Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Katedra Informatyki

Praca dyplomowa

Temat pracy:

Wieloprotokołowy komunikator Internetowy dla systemu JBoss Portal, zrealizowany z wykorzystaniem technologii AJAX.

Autor: Daniel Para

Rodzaj pracy: Inżynierska

Promotor: dr inż. Dominik Radziszowski

Kraków, 2008

Spis treści

Indeks ilustracji

1. Wstęp

Internet jest obecnie najbardziej rozpowszechnionym i rozległym medium komunikacji, które ma praktycznie niczym nieograniczony zasięg działania oraz różnorodną funkcjonalność. Jako nowoczesny środek komunikacji, pokonuje bariery geograficzne, zapewnia tani i szybki dostęp do informacji w niemal wszystkich dziedzinach działalności człowieka, zwiększa uczestnictwo społeczności lokalnych w życiu publicznym. Wśród najpopularniejszych usług wyróżnić możemy: strony internetowe, chaty, fora dyskusyjne, telefonię internetową oraz komunikatory internetowe (ang. *instant messaging (IM)* – komunikacja natychmiastowa).

Komunikator internetowy to program, który umożliwia bezpośrednie komunikowanie się poprzez Internet lub sieć lokalną. Komunikacja w przypadku IM prowadzona jest w czasie rzeczywistym, a oparta może być na wiadomościach tekstowych, głosowych lub przekazach multimedialnych.

Komunikatory wykorzystywane są zarówno przez prywatnych użytkowników do celów towarzyskich, rozrywkowych, jak i przez wielkie międzynarodowe korporacje do komunikacji biznesowej.

Niniejsza praca inżynierska ma na celu stworzenie wieloprotokołowego komunikatora internetowego, dostępnego z poziomu przeglądarki internetowej. Tak sprecyzowany temat może być interesujący z kilku powodów. Jednym z nich jest utworzenie w pełni interaktywnego interfejsu użytkownika, który spotykany jest w aplikacjach stacjonarnych. Drugim, przedstawienie wspólnego abstrakcyjnego modelu opisującego i grupującego wybrane implementacje komunikatorów. Ostatnim zaś sposób połączenia tych dwóch elementów w jeden w postaci portletu i uruchomienia go w systemie JBoss Portal. W rezultacie użytkownik w bardzo prosty sposób, mając tylko przeglądarkę internetową może korzystać w dowolnym momencie z potrzebnego mu komunikatora.

1.1 Cel i zakres pracy

Celem niniejszej pracy dyplomowej jest stworzenie wieloprotokołowego komunikatora Internetowego działającego w środowisku portletowym dla systemu JBoss Portal. Aplikacja ma zapewnić możliwość komunikacji za pomocą różnych technologicznie komunikatorów, takich jak Gadu-Gadu, Jabber, GoogleTalk czy Windows Live Messenger, a także ma dostępna być poprzez przeglądarkę internetową.

1.2 Struktura pracy

W rozdziale pierwszym zawarto wstęp, cel i zakres oraz strukturę pracy. Rozdział drugi opisuje historię, model działania oraz najbardziej znane protokoły komunikatorów internetowych. W kolejnym trzecim rozdziale zawarto specyfikację wymagań projektowanej aplikacji. Znajduje się tam cel projektu oraz wymagania funkcjonalne i niefunkcjonalne. Rozdział czwarty zawiera analizę możliwych rozwiązań a także wykorzystane narzędzia, biblioteki i technologie. W rozdziale piątym została zamieszczona architektura tworzonej aplikacji, a także ważniejsze zagadnienia fazy projektowania i implementacji. Kolejny, szósty rozdział, charakteryzuje sposoby weryfikacji, rozwiązania oraz testy. Następnie dokonano podsumowania rozważań i przedstawiono listę literatury. Na końcu pracy umieszczono dodatek A, który zawiera niezbędne informacje do instalacji i uruchomienia aplikacji, listę kont testowych oraz słownik akronimów.

2. Zasada działania komunikatorów internetowych

W rozdziale tym przedstawiona jest krótka historia komunikatorów internetowych, model działania oraz najbardziej powszechne protokoły wykorzystywane przy tworzeniu komunikatorów internetowych.

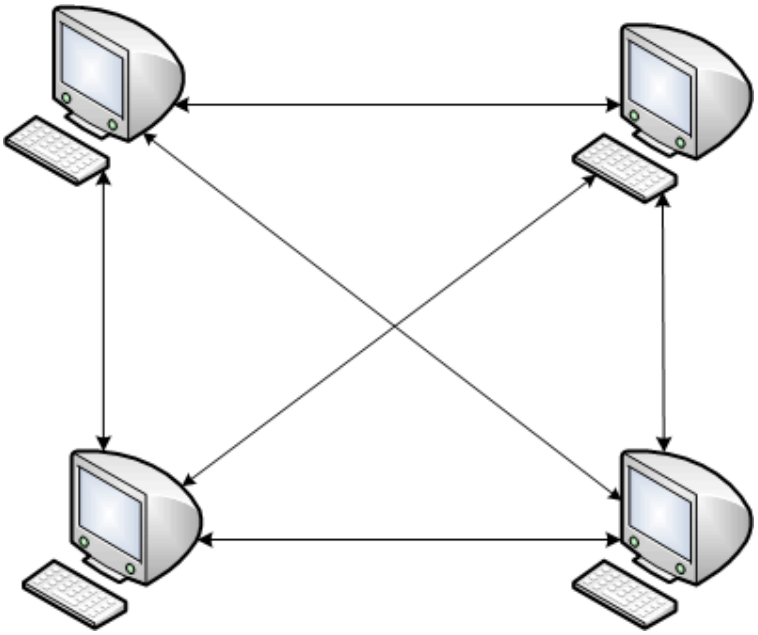
2.1 Czym jest Instant Messaging?

Instant Messaging (Komunikacja natychmiastowa) – jest to forma wymiany komunikatów tekstowych pomiędzy dwójką lub większą liczbą osób, poprzez Internet lub sieć wewnętrzną danej instytucji. Podstawową różnicą między tą formą komunikacji a pocztą elektroniczną jest fakt, że oprócz przesyłania tekstu wiadomości, użytkownik posiada także informację o statusie dostępności pozostałych użytkowników, określającą chęć i zdolność osoby do przyjęcia rozmowy.

2.2 Historia

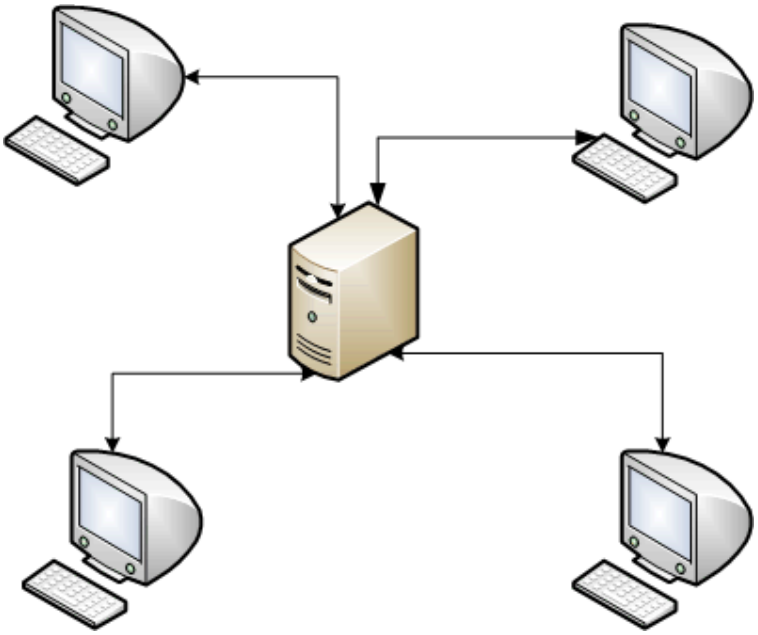
Pierwsze komunikatory, które wyprzedzały erę Internetu i powstawały w połowie lat 60-tych, wywodziły się z systemów obsługiwanych przez wielu zalogowanych jednocześnie użytkowników, tj. CTSS [5] oraz Multics [6]. Systemy te były pierwszymi systemami operacyjnymi z podziałem czasu i zyskały bardzo dużą popularność w tamtym okresie. Komunikacja pomiędzy użytkownikami następowała w obrębie jednego systemu operacyjnego. Rozwój sieci, a także protokołów sieciowych umożliwił powstanie kolejnych odmian komunikatorów. Należy tu wyróżnić dwa typy komunikacji:

- nie wymagającego zalogowanie się do serwera, model peer-to-peer [7] np.: talk [8], ntalk [8]
- wymagającego zalogowania się do serwera, model klient-serwer [9] np.: talker [10] lub do niedawna jeszcze bardzo popularny IRC [11]



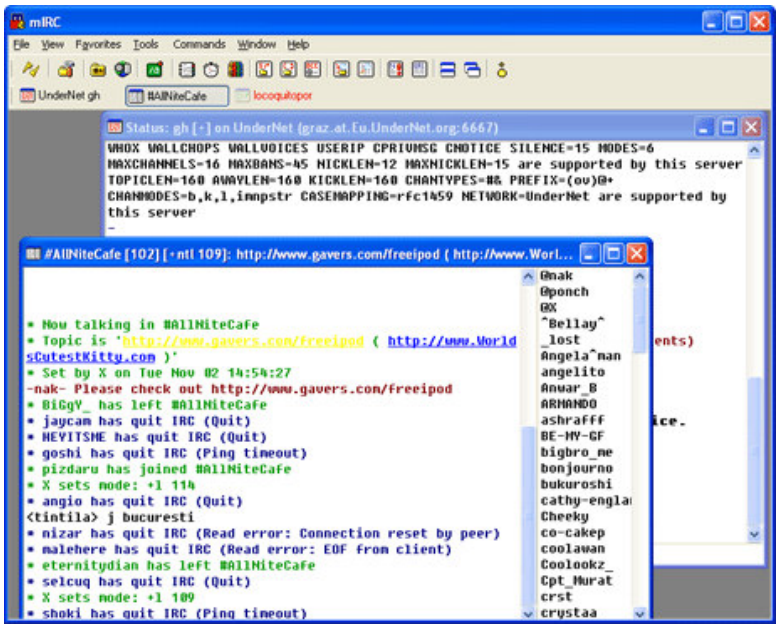
Rysunek 1 – Model komunikacji peer-to-peer

Model komunikacji peer-to-peer, czyli równy z równym, gwarantuje obydwu stronom równe prawa. W tego typu sieciach komputer może pełnić zarówno rolę serwera jak i klienta, stąd do zapewnienia komunikacji nie jest potrzebny jeden centralny serwer.



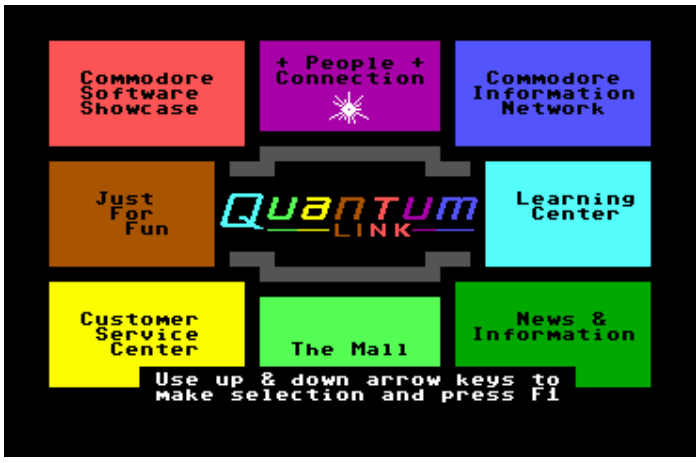
Rysunek 2 – Model komunikacji klient - serwer

W modelu klient-serwer mamy do czynienia z rozdzieleniem funkcjonalności pomiędzy serwer – dostawcę usług, a klientów – odbiorców usług. Klienci w celu zaspokojenia swoich potrzeb komunikują się za pomocą żądań wysyłanych do serwera.



Rysunek 3 – mIRC klient IRC dla Windows [12]

W drugiej połowie lat 80-tych, firma Quantum Link, później wcielona do America Online, oferowała możliwość przesyłania wiadomości pomiędzy swoimi abonentami, użytkownikami komputerów klasy m.in. Commodore 64.



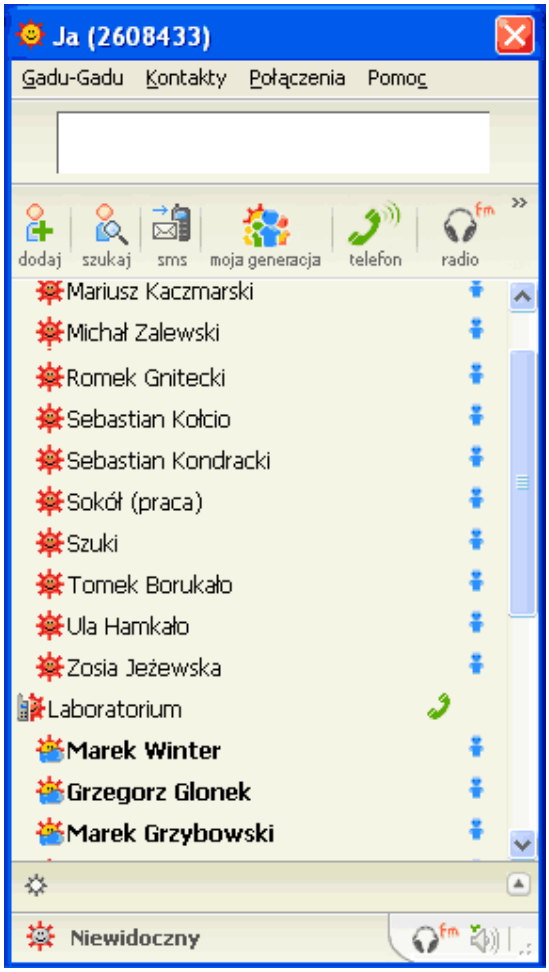
Rysunek 4 – Usługa sieciowa Quantum Link [13]

Pierwszym z nowoczesnych, bazujących na graficznym interfejsie użytkownika, działającym do dziś komunikatorem internetowym jest ICQ (ang. I seek you), wydany w listopadzie 1996 roku. Do czerwca 1997 roku na serwerach firmy Mirabilit właściciela ICQ zarejestrowało się 850 tys. użytkowników, a klient ten stał się największym na świecie komunikatorem internetowym. W czerwcu 1998 roku, firma Mirabilis wraz z ICQ została wykupiona za kwotę 287 milionów dolarów przez America Online. Mimo wszystko AOL stworzyła swój własny komunikator internetowy „AOL Instant Messenger", w skrócie AIM. W tym samym czasie inne firmy Microsoft oraz Yahoo! stworzyły własne aplikacje, odpowiednio MSN Messenger oraz Yahoo! Messenger. W dużej większości komunikatory zostały napisane aby przynosić zyski dla firm je tworzących i dlatego nie jest możliwa komunikacja pomiędzy nimi. Wiele z nich posiada swój własny protokół lub utworzone są na bazie kilku ogólnodostępnych protokołów. W roku 2000 powstał bazujący na otwartych standardach i otwartym kodzie protokół Jabber. Serwery Jabbera, dzięki wykorzystaniu tzw. transportów, umożliwiają komunikację z innymi komunikatorami, używającymi odmiennych protokołów. Na podstawie rozwiązań Jabbera powstał protokół XMPP, który szerzej opisany zostanie w dalszych rozdziałach.

Obecnie ukazujące się komunikatory udostępniają coraz więcej dodatkowych usług, takich jak wideokonferencje, telefonie VoIP, udostępnianie pulpitu, IP radio, bramki SMS, IPTV, gry oraz wiele innych dodatków mających na celu przyciągnięcie uwagi potencjalnego użytkownika.

Rysunek 5 – Komunikator ICQ

Z pośród wielu komunikatorów internetowych na uwagę zasługuje polski komunikator Gadu-Gadu. Pierwsze wydanie ukazało się 15 sierpnia 2000 roku i opierało się na rozwiązaniu zaproponowanym przez twórców ICQ. Szybko zyskało bardzo dużą popularność w Polsce i cieszy się nią do dziś. Autorem Gadu-Gadu jest Łukasz Foltyn, zaś obecnie komunikator należy do spółki Gadu-Gadu S.A. i posiada 6 milionów aktywnych użytkowników (stan czerwiec 2008), co w skali kraju jest zdecydowaną większością.



Rysunek 6 – Komunikator Gadu-Gadu

2.3 Model działania komunikatorów internetowych

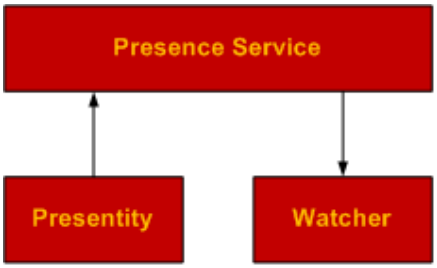
W 1986 roku powstała organizacja IETF (ang. Internet Engineering Task Force), która zajmuje się ustanawianiem standardów technicznych i organizacyjnych. Opublikowała ona w lutym 2000 roku dokumenty RFC 2778 [3] oraz RFC 2779 [4]. Znajduje się w nich definicja abstrakcyjnego modelu dwóch podstawowych usług dostarczanych przez komunikatory internetowe jakimi są:

- **Presence Service** – usługa informująca o obecności użytkownika,
- **Instant Messaging Service** – usługa odpowiedzialna za dostarczanie wiadomości.

Dokumenty te nie definiują żadnego standardu, lecz mają na celu wprowadzenie terminologii będącej podstawą do dalszych rozważań przy tworzeniu konkretnych implementacji protokołów. Wprowadzone terminy mogą nie występować w rzeczywistej implementacji protokołu lub mogą stanowić jeden wspólny element.

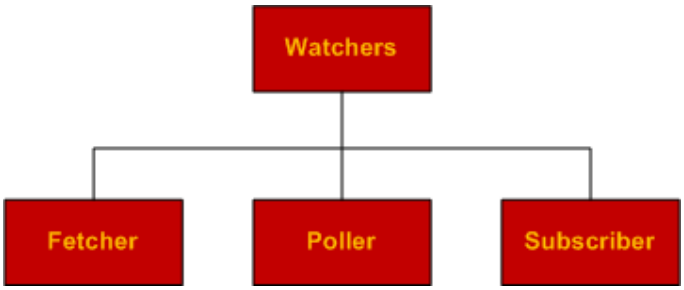
2.3.1 Presence service – usługa obecności

Usługa obecności zakłada istnienie dwóch typów klientów: dostarczycieli informacji na temat obecności ang. *presentities* oraz odbiorców tej wiadomości ang. *watchers*.



Rysunek 7 – Usługa obecności

Druga grupa dodatkowo podzielona jest na trzy podtypy:
- ang. *fetcher* – umożliwia wysyłanie żądań do usługi w celu zdobycia informacji o obecności
- ang. *pooler* – umożliwia wysyłanie żądań do usługi w pewnych odstępach czasu
- ang. *subscriber* – otrzymuje z usługi powiadomienia o zmianach obecności



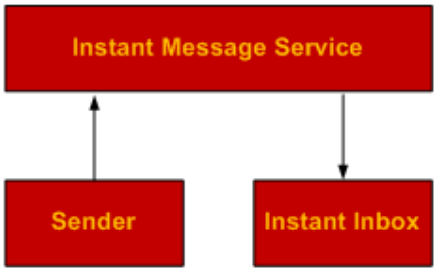
Rysunek 8 – Podtypy Watchers

Protokół obecności definiuje sposób interakcji pomiędzy usługą dostępności a jej klientami, czyli dostarczycielami oraz odbiorcami. Informacja o dostępności użytkownika komunikatora transportowana jest za pomocą tego protokołu. Na istotną uwagę zasługują także prawa dostępu do zasobów, tzw. *access rules*. Funkcjonalność ta ma na celu określenie sposobu, w jaki informacja o osiągalności użytkownika ma być dostępna dla obserwatorów.

Komunikator powinien zapewniać możliwość ukrywania statusu swojej dostępności przed innymi użytkownikami.

2.3.2 Instant message service – usługa dostarczania wiadomości

Usługa dostarczania wiadomości zakłada istnienie dwóch typów klientów: nadawców (z ang. *senders)* oraz skrzynek (z ang. *instant inboxes)* – odbiorców wiadomości. Każda wysłana wiadomość oprócz treści posiada także adres skrzynki. Usługa po otrzymaniu wiadomości, ma za zadanie dostarczenie jej do odpowiedniej skrzynki.



Rysunek 9 – Usługa dostarczania wiadomości

Protokół wysyłania wiadomości definiuje sposób interakcji pomiędzy usługą wysyłania wiadomości, nadawcami oraz odbiorcami w postaci skrzynek. Wiadomość do użytkownika transportowana jest za pomocą tego protokołu. Protokół powinien określać zasady dostarczania wiadomości, tzw. *delivery rules,* czyli sposób w jaki usługa dostarcza je do skrzynek. Komunikator powinien udostępniać filtrowanie wiadomości, np. ze względu na typ wiadomości: wiadomość systemowa, wiadomość od innego użytkownika.

2.4 Protokoły komunikatorów internetowych

Obecnie na rynku komunikatorów internetowych nie istnieje jeden najlepszy i używany przez wszystkich producentów protokół zapewniający zarówno dostarczanie wiadomości, jak i usługę dostępności. Każdy z producentów stara się tak udoskonalać swój własny protokół, aby ciągle być konkurencyjnym na rynku, a tym zyskać sobie jak największą rzeszę użytkowników. Tworzone protokoły są przeważnie mieszanką protokołu SIP (Session Initiation Protocol) [14] oraz SIMPLE (SIP for IM and Presence Leveraging Extensions) [15] w stosunku do konkurencyjnego protokołu XMPP (Extensible Messaging and Presence Protocol) [16].

2.4.1 Protokół XMPP (Extensible Messaging and Presence Protocol)

Protokół XMPP[16] w przeciwieństwie do większości tego typu protokołów jest otwartym standardem. Oznacza to, że specyfikacja protokołu jest opublikowana i dostępna dla wszystkich bezpłatnie. Wszelkie prawa autorskie i patenty są nieodwołalnie udostępnione bez opłat, a sposób wykorzystania protokołu nie posiada żadnych ograniczeń. Podwalinami pod przyszłą specyfikację protokołu były prace nad komunikatorem Jabber, zapoczątkowane w 1999 roku przez Jeremiego Millera. Na potrzeby sformalizowania protokołu, organizacja IETF utworzyła specjalną grupę pod nazwą XMPP Working Group, dzięki której w październiku 2004 roku opublikowano cztery dokumenty definiujące ten standard:

- RFC 3920: XMPP Core [16] – opisuje główne cechy protokołu a także sposób dostarczania niemalże w czasie rzeczywistym strumienia danych w postaci XML pomiędzy dwoma punktami końcowymi sieci.

- RFC 3921: XMPP Instant Messaging and Presence Protocol [17] – rozszerza podstawowe cechy o sposób w jaki protokół XMPP zapewnia usługę dostarczania wiadomości (Instant Messaging) oraz usługę dostarczającą informację o dostępności użytkowników (Presence Protocol).

- RFC 3922: Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM) [18] – opisuje sposób tworzenia bram pomiędzy protokołem XMPP a innymi nie związanymi z XMPP protokołami.

Rysunek 10 – Usługa XMPP-CPIM

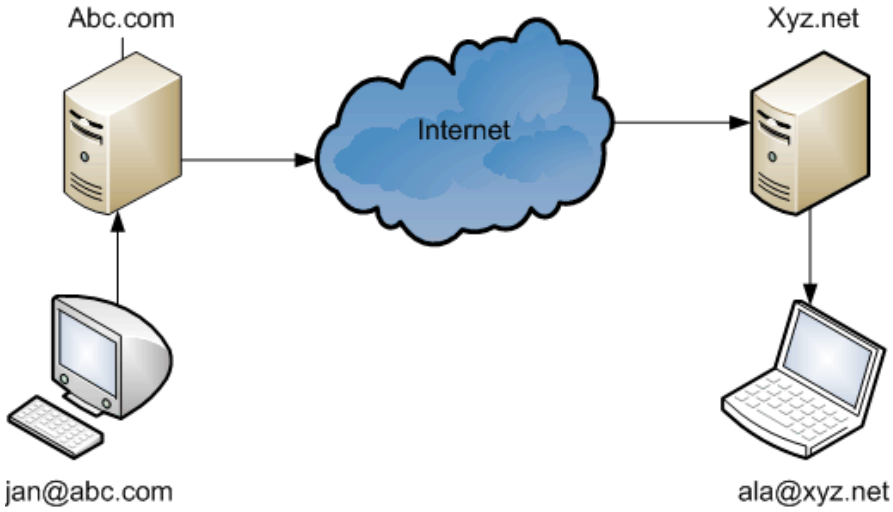
- RFC 3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP) [19] - opisuje m.in. sposób szyfrowania przesyłanych widomości, informacji o obecności, obsługi błędów oraz sposoby zapewniania bezpieczeństwa przy komunikacji przez bramy XMPP-CPIM.

Zasada działania XMPP

Przyjęto, że pierwszy użytkownik Jan posiada konto na serwerze abc.com swojej firmy o identyfikatorze jan@abc.com, drugi użytkownik, Ala, posiada konto na serwerze xyz.net o identyfikatorze ala@xyz.net. Ich identyfikatory wyglądają dokładnie tak samo, jak adresy e-mail, co umożliwia ich łatwe zapamiętanie.

Komunikacja pomiędzy użytkownikami wygląda następująco:

1. Jan loguje się do swojego serwera abc.com.
2. Jan wysyła wiadomość zaadresowaną do ala@xyz.net.
3. Wiadomość zostaje przechwycona przez serwer abc.com.
4. Serwer abc.com otwiera połączenie z serwerem xyz.net jeśli takowe jeszcze nie jest nawiązane.
5. W przypadku gdy administratorzy serwerów nie zablokowali komunikacji pomiędzy serwerem abc.com a xyz.net. Wiadomość Jana jest przesyłana do serwera xyz.net.
5. Serwer xyz.net zauważa, że wiadomość którą otrzymał zaadresowana jest do użytkownika Ala znajdującego się na tym serwerze. Serwer przesyła wiadomość do klienta uruchomionego na komputerze Ali.



Rysunek 11 – Przykładowa komunikacja za pomocą protokołu XMPP

Podczas takiej prostej wymiany wiadomości wykorzystywana jest duża liczba różnorodnych elementów. Różne mogą być aplikacje klienckie, którymi posługują się użytkownicy. Aplikacje te mogą pracować na różnych systemach operacyjnych, a także różnych maszynach. Odmienne mogą być także kanały komunikacji: pomiędzy klientami a serwerami, a także serwerami.

Cechy protokołu XMPP

Otwarty protokół

Z uwagi na to, iż protokół XMPP jest otwartym protokołem, bardzo szybo zyskał uznanie wśród użytkowników Internetu, a z tym rozrósł się. Powstało bardzo wiele nowych klientów, a także serwerów świadczących usługi bazujące na tym protokole.

Dane w formacie XML

XML jest integralną częścią protokołu, gdyż komunikacja następuje w postaci strumienia danych w formacie XML. Dane przesyłane są za pomocą małych porcji tzw. "stanzas" np.:

- przykładowa wiadomość

```
<message from='Jan@ABC.com' to='Ala@XYZ.net'>
     <body>Gdzie jesteś, Jan?</body>
</message>
```

- status użytkownika

```
<presence xml:lang='en'>
  <show>dnd</show>
  <status>Przykładowy status</status>
</presence>
```

Sposób adresowania

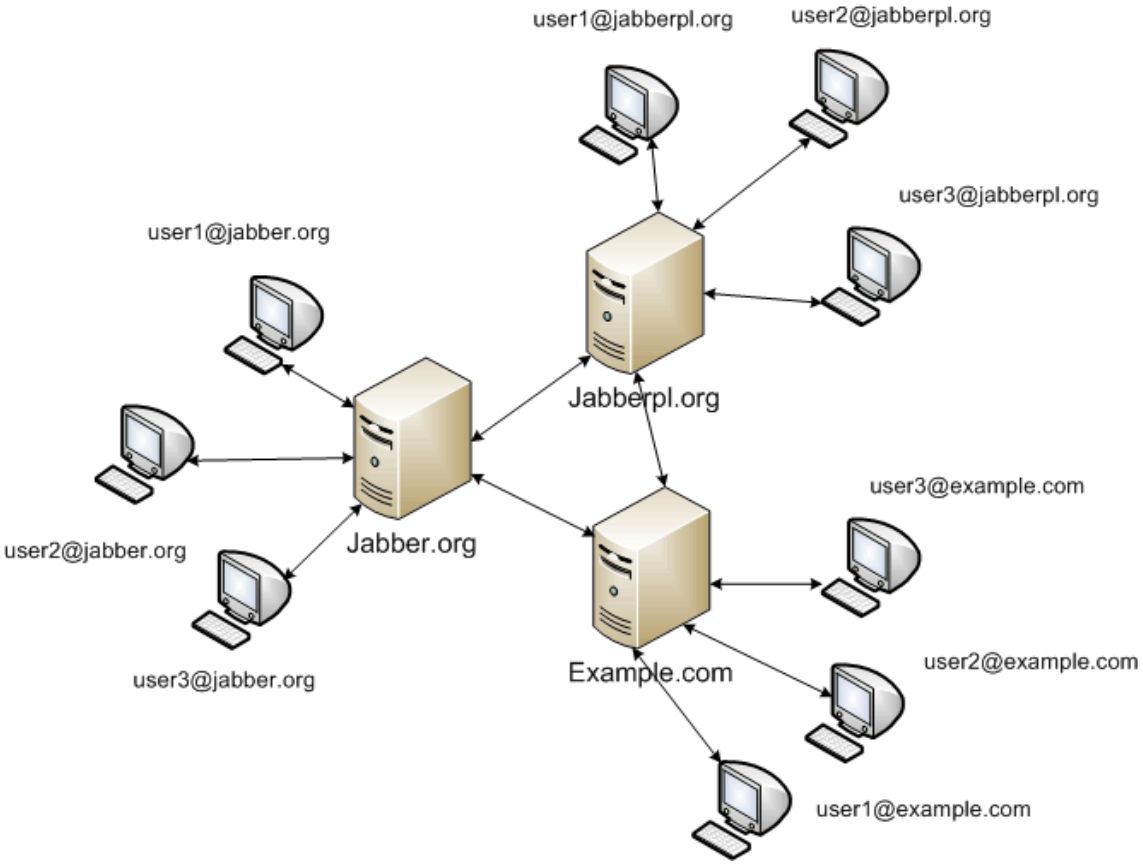
W sieci opartej na protokole XMPP wiele różnych elementów musi ze sobą współpracować. Każdy użytkownik sieci posiada swój identyfikator tzw. JID który wyraża przynależność do danego serwera oraz jednocześnie informacje o routing-u. Każdy JID identyfikuje jednoznacznie użytkownika i posiada następujący format:

```
[node]@domain[/resource] czyli [węzeł]@domena[/zasób]
```

Domena jest głównym identyfikatorem, gdyż reprezentuje adres serwera na którym użytkownik posiada swoje konto. Drugim identyfikatorem jest węzeł, który reprezentuje konkretnego użytkownika. Wszystkie węzły istnieją w obrębie domeny. Trzecim opcjonalnym identyfikatorem jest zasób i może on służyć jako zbiór specyficznych parametrów opisujących użytkownika, np.: jego lokalizacja.

Architektura klient-serwer

Protokół XMPP zbudowany jest w oparciu o architekturę klient-serwer, tzn. każdy element danych wysłany przez klienta musi przejść przez przynajmniej jeden serwer. Każdy klient łączy się z serwerem poprzez gniazdo TCP używając portu 5222. Serwery łączą się ze sobą z wykorzystaniem portu 5269, a połączenie to utrzymywane jest na czas trwania sesji klienta z serwerem. Oznacza to, że klient nie musi odpytywać serwera w celu uzyskania nowych wiadomości jak ma to miejsce w przypadku klientów e-mail. Serwer dba o weryfikację dostępności użytkownika, a w momencie gdy serwer wykryje, że użytkownik jest „offline", wszystkie wiadomości wysyłane do tego użytkownika zostają na serwerze i zostaną dostarczone w momencie gdy użytkownik znów pojawi się „online".



Rysunek 12 – XMPP – przykładowa sieć połączeń

Serwery

Serwery obsługujące protokół XMPP pełnią następujące funkcje:

- komunikacji bezpośrednio z klientami jak i pomiędzy sobą,
- rejestracji i autentykacji użytkowników,
- usługi wysyłania wiadomości,
- usługi dostępności,
- przechowywania listy kontaktów użytkownika,
- gromadzenie wiadomości użytkowników będących offline.

Podstawowe usługi serwerów mogą być rozszerzane przez administratorów o dodatkowe funkcje, takie jak specjalne bramy XMPP-CPIM umożliwiające komunikację z innymi systemami jak np.: ICQ.

Klienci

Jednym z kryteriów jakim kierowali się twórcy protokołu XMPP jest prostota i elastyczność, a zatem protokół skonstruowany został tak, aby napisanie klienta było jak najłatwiejsze.
Klient dla protokołu XMPP musi posiadać następujące cechy:

- umiejętność nawiązywania połączenia z użyciem gniazd,
- poprawna interpretacja strumienia danych w postaci XML,
- umiejętność rozpoznawania podstawowych typów danych protokołu XMPP.

Podczas projektowania starano się, aby całą złożoność problemu przenieść maksymalnie na stronę serwera tak aby zniwelować konieczność późniejszej aktualizacji klientów w miarę powstawania nowych funkcjonalności.

2.4.2 Protokół SIP (Session Initiation Protocol)

SIP [14] jest protokołem służącym do inicjowania sesji pomiędzy punktami końcowymi sieci IP. Został zaproponowany przez organizację IETF i jest obecnie dominującym protokołem sygnalizacyjnym dla sieci VoIP [20].

Zestawiona za pomocą protokołu SIP sesja może być zwykłą dwustronną rozmową telefoniczną, jak i konferencją multimedialną, gdyż protokół SIP ma w zamierzeniu dostarczać zestaw funkcji obsługi połączenia i innych cech obecnych w publicznej sieci telefonicznej (PSTN [21]). Jako taki, zawiera funkcje które umożliwiają znane ze stacjonarnej telefonii operacje: wybieranie numeru, dzwonek w telefonie, sygnał zajętości itp., jednakże ich implementacja i używana terminologia jest odmienna.

Protokół SIP najczęściej spotykany jest w parze z protokołem RTP (ang. Real-time Transport Protocol) [22]. Protokół SIP odpowiada za zestawienie połączenia zaś protokół RTP za transmisję danych.

Elementy sieci SIP

Terminale końcowe (nodes) - mogą być to aplikacja uruchamiane na komputerze (softphone) lub urządzenia podobne w wyglądzie i obsłudze do tradycyjnych telefonów

(hardphone), ale podłączane do gniazdek sieciowych. Jeśli dostawca usług zapewni odpowiednią infrastrukturę sieci, można z tych terminali prowadzić zwykłą rozmowę z abonentami sieci PSTN.

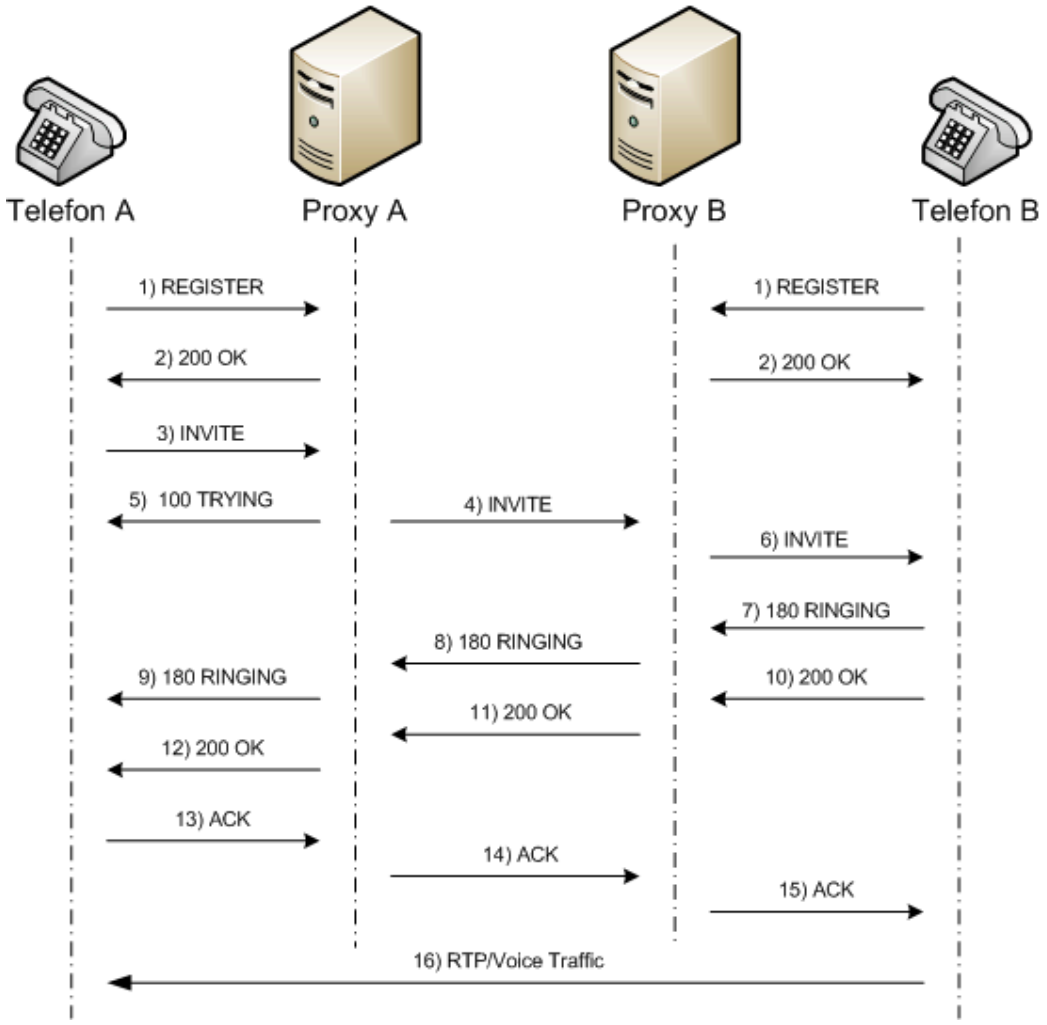
Rejestrator SIP (register) - jest specjalną bazą danych, która komunikuje się z węzłami SIP, w celu zbierania i archiwizacji informacji na temat użytkowników SIP. Dane te są używane w trakcie nawiązywania połączenia do odnajdywania w sieci węzłów docelowych. Każdy węzeł rejestruje się w rejestrze, któremu przekazuje informacje o adresie IP i porcie węzła, wykorzystywane przy dalszej komunikacji. Rejestrator działa jak router na poziomie protokołu SIP, pozwalając węzłowi inicjującemu połączenie odnaleźć węzeł docelowy.

SIP proxy - są to serwery mające za zadanie przekazywać do odpowiedniej domeny żądania połączenia. Proxy analizuje polecenie INVITE i na podstawie zawartego w nim adresu podejmuje decyzje czy połączenie jest lokalne (podłączone do tego samego rejestratora SIP), jeśli nie to kieruje je do innego węzła w sieci. SIP Proxy mogą pełnić też inne zadania takie jak możliwość połączenia sieci VoIP z siecią PSTN.

Z uwagi na to, że protokół SIP jest bardzo elastyczny i z powodzeniem wykorzystywany w sieciach IP zwrócił on uwagę twórców komunikatorów internetowych. W związku z tym zawarty jest w wielu implementacjach tego typu aplikacji.

Metoda wykonywania wywołań przez SIP przedstawiona jest na rysunku 13:

1. Oba telefony rejestrują się u swoich serwerów pośredniczących protokołu SIP (SIP Proxy Server) za pomocą komunikatów REGISTER. Ten etap nie jest wykonywany za każdym razem, a jedynie przy rejestracji stacji.
2. Telefony otrzymują komunikat OK od swoich serwerów.
3. Telefon A wysyła komunikat INVITE zawierający informacje o wspieranych kodekach, portach i adresie IP przeznaczonych dla strumieni dźwiękowych.
4. Używając zapytań DNS, Proxy A określa, który serwer jest odpowiedzialny za obsługę wywołanego URI, i przekazuje wiadomość INVITE do serwera Proxy B.
5. Równocześnie wiadomość 100 TRYING jest wysyłana zwrotnie do Telefonu A.
6. Proxy B odbiera INVITE, sprawdza czy Telefon B jest aktualnie zarejestrowany i przekazuje do niego wiadomość.
7. Telefon B akceptuje komunikat INVITE i zwraca wiadomość 180 RINGING.



Rysunek 13 – Komunikacja w SIP

8. Proxy B przekazuje wiadomość 180 RINGING do Proxy A.
9. Proxy A przekazuje wiadomość 180 RINGING do telefonu A.
10. Użytkownik B w końcu „podnosi słuchawkę”, w efekcie Telefon B akceptuje połączenie (wywołane) i wysyła 200 OK do Proxy B. Wiadomość zawiera informacje o wspieranych kodekach, portach i adresie IP, przeznaczonych dla strumieni dźwiękowych.
11. Proxy B przekazuje wiadomość 200 OK do Proxy A.
12. Proxy A przekazuje wiadomość 200 OK do Telefonu A.
13. Telefon A zwraca potwierdzenie (zgodę) ACK do Proxy A. ACK może zawierać informację dotyczącą preferowanych mediów transmisji.
14. Proxy A przekazuje ACK do Proxy B.
15. Proxy B przekazuje ACK do Telefonu A.

16. Dwa telefony mogą teraz nawiązać komunikację bezpośrednio ze sobą. Serwery SIP nie będą już angażowane.

2.4.3 Protokół SIMPLE (SIP for IM and Presence Leveraging Extensions)

SIMPLE[13] jest to protokół zapewniający dwie podstawowe usługi komunikatorów internetowych: usługę wysyłania wiadomości oraz usługę obecności. Powstał na bazie protokołu SIP na potrzeby komunikatorów internetowych i korzysta z niego w celu rozwiązania następujących problemów:

- odpowiada za zarządzanie sesją,
- rejestruje oraz odbiera powiadomienia na temat obecności użytkownika,
- odpowiada za wysyłanie krótkich wiadomości tekstowych analogicznych do SMS.

Usługa dostępności

Usługa dostępności realizowana jest za pomocą metod SUBSCRIBE, NOTIFY oraz PUBLISH udostępnianych przez protokół SIP. Metoda SUBSCRIBE pozwala na zarejestrowanie zdarzenia na serwerze który to odpowie za pomocą metody NOTIFY w momencie nadejścia zdarzenia. Istnieją dwa modele realizacji usługi dostępności:

- model end-to-end,
- model scentralizowany.

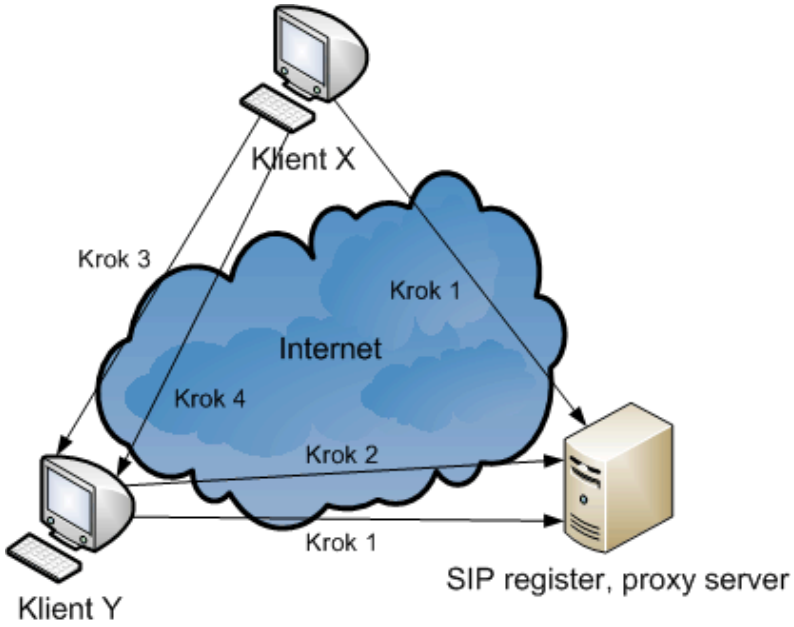
W modelu end-to-end każdy z uczestników sam odpowiada za subskrypcję, zaś w modelu scentralizowanym wykorzystywany jest serwer. Każdy z użytkowników ma obowiązek informować go za pomocą metody PUBLISH o swoim stanie.

Usługa wysyłania wiadomości

SIP definiuje dwa modele wysyłania wiadomości:

- model stron (ang. Page Mode),
- model sesji (ang. Session Mode).

W modelu stron wykorzystywana jest metoda MESSAGE protokołu SIP i nie nawiązywana jest sesja pomiędzy użytkownikami. W model sesji, jak sama nazwa wskazuje, zestawiania jest sesja, zaś za wymianę wiadomości odpowiedzialny jest protokół MSRP[23].



Rysunek 14 – Komunikacja w SIMPLE

Zasada działania:

1. Klient X i Y wykorzystują protokół SIP (metoda REGISTER) aby zarejestrować swoją obecności.
2. Klient Y dokonuje subskrypcji (metoda SUBSCRIBE) na serwerze w celu otrzymywania powiadomień o zmianie obecności klienta X.
3. Klient X wysyła powiadomienie (metoda NOTIFY) z informacją o jego dostępności do jego znajomego Y.
4. Klient X wysyła wiadomość do klienta Y (metoda MESSAGE).

3. Specyfikacja problemu

W rozdziale tym zawarto cel projektu, wymagania funkcjonalne oraz niefunkcjonalne.

3.1 Cel projektu

Celem projektu jest stworzenie aplikacji będącej wieloprotokołowym komunikatorem internetowym, zapewniającym obsługę dwóch podstawowych usług: usługę obecności oraz usługę wysyłania wiadomości. Tworzona aplikacja powinna być dostępna z poziomu przeglądarki internetowej.

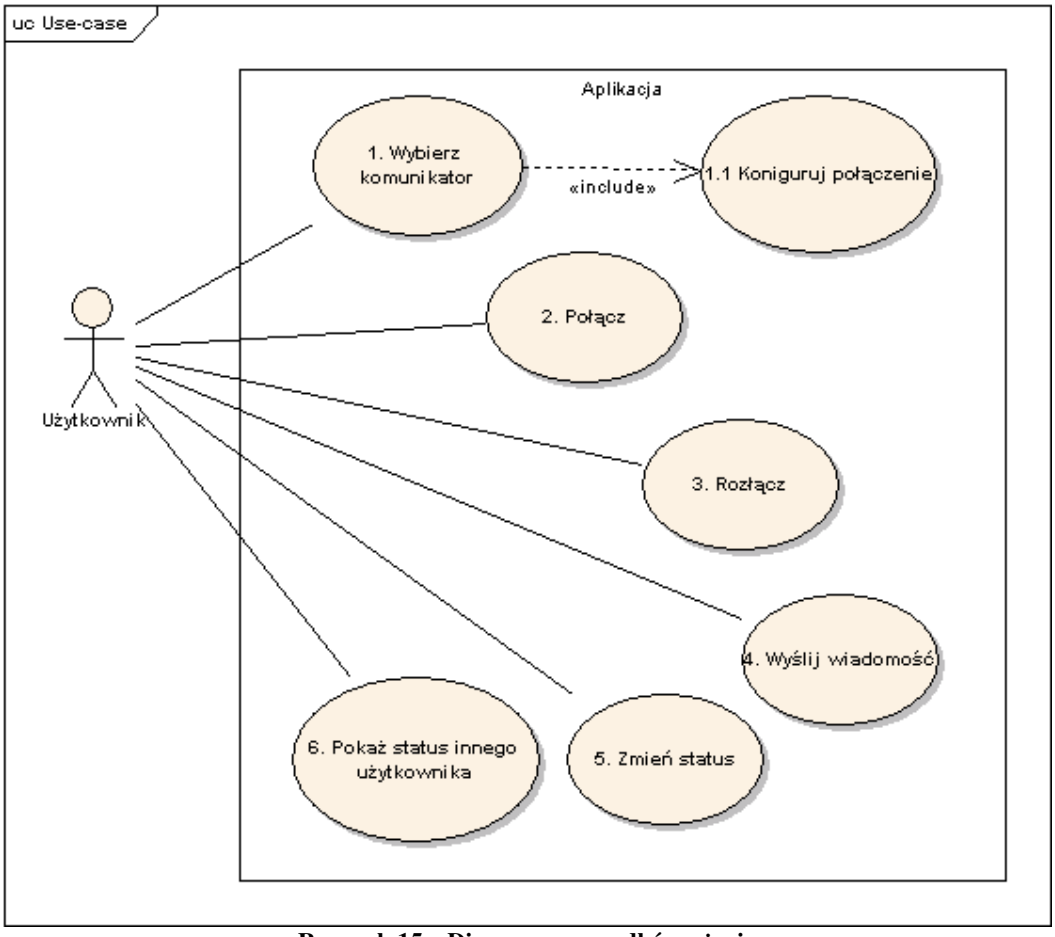
3.2 Wymagania funkcjonalne

Tworzona aplikacja powinna udostępniać użytkownikowi możliwość korzystania z najbardziej znanych komunikatorów internetowych, takich jak GoogleTalk, Gadu-Gadu, Jabber oraz Windows Live Messenger.

Przewidziano obsługę następujących, podstawowych przypadków użycia:

- Wybierz komunikator,
- Konfiguruj połączenie,
- Połącz,
- Rozłącz,
- Zmień status,
- Pokaż status innego użytkownika.

Zostały one uszczegółowione i przedstawione w kolejnych punktach.



Rysunek 15 – Diagram przypadków użycia

Przypadek użycia	**Scenariusz główny**
1. Wybierz komunikator	1. Użytkownik wybiera komunikator z listy dostępnych komunikatorów. 2. Użytkownik konfiguruje parametry połączenia inne w zależności od wybranego komunikatora.
	Scenariusze alternatywne
	2.a.1 System waliduje wprowadzone przez użytkownika dane i zgłasza błąd z prośbą o poprawę wprowadzonych danych.
	Warunek wstępny
	1. Klient jest rozłączony.

Przypadek użycia	Scenariusz główny
2. Połącz	1. Użytkownik wybiera opcję połącz. 2. Po nawiązaniu połączenia system odblokowuje odpowiednie elementy interfejsu użytkownika dostępne jedynie po zalogowaniu.
	Scenariusze alternatywne
	1.a.1 System wykrywa błąd podczas nawiązywania połączenia i wyświetla stosowny komunikat.
	Warunek wstępny
	1. Klient jest rozłączony.

Przypadek użycia	Scenariusz główny
3. Rozłącz	1. Użytkownik wybiera opcję rozłącz. 2. Po rozłączeniu system blokuje odpowiednie elementy interfejsu użytkownika dostępne jedynie po wylogowaniu.
	Scenariusze alternatywne
	1.a.1 System wykrywa błąd podczas zawieszania połączenia i wyświetla stosowny komunikat.
	Warunek wstępny
	1. Klient jest połączony.

Przypadek użycia	Scenariusz główny
4. Wyślij wiadomość	1. Użytkownik wysyła wiadomość do innego użytkownika.
	Scenariusze alternatywne
	1.a.1 System wykrywa błąd podczas wysyłania wiadomości i wyświetla stosowny komunikat.
	Warunek wstępny
	1. Klient jest połączony.

Przypadek użycia	Scenariusz główny
5. Zmień status	1. Użytkownik zmienia swój status.
	Scenariusze alternatywne
	1.a.1 System wykrywa błąd podczas zmiany statusu i wyświetla stosowny komunikat.
	Warunek wstępny
	1. Klient jest połączony.

Przypadek użycia	Scenariusz główny
6. Pokaż status innego użytkownika	1. Użytkownik wybiera z listy użytkownika. 2. W stosownym miejscu interfejsu system pokazuje status wybranego użytkownika.
	Scenariusze alternatywne
	1.a.1, 2.a.1 System wykrywa błąd podczas pobrania statusu i wyświetla stosowny komunikat.
	Warunek wstępny
	1. Klient jest połączony.

3.3 Wymagania niefunkcjonalne

Rozdział ten zawiera wymagania niefunkcjonalne dotyczące zarówno sprzętu jak i aplikacji.

3.3.1 Wobec aplikacji

Aplikacja powinna być dostępna z poziomu przeglądarki internetowej oraz systemu JBoss Portal [24].

JBoss portal jest implementacją portalu stworzoną przez firmę JBoss. Portal to platforma dostępna z poziomu przeglądarki internetowej umożliwiająca łatwe tworzenie i zarządzanie treścią stron oraz personalizację użytkowników. Jest także kontenerem portletów, dzięki czemu możliwe jest umieszczanie w nim dodatkowych aplikacji.

3.3.2 Wobec sprzętu

Z uwagi na wykorzystanie systemu JBoss Portal minimalne wymagania wobec sprzętu to:

- JDK™ w wersji 5
- 512 MB pamięci RAM
- 100 MB wolnego miejsca na dysku
- minimalna częstotliwość procesora 400 MHz

Zainstalowany system operacyjny może być dowolnym systemem umożliwiającym uruchomienie wirtualnej maszyny Java, czyli mogą to być systemy z rodziny Linux, Windows, Unix oraz Mac OS X.

Szczegółowe dane na temat instalacji oraz wersji komponentów znajdują się w Dodatku A.

4. Wykorzystane technologie

W rozdziale tym przedstawiono analizę możliwych rozwiązań, wykorzystywane w procesie tworzenia aplikacji narzędzia, biblioteki oraz technologie.

4.1 Analiza możliwych rozwiązań

Podczas analizy możliwych sposobów implementacji zdecydowano się wykorzystać język Java, który znakomicie nadaje się zarówno do aplikacji sieciowych, jak i rozproszonych. Język ten jest powszechnie znany, a także istnieje duża liczba sprawdzonych narzędzi oraz bibliotek obsługujących popularne protokoły komunikatorów internetowych.

Na dalszym etapie ustalono, iż tworzona aplikacja powinna być dostępna z poziomu przeglądarki internetowej oraz udostępniona w formie portletu, z uwagi na możliwość wielokrotnego wykorzystania. W fazie analizy pod uwagę brano także inne rozwiązania:

- **aplikacja stacjonarna (standalone)** - większość znanych komunikatorów internetowych jest w ten sposób udostępniana swoim użytkownikom. Ten sposób implementacji jest bardzo prosty postanowiono utworzyć taką aplikację z wykorzystaniem technologii Java Swing. Utworzona aplikacja ma służyć tylko i wyłącznie do celów testowych - weryfikacji poprawnej pracy tworzonej biblioteki oraz zbierania logów do dalszej analizy.

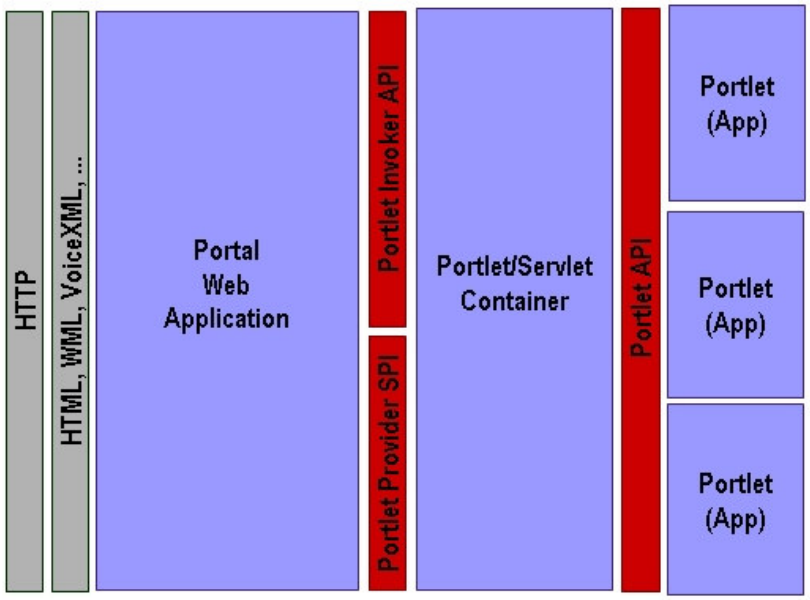
- **aplikacja uruchamiana za pomocą Java Web Start [25]** - podobnie jak aplikacja stacjonarna uruchamiana jest na komputerze użytkownika, jedyną zaletą tego rozwiązania jest fakt iż użytkownik zawsze będzie dysponował aktualną wersją aplikacji klienckiej.

- **klient w formie apletu** - jest jednym ze sposobów przeniesienie "grubego" klienta do przeglądarki internetowej. Aplety są bezpieczne w użyciu, gdyż nie mają prawa dostępu do systemu plików znajdujących się na dysku użytkownika, ale mogą wymieniać dane z innymi serwerami. Aplety nie uzyskały dużej popularności z uwagi na to, iż witryny z apletami ładują się znacznie dłużej.

- **klient z poziomu WWW** - w chwili obecnej wiele komunikatorów internetowych jest dostępnych z poziomu WWW, dzięki wykorzystaniu technologii AJAX. W przypadku aplikacji tego typu, użytkownik może z dowolnego miejsca i bardzo szybko skorzystać z usługi. Aplikacja napisana w ten sposób rozwiązuje także problem aktualizacji wersji oprogramowania, jaki istnieje w przypadku aplikacji stacjonarnych. W ramach niniejszej pracy zdecydowano się właśnie na takie rozwiązanie, gdyż tworzone za pomocą technologii AJAX aplikacje stają się coraz bardziej powszechne, a proces ich tworzenia nie jest sprawą do końca oczywistą.

4.2 Technologie portalowe dla języka Java

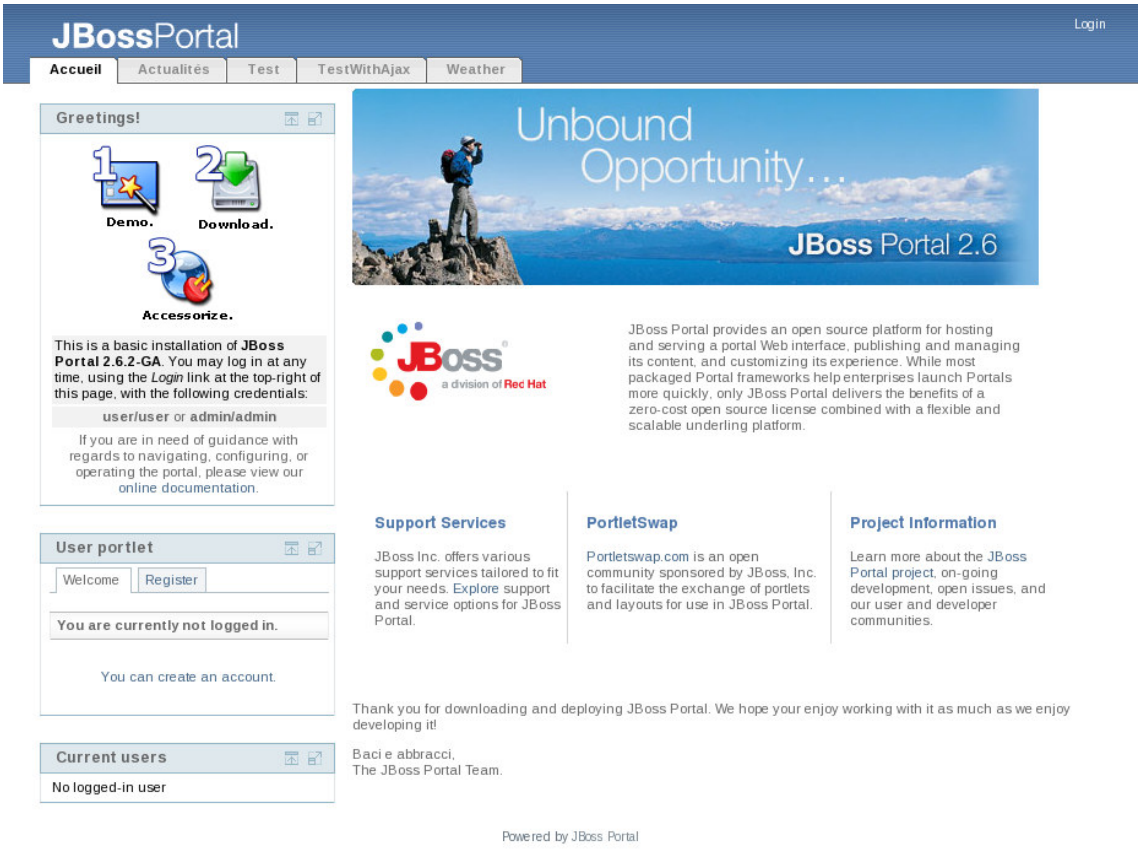
Portal – jest aplikacją dostępną z poziomu WWW, umożliwiającą łatwą publikację i zarządzanie treścią stron, a także personalizacje użytkowników. Portal jest także kontenerem portletów, co umożliwia bardzo łatwe wzbogacanie treści strony o dodatkowe elementy, takie jak np.: prognoza pogody, tematy z forum, indeksy giełdowe itp. W rezultacie każdy użytkownik po zalogowaniu może korzystać z innego zbioru portletów.



Rysunek 16 – Architektura Portalu

Na rysunku 16 przedstawiona jest architektura portalu. Po otrzymaniu żądania od klienta, kontener portletów odpytywany jest w celu uzyskania treści portletu. Kontener portletów dostarcza środowiska uruchomieniowego dla portletów i komunikuje się z nimi za pomocą Portlet API. Kontener portletów jest wywoływany przez portal za pomocą Portlet Invoker API, zaś kontener otrzymuje informacje na temat portalu za pomocą Portlet Provider SPI (ang. Sernice Provider Interface). Tworzona w ramach pracy aplikacja docelowo będzie umieszczona na tego typu portalu jakim jest JBoss Portal.

JBoss Portal – jest implementacją portalu stworzoną przez firmę JBoss. W ramach niniejszej pracy wykorzystywany jest JBoss Portal w wersji 2.6.7.GA wraz z serwerem aplikacji JBoss Application Server w wersji 4.2.3.GA. – wybór ten wynika z założeń pierwotnego projektu.



Rysunek 17 – JBoss Portal – strona startowa

Portlet – jest to komponent bazujący na języku Java, który osadzony na stronie WWW przetwarza żądania od użytkowników i generuje dynamiczną treść. Treść wygenerowana

przez portlet nosi nazwę fragmentu i jest to kawałek kodu Np.: HTML lub XHTML. Portlet osadzony jest w kontenerze portletów który zarządza jego cyklem życia. Szczegółową specyfikacje portletów określają dokumenty JSR-168 [26] i JSR-286 [27]. Tworzona w ramach niniejszej pracy aplikacja będzie takim komponentem. Jej podstawowe składowe stanowią:

JavaServer Faces (JSF) – bazujący na języku Java framework umożliwiający dynamiczne tworzenie stron WWW stworzony w celu uproszczenia tworzenia interfejsu użytkownika dla aplikacji J2EE. Cechą, która wyróżnia ten framework od innych, jest podejście komponentowe do tworzenia elementów interfejsu. Szczegółową specyfikację JSF dla wersji 1.2 opisuje dokument JSR-252 [28]. Framework ten jest bezpośrednio wykorzystywany do warstwy prezentacji tworzonej aplikacji.

RichFaces – jest to biblioteka komponentów dla JavaServer Faces bazująca na frameworku Ajax4jsf. Pozwala na bardzo łatwą integrację modelu AJAX z aplikacjami pisanymi w technologii JSF. Tagi biblioteki RichFaces wykorzystywane są do utworzenia interaktywnych elementów interfejsu użytkownika tworzonej aplikacji.

AJAX - (ang. Asynchronous JavaScript and XML, Asynchroniczny JavaScript i XML) – technika tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się bez przeładowywania całego dokumentu.

JBoss Portlet Bridge – jest to implementacja specyfikacji JSR-301 [29] wspierającej możliwość uruchamiania aplikacji utworzonych z wykorzystaniem technologii JSF w środowisku portletowym. W chwili obecnej bridge wspiera dowolną kombinację JSF, Seam oraz RichFaces przy uruchamianiu z poziomu portletu. Dzięki wykorzystaniu tego rozwiązania możliwe jest uruchomienie tworzonej aplikacji w systemie JBoss Portal.

4.3 Wybrane biblioteki komunikatorów internetowych

Podczas początkowych prac nad projektem przeanalizowano sposób działania oraz przydatność bibliotek najbardziej popularnych protokołów komunikatorów internetowych opartych na języku Java:

Smack – biblioteka umożliwiająca korzystanie z protokołu XMPP stworzona przez firmę JIVE Software, wykorzystana do implementacji komunikatorów Jabber oraz GoogleTalk.

Muse – alternatywna biblioteka do biblioteki Smack, nie wykorzystana bezpośrednio w projekcie.

JGGApi – biblioteka obsługująca protokół Gadu-Gadu, nie rozwijana od 2006 roku wymagająca kilku drobnych poprawek, wykorzystana w projekcie do implementacji komunikatora Gadu-Gadu.

Daim, JOscar - biblioteki umożliwiające korzystanie z protokołu OSCAR słabo udokumentowane, pełne błędów i niejasności dlatego nie wykorzystane bezpośrednio w projekcie.

JML – biblioteka utworzona przez Daniela Henningera umożliwiająca korzystanie z komunikatora Windows Live Messenger.

4.4 Narzędzia wykorzystane w procesie implementacji

W rozdziale tym przedstawiono krótki opis wykorzystanych narzędzi oraz wtyczek dla środowiska Eclipse.

Eclipse – zintegrowane środowisko programistyczne do tworzenia programów w języku Java. Tworzona w ramach niniejszej pracy aplikacja, napisana została z wykorzystaniem tego narzędzia.

Apache Maven - jest narzędziem automatyzującym budowę oprogramowania na platformę Java. Poszczególne funkcjonalności Mavena realizowane są poprzez wtyczki, które są automatycznie pobierane przy ich pierwszym wykorzystaniu. Plik określający sposób budowy aplikacji nosi nazwę POM-u (ang. Project Object Model). Struktura tworzonej aplikacji oparta w oparciu o to narzędzie.

Subversion (znany również jako SVN) - system kontroli wersji, który powstał w celu zastąpienia CVS. Podczas procesu implementacji narzędzie to znalazło zastosowanie w celu archiwizacji tworzonego kodu.

Apache log4j - biblioteka języka programowania Java służąca do tworzenia logów podczas działania aplikacji. Wykorzystana w procesie implementacji do zbierania logów.

jUnit - jest narzędziem służącym do tworzenia powtarzalnych testów jednostkowych oprogramowania pisanego w języku Java. W oparciu o narzędzie jUnit powstał zbiór testów jednostkowych dla biblioteki agregującej komunikatory.

Wtyczki dla środowiska Eclipse:

m2eclipse - wtyczka ułatwiająca korzystanie z Mavena. Wykorzystywana w procesie implementacji do zarządzania zależnościami oraz budowania projektu.

Subclipse - wtyczka ułatwiająca pracę z systemem kontroli wersji SVN. Umożliwia łatwe synchronizowanie stanu kodu aplikacji z repozytorium.

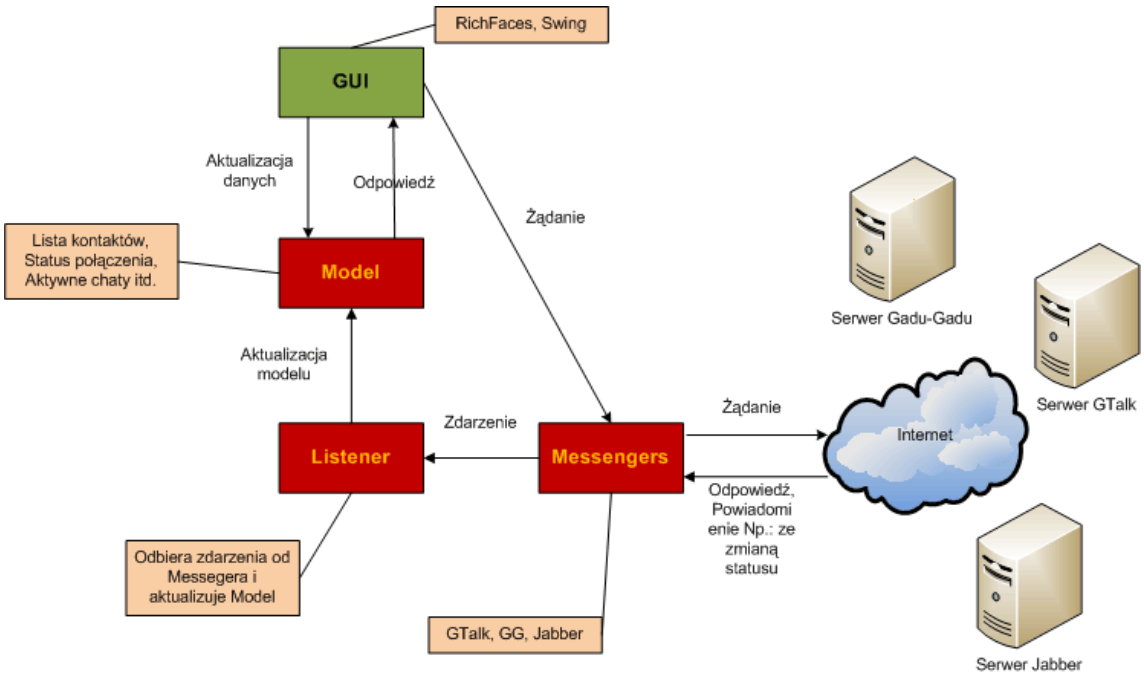
JBoss Tools - zbiór wtyczek ułatwiających tworzenie interfejsu za pomocą JSF oraz RichFaces. Wyczytano podczas tworzenia widoków dla klienta dostępnego z poziomu przeglądarki internetowej.

Jigloo – wtyczka ułatwiająca tworzenie interfejsu użytkownika z wykorzystaniem Java Swing. Wykorzystana do utworzenia aplikacji testowej.

5. Projekt i implementacja

Rozdział ten przedstawia najistotniejsze elementy opisujące projekt jak i implementację tworzonej w ramach niniejszej pracy aplikