testy to:
Athlon64 X2 Turion, 1.8Ghz, 4GiB RAM, System: Linux Debian Lenny amd64 (http://debian.org/).

Wyniki testu przedstawiają się następująco:

Zużycie pamięci podczas bezczynności (idle)	
Skype	**25.1 MiB RSS** (Resident Set Size)
dSipCom	**6.2 MiB RSS**

Zużycie pamięci podczas połączenia	
Skype	**27.3 MiB RSS** (Resident Set Size)
dSipCom	**6.6 MiB RSS**

Zużycie procesora podczas bezczynności (idle)	
Skype	**1% – 3%**
dSipCom	**1% – 3%**

Zużycie procesora podczas połączenia	
Skype	**1% – 15%**
dSipCom	**3% – 10%**

Widać zatem niewielką różnicę zużycia procesora pomiędzy obiema aplikacjami niezależnie od trybu pracy, przy znacznej różnicy zużycia pamięci RAM. DsipCom zużywa kilkakrotnie mniej pamięci niż jego konkurent, choć jest to w zasadzie jedyna jego przewaga. Skype posiada bowiem lepszej jakości, komercyjne kodeki audio, przewyższające darmowe

odpowiedniki domyślnie dołączane do projektu DSipCom.

3. Modelowanie i projekt

W tym punkcie przedstawię diagramy stworzone przy projektowaniu aplikacji, w tym diagram przypadków użycia, klas i architektury. Przedstawiam tu także dokładne opisy wszystkich diagramów.

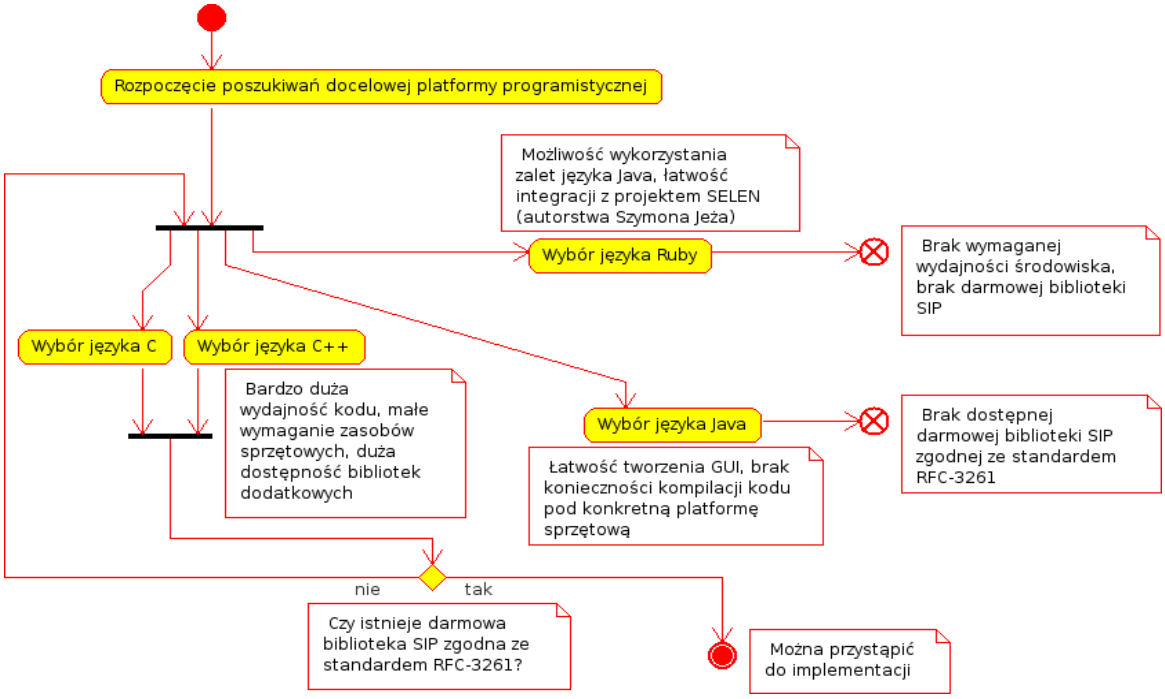


Diagram 1. Schemat poszukiwań platformy programistycznej.

Projekt rozpocząłem od poszukiwania platformy programistycznej. Celem poszukiwań było określenie jaki język programowania będzie najlepszy do zastosowania go w moim projekcie. Wybór padł na język C i C++. Diagram nr 1, przedstawia mój proces poszukiwań:

Tabela 1. Opis scenariuszy na podstawie przypadków użycia.

Przypadek użycia.	*Opis akcji.*
„Zmiana ustawień	*Użytkownik* wybiera zakładkę preferencji, wczytuje bieżącą

Przypadek użycia.	*Opis akcji.*
konfiguracji”	konfiguracje z pliku, dokonuje zmian konfiguracji komunikatora, zapisuje zmiany. Zmiany nie zapisane przez użytkownika działają tylko podczas bieżącej sesji i zostają utracone podczas wyjścia z programu.
„Dodanie nowego kontaktu do listy kontaktów”	*Użytkownik* dodaje kontakt do listy”, uzupełnia pola wszystkie pola wymagane do identyfikacji nowego kontaktu. Po zatwierdzeniu zmiany, na liście kontaktów programu znajdzie się wprowadzona pozycja. Zmiany nie zatwierdzone przez użytkownika zostają utracone po anulowaniu akcji dodania nowego kontaktu.
„Usunięcie kontaktu z listy kontaktów”	*Użytkownik* aktywuje opcję usunięcia kontaktu z listy. Z listy użytkowników natychmiast usuwana jest zaznaczona pozycja. Aby zatwierdzić zmiany, *Użytkownik* musi zatwierdzić zmiany zapisując je do pliku. W przeciwnym wypadku po wyjściu z programu zmiany nie zostaną zapamiętane.
„Przeglądanie logów połączeń”	*Użytkownik* wybiera zakładkę Raportów. Aby przejrzeć log z danego dnia, wybiera w kalendarzu interesujący go dzień, po czym może przeczytać wpis w przeglądarce logów.
„Wpis raportu z połączenia”	Gdy *Użytkownik* zakończy wykonywanie połączenia, dostaje automatyczny monit o wprowadzenie raportu z wykonanego połączenia. (Domyślnie program automatycznie generuje taki raport i ingerencja użytkownika jest niekonieczna.)
„Wybieranie numeru lub adresu kontaktu docelowego” (dla połączeń bez pośrednika – proxy)	*Użytkownik* wybiera z listy użytkowników interesujący go kontakt po czym próbuje nawiązać połączenie. Program sygnalizuje próbę nawiązywania połączenia poprzez sygnał dźwiękowy. W przypadku wystąpienia błędu, *Użytkownik* jest informowany wyskakującą informacją o błędzie.
„Wybieranie numeru lub	*Użytkownik* postępuje analogicznie jak dla połączeń bez

Przypadek użycia.	*Opis akcji.*
adresu kontaktu docelowego" (dla połączeń z pośrednikiem – proxy)	pośrednika, z tą różnicą, że przed wykonaniem połączenia należy w preferencjach aplikacji ustawić dane serwera proxy. Następnie *Użytkownik* nawiązuje połączenie z serwerem proxy wybierając opcję z menu. Aplikacja poinformuje o wyniku działania żądania połączenia z serwerem proxy odpowiednim komunikatem.
„Odebranie przychodzącego połączenia"	*Użytkownik* w zależności od preferencji ustawień sieciowych (w tym ustawień serwera Proxy), może odbierać połączenia przychodzące, które sygnalizowane są dźwiękowym sygnałem oczekiwania (identycznym jak w zwykłych telefonach analogowych).

3.1 Schemat przypadków użycia

Kolejnym diagramem, który przestawię jest diagram przypadków użycia (diagram 2). Przedstawia on listę możliwych akcji, które może wykonać użytkownik pracując w programie dSipCom. Tabela nr 1, przedstawia opis scenariuszy na podstawie tych przypadków użycia.

3.2 Diagram aktywności

Kolejnym ważnym krokiem, jest przedstawienie procesów dynamicznych zachodzących w mojej aplikacji. Najważniejsze dynamiczne procesy wraz z opisem, przedstawia diagram 3.

3.3 Architektura

Kolejnym diagramem mającym bezpośredni związek z moją aplikacją jest diagram architektury, przedstawiony na diagramie 4, w którym pokazuję kompozycję elementów (bibliotek), wymaganych przez moją aplikację i ich wzajemne zależności.

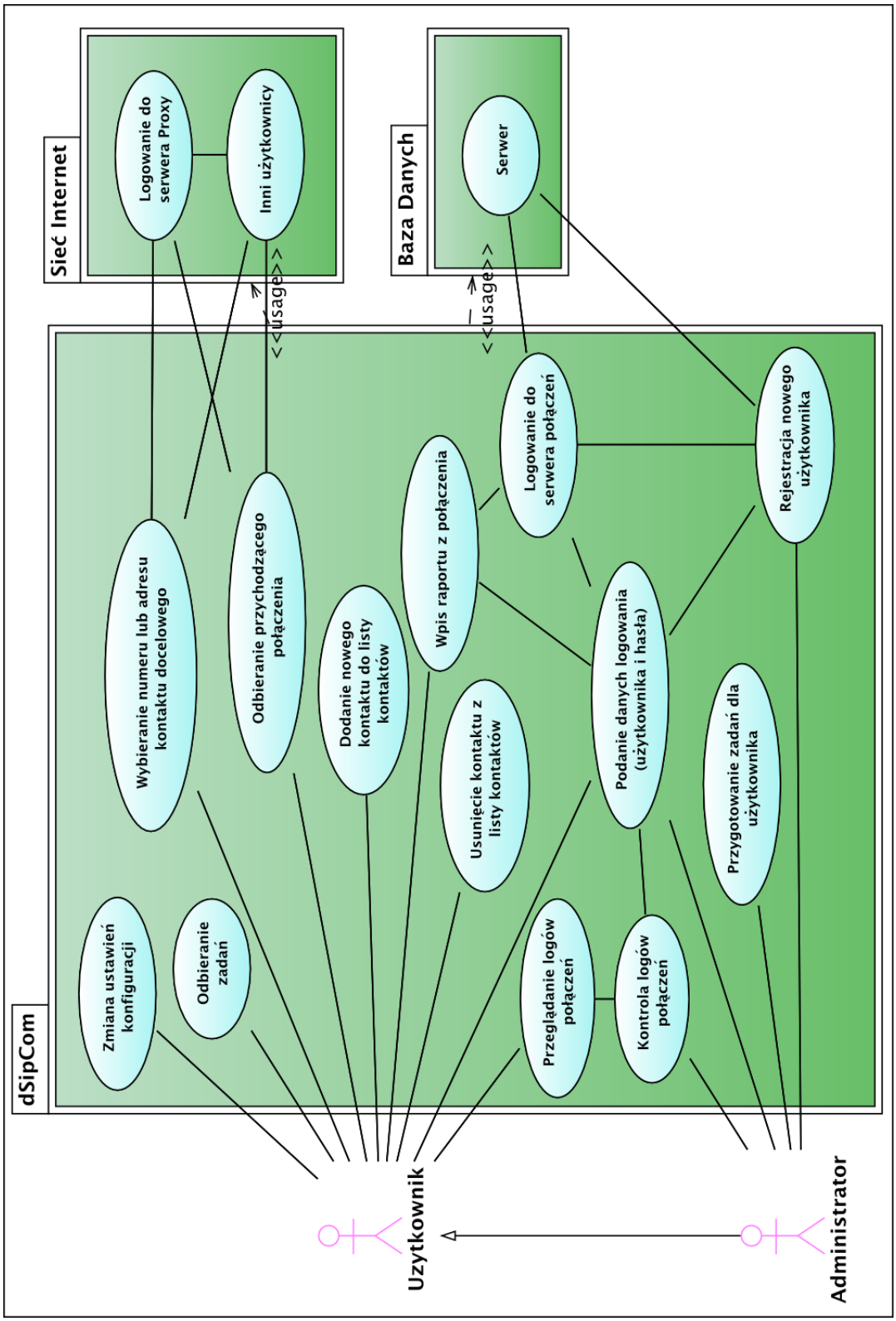


Diagram 2. Diagram przypadków użycia.

Na żółto przedstawione są komponenty składające się na pakiety – które przedstawione są jako zielone przestrzenie. Każdy pakiet to zbiór funkcji należących do aplikacji. Pakiety dodatkowe Qt4 i Boost, to pakiety dodatkowe wspomagające tylko pracę aplikacji. Pakiet *dSipCom* to główny, napisany przeze mnie moduł aplikacji. Pakiet ten korzysta bezpośrednio z komponentu *Qt4* w którym budowany jest interfejs użytkownika, oraz głównego komponentu *dSipCom* w którym zaimplementowana jest moja aplikacja. Komponent *Linphone*, składa się z pakietów kodeków audio (*Speex*, *GSM*, *FFMPEG*), pakietów do przesyłania i zarządzania strumieniami danych (*MediaStreamer*, *oRTP*) oraz pakietów odpowiedzialnych za parsowanie i wybieranie numerów i adresów SIP.

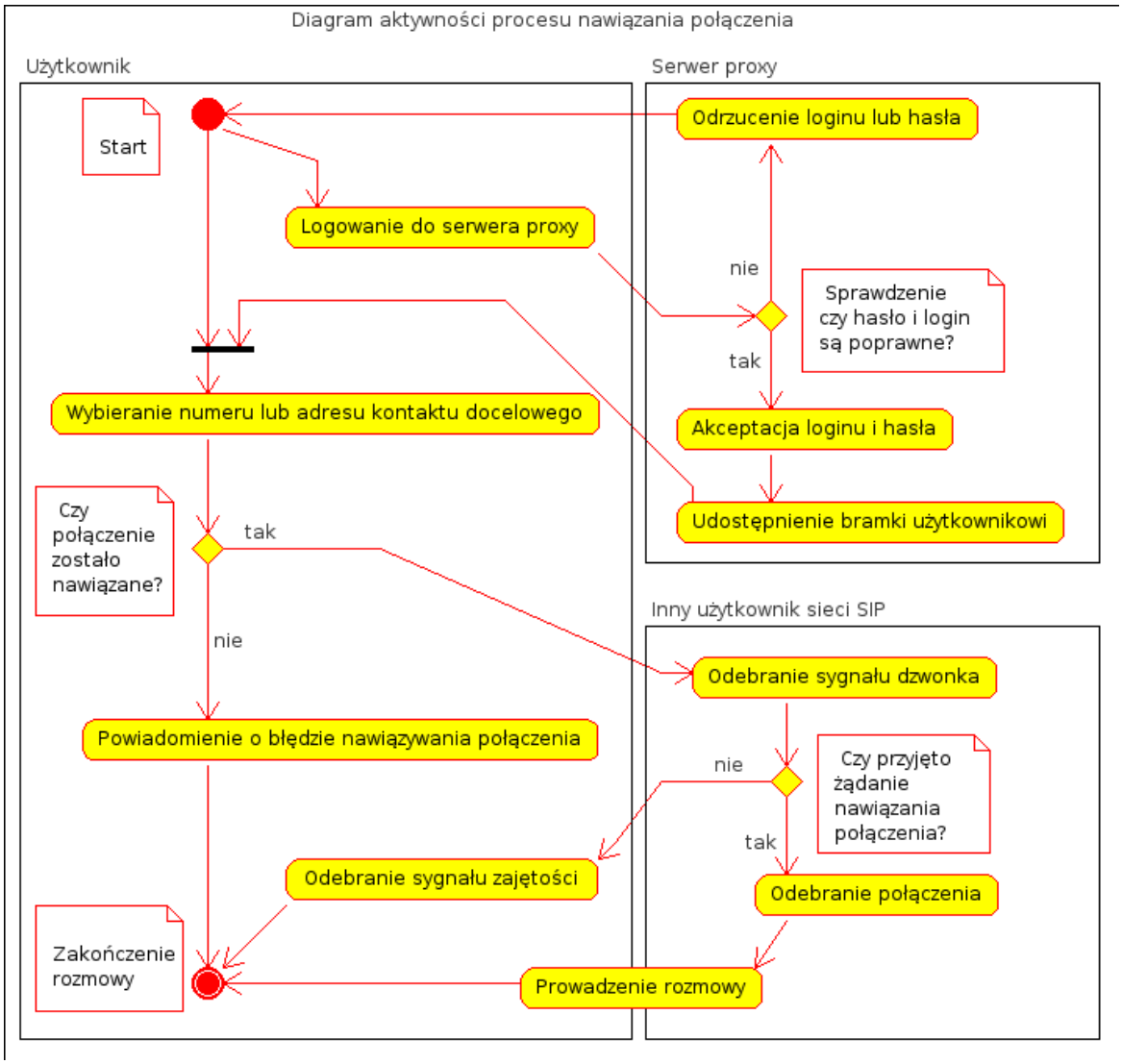


Diagram 3: „Diagram aktywności procesu nawiązywania połączenia”.

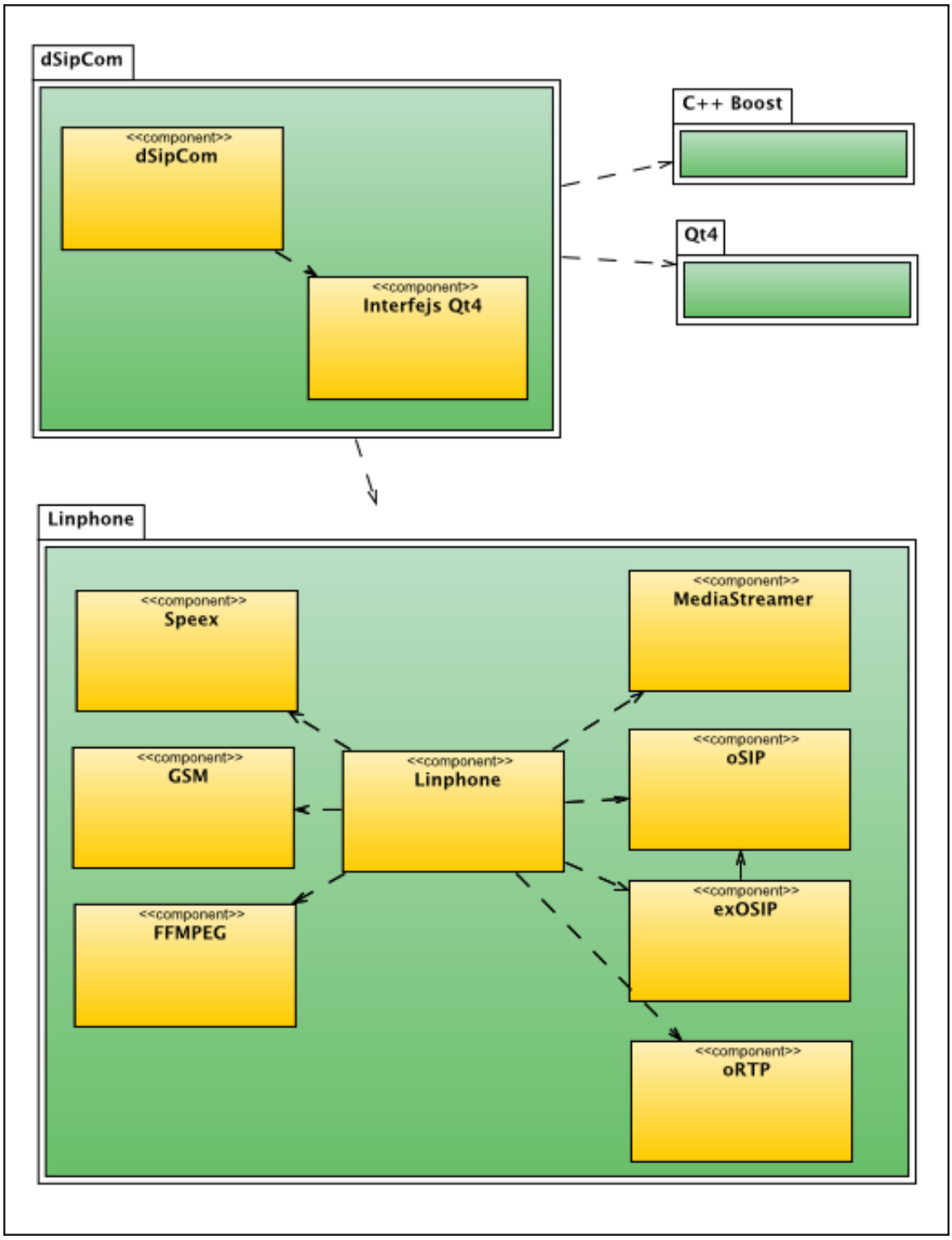


Diagram 4: „Diagram architektury aplikacji dSipCom”

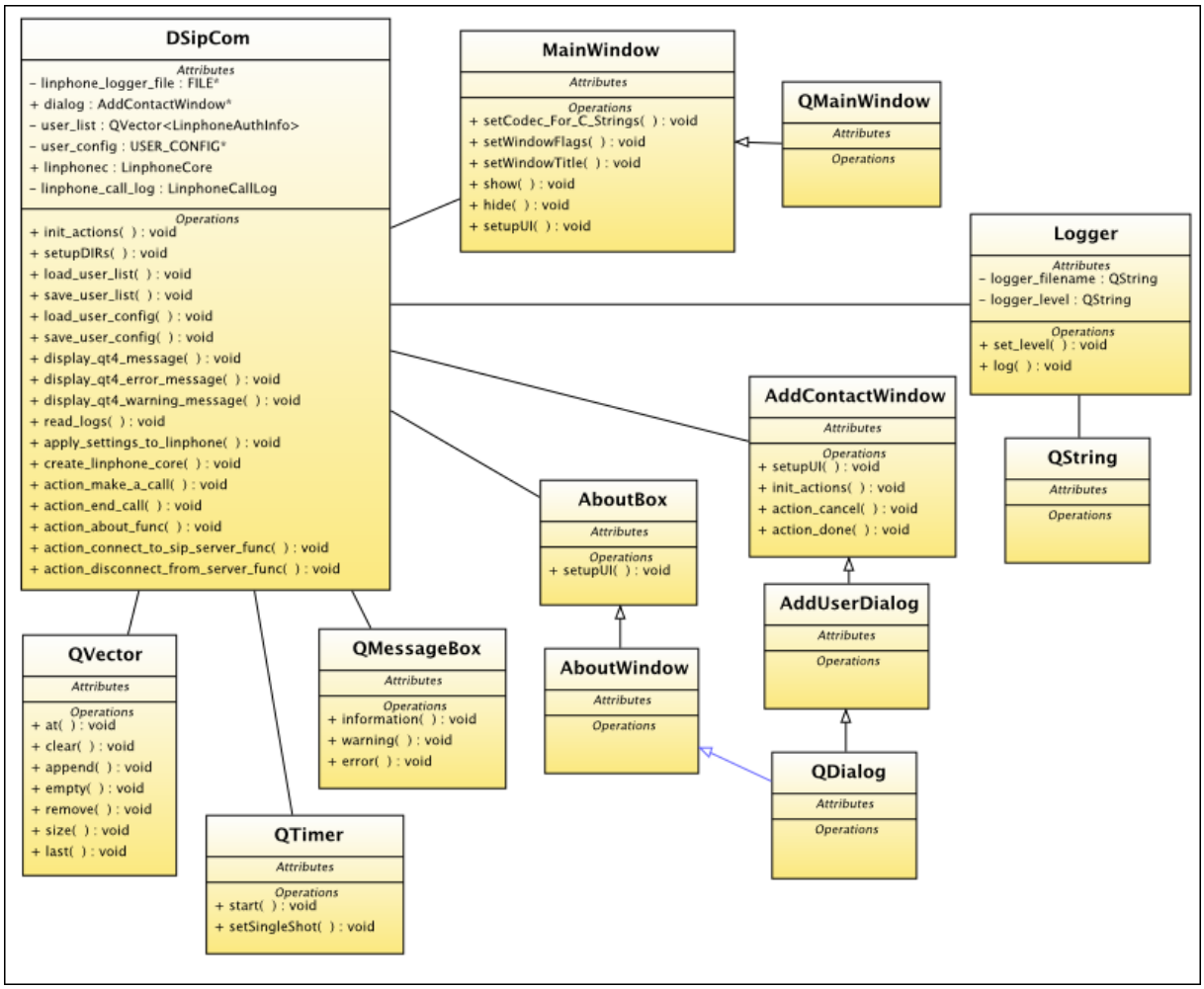


Diagram 5. Diagram klas.

3.4 Diagram klas

Diagram 5, przedstawia wzajemne relacje pomiędzy klasami aplikacji. Okno główne aplikacji zaimplementowane zostało w klasie DSipCom, która dziedziczy po wygenerowanej automatycznie (na podstawie stworzonego okna interfejsu) klasie *MainWindow*, która z kolei dziedziczy po szkieletowej klasie *QMainWindow*, służącej do tworzenia głównego okna programu. Pierwsze okno dodatkowe zaimplementowane w klasie *AddContactWindow*, jest dokowanym oknem dziedziczącym po wygenerowanej klasie *AddUserDialog*, która swój szkielet dziedziczy po klasie *QDialog*. Służy ono do wprowadzania nowych kontaktów do listy w głównym oknie programu. Trzecim oknem, które podobnie jak *AddContactWindow*

dziedziczy po klasie szkieletowej *Qdialog* to okno *AboutBox*. Dziedziczy ona po wygenerowanej przez *QT4-Designer'a*, klasie *AboutWindow*. Dodatkowe klasy składowe: *QTimer*, *QmessageBox* i *QVector*, służą kolejno do wykonywania czasowo określonych działań, wyświetlania wiadomości (powiadomień) i przechowywania list obiektów na listach wektorowych. Klasa *Logger*, to klasa służąca do logowania zdarzeń występujących w programie. Służy ona wyłącznie do logowania zdarzeń, mając na celu ułatwienie odnajdowania ewentualnych błędów mogących wystąpić w programie.

4. Wybór i uzasadnienie technologii i narzędzi.

W rozdziale tym uzasadniam wybór języka programowania, bibliotek dodatkowych i interfejsu użytkownika. Następnie uzasadniam wybór środowiska programistycznego, którego użyłem do napisania mojej aplikacji. Na końcu opisuję system kontroli wersji, użyty do zarządzania repozytorium mojego projektu.

4.1 Wybór i uzasadnienie języka programowania oraz wybór bibliotek dodatkowych

Przy wyborze języka programowania w którym chciałem napisać swoją aplikację kierowałem się wydajnością gotowego kodu, przenośnością między architekturami kodu źródłowego, oraz otwartością kodu źródłowego. Wybrałem język C++ i kompilator g++ (Gnu C++ Compiler). Dodatkowo użyłem elementów języka Ruby.

Uzasadniam swój wybór faktem szerokiej dostępności bibliotek użytkowych, w tym obsługujących technologie z których korzystam. Wśród nich są biblioteki:

- Linphone – Biblioteka łącząca wszystkie wymienione poniżej technologie w jedną całość.
- MediaStreamer – Biblioteka do przesyłania strumieni danych multimedialnych

dostarczana wraz z pakietem Linphone (http://www.linphone.org/).

- ORTP – Biblioteka wspierająca protokół czasu rzeczywistego (Real Time Protocol) dostarczana w pakiecie Linphone.
- OSIP2 – Biblioteka protokołu SIP – zgodna z RFC-3261. (http://savannah.gnu.org/projects/osip)
- oSIP2Parser – Biblioteka dołączana do pakietu OSIP2 – parsująca adresy SIP.
- Boost – Biblioteka algorytmów i funkcji opartych na STL (Standard Template Library) języka C++ (http://boost.org/).

Nie miałem dostępu do komercyjnych bibliotek z powodu braku środków na zakup licencji. Użycie elementów języka Ruby w mojej aplikacji, ma na celu możliwie najłatwiejszą integrację z projektem Szymona Jeża („Aplikacja internetowa wspomagająca zarządzanie sprzedawcami aktywnymi (telesprzedawcami)”).

4.2 Wybór i uzasadnienie biblioteki interfejsu użytkownika

Jednym z elementów, których potrzebowałem w zaprogramowaniu mojej aplikacji w C++, są biblioteki oferujące tzw GUI (Graphical User Interface). Pośród szerokiego wyboru różnych rozwiązań, wybrałem trzech kandydatów:

- WINGUI.
- GTK2.
- QT4.

Pierwsza, to jedna z bardziej znanych, na której oparta jest większość aplikacji pod systemy Windows. Jej wadą jest głównie brak możliwości natywnego wykorzystania/ uruchomienia w systemach innych niż sam Windows.

Kolejna, to niekomercyjna wydana na licencji GPL i LGPL biblioteka, wywodząca się

głównie z systemów zgodnych z systemami standardu POSIX (UNIX, BSD, Linux, MacOS), stworzona by umożliwić dorównanie aplikacjom z graficznym interfejsem użytkownika z tymi w systemach Windows. Jej wadą jest fakt, że została napisana w C i interfejs tej biblioteki także tworzony jest w strukturalnym kodzie C.

Ostatnią biblioteką, która spełniła wszystkie moje oczekiwania była QT4. Biblioteka ta wydana została na podwójnej licencji GPL i LGPL i stawia na niezależność od systemu operacyjnego użytkownika. Oferuje obiektowo zorientowany interfejs i sama napisana jest w C++. W chwili obecnej, natywnie obsługuje ona systemy: Linux, Windows i MacOS oferując przy tym łatwość tworzenia interfejsów podobną do znanego, komercyjnego produktu firmy Borland: C++ Builder.

Biblioteka QT4, składa się z wielu współpracujących ze sobą elementów:

- Designer-QT4 – Aplikacja do graficznego tworzenia interfejsów metodą WYSIWYG.
- UIC-QT4 – skrót od User Interface Compiler – zamienia pliki stworzone przez Designer'a na postać zrozumiałą dla kompilatora języka programowania.
- RCC – skrót od QT Resource Compiler – zamienia pliki map bitowych na postać struktur języka C.
- Qmake-QT4 – Tworzy plik Makefile z pliku projektu.

4.3 Wybór i uzasadnienie środowiska pracy

Kolejnym aspektem był wybór odpowiedniego środowiska pracy – IDE (Integrated Development Environment).

Rozpatrywałem następujące IDE: Netbeans, Eclipse i GVim

Wybrałem środowisko Netbeans w wersji 6.1.

Swój wybór uzasadniam tym, że od lat używam i cenię sobie produkt firmy SUN Microsystems. Netbeans jest środowiskiem dojrzałym, stabilnym, ciągle rozwijanym, z dostępnością wielu dodatków (wtyczek) pochodzących ze źrodeł niezależnych (Netbeans Community). Udostepnia dodatkowo większość wykorzystywanych dziś systemów kontroli wersji (w tym Subversion, CVS, Mercurial, SVK i GIT).

Co do Eclipse: uczestniczyłem w testowaniu tego środowiska i zauważyłem, że na 64 bitowych maszynach jest niestabilny, trudny w konfiguracji (niektórych oczywistych opcji nie da się nawet ustawić) i nie obsługuje takiej ilości systemów kontroli wersji jak np. Netbeans.

Co do GVim: okazał się w mojej opinii zbyt ubogi w możliwości (mimo ogromnych ilości dodatków ciągle tworzonych przez członków VIM Community)

4.4 Wybór systemu kontroli wersji

Kolejnym elementem do wyboru, był odpowiadający mi system kontroli wersji. Przeanalizowałem dwie opcje: Subversion i GIT. Wybrałem w moim mniemaniu lepszego GIT'a, ze względu na to, że posiada bogatszą funkcjonalność. Istotne dla mnie były następujące cechy GIT'a:

- GIT jest systemem rozproszonym. Oznacza to, że każda kopia repozytorium jest sama w sobie kompletnym repozytorium. W przeciwieństwie do SVN, GIT nie ma (choć może mieć) jednego głównego repozytorium. (Daje to na przykład możliwość pracy w zespole i wymianę danych lokalnie bez połączenia z siecią Internet)
- Propagowanie porządku w projekcie – mam tu na myśli fakt, iż w katalogu projektu GIT umieszcza ukryty katalog .git tylko raz, w głównym katalogu projektu. Subversion tworzy ukryte katalogi .svn w każdym podfolderze projektu, czego wynikiem może być nieład w repozytorium, szczególnie w przypadku dużych projektów.

- Brak konieczności posiadania dodatkowego oprogramowania – system kontroli GIT, po zainstalowaniu ma gotowe graficzne oraz tekstowe, pełnowartościowe narzędzia. (Należy wspomnieć o tym jak SVN nie potrafi automatycznie dodać nowych plików z drzewa katalogów do projektu bez dodatkowych narzędzi skanujących i wspierających jego działanie.)
- GIT jest bardziej odporny na błędy użytkownika niż SVN. (Przykładem może być sytuacja w której użytkownik projektu SVN spowoduje przypadkowe usunięcie katalogu, zawierającego podfolder .svn)

5. Implementacja i testowanie

W tym rozdziale znajdują się różnice implementacji względem projektu, listingi kodu źródłowego aplikacji (w tym testy) oraz opis i przedstawienie interfejsu użytkownika.

5.1 Różnica implementacji względem projektu

Najczęstszymi problemami przy projektowaniu aplikacji z założenia „idealnych", są ograniczenia wykorzystywanych bibliotek. Jako gotowe narzędzia przystosowane do wykorzystania, zawierają w sobie całą serię ograniczeń, do których musi dostosować się osoba chcąca z nich skorzystać.

Pierwszym ograniczeniem, które napotkałem to ograniczona ilość kodeków audio i wideo. Jest to oczywiście narzut spowodowany wykorzystaniem wyłącznie otwartego (darmowego) kodu, co uniemożliwia w aktualnej implementacji na korzystanie z wysokiej klasy komercyjnych bibliotek. Obecnie możliwe jest wykorzystanie następujących kodeków audio: GSM, Speex i G.711.

GSM (od ang. „Global System for Mobile Communications") jest implementacją jednego ze standardów kodowania i dekodowania głosu, wywodzący się ze standardu GSM

telefonii komórkowej w której używany jest do dziś. Charakteryzuje się on niskimi wymaganiami związanymi z prędkością transmisji przy akceptowalnej jakości zakodowanej mowy, oraz stosunkowo niewielką złożonością obliczeniową potrzebną do kodowania dźwięku. Istnieją cztery rodzaje tego kodeka, różniące się jakością i wymaganiami na przepustowość łącza sieciowego:

- ***HR*** (Half Rate). Prędkość transmisji 5.6Kibit/s. Używa algorytmu VSELP (Vector-Sum Excited Linear Prediction).
- ***FR*** (Full Rate). Prędkość transmisji 13Kibit/s. Używa algorytmu Regular Pulse Excitation - Long Term prediction Linear Predictive Coder (RPE-LTP).
- ***EFR*** (Enhanced Full Rate). Prędkość transmisji 12Kibit/s. Oferuje lepszą jakość niż FR, przez użycie algorytmu Algebraic-Code-Excited Linear Predictive (ACELP).
- ***AMR*** (Adaptive Multi Rate). Prędkość transmisji zmienna w zakresie od 4.75Kibit/s do 12.2Kibit/s. Używa kodeka Algebraic-Code-Excited Linear Predictive (ACELP), korzystając ze zmiennych współczynników kompresji.

Speex to format stratnej kompresji dźwięku z rodziny kodeków Ogg (http://en.wikipedia.org/wiki/Ogg). Jest przeznaczony tylko do kompresji mowy. Jego częstotliwość próbkowania to od 8 do 32Kibit/s. Jest to kodek powszechnie używany w telefonii VoIP, korzysta z algorytmu CELP i w zależności od warunków zajmuje pasmo 2.15 – 22.4 Kibit/s. Speex jest rozwijany przez Xiph Org Foundation (http://en.wikipedia.org/wiki/Xiph.Org_Foundation) na licencji BSD.

G.711 jest międzynarodowym standardem modulacji sygnałów mowy w kanałach o prędkości transmisji do 64Kibit/s i częstotliwości próbkowania 8 kHz z rozdzielczością 8 bitów na próbkę. Biorąc pod uwagę twierdzenie Kotielnikowa-Shannona, zgodnie z G.711 można kodować sygnały o częstotliwości do 4kHz. Istnieją dwa rodzaje tego kodeka:

- G.711U (PCMU). Stosuje algorytm μ-law, używany w Ameryce Północnej i

Japonii

- G.711A (PCMA). Stosuje algorytm A-law, używany przez resztę świata.

Aplikacja dSipCom automatycznie dostosuje odpowiedni kodek obsługiwany przez obydwie strony połączenia w chwili nawiązywania połączenia. W przypadku gdy strona nawiązująca połączenie wymusi kodowanie strumienia kodekiem nieobsługiwanym przez dSipCom, połączenie nie zostanie nawiązane i wystąpi błąd. Problem ten może być jednak łatwo rozwiązany. Nic nie stoi na przeszkodzie by zrekompilować odpowiednio biblioteki, z których korzysta moja aplikacja z wykorzystaniem dowolnego z kodeków komercyjnych. Z założenia standard SIP nie sugeruje korzystania z żadnej konkretnej technologii multimedialnej. Robi to dopiero warstwa SDP (Session Description Protocol), która także podtrzymuje niezależność od wykorzystanych bibliotek, opisując jedynie zasady, których trzeba się trzymać przy projektowaniu.

Kolejnym problemem jest fakt, iż korzystam z dość niebezpiecznych wersji oprogramowania. Skorzystałem z wersji trunk biblioteki Linphone, która jak sam autor wspomniał nie jest jeszcze w 100% skończona. Ze znaczących niedogodności jakie za tym idą jest fakt, że nie można jeszcze prowadzić konferencji głosowych z wykorzystaniem tej biblioteki. Istnieje jeszcze parę mało istotnych problemów, które jednak nie wpływają znacząco na zgodność z wymaganym standardem SIP (RFC-3261).

Zasadniczą niezgodnością implementacji z projektem jest fakt, iż biblioteka Linphone i biblioteki dodatkowe współpracujące z nią, są napisane w czystym ANSI C, a interfejs i logika aplikacji w C++. Co za tym idzie, musiałem pogodzić dwa dość odmienne podejścia do programowania: Strukturalne i Obiektowe. Dzięki paradygmatowości języka C++, nie sprawiło to większych problemów w implementacji, jednak nie pozwoliło to uniknąć kompromisów po stronie projektu. Zadeklarowany obiekt klasy DsipCom zawiera kod łączący obiektową strukturę ze strukturalnym kodem ANSI C. W pliku klasy DsipCom znajduje się deklaracja głównej struktury dynamicznej biblioteki Linphone – LinphoneCoreVTable, która ma za zadanie powiązać zdarzenia aplikacji (np. dzwonienie,

odbieranie połączeń) z konkretnymi metodami (także określonej struktury), zatem nie było możliwości by włączyć ją bezpośrednio do kodu obiektów głównej klasy programu . Dodatkowo ze strukturą tą, powiązana jest główna, krytyczna struktura, dzięki której program może wykonywać połączenia telefoniczne – LinphoneCore. Jest to struktura, na której oparty jest cały kod połączeń wywoływanych w programie.

Kolejną kwestią, która ma krytyczny wpływ na implementację aplikacji jest pojęcie wielozadaniowości. Próba tworzenia aplikacji jednowątkowej spowodowałaby oczywiste konsekwencje w tym niemożność sterowania programem gdy wykonujemy rozmowę. Rozwiąznia były dwa. Jedno z nich to wykorzystanie funkcji fork() systemów zgodnych ze standardem POSIX. Jednak jej implementacja w systemach niezgodnych z POSIX jest mocno utrudniona. Zdecydowałem się więc na korzystanie z natywnych wątków C (threads). Interfejs biblioteki Linphone domyślnie pisany jest z wykorzystaniem wątków. Nie musiałem dzięki temu ingerować w wewnętrzną strukturę jej działania. Pozostało jedynie napisać kod, który byłby bezpieczny (thread safe) względem pozostałych istniejących już wątków. Pierwszym wątkiem, który zaimplementowałem, jest nieskończoną pętlą, iterującą obiekt struktury Linphone, co było wymaganym krokiem narzuconym przez kod biblioteki. Drugi wątek, to cały interfejs użytkownika aplikacji, czyli wątek główny, który z punktu widzenia działania kodu jest wątkiem „głównodowodzącym”, mającym władzę i kontrolę nad pozostałymi. Trzeci i ostatni wątek zainicjowany przeze mnie to „pomocnik”, oferujący jedynie spusty czasowe. (time triggers).

5.2 Testy

Pisząc kod swojej aplikacji, sugerowałem się napisanymi wcześniej testami jednostkowymi, które testują wątpliwej jakości fragmenty kodu. Zacząłem od problemu multiplatformowości. Zakładając, że aplikacja ma działać niezależnie od architektury (w tym przypadku 32bit i 64bit), należało odpowiednio przygotować struktury zapisywane na dysk w postaci plików danych. Do wyboru miałem zapis tekstowy i binarny. Wybrałem binarny ponieważ jest on łatwiejszy do zaimplementowania. W przypadku pliku tekstowego

musiałbym dopisywać odpowiedni parser. Wymaganym założeniem, które musiałem przyjąć wykorzystując pliki binarne, było ustalenie stałego, niezależnego od architektury formatu pliku. Problem pojawia się przy zapisie liczb całkowitych (np. numeru portu połączenia SIP), a konkretniej długości słowa dla typu *int*. W zależności od architektury może być to 4 i 8 bajtów, a ponieważ pliki danych i konfiguracji mają być przenośne pomiędzy 32bit i 64bit architekturami, przyjąłem że zastosuję zunifikowane typy danych – *uint64_t* i *uint32_t*. Wymieniłem dwa, gdyż tylko tych używam w swojej aplikacji. Jest ich znacznie więcej a charakteryzuje je stała szerokość słowa niezależnie od procesora. Prócz testów typów danych, testowałem zapis i odczyt struktur danych do i z plików. W teście tym przedstawione są niemal wszystkie scenariusze wystąpienia problemów które mogą nastąpić. W testach zainicjowałem też sprawdzanie obsługi wyjątków, co ma chronić aplikację przed błędami użytkownika. Przetestowane zostały też niektóre funkcje zaczerpnięte z sieci np. te do konwersji typów C na typy C++ i vice versa oraz inne mniej ważne fragmenty kodu.

Testowanie kodu w postaci testów jednostkowych to jedno. Jednak aplikacje pisane w C i C++ są podatne na kilka rodzajów błędów. Jednym z nich są wycieki pamięci (memory leaks). Jednym z najlepszych otwartych narzędzi do testowania problemów związanych z pamięcią aplikacji jest ***Valgrind*** (strona projektu: http://valgrind.org/). Jest to aplikacja mająca bardzo wiele zastosowań, oferująca własne wirtualne środowisko uruchomieniowe, dzięki czemu może ona przetestować poprawność wszystkich operacji wykonywanych przez kod wynikowy aplikacji. Kolejnym narzędziem wykorzystywanym do testowania był znany wszystkim programistom gdb (strona projektu: http://gnu.org/software/gdb**)**. Gnu Debugger to kolejna aplikacja do testowania poprawnej pracy kodu wynikowego, ma jednak możliwość wskazania, w razie błędu, (np. sygnału SIGSEGV) dokładnej lokalizacji (linii kodu źródłowego), wraz z ostatnimi krokami wywołanych funkcji. (backtrace) Pomaga to w zlokalizowaniu wielu błędów, które byłyby trudne do odnalezienia w inny sposób.

Kolejnym krokiem, w kierunku uzyskania łatwej kontroli nad działaniem aplikacji, jest kod debugujący. Umieściłem go w ciele kodu aplikacji, korzystając z dyrektywy preprocesora

```
#ifdef DEBUG
```

W zależności od istnienia definicji DEBUG, do kodu dołączane są bloki z kodem wypisujące

komunikaty wprost na konsolę (terminal).

Ważnym aspektem przy kompilacji Aplikacji jest ustawienie flag kompilatora. (dla **g++** w systemie Linux, trzymane są one w zmiennej powłoki: CXXFLAGS) Żeby uzyskać maksymalną wydajność, ustawiłem flag:

```
-Os -march=athlon64
```

Opcja -O jest odpowiedzialna za stopień optymalizacji kodu, -march określa architekturę procesora pod który ma być generowany kod wynikowy. W trybie Debug użyłem flag:

```
-O0 -ggdb
```

Gdzie -ggdb jest parametrem umieszczającym w kodzie wynikowym dodatkowe informacje dla debuggera gdb. Dla trybu Debug użyłem optymalizacji zerowej, co ma szczególne znaczenie przy szukaniu przyczyny nieoczekiwanych błędów aplikacji. Spowodowane jest to tym, że podczas zbyt agresywnej optymalizacji kodu (np. opcja -O3), może ulec zmianie przebieg działania programu, a nawet może to spowodować sytuację w której program nie uruchomi się wcale. Uzależnione jest to w dużej mierze od rodzaju kodu i operacji jakie wykonuje.

5.3 Budowa interfejsu

W górnej części okna znajduje się menu aplikacji. Podzielone jest ono na trzy główne grupy:

- dSipCom – W której można znaleźć aktywatory połączenia i rozłączenia z serwerem proxy oraz wyjścia z programu.
- Kontakty – W tej grupie znajdują się aktywatory dodania i usunięcia elementów listy kontaktów.
- Pomoc – W której znajduje się aktywator pomocy i informacji o programie.

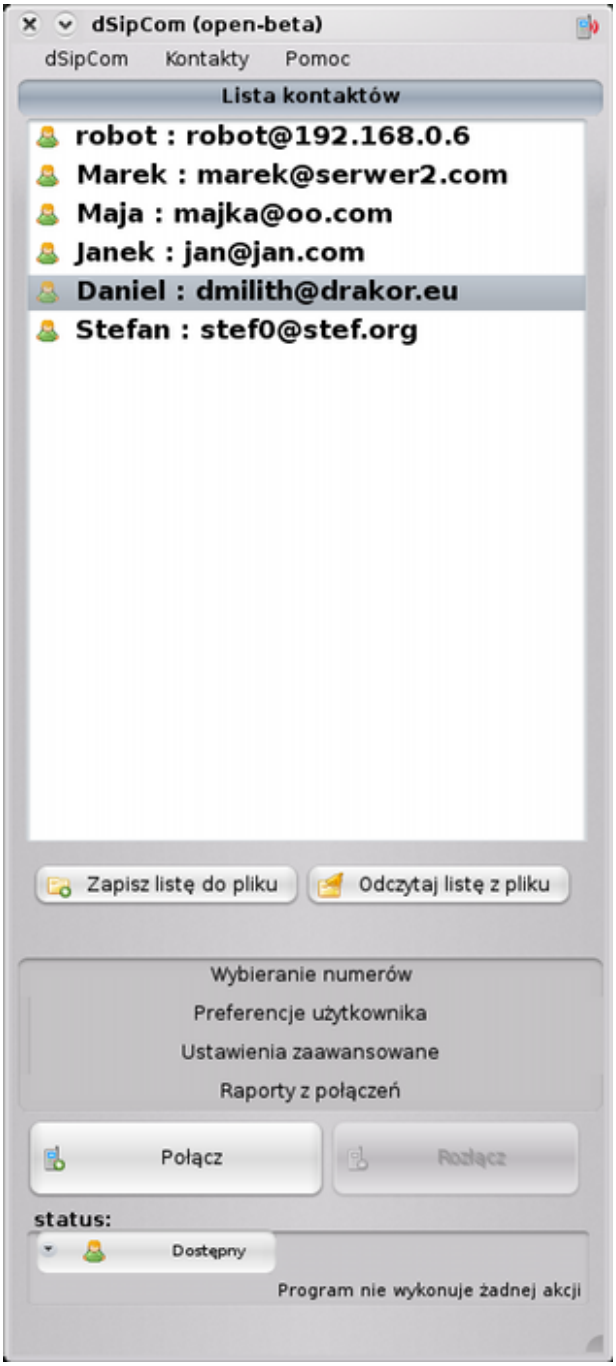
Interfejs główny aplikacji, podzielony jest na pięć zakładek. Pierwsza, to lista kontaktów SIP użytkownika, przedstawiona na rysunku 1.

Z poziomu widoku tej listy, można wykonywać połączenia oraz zapisywać lub/ i

odczytywać listę z pliku listy kontaktów. Druga zakładka, przedstawiona na rysunku 2, ma podobną funkcjonalność co pierwsza. Służy do ręcznego wprowadzania adresów SIP lub numerów telefonów, pod które można następnie zadzwonić.

Trzecia zakładka, przedstawiona na rysunku 3, zawiera preferencje podstawowe programu, w tym dane użytkownika i hasła gdy korzysta on z pośrednika (proxy) do wykonywania połączeń.

Z poziomu tej zakładki można zapisać lub wczytać całą konfigurację programu z plików konfiguracji. Czwarta zakładka, przedstawiona na rysunku 4, zawiera ustawienia zaawansowane i przeznaczona jest dla zaawansowanych użytkowników.



Rysunek 1. Lista kontaktów

Można tu skonfigurować parametry połączenia sieciowego, ustawienia głośności urządzeń podpiętych do karty dźwiękowej oraz sygnał dzwonka. Ostatnia zakładka przedstawiona została na rysunku 5. Zawiera ona listę wszystkich wchodzących i

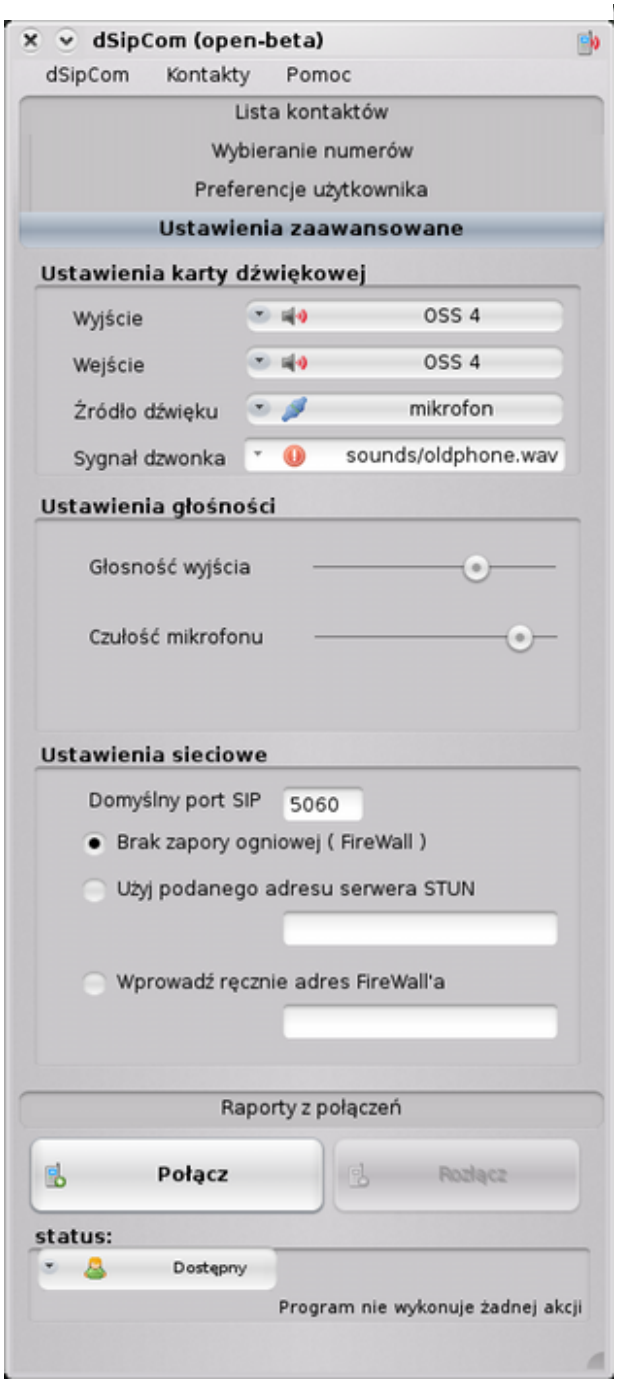
wychodzących połączeń, wraz z kalendarzem ułatwiającym wgląd w log z danego dnia:

Rysunek 2. Wybieranie numerów.

Poniżej zakładek głównych, znajdują się przyciski „Połącz” i „Rozłącz”. Mają one swoją funkcjonalność tylko przy jednoczesnej aktywności pierwszej lub drugiej zakładki głównej.

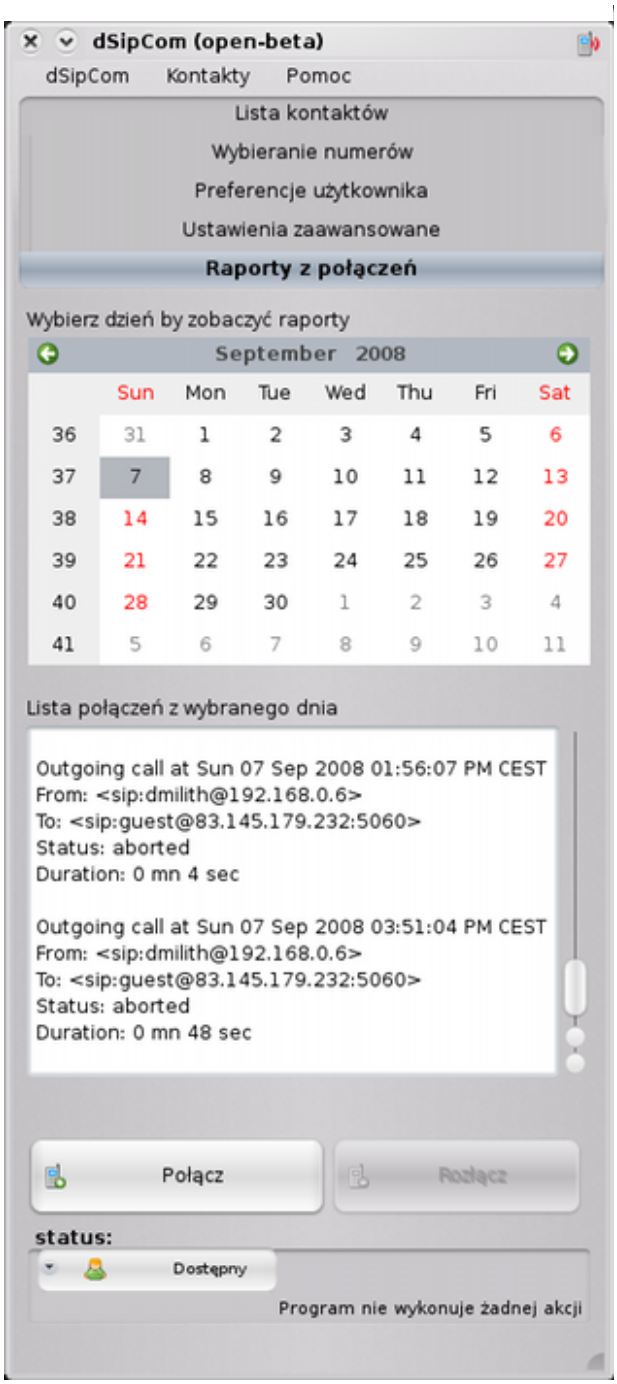
Rysunek 3. Preferencje użytkownika.

W przypadku wciśnięcia przycisku „Połącz” np. w trybie konfiguracji, użytkownik automatycznie zostanie przeniesiony do zakładki listy kontaktów.



Rysunek 4. Ustawienia zaawansowane.

W dolnej części interfejsu, znajduje się pole statusu aplikacji, w którym użytkownik informowany jest o bieżących działaniach programu oraz pole wyboru statusu dostępności.



Rysunek 5. Raporty z połączeń.

5.4 Listingi

Listing 1.Główny plik projektu dla Qmake.Nazwa pliku: dsipcom.pro.

```
CONFIG += debug
HEADERS += main.h \
          version.h \
          logger.h \
          dsipcom_ui.h \
          platform.h \
          d_utils.h

SOURCES += dsipcom_ui.cpp \
          logger.cpp \
          d_utils.cpp \
          platform.cpp \
          main.cpp

FORMS += dsipcom.ui \
         add_contact_dialog.ui \
         about.ui

RESOURCES += dsipcom.qrc
LIBS += -llinphone -lboost_filesystem

QMAKE_CXXFLAGS += -O2 -w -march=athlon64 -ggdb
TARGET = ../dsipcom
target.path = ../
sources.files = $$SOURCES $$HEADERS $$RESOURCES $$FORMS *.pro
sources.path = .
INSTALLS += target sources
```

Listing 2. Test Wejścia/ Wyjścia. Nazwa pliku: IO_file_test.cpp.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor . eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#include <iostream>
#include <sstream>
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cassert>

#define CONFIG_FILE "test.dcnf"
```

Listing 2. Test Wejścia/ Wyjścia. Nazwa pliku: IO_file_test.cpp.

```
using namespace std;

typedef struct {
  char contact_name[50];
  char contact_sip_address[50];
} USER_LIST;

typedef struct {
  char user_sip_server[50];
  char user_sip[50];
  char user_password[50];
  char user_name[50];
} USER_CONFIG;


string
strip( string s, char sign ) {
  string::iterator it = remove_if(s.begin(), s.end(),
  bind2nd( equal_to<char>(), sign ));
  s = string( s.begin(), it );
  return s;
}

const char*
uint2cstr( uint64_t i ) {
  stringstream ss;
  string temp;
  ss << i;
  ss >> temp;
  return temp.c_str();
}

int
main() {
// uint2cstr test
  uint32_t z32 = 1234567890;
    assert( uint2cstr( z32 ) == (string)"1234567890" );
  uint64_t z64 = 1234567890;
    assert( uint2cstr( z64 ) == (string)"1234567890" );

  z32 = 12345678901234567890;
    assert( uint2cstr( z32 ) != (string)"12345678901234567890" );
  z64 = 12345678901234567890;
    assert( uint2cstr( z64 ) == (string)"12345678901234567890" );

  // data writing to file test 1
    USER_CONFIG *user_config = NULL, *readed = NULL;
    user_config = new USER_CONFIG;
    readed = new USER_CONFIG;
```

Listing 2. Test Wejścia/ Wyjścia. Nazwa pliku: IO_file_test.cpp.

```
      strcpy( user_config->user_name, "dmilith" );
      strcpy( user_config->user_password, "alaniemakota_nieszyfrowane!");
      strcpy( user_config->user_sip, "sip:dmilith@drak.kill.pl");
      strcpy( user_config->user_sip_server, "ekiga.net");
    FILE* config_file;
      config_file = fopen( CONFIG_FILE, "wb" );
      fwrite( user_config, sizeof( USER_CONFIG ), 1, config_file );
      fclose( config_file );
      config_file = fopen( CONFIG_FILE, "rb" );
      fread( readed, sizeof( USER_CONFIG ), 1, config_file );
      fclose( config_file );
      assert( (string)user_config->user_name == (string)readed->user_name );
      assert( (string)user_config->user_password == (string)readed->user_password );

    // removing spaces from string (utils.h):
      assert( strip( " V er   tic es ", ' ' ) == (string)"Vertices" );
      assert( strip( " v d f  ", ' ' ) == (string)"vdf" );
      assert( strip( " z***łto s,23*(%^&#  *", '*') == (string)" złto s,23(%^&#  " );
      assert( strip( "  ", ' ') == (string)"" );
      assert( strip( "123", '*') == (string)"123" );
      assert( strip( "% % %", '%') == (string)"  " );

  return 0;
}
```

Listing 3. Test rozmiarów zmiennych w zależności od architektury. Nazwa pliku: varsizes_test.cpp.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor . eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#include <cstdio>
#include <cstdlib>
#include <cassert>
#include <cstring>
#include <iostream>
#include <string>
#include <cstdint>

using namespace std;

int
main( int argc, char** argv ) {
```

Listing 3. Test rozmiarów zmiennych w zależności od architektury. Nazwa pliku: varsizes_test.cpp.

```
    if ( sizeof( void * ) == 8 ) { // 64bit
        assert( sizeof( int ) == 4 );
        assert( sizeof( long int ) == 8 );
        assert( sizeof( short int ) == 2 );
        assert( sizeof( double ) == 8);
      char a[] = "dulf0";
        assert( sizeof( a ) == 5+1 );
        string b = "dulf0";
        assert( sizeof( b ) == 8 );
      char* HOME = getenv( "HOME" );
        assert( getenv( "HOME" ) == HOME );
        string HOME2 = getenv( "HOME" );
        assert( getenv( "HOME" ) == HOME2 );
      uint64_t number = 123;
        assert( sizeof( number ) == 8 );
        assert( sizeof( char ) == 1 );
        assert( sizeof( char[5] ) == 5 );
    }

    if ( sizeof( void * ) == 4 ) { // 32bit
        assert( sizeof( int ) == 4 );
        assert( sizeof( long int ) == 4 );
        assert( sizeof( short int ) == 2 );
        assert( sizeof( double ) == 8);
      char a[] = "dulf0";
        assert( sizeof( a ) == 5 + 1 );
        string b = "dulf0";
        assert( sizeof( b ) == 4 );
      char* HOME = getenv( "HOME" );
        assert( getenv( "HOME" ) == HOME );
        string HOME2 = getenv( "HOME" );
        assert( getenv( "HOME" ) == HOME2 );
      uint64_t liczba = 123;
        assert( sizeof( liczba ) == 8 );
        assert( sizeof( char ) == 1 );
        assert( sizeof( char[5] ) == 5 );
    }
    return (EXIT_SUCCESS);
}
```

Listing 4. Test modułu logowania. Nazwa pliku: log_module_test.cpp.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor . eu
 * released under GPL2 & LGPL license
```

Listing 4. Test modułu logowania. Nazwa pliku: log_module_test.cpp.

```
 * (c) 2oo8
 *
 */

#include <cstring>
#include <cassert>
#include <algorithm>
#include <sstream>
#include <vector>
#include <fstream>
#include <iostream>

#define DEBUG

using namespace std;

typedef struct {
  uint32_t day, month, year;
  char str1[50000];
} LOG_ELEMENT;

void
write_one_log_by_date( string log, uint32_t day, uint32_t month, uint32_t year, const char* filename ) {
  ofstream file( filename, ios_base::out | ios_base::binary | ios_base::app );
  LOG_ELEMENT *log_n = new LOG_ELEMENT;
    log_n->day = day;
    log_n->month = month;
    log_n->year = year;
    strcpy( log_n->str1, log.c_str() );
    file.write( (char*)log_n, sizeof( LOG_ELEMENT ) );
    file.close();
  delete log_n;
  return;
}

const string
read_one_log_by_date( uint32_t day, uint32_t month, uint32_t year, const char* filename ) {
        ifstream file( filename, ios_base::in | ios_base::binary );
  LOG_ELEMENT *str_result = new LOG_ELEMENT;
        string result = "";
                if ( !file ) return "";
      while ( file.good() ) {
        file.read( (char*)str_result, sizeof( LOG_ELEMENT ) );
        if ( file.eof() ) {
#ifdef DEBUG
        std::cout << "EOF" << std::endl;
        std::cout.flush();
#endif
                                    break;
```

Listing 4. Test modułu logowania. Nazwa pliku: log_module_test.cpp.

```
                        }
                        if ( ( day == str_result->day) && ( month == str_result->month ) &&
            ( year == str_result->year ) ) result += str_result->str1;
                  }
        file.close();
        delete str_result;
        return result;
}

int
main() {

        assert( read_one_log_by_date( 8, 2, 35, "test_Logs" ) == "" );
  write_one_log_by_date( "jakas tresc\ndruga.", 1,2,3, "test_Logs" );
  write_one_log_by_date( "ABC.", 6,6,6, "test_Logs" );
  write_one_log_by_date( "DEF.", 6,6,6, "test_Logs" );
  write_one_log_by_date( "GHI..", 6,6,6, "test_Logs" );

  assert( read_one_log_by_date( 1, 2, 3, "test_Logs" ) == "jakas tresc\ndruga." );
        string z = read_one_log_by_date( 1, 2, 3, "test_Logs" );
  assert( read_one_log_by_date( 1, 2, 3, "test_Logs" ) != "jakas tresc\ndruga ." );
  assert( read_one_log_by_date( 1, 2, 3, "test_Logs" ) != "jakas  tresc\ndruga." );
  assert( read_one_log_by_date( 6, 6, 6, "test_Logs" ) == ( "ABC.DEF.GHI.." ) );
  assert( read_one_log_by_date( 4, 2, 99, "test_Logs" ) == "" );
  assert( read_one_log_by_date( 4, 2, 99, "test_Logs" ) != " " );

  return 0;
}
```

Listing 5. Zbiór narzędzi pomocniczych wykorzystanych w kodzie głównym. Nazwa pliku: d_utils.h.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#ifndef D_UTILS_H
#define D_UTILS_H

#include <iostream>
#include <algorithm>
#include <cstring>
#include <string>
#include <sstream>
```

Listing 5. Zbiór narzędzi pomocniczych wykorzystanych w kodzie głównym. Nazwa pliku: d_utils.h.

```
#include <vector>
#include <fstream>

using namespace std;

typedef struct {
  uint32_t day, month, year;
  char str1[50000]; // XXX: 50KB per log. it's awful way but easiest
} LOG_ELEMENT;


// strips specified string from all ''sign'' chars
  string
  strip( string s, char sign );

// integer to C string converter
  const char*
  uint2cstr( uint64_t i );

  void
        write_one_log_by_date( string log, uint32_t day, uint32_t month, uint32_t year, const char*
filename );

        const string
  read_one_log_by_date( uint32_t day, uint32_t month, uint32_t year, const char* filename );

#endif /* D_UTILS_H */
```

Listing 6. Zbiór narzędzi pomocniczych, wykorzystanych w kodzie głównym. Nazwa pliku: d_utils.cpp.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#include "d_utils.h"

using namespace std;

string strip( string s, char sign ) {
  string::iterator it = remove_if( s.begin(), s.end(),
  bind2nd( equal_to<char>(), sign ) );
  s = string( s.begin(), it );
```

Listing 6. Zbiór narzędzi pomocniczych, wykorzystanych w kodzie głównym. Nazwa pliku: d_utils.cpp.

```
  return s;
}

const char*
uint2cstr( uint64_t i ) {
  stringstream ss;
  string temp;
  ss << i;
  ss >> temp;
  return temp.c_str();
}

void write_one_log_by_date( std::string log, uint32_t day, uint32_t month, uint32_t year, const char*
filename ) {
    ofstream file( filename, ios_base::out | ios_base::binary | ios_base::app );
    LOG_ELEMENT *log_n = new LOG_ELEMENT;
                log_n->day = day;
    log_n->month = month;
    log_n->year = year;
    strcpy( log_n->str1, log.c_str() );
                file.write( (char*)log_n, sizeof( LOG_ELEMENT ) );
                file.close();
                delete log_n;
                return;
}

const string read_one_log_by_date( uint32_t day, uint32_t month, uint32_t year, const char* filename ) {
        ifstream file( filename, ios_base::in | ios_base::binary );
  LOG_ELEMENT *str_result = new LOG_ELEMENT;
        string result = "";
                if ( !file ) return "";
                while ( file.good() ) {
                        file.read( (char*)str_result, sizeof( LOG_ELEMENT ) );
                        if ( file.eof() ) {
#ifdef DEBUG
        std::cout << "EOF" << std::endl;
        std::cout.flush();
#endif
                                break;
                        }
                        if ( ( day == str_result->day) && ( month == str_result->month ) && ( year ==
str_result->year ) )
            result += str_result->str1;
                }
        file.close();
        delete str_result;
        return result;
}
```

Listing 7. Plik wersji aplikacji. Nazwa pliku: version.h.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#ifndef _DSIPCOM_VERSION_H
#define _DSIPCOM_VERSION_H

using namespace std;

static const string DSIPCOM_VERSION = "v0.6.6b";
static const string MAIN_WINDOW_TITLE = "dSipCom (open-beta)";

#endif
```

Listing 8. Plik loggera aplikacji. Nazwa pliku: logger.h.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#ifndef _LOGGER_H
#define _LOGGER_H

#include <qt4/QtCore/QTextOStream>
#include "main.h"

using namespace std;

namespace Log {

  class Logger {
    public:
      QString logger_filename;
      QString logger_level;

      Logger( const QString& log_filename, const QString& level = "notice" );
      void set_level( const QString& level_name );
      void log( const QString& message );
```

Listing 8. Plik loggera aplikacji. Nazwa pliku: logger.h.

```
  };

}

#endif  /* _LOGGER_H */
```

Listing 9. Plik loggera aplikacji. Nazwa pliku: logger.cpp.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#include <qt4/QtCore/QFile>
#include "logger.h"
#include "main.h"

using namespace Log;

Logger::Logger( const QString& log_filename, const QString& level ) {
  logger_filename = log_filename;
  logger_level = level;
}

void
Logger::log( const QString& message ) {
  QFile logger_file( logger_filename );
  logger_file.open( QIODevice::Text | QIODevice::Append | QIODevice::WriteOnly );
  QTextStream logger( &logger_file ); // logger
  logger.setCodec( "UTF-8" );
  logger << logger_level << ": "<< message << "\n";
}

void
Logger::set_level( const QString& level_name ) {
  logger_level = level_name;
}
```

Listing 10. Kod głównodowodzący aplikacji. Nazwa pliku: main.h.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
```

Listing 10. Kod głównodowodzący aplikacji. Nazwa pliku: main.h.

```
 * © 2oo8
 */

#ifndef _DSIPCOM_MAIN_H
#define _DSIPCOM_MAIN_H

#include <qt4/QtGui/QApplication>
#include <string>
#include <execinfo.h>

// enabling debuging options
#define DEBUG
//#define WIN32
// enable asserts in code
#define D_ASSERT

using namespace std;

const string SLASH = "/";
const string BACKSLASH = "\\";
static string pp = SLASH; // "normal" UNIX slash to separate parts path

// main declarations and constants
#ifndef WIN32
 #include <cstdlib>
 const string DSIP_MAIN_DIR = string( getenv( "HOME" ) ) + pp + ".dSipCom";
#else
 pp = BACKSLASH;
 // TODO: fix static path for windows
 const string DSIP_MAIN_DIR = string( "C:" + pp + "dSipCom" );
#endif

 const string LOGS_DIR = DSIP_MAIN_DIR + pp + "logs";
 const string CONF_DIR = DSIP_MAIN_DIR + pp + "config";
 const string ULIST_DIR = DSIP_MAIN_DIR + pp + "user_list";

 const string LOGGER_FILE = LOGS_DIR + pp + "dsipcom.main.log";
//      static const string LOGGER_LINPHONE = LOGS_DIR + "dsipcom.linphone.log";
 const string LOGGER_DSIPCOM_UI = LOGS_DIR + pp + "dsipcom.logger.ui.log";

 const string CONFIG_FILE = CONF_DIR + pp + "dsipcom.dcnf";
 const string LINPHONE_CONFIG = CONF_DIR + pp + "linphone.conf";
 const string CALL_LOG_FILE = DSIP_MAIN_DIR + pp + "call_logs.dlog";

 const string USER_LIST_FILE = ULIST_DIR + pp + "dsipcom.dulf";
#endif
```

Listing 11. Kod głównodowodzący aplikacji. Nazwa pliku: main.cpp.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#include "main.h" // main constants and settings
#include "version.h" // main program version
#include "platform.h" // platform specific setting
#include "dsipcom_ui.h" // main user interface

using namespace Ui;


void
backtrace( void ) {
  void *addresses[ 10 ];
  char **strings;
  uint64_t size = backtrace( addresses, 10 );
  strings = backtrace_symbols( addresses, size );
   cout << "Stack frames: " << size << endl;
    for ( uint64_t i = 0; i < size; i++ ) {
      cout << i << " : " << (uint64_t)addresses[ i ] << endl;
      cout << strings[ i ];
    }
  free(strings);
}

void
received_normal_signal( int param ) {
  cout << "\n\nReceived signal : " << param << endl;
  backtrace();
}

void
received_SIGSEGV_signal( int param ) {
  cout << "\n\nSIGSEGV : " << param << " !\n";
  backtrace();
  exit(11);
}

int
main( int argc, char *argv[] ) {
  /* macro to load images from dsipcom.qrc
     could be problematic on some archs.. but on linux seems to be unecessary
     TODO: make dependancy to running OS,
     Q_INIT_RESOURCE(dsipcom); */
```

Listing 11. Kod głównodowodzący aplikacji. Nazwa pliku: main.cpp.

```
   QApplication app( argc, argv );
   DSipCom* main_obj = new DSipCom( MAIN_WINDOW_TITLE.c_str() );

   //signal handling
   signal( SIGINT, received_normal_signal );
   signal( SIGSEGV, received_SIGSEGV_signal );

   if ( app.exec() == 0 ) {
     delete main_obj;
   }

  return 0;
}
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.h.

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

#ifndef _DSIPCOM_UI_H
#define _DSIPCOM_UI_H

#include <cstdio>
#include <csignal>
#include <cassert>

#include <qt4/QtCore/QTimer>
#include <qt4/QtCore/Qt>
#include <qt4/QtCore/QFile>
#include <qt4/QtGui/QDialog>
#include <qt4/QtCore/QTextCodec>
#include <qt4/QtGui/QMessageBox>
#include <qt4/QtCore/QQueue>
#include <qt4/QtGui/QSlider>
// interesting way to make widget int omain window.. #include <qt4/QtGui/QDockWidget>

#include <linphone/config.h>
#include <linphone/linphonecore.h>

#include <string>
#include <iostream>
#include <sstream>
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.h.

```
#include <boost/filesystem/operations.hpp>

#include "version.h"
#include "d_utils.h"
#include "ui_dsipcom.h" // automaticly generated interface from ui file
#include "ui_add_contact_dialog.h" // automaticly generated widget window
#include "ui_about.h" // obviously, just about window
#include "logger.h" //logger defs
#include "main.h"

#define MAX_PENDING_AUTH 8
#define PROMPT_MAX_LEN 256        /* max len of prompt string */
#define LINE_MAX_LEN 256 /* really needed ? */

// Linphone definitions
typedef struct {
  LinphoneAuthInfo *elem[MAX_PENDING_AUTH];
  int nitems;
} LPC_AUTH_STACK;

typedef struct {
  char user_sip_server[50];
  char user_sip[50];
  char user_password[50];
  char user_name[50];
        // 64bit numbers no matter system architecture
  char out_soundcard[50];
  char in_soundcard[50];
  char recording_source[50];
  char ring_sound[255];
  char default_port[5];
  uint32_t no_firewall;
  uint32_t use_stun_server;
  char stun_address[50];
  uint32_t manual_firewall_address;
  char firewall_address[50];
  uint8_t microphone_volume;
  uint8_t output_volume;
} USER_CONFIG;

/* User Interface namespace is providing main UI inherited from automaticaly generated by qt-designer
templates */
namespace Ui {

  class AddContactWindow : public QDialog, public Ui::addUserDialog {
    // qt4 ui macro (for actions)
    Q_OBJECT

    public:
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.h.

```
    AddContactWindow( QWidget *parent );
    ~AddContactWindow();
    void init_actions();

  public slots:
    void action_done();
    void action_cancel();

  signals:
    void clicked();
};

class AboutBox : public QDialog, public Ui::AboutWindow {
  // qt4 ui macro (for actions)
  Q_OBJECT

  public:
    AboutBox();
    ~AboutBox();
};

class DSipCom : public QMainWindow, public Ui::MainWindow {
  // qt4 ui macro (for actions)
  Q_OBJECT

  public:
    DSipCom( const QString& title = "dSipCom" );
   ~DSipCom();

    // init qt4 actions (ui slots and signals)
    void init_actions();

    // loading data from files
    void setupDIRs();
    void load_user_list();
    void save_user_list();
    void load_user_config();
    void save_user_config();

    void display_qt4_error_message( const char* message );
    void display_qt4_warning_message( const char* message );
    void display_qt4_message( const char* message );

    void read_logs();

    // init linphone
    void apply_settings_to_linphone();
    void create_linphone_core();
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.h.

```
      QVector<LinphoneAuthInfo> user_list;
      USER_CONFIG *user_config;

    private:
      // add contact dialog:
      AddContactWindow *dialog;

    // qt4 action slots
    public slots:
      void linphonec_main_loop();
      void reset_status_bar();
      void action_help_func();
      void action_make_a_call();
      void action_end_call();
      void action_about_func();
      void action_add_contact_func();
      void action_remove_contact_func();
      void action_connect_to_sip_server_func();
      void action_disconnect_from_sip_server_func();
      void action_enter_0();
      void action_enter_1();
      void action_enter_2();
      void action_enter_3();
      void action_enter_4();
      void action_enter_5();
      void action_enter_6();
      void action_enter_7();
      void action_enter_8();
      void action_enter_9();
      void action_enter_star();
      void action_enter_hash();
      void action_save_user_config();
      void action_load_user_config();
      void action_save_user_list();
      void action_load_user_list();
      void action_get_log_func();

    // qt4 action signals
    signals:
      void clicked();
  };

} // of namespace

#endif   /* _DSIPCOM_UI_H */
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
/*
 * author: Daniel (dmilith) Dettlaff
 * email: dmilith at drakor.eu
 * released under GPL2 & LGPL license
 * (c) 2oo8
 *
 */

// TODO: make header check for dsipcom.dcnf

#include "dsipcom_ui.h"

using namespace Log;
using namespace Ui;
using namespace std;
using namespace boost::filesystem;

// Linphone core variables & consts
//
LinphoneCore linphonec;
LinphoneCallLog linphone_call_log;
LinphoneProxyConfig *pcfg = NULL;
const MSList *proxy_list;
// today_log will contain current session call log
static string today_log = "";
// List of sound devices
static const char **sound_dev_names;
// List of sound codecs
static const MSList *audio_codec_list, *video_codec_list;
FILE* linphone_logger_file;
LPC_AUTH_STACK auth_stack; // stack of auth requests (?)
//char prompt[PROMPT_MAX_LEN];
static bool_t auto_answer = FALSE;
static bool_t vcap_enabled = FALSE;
static bool_t display_enabled = FALSE;
// pending_call_sip contains sip address of caller
static string pending_call_sip;

        /* Linphone structs
            These are callbacks for linphone core */
                static void linphonec_call_received( LinphoneCore *lc, const char *from );
                static void linphonec_prompt_for_auth( LinphoneCore *lc, const char *realm, const char
*username );
                static void linphonec_display_something ( LinphoneCore * lc, const char *something );
                static void linphonec_display_url ( LinphoneCore * lc, const char *something, const char
*url );
                static void linphonec_display_warning ( LinphoneCore * lc, const char *something );
                static void stub () {}
                static void linphonec_notify_received( LinphoneCore *lc, LinphoneFriend *fid,
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
                const char *from, const char *status, const char *img );
                static void linphonec_new_unknown_subscriber( LinphoneCore *lc,

                LinphoneFriend *lf, const char *url );
                static void linphonec_bye_received( LinphoneCore *lc, const char *from );
/*      TODO:   static void linphonec_text_received( LinphoneCore *lc, LinphoneChatRoom *cr,

                const char *from, const char *msg ); */
                static void linphonec_display_status ( LinphoneCore * lc, const char *something );
    static void linphonec_call_log_updated( LinphoneCore *lc, LinphoneCallLog *call_log );

                // main Linphone table.
                LinphoneCoreVTable linphonec_vtable = {
                        show:(ShowInterfaceCb) stub,
                        inv_recv: linphonec_call_received,
                        bye_recv: linphonec_bye_received,
                        notify_recv: linphonec_notify_received,
                        new_unknown_subscriber: linphonec_new_unknown_subscriber,
                        auth_info_requested: linphonec_prompt_for_auth,
                        display_status: linphonec_display_status,
                        display_message: linphonec_display_something,
                        display_warning: linphonec_display_warning,
                        display_url: linphonec_display_url,
                        display_question: (DisplayQuestionCb)stub,
        call_log_updated: linphonec_call_log_updated,
                  // TODO: text_received:linphonec_text_received,
                };

      void
      display_qt4_error_message( const char* message ) {
        QMessageBox::critical( 0, MAIN_WINDOW_TITLE.c_str(), message );
      }

      void
      display_qt4_warning_message( const char* message ) {
        QMessageBox::warning( 0, MAIN_WINDOW_TITLE.c_str(), message );
      }

      void
      display_qt4_message( const char* message ) {
        QMessageBox::information( 0, MAIN_WINDOW_TITLE.c_str(), message );
      }

                /* Linphone callbacks definitions */
      static void
      linphonec_call_log_updated( LinphoneCore *lc, LinphoneCallLog *call_log ) {
        lc = &linphonec;
        call_log = &linphone_call_log;
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
        MSList *elem = linphone_core_get_call_logs( lc );
        for ( ; elem != NULL; elem = ms_list_next( elem ) ) {
                LinphoneCallLog *cl = (LinphoneCallLog*)elem->data;
                char *str = linphone_call_log_to_str( cl );
             #ifdef DEBUG
                cout << endl << "CallLog:" << str << endl << endl << flush;
             #endif
                today_log += (string)str + "\n"; // adding call logs to common log
                ms_free( str );
        }
      }

                        static void
                        linphonec_display_something ( LinphoneCore * lc, const char *something ) {
        lc = &linphonec;
        #ifdef DEBUG
          cout << "\ndebug_linphonec_display_something_: " << something << endl << flush;
        #endif
        display_qt4_message( something );
                        }

                        static void
                        linphonec_display_status ( LinphoneCore * lc, const char *something ) {
        lc = &linphonec;
        #ifdef DEBUG
          cout << "\ndebug_linphonec_display_status_: " << something << endl << flush;
        #endif
        // inform about everything but Ready
        if ( (string)"Ready" == (string)something ) {
          //display_qt4_message( something );
        } else if ( (string)something == (string)"Could not reach destination." ) {
          display_qt4_error_message( something );
          linphone_core_terminate_call( &linphonec, pending_call_sip.c_str() );
        }
                        }

                        static void
                        linphonec_display_warning ( LinphoneCore * lc, const char *something ) {
        lc = &linphonec;
        #ifdef DEBUG
          cout << "\ndebug_linphonec_display_warning_: " << something << endl << flush;
        #endif
        display_qt4_warning_message( something );
                        }

                        static void
                        linphonec_display_url ( LinphoneCore * lc, const char *something, const char *url
) {
                          lc = &linphonec;
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
        #ifdef DEBUG
          cout << "\ndebug_linphonec_display_url_: " << something << ", url: " << url << endl << flush;
        #endif
        display_qt4_message( something );
      }

                        static void
                        linphonec_call_received( LinphoneCore *lc, const char *from ) {
        lc = &linphonec;
                                #ifdef DEBUG
          cout << "\ndebug_linphonec_call_received_: from: " << from << endl << flush;
        #endif
        if ( auto_answer )  {
        #ifdef DEBUG
          cout << "\ndebug_linphonec_call_received_: Auto answered call" << endl << flush;
        #endif
                                }
                        }

                        static void
                        linphonec_prompt_for_auth( LinphoneCore *lc, const char *realm, const char
*username ) {
        lc = &linphonec;
        LinphoneAuthInfo *pending_auth;
        #ifdef DEBUG
          cout << "\ndebug_linphonec_prompt_for_auth_: realm:" << realm << ", username: " << username <<
endl << flush;
        #endif
                                if ( auth_stack.nitems + 1 > MAX_PENDING_AUTH ) {
                                        cout << "\n\nCan't accept another authentication request.\n" <<
                "Consider incrementing MAX_PENDING_AUTH macro." << endl << flush;
                                        return;
                                }
                                pending_auth = linphone_auth_info_new( username, NULL, NULL, NULL, realm
);
                                auth_stack.elem[ auth_stack.nitems++ ] = pending_auth;
        string concated = "Odebrano żądanie autoryzacji od " + (string)username + " (" + (string)realm +
") ";
        display_qt4_message( concated.c_str() );
                        }

                        static void
                        linphonec_notify_received( LinphoneCore *lc, LinphoneFriend *fid,

                  const char *from, const char *status, const char *img ) {
        lc = &linphonec;
                                // TODO: update Friend list state (unimplemented in linphonec)
        // TODO: do something with LinphoneFriend struct
        #ifdef DEBUG
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
         cout << "\ndebug_linphonec_notify_received_: From: " << from << " Status: " << status << " img: "
<< img << endl << flush;
        #endif
        string concated = "Odebrano zdarzenie od " + (string)from + " ( status:" + (string)status + ") ";
        display_qt4_message( concated.c_str() );
                        }

                        static void
                        linphonec_new_unknown_subscriber( LinphoneCore *lc, LinphoneFriend *lf, const
char *url ) {
        lc = &linphonec;
        #ifdef DEBUG
          cout << "\ndebug_linphonec_new_unknown_subscriber_: friend: " << url <<
                  " requested subscription (accept/deny is not implemented yet)" << endl << flush;
        // This means that this person wishes to be notified
                                // of your presence information (online, busy, away...).
        #endif
                        }

                        static void
                        linphonec_bye_received( LinphoneCore *lc, const char *from ) {
                                // printing this is unneeded as we'd get a "Communication ended"
                                // message trough display_status callback anyway
        lc = &linphonec;
                                #ifdef DEBUG
          cout << "\ndebug_linphonec_bye_received_: from: " << from << endl << flush;
        #endif
                        }

 // TODO: text chats should be implemented soon
 /*
                        static void
                        linphonec_text_received( LinphoneCore *lc, LinphoneChatRoom *cr, const char
*from, const char *msg) {
                                // TODO: provide mechanism for answering.. ('say' command?)
                                printf("\n\nFrom: %s: Msg: %s\n", from, msg);
                                fflush( stdout );
                        }
 */
                        void
      DSipCom::linphonec_main_loop() {
        linphone_core_iterate( &linphonec );
        if ( linphonec.call != NULL ) {
          #ifdef DEBUG
            cout << "." << flush;
          #endif
        }
                        }
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
      void
      DSipCom::reset_status_bar() {
         this->status_bar->setText( "Program nie wykonuje żadnej akcji" );
      }

//DSipCom objects
#ifdef DEBUG
  Logger
  logger( LOGGER_DSIPCOM_UI.c_str(), "debug" );
#endif

  void DSipCom::read_logs() {
    string log;
    if ( read_one_log_by_date( this->calendar->selectedDate().day(),
                          this->calendar->selectedDate().month(),
                          this->calendar->selectedDate().year(), CALL_LOG_FILE.c_str() ) == "" )
      log = "Brak logów";
    else {
      log = read_one_log_by_date( this->calendar->selectedDate().day(),
                          this->calendar->selectedDate().month(),
                          this->calendar->selectedDate().year(), CALL_LOG_FILE.c_str() );
    }
#ifdef DEBUG
  cout << "debug_read_logs_:" << log;
  cout.flush();
#endif
    raport_viewer->setPlainText( (QString)log.c_str() );
  }

//DSipCom methods
DSipCom::DSipCom( const QString& title ) {
  #ifdef DEBUG
    logger.log( "Checking HOME and DIRS" );
  #endif
    setupDIRs();
  #ifdef DEBUG
    logger.log( "Initializing UI" );
  #endif
  setupUi( this );
   // global ui encoding => utf8
   QTextCodec::setCodecForCStrings( QTextCodec::codecForName( "UTF-8" ) );
   // setting window flags
   Qt::WindowFlags flags;
   flags = Qt::Window | Qt::WindowMinimizeButtonHint | Qt::WindowStaysOnTopHint;
   setWindowFlags( flags );
   // ui settings
   setWindowTitle( title );
   // contacts list in front by default
   toolBox->setCurrentIndex( 0 );
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
   show();
   #ifdef DEBUG
    logger.log( "Initializing QT4 actions" );
   #endif
    init_actions();
   #ifdef DEBUG
    logger.log( "DSipCom initialized" );
    logger.log( "Loading User List" );
   #endif
   //save_user_list();
   user_list.reserve( 100 ); // reserve place for 100 elements
   load_user_list();

   #ifdef DEBUG
    logger.log( "Loading User Config" );
   #endif
  user_config = new USER_CONFIG;
  create_linphone_core();
  load_user_config();
  //reading logs for calendar
  read_logs();
   #ifdef DEBUG
    logger.log( "Loading Linphone, version: " + (QString)linphone_core_get_version() );
   #endif
}

DSipCom::~DSipCom() {
  // destroing main linphone core structure and friends
  linphone_core_uninit( &linphonec );
 #ifdef DEBUG
  cout << "\nDsipCom destructor." << endl;
  cout << today_log;
  cout.flush();
 #endif
  if ( today_log != "" ) {
        write_one_log_by_date( today_log,
                        this->calendar->selectedDate().day(),
                        this->calendar->selectedDate().month(),
                        this->calendar->selectedDate().year(), CALL_LOG_FILE.c_str() );
        }
}

void
DSipCom::setupDIRs() {
  // this method will check existance of main program directories and it will try to create them if they
doesn't exist
  if ( !exists( DSIP_MAIN_DIR ) ) create_directory( DSIP_MAIN_DIR );
  if ( !exists( LOGS_DIR ) ) create_directory( LOGS_DIR );
  if ( !exists( CONF_DIR ) ) create_directory( CONF_DIR );
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
  if ( !exists( ULIST_DIR ) ) create_directory( ULIST_DIR );
}

void
DSipCom::create_linphone_core() {
    #ifdef DEBUG
       logger.log( "Linphone config: " + (QString)( LINPHONE_CONFIG.c_str() ) );
       logger.log( "Initializing Linphone core logger" );
       linphone_core_enable_logs( stdout );
       TRACE_INITIALIZE( (trace_level_t)0, stdout );
    #endif
    #ifndef DEBUG
       linphone_core_disable_logs();
    #endif
    #ifdef DEBUG
      logger.log( "Linphone logger initialized" );
      logger.log( "Initializing LinPhone" );
    #endif
    // TODO: make configurable choosing ipv4/v6, IPv6 is now disabled by default.
    linphone_core_enable_ipv6( &linphonec, FALSE );
    auth_stack.nitems = 0;
    linphone_core_init ( &linphonec, &linphonec_vtable, LINPHONE_CONFIG.c_str(), NULL );
    linphone_core_enable_video( &linphonec, vcap_enabled, display_enabled );
    // CRITICAL SECTION OF DSIPCOM:
    // Creating timer with 60ms trigger, and launch it in the background thread
    // Here we going to iterate main Linphone engine.
    QTimer *timer = new QTimer( this );
    connect( timer, SIGNAL( timeout() ) , this, SLOT( linphonec_main_loop() ) );
    timer->start( 60 ); // 60ms is enough
    // char** with list of sound devices
    sound_dev_names = linphone_core_get_sound_devices( &linphonec );
    // MSlist with audio codecs list
    audio_codec_list = linphone_core_get_audio_codecs( &linphonec );
    video_codec_list = linphone_core_get_video_codecs( &linphonec );
   // linphone_core_set_audio_codecs( &linphonec, (MSList*)audio_codec_list->next );
   // linphone_core_set_video_codecs( &linphonec, (MSList*)video_codec_list );
    #ifdef DEBUG
      logger.log( "Linphone core Ready!" );
    #endif
}

void
DSipCom::save_user_list() {
// TODO: implement User Authorisation for linphone core (not required but could improve compatibility with
other linphone core
// based apps)
  LinphoneAuthInfo* temp;
  FILE* userlist_file;
  userlist_file = fopen( USER_LIST_FILE.c_str(), "wb+" );
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
  if ( userlist_file == 0 ) {
    cout << "Error writing userlist file!\nCannot continue. Check Your user access and try again." <<
endl;
    exit( 1 );
  }
  // writing header
  char user_list_header[] = "dulf2";
  fwrite( user_list_header, sizeof( user_list_header ), 1, userlist_file );
  // writing amount of users
  uint32_t user_list_size = user_list.size();
  fwrite( &user_list_size, sizeof( uint32_t ), 1, userlist_file );
  cout << "\nuser_list_size_:" << user_list_size << endl;
  // writing data
  if ( user_list_size > 0 ) {
    for (int i = 0; i < user_list.size(); i++ ) {
      char realm[255] = "";
      char username[255] = "";
      temp = linphone_auth_info_new( user_list.at( i ).username, NULL, NULL, NULL, user_list.at( i ).realm
);
      strcpy( username, user_list.at( i ).username );
      strcpy( realm, user_list.at( i ).realm );
      cout << username << " " << realm << endl;
      fflush( stdout );
      #ifdef DEBUG
        cout << "\nsave_user_list_: " << username << "@" << realm << " vs " <<
                user_list.at( i ).username << "@" << user_list.at( i ).realm << endl << flush;
      #endif
      fwrite( username, sizeof( username ), 1, userlist_file );
      fwrite( realm, sizeof( realm ), 1, userlist_file );
    }
  }
  #ifdef DEBUG
    cout << "\nsave_user_list_: amount of records written to file: " << (uint32_t)user_list_size << endl
<< flush;
  #endif
  fclose( userlist_file );
}

void
DSipCom::load_user_list() {
  // TODO: each contact on DSipCom's user list should get linphone presence info
  //void linphone_core_set_presence_info(LinphoneCore *lc,int minutes_away,const char
*contact,LinphoneOnlineStatus os);
  //linphone_core_set_presence_info( &linphonec, 0, )
  // clear user_list QVector
  this->user_list.clear(); // == .resize(0)
  // clear items on contacts list
  this->contacts_list->clear();
  // reading user_list from file
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
  uint32_t size_of_list;
  FILE* userlist_file;
  userlist_file = fopen( USER_LIST_FILE.c_str(), "rb+" );
  // checking existance of list file
  if ( userlist_file == 0 ) {
    cout << "Error reading userlist file!\nNew user_list file will be created." << endl;
    save_user_list();
    userlist_file = fopen( USER_LIST_FILE.c_str(), "rb+" );
  }
  // checking userlist file header
  char user_list_header_correct[] = "dulf2";
  char* user_list_header = new char[ sizeof( user_list_header_correct ) + 1 ];
  fread( user_list_header, sizeof( user_list_header_correct ), 1, userlist_file );
  #ifdef DEBUG
    logger.log( "Userlist file header check: " + (QString)user_list_header + " vs " +
(QString)user_list_header_correct );
  #endif
  if ( strcmp( user_list_header, user_list_header_correct ) != 0 ) {
    cout << "Error in user_list file header. (" << user_list_header << " instead of " <<
            user_list_header_correct << ") Probably I tried to read bad format user_list" <<
            " file! Delete this file, maybe it's broken or smth" << endl;
    exit( 1 );
  }
  delete[] user_list_header;
  // reading number of elements
  fread( &size_of_list, sizeof( uint32_t ), 1, userlist_file );
  // reading elements
  if ( size_of_list > 0 ) {
    char realm[255];
    char username[255]; //temp ones
    for ( uint32_t i = 0; i < size_of_list; i++ ) {
      fread( username, sizeof( username ), 1, userlist_file );
      fread( realm, sizeof( realm ), 1, userlist_file );
      LinphoneAuthInfo* temp = linphone_auth_info_new( username, "", "", "", realm ); // XXX XXX
      user_list.append( *temp );
    }
    // putting elements to user_list plus icons
    if (! user_list.empty() ) {
      for ( uint32_t i = 0; i< size_of_list; i++ ) {
        // this will set specified icon to current list element, then will set caption, and add object to
user_list
        QIcon icon1;
        icon1.addPixmap( QPixmap( QString::fromUtf8( ":/images/images/user_green.png" ) ), QIcon::Active,
QIcon::On );
        QListWidgetItem *__listItem = new QListWidgetItem( this->contacts_list );
        __listItem->setIcon( icon1 );
        __listItem->setText( QString( user_list.at( i ).username ) + QString( " : " ) +
QString( user_list.at( i ).realm ) );
      }
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
    }
  }
  fclose( userlist_file );
  // matter of security - always, one element on user list need to be choosen: ( SEGV when accessing
unchoosen element )
  this->contacts_list->setCurrentRow( 0 );
}

void
DSipCom::apply_settings_to_linphone() {
  // applying settings to linphone core:
  uint64_t port = strtol( user_config->default_port, NULL, 10 ); //conversion from char[5] to uint64_t, 10
=> decimal number sys.
  if ( ( port > 65535 ) || ( port < 1024 ) ) { // 65535 is max port, greater than 1024 cause 0...1024 are
root ports (POSIX)
    // stupid workaround..
    linphone_core_set_sip_port( &linphonec, 5060 );
    strcpy( user_config->default_port, "5060" );
  } else {
    linphone_core_set_sip_port( &linphonec, port );
  }
  #ifdef DEBUG
    cout << "\nConfig port value/ after conversion: " << user_config->default_port << "/ " << port <<
endl;
    cout << "\nSetting default port to: " << (uint64_t)linphone_core_get_sip_port( &linphonec ) << endl <<
flush;
  #endif
  linphone_core_set_inc_timeout( &linphonec, 60 ); // 60 to timeout
  linphone_core_set_firewall_policy( &linphonec, LINPHONE_POLICY_NO_FIREWALL );
  if ( user_config->use_stun_server ) {
    linphone_core_set_stun_server( &linphonec, user_config->stun_address );
    linphone_core_set_firewall_policy( &linphonec, LINPHONE_POLICY_USE_STUN );
  }
  if ( user_config->manual_firewall_address ) {
    linphone_core_set_nat_address( &linphonec, user_config->firewall_address );
    linphone_core_set_firewall_policy( &linphonec, LINPHONE_POLICY_USE_NAT_ADDRESS );
  }
  #ifdef DEBUG
    PayloadType *pt = NULL;
    for( MSList* elem = (MSList*)audio_codec_list; elem != NULL; elem = elem->next ) {
      cout << elem << endl;
      cout.flush();
    }
  #endif
  // void linphone_core_set_ring_level(LinphoneCore *lc, int level);
  linphone_core_set_ring_level( &linphonec, user_config->output_volume );
  // void linphone_core_set_play_level(LinphoneCore *lc, int level);
  linphone_core_set_play_level( &linphonec, user_config->output_volume );
  // void linphone_core_set_rec_level(LinphoneCore *lc, int level);
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
  linphone_core_set_rec_level( &linphonec, user_config->microphone_volume );
  // TODO: add option to manually choose ring sound, now user needs to type path to sound file..
  strcpy( user_config->ring_sound, "sounds/toyphone.wav" );
  linphone_core_set_ring( &linphonec, user_config->ring_sound );
  // TODO: add support for echo cancelation:
  // void linphone_core_enable_echo_cancelation(LinphoneCore *lc, bool_t val);
  linphone_core_set_ringer_device( &linphonec, user_config->out_soundcard  );
  #ifdef DEBUG
    cout << "\nSound RING OUT device: " << linphone_core_get_ringer_device( &linphonec ) << endl;
  #endif
  linphone_core_set_playback_device( &linphonec, user_config->out_soundcard );
  #ifdef DEBUG
    cout << "\nSound PLAYBACK OUT device: " << linphone_core_get_playback_device( &linphonec ) << endl;
  #endif
  linphone_core_set_capture_device( &linphonec, user_config->in_soundcard );
  #ifdef DEBUG
    cout << "\nSound CAPTURE IN device: " << linphone_core_get_capture_device( &linphonec ) << endl <<
flush;
  #endif
  linphone_core_set_guess_hostname( &linphonec, TRUE );
  linphone_core_set_download_bandwidth( &linphonec, 0 ); // bandwidth unlimited
  linphone_core_set_upload_bandwidth( &linphonec, 0 ); // same as above.
  // create proxy structure and
  // get proxy list ( not specially used but needed by core )
  proxy_list = linphone_core_get_proxy_config_list( &linphonec );
  pcfg = linphone_proxy_config_new();
  linphone_core_get_default_proxy( &linphonec, &pcfg );
}

// load_user_config() it's method which load application settings and apply them in linphone core right
after init
void
DSipCom::load_user_config() {
  FILE* config_file;
  config_file = fopen( CONFIG_FILE.c_str(), "rb+" );
  if ( config_file == 0 ) {
    cout << "Error reading user config file!\nNew user config will be created." << endl;
    save_user_config();
    config_file = fopen( CONFIG_FILE.c_str(), "rb+" );
  }
  // reading user config structure at once
  fread( user_config, sizeof( USER_CONFIG ), 1, config_file );
  fclose( config_file );
  // putting values from file to edit boxes
  this->user_name->setText( user_config->user_name );
  this->user_password->setText( user_config->user_password );
  this->user_sip_server->setText( user_config->user_sip_server );
  // FIXME: it should set properly those, now we'll set default as CONST!:
  this->out_soundcard->setCurrentIndex( 0 ); //user_config->out_soundcard );
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
  this->in_soundcard->setCurrentIndex( 0 ); //user_config->in_soundcard );
  this->recording_source->setCurrentIndex( 0 ); //user_config->recording_source );
  strcpy( user_config->out_soundcard, sound_dev_names[ 1 ] );
  strcpy( user_config->in_soundcard, sound_dev_names[ 1 ] );
  strcpy( user_config->recording_source, sound_dev_names[ 1 ] );
  this->ring_sound->setItemText( this->ring_sound->currentIndex(), user_config->ring_sound );
  this->ring_sound->setEditable( true );
  this->default_port->setText( user_config->default_port );
  this->no_firewall->setChecked( user_config->no_firewall );
  this->use_stun_server->setChecked( user_config->use_stun_server );
  this->stun_address->setText( user_config->stun_address );
  this->manual_firewall_address->setChecked( user_config->manual_firewall_address );
  this->firewall_address->setText( user_config->firewall_address );
  this->output_volume->setValue( user_config->output_volume );
  this->microphone_volume->setValue( user_config->microphone_volume );
  apply_settings_to_linphone();
}

void
DSipCom::save_user_config() {
  // getting values from main window objects
  strcpy( user_config->user_name, this->user_name->text().toUtf8() );
  strcpy( user_config->user_password, this->user_password->text().toUtf8() );
  strcpy( user_config->user_sip_server, this->user_sip_server->text().toUtf8() );
  if ( this->out_soundcard->currentIndex() == 0 ) {
    // index 0 means default sound card on dSipCom device list, but it's 1 on sound_dev_names list..
    strcpy( user_config->out_soundcard, sound_dev_names[ 1 ] );
  } else {
    strcpy( user_config->out_soundcard, sound_dev_names[ 0 ] );
  }
  if ( this->in_soundcard->currentIndex() == 0 ) {
    strcpy( user_config->in_soundcard, sound_dev_names[ 1 ] );
  } else {
    strcpy( user_config->in_soundcard, sound_dev_names[ 0 ] );
  }
  if ( this->recording_source->currentIndex() == 0 ) {
    strcpy( user_config->recording_source, sound_dev_names[ 1 ] );
  } else {
    strcpy( user_config->recording_source, sound_dev_names[ 0 ] );
  }
  strcpy( user_config->ring_sound, this->ring_sound->currentText().toUtf8() );
  strcpy( user_config->default_port, this->default_port->text().toUtf8() );
  user_config->no_firewall = this->no_firewall->isChecked();
  user_config->use_stun_server = this->use_stun_server->isChecked();
  strcpy( user_config->stun_address, this->stun_address->text().toUtf8() );
  user_config->manual_firewall_address = this->manual_firewall_address->isChecked();
  strcpy( user_config->firewall_address, this->firewall_address->text().toUtf8() );
  user_config->output_volume = this->output_volume->value();
  user_config->microphone_volume = this->microphone_volume->value();
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
  FILE* config_file;
  config_file = fopen( CONFIG_FILE.c_str(), "wb+" );
  if ( config_file == 0 ) {
    cout << "Error writing user config file!\nCannot continue. Check Your user access and try again." <<
endl << flush;
    exit( 1 );
  }
  // writing whole structure with data to config file
  fwrite( user_config, sizeof( USER_CONFIG ), 1, config_file );
  fclose( config_file );
  apply_settings_to_linphone();
}

// init_actions will init all actions and binds in application
void
DSipCom::init_actions() {
  // buttons
  QObject::connect( call_button, SIGNAL( clicked() ), this, SLOT( action_make_a_call() ));
  QObject::connect( hang_button, SIGNAL( clicked() ), this, SLOT( action_end_call() ));
  QObject::connect( dial_0, SIGNAL( clicked() ), this, SLOT( action_enter_0() ));
  QObject::connect( dial_1, SIGNAL( clicked() ), this, SLOT( action_enter_1() ));
  QObject::connect( dial_2, SIGNAL( clicked() ), this, SLOT( action_enter_2() ));
  QObject::connect( dial_3, SIGNAL( clicked() ), this, SLOT( action_enter_3() ));
  QObject::connect( dial_4, SIGNAL( clicked() ), this, SLOT( action_enter_4() ));
  QObject::connect( dial_5, SIGNAL( clicked() ), this, SLOT( action_enter_5() ));
  QObject::connect( dial_6, SIGNAL( clicked() ), this, SLOT( action_enter_6() ));
  QObject::connect( dial_7, SIGNAL( clicked() ), this, SLOT( action_enter_7() ));
  QObject::connect( dial_8, SIGNAL( clicked() ), this, SLOT( action_enter_8() ));
  QObject::connect( dial_9, SIGNAL( clicked() ), this, SLOT( action_enter_9() ));
  QObject::connect( dial_star, SIGNAL( clicked() ), this, SLOT( action_enter_star() ));
  QObject::connect( dial_hash, SIGNAL( clicked() ), this, SLOT( action_enter_hash() ));
  QObject::connect( save_config_button, SIGNAL( clicked() ), this, SLOT( action_save_user_config() ));
  QObject::connect( load_config_button, SIGNAL( clicked() ), this, SLOT( action_load_user_config() ));
  QObject::connect( save_contact_list_button, SIGNAL( clicked() ), this, SLOT( action_save_user_list() ));
  QObject::connect( load_contact_list_button, SIGNAL( clicked() ), this, SLOT( action_load_user_list() ));
  // menu bar:
  QObject::connect( action_help, SIGNAL( activated() ), this, SLOT( action_help_func() ));
  QObject::connect( action_about, SIGNAL( activated() ), this, SLOT( action_about_func() ));
  QObject::connect( action_connect_to_sip_server, SIGNAL( activated() ), this,
SLOT( action_connect_to_sip_server_func() ));
  QObject::connect( action_disconnect_from_sip_server, SIGNAL( activated() ), this,
SLOT( action_disconnect_from_sip_server_func() ));
  QObject::connect( action_add_contact_to_list, SIGNAL( activated() ), this,
SLOT( action_add_contact_func() ));
  QObject::connect( action_remove_contact_from_list, SIGNAL( activated() ), this,
SLOT( action_remove_contact_func() ));
  // calendar
  QObject::connect( calendar, SIGNAL( selectionChanged() ), this, SLOT( action_get_log_func() ));
}
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
void
DSipCom::action_get_log_func() {
  QDate selected = this->calendar->selectedDate();
  #ifdef DEBUG
    cout << endl << "Current selected day: " << selected.day() << endl;
    cout.flush();
  #endif
  read_logs();
}

void
DSipCom::action_save_user_config() {
  save_user_config();
}

void
DSipCom::action_load_user_config() {
  load_user_config();
}

// TODO: add support for void linphone_core_add_friend(LinphoneCore *lc, LinphoneFriend *fr), and
LinphoneFriend structure in place of actual two user info fields
void
DSipCom::action_load_user_list() {
  load_user_list();
}

void
DSipCom::action_save_user_list() {
  save_user_list();
}

/* numbers enterance: */
void
DSipCom::action_enter_0() {
  this->number_entry->setText( this->number_entry->text() + "0" );
}

void
DSipCom::action_enter_1() {
  this->number_entry->setText( this->number_entry->text() + "1" );
}

void
DSipCom::action_enter_2() {
  this->number_entry->setText( this->number_entry->text() + "2" );
}
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
void
DSipCom::action_enter_3() {
  this->number_entry->setText( this->number_entry->text() + "3" );
}

void
DSipCom::action_enter_4() {
  this->number_entry->setText( this->number_entry->text() + "4" );
}

void
DSipCom::action_enter_5() {
  this->number_entry->setText( this->number_entry->text() + "5" );
}

void
DSipCom::action_enter_6() {
  this->number_entry->setText( this->number_entry->text() + "6" );
}

void
DSipCom::action_enter_7() {
  this->number_entry->setText( this->number_entry->text() + "7" );
}

void
DSipCom::action_enter_8() {
  this->number_entry->setText( this->number_entry->text() + "8" );
}

void
DSipCom::action_enter_9() {
  this->number_entry->setText( this->number_entry->text() + "9" );
}

void
DSipCom::action_enter_star() {
  this->number_entry->setText( this->number_entry->text() + "*" );
}

void
DSipCom::action_enter_hash() {
  this->number_entry->setText( this->number_entry->text() + "#" );
}

void
DSipCom::action_end_call() {
  if ( linphonec.call != NULL ) {
    // section is equivalent of ruby split method:
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
    this->status_bar->setText( "Rozłączam z " + ( (QString)pending_call_sip.c_str() ).section( ':', 1 ) );
    //this->call_button->setEnabled( true );
    //this->hang_button->setEnabled( false );
  #ifdef DEBUG
    cout << "Ending call with: " << pending_call_sip.c_str() << endl;
    cout.flush();
  #endif
    linphone_core_terminate_call( &linphonec, pending_call_sip.c_str() );
    QTimer *timer = new QTimer( this );
    connect( timer, SIGNAL( timeout() ) , this, SLOT( reset_status_bar() ) );
    timer->setSingleShot ( true ); //activate only once
    timer->start( 3000 ); // 3s
    // filling raport viewer log for current day:
    raport_viewer->setPlainText( "\n" + (QString)today_log.c_str() );
  }
}

void
DSipCom::action_make_a_call() {
  // TODO: DSipCom should ask for video port. codecs should be choosen automaticly
linphone_core_set_video_port
  // TODO: void linphone_core_enable_video_preview(LinphoneCore *lc, bool_t val) - it should be "enable
video window" setting somewhere with default FALSE.
  // if we're on contacts list tab and this list isn't empty
  if ( ( ( this->contacts_list->count() != 0 ) && ( this->toolBox->currentIndex() == 0 ) ) ||
      // or number entry is at least one char long and we're on number entry page
       ( ( this->number_entry->text().length() > 0 ) && ( this->toolBox->currentIndex() == 1 ) ) ) {
          switch ( this->toolBox->currentIndex() ) {
            case 0:
              // 0 => contact list page
              this->status_bar->setText( "Dzwonię do: " +
                this->contacts_list->item( this->contacts_list->currentRow() )->text().section( ':',
1 ) ); // str == "myapp" );

               pending_call_sip = (string)"sip:" + (string)( this->contacts_list->item(
                                   this->contacts_list->currentRow() )->text().section( ':',
1 ) ).toUtf8() +

                                          (string)":" + (string)user_config->default_port;
               pending_call_sip = strip( pending_call_sip, ' ' );
                                                  #ifdef DEBUG
                                                          cout << "\ndebug_action_make_a_call_:Making new
call with: " << pending_call_sip.c_str() << endl << flush;
                                                  #endif
              break;
            case 1:
              // 1 => dialing page
              this->status_bar->setText( "Dzwonię do: " + this->number_entry->text() );
                                                          // SIP address format is
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
"sip:ADDR_OR_NUMBER_HERE:port"
              pending_call_sip = (string)"sip:" + (string)( this->number_entry->text() ).toUtf8() +

                                 (string)":" + (string)user_config->default_port;
              pending_call_sip = strip( pending_call_sip, ' ' );
                                                  #ifdef DEBUG
                                                          cout << "Making new call with: " <<
pending_call_sip.c_str() << endl << flush;
                                                  #endif
              break;
          }

          if ( linphonec.call != NULL ) linphone_core_accept_call( &linphonec, pending_call_sip.c_str() );
          else linphone_core_invite( &linphonec, pending_call_sip.c_str() ); // to invite
//   this->call_button->setEnabled( false );
      this->hang_button->setEnabled( true );
  } else {
      this->toolBox->setCurrentIndex( 0 );
  }
}

void
DSipCom::action_help_func() {
  #ifdef DEBUG
    logger.log( "Visited -> Help" );
  #endif
  // TODO: add own help dialog instead of QMessageBox
  QMessageBox::information( this, MAIN_WINDOW_TITLE.c_str(), " Brak pliku pomocy [ niezainicjowano ] ");
}

void
DSipCom::action_about_func() {
  #ifdef DEBUG
    logger.log( "Visited -> about dialog!" );
  #endif
  new AboutBox();
}

void
DSipCom::action_connect_to_sip_server_func() {
  #ifdef DEBUG
    logger.log( "Trying to connect to server" );
  #endif
    if ( strcmp( user_config->user_sip_server, "" ) == 0 ) {
      QMessageBox::information( this, MAIN_WINDOW_TITLE.c_str(), " Proszę podać w preferencjach
użytkownika nazwę \
          serwera SIP proxy i zapisać ustawienia! " );
    } else if ( strcmp( user_config->user_password, "" ) == 0 ) {
      QMessageBox::information( this, MAIN_WINDOW_TITLE.c_str(), " Proszę podać w preferencjach hasło SIP
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
użytkownika i\
          zapisać ustawienia! " );
    } else if ( strcmp( user_config->user_name, "" ) == 0 ) {
      QMessageBox::information( this, MAIN_WINDOW_TITLE.c_str(), " Proszę podać w preferencjach nazwę
użytkownika i \
          zapisać ustawienia! " );
    } else {
      QMessageBox::information( this, MAIN_WINDOW_TITLE.c_str(), " Połączono z serwerem: " +
(QString)user_config->user_sip_server );
      // all required settings are ok
      #ifdef DEBUG
        logger.log( "All required config data is OK!" );
      #endif
      linphone_proxy_config_set_server_addr( pcfg, user_config->user_name );
      linphone_proxy_config_set_identity( pcfg, user_config->user_sip_server );
      linphone_core_set_default_proxy( &linphonec, pcfg ); // apply proxy config as default
    }
}

void
DSipCom::action_disconnect_from_sip_server_func() {
  QMessageBox::information( this, MAIN_WINDOW_TITLE.c_str(), " Rozłączono z serwerem: " + (QString)this-
>user_config->user_sip_server );
        #ifdef DEBUG
                logger.log( "Trying to disconnect from server" );
        #endif
}

void
DSipCom::action_add_contact_func() {
  //creating new window with parent of current one
  dialog = new AddContactWindow( this );
  //switching to contacts list view
  toolBox->setCurrentIndex( 0 );
  //moving all main window content down
  dialog->setGeometry( toolBox->x(), toolBox->y() + 20, toolBox->width(), toolBox->height() + 20 );
  toolBox->setGeometry( toolBox->x(), toolBox->y() + 220, toolBox->width(), toolBox->height() + 220 );
  status_box->setGeometry( status_box->x(), status_box->y() + 220, status_box->width(), status_box-
>height() + 220 );
  dialog->show();
}

void
DSipCom::action_remove_contact_func() {
  if ( ( toolBox->currentIndex() == 0 ) && ( this->contacts_list->count() > 0 ) ) {
    // and from user_list QVector
        #ifdef DEBUG
                cout << "Removed contact with index: " << this->contacts_list->currentRow() << endl;
        #endif
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
    this->user_list.remove( this->contacts_list->currentRow() );
    // delete item from list
    delete this->contacts_list->item( this->contacts_list->currentRow() );
        #ifdef DEBUG
                cout << "Remove contact func contacts list: " << this->contacts_list->count() << endl;
                cout << "Remove contact func list size: " << user_list.size() << endl << flush;
        #endif
  } else {
        #ifdef DEBUG
                cout << "\nNo elements on list." << endl << flush;
        #endif
  }
}

AddContactWindow::AddContactWindow( QWidget *parent ) {
  setupUi( this );
  init_actions();
  // we neet to tell child widget that it's parent is main window
  setParent( parent );
}

AddContactWindow::~AddContactWindow() {
}

void
AddContactWindow::init_actions() {
  // buttons
  QObject::connect( add_button, SIGNAL( clicked() ), this, SLOT( action_done() ));
  QObject::connect( cancel_button, SIGNAL( clicked() ), this, SLOT( action_cancel() ));
}

void
AddContactWindow::action_done() {
  // finding parent
  LinphoneAuthInfo* temp = new LinphoneAuthInfo;
  char username[255];
  char realm[255];
  DSipCom *object = ( (DSipCom*)this->parent() );
  // adding lineedit content from dialog on contact list
  if ( ( contact_name->text().length() > 0 ) && ( contact_sip_address->text().length() > 0 ) ) {
    QIcon icon1;
    icon1.addPixmap( QPixmap( QString::fromUtf8( ":/images/images/user_green.png" ) ), QIcon::Active,
QIcon::On );
    // after setting icon, we'll bind it to an item, then update text elements
    QListWidgetItem *__listItem = new QListWidgetItem( object->contacts_list );
    __listItem->setIcon( icon1 );
    __listItem->setText( this->contact_name->text() + QString( " : " ) + this->contact_sip_address->text()
);
    // marking last element ( just added one )
```

Listing 12. Główny moduł aplikacji. Nazwa pliku: dsipcom_ui.cpp

```
    // creating new user list element and appending it to user_list object
    strcpy( username, this->contact_name->text().toUtf8() ); //.toUtf8();
    strcpy( realm, this->contact_sip_address->text().toUtf8() );
    #ifdef DEBUG
      cout << "\ndebug_action_done_: " << "UN: " << username << ", RL: " << realm << endl;
    #endif
    temp = linphone_auth_info_new( username, NULL, NULL, NULL, realm );
    #ifdef DEBUG
      cout << "\ndebug_action_done_: " << "TUN: " << temp->username << ", TRL: " << temp->realm << endl;
    #endif
    // TODO: only for dsipcom local user: strcpy( temp->passwd, "password" );
    object->user_list.append( *temp );
    #ifdef DEBUG
      cout << "\nLast username on list: " << object->user_list.last().username << endl << flush;
    #endif
    //delete temp;
    object->toolBox->setGeometry( object->toolBox->x(), object->toolBox->y() - 220, object->toolBox-
>width(), object->toolBox->height() - 220 );
    object->status_box->setGeometry( object->status_box->x(), object->status_box->y() - 220, object-
>status_box->width(), object->status_box->height() - 220 );
    this->close();
  }
}

void
AddContactWindow::action_cancel() {
  //object will be object pointing to parent window
  DSipCom *object = ( (DSipCom*)this->parent() );
  // moving all parent elements back up
  object->toolBox->setGeometry( object->toolBox->x(), object->toolBox->y() - 220, object->toolBox-
>width(), object->toolBox->height() - 220 );
  object->status_box->setGeometry( object->status_box->x(), object->status_box->y() - 220, object-
>status_box->width(), object->status_box->height() - 220 );
  close();
}

AboutBox::AboutBox() {
  setupUi( this );
  version_label->setText( DSIPCOM_VERSION.c_str() );
  show();
}

AboutBox::~AboutBox() {
  #ifdef DEBUG
    cout << "AboutBox destructor." << endl << flush;
  #endif
}
```

8. Podsumowanie

Wynikiem mojej pracy jest aplikacja „*dSipCom”*, która przy nieznacznych przeróbkach kodu, mogłaby znaleźć komercyjne zastosowanie. Jest bardzo mało wymagająca na zasoby systemu komputerowego. Do prawidłowego działania wystarczy już procesor klasy *Pentium III* i 64MiB pamięci operacyjnej, a do nawiązywania połączeń łącze o przepustowości 56Kibit/s (połączenie modemowe). W chwili pisania tej pracy, wersja oprogramowania dSipCom to ***0.6.6b***. Po zakończeniu pracy dyplomowej, aplikacja będzie nadal dostępna na serwisie GitHub (jako publiczne repozytorium), pod adresem: http://github.com/dmilith/dsipcom/tree/master.

7. Załączniki i dodatki.

Zawartość krążka CD dołączonego do tej pracy przedstawiona jest w tabeli 2.

Tabela 2. Zawartość krążka CD dołączonego do pracy.

debian_amd64_lenny_apt_repository	Dodatkowe dowiązanie symboliczne do folderu z repozytorium Debiana Lenny w wersji dla architektur amd64.
Debian_amd64_lenny_precompiled_bin	Prekompilowana aplikacja dSipCom, skompilowana dynamicznie pod systemem Debian Lenny dla architektury amd64. Wymaga do uruchomienia bibliotek Qt4 oraz prekompilowanych bibliotek dostarczanych w dSipCom'owym repozytorium Debiana, które jest dołączone do tego CD.
dsipcom_git_repository	Pełny klon repozytorium GIT'a, zawierająca:
dsipcom_git_repository/docs	Folder zawierający pełną dokumentację projektu oraz komponenty (diagramy w postaci plików graficznych i wektorowych, oraz zrzuty ekranu). Praca podzielona jest na rozdziały znajdujące się w oddzielnych plikach. W podfolderze ***UML/*** znajduje się projekt Netbeans z diagramami UML.
dsipcom_git_repository/sounds	Folder z sygnałami dźwiękowymi wykorzystywanymi w programie.
Dsipcom_git_repository/src	Folder zawierający pełny kod źródłowy aplikacji oraz skrypty pomocnicze w kompilacji aplikacji (Make_all, Make_clean, Make_all_tests). Dodatkowo w podfolderze ***tests/*** znajdują się źródła testów, a w ***images/*** zbiór plików graficznych używanych w aplikacji. W podfolderze ***external/*** znajduje się repozytorium Debiana Lenny dla architektury amd64 z prekompilowanymi bibliotekami wymaganymi do skompilowania i uruchomienia aplikacji.
dsipCom_git_repository/nbproject	Folder projektu Netbeans 6.1.

8. Bibliografia i Netografia

8.1 Bibliografia

1. Bruce Eckel „Thinking in C++" 2nd edition , Patience Hall Inc, 2000.
2. Jerzy Grebosz „Symfonia C++ Standard", Editions 2000 Kraków, 2005.
3. Hal Fulton „The Ruby Way" 2nd edition, Addison-Wesley Professional, 2006.

8.2 Netografia

1. Wikipedia i źródła pokrewne w wersji angielskiej i polskiej: http://en.wikipedia.org/ oraz http://pl.wikipedia.org/
2. RFC 3261, czyli specyfikacja standardu SIP: http://tools.ietf.org/html/3261
3. Strona domowa twórcy protokołu SIP: http://www.cs.columbia.edu/sip/

9. Słowniczek pojęć.

Makefile – to plik skryptu kompilacji dla programu make (autotools).

Link symboliczny – Wskazuje on, odwołując się za pomocą nazwy, na dowolny inny plik lub katalog (który może nawet w danej chwili nie istnieć). (tylko w systemach plików zgodnych z POSIX, np. ext3, xfs itd.)

Node – Gałąź – Tutaj odbiorca lub nadawca połączenia SIP.

RSS – Resident Set Size – część (porcja) pamięci procesu, która rezyduje w pamięci RAM (POSIX).

SIGSEGV – sygnał systemów POSIX (tutaj systemu Linux), mający zazwyczaj kod 11: oznaczający błąd segmentacji.

STL – Standard Template Library – czyli standardowa biblioteka szablonów C++ dostarczana z kompilatorem, jest częścią standardu C++.

IDE – Zintegrowane środowisko developerskie

WYSIWYG – Od Angielskiego „What You See Is What You Get".

parser – analizator składniowy, czyli program/ narzędzie/ biblioteka mająca na celu dokonanie analizy poprawności gramatycznej danych wejściowych na podstawie określonych reguł i struktur (zazwyczaj konkretnego języka)

GPL – Gnu Public License

LGPL – Limited Gnu Public License

UNIX – system operacyjny napisany w 1969 roku, stał się wzorem dla twórców innych systemów takich jak np. Linux, BSD czy Mac OSX

GUI – graficzny interfejs użytkownika, oferuje możliwość przedstawienia programu w postaci graficznej (okienkowej)

portowalność – możliwość uruchamiania/ pisania aplikacji jednakowo działających/ kompilowalnych, niezależnie od systemu operacyjnego

adres nieroutowalny – adres IP który jest pomijany przy routingu w sieci Internet (np. sieci 192.168.0.0 czy 10.0.0.0), mający zastosowanie w tworzeniu sieci lokalnych LAN.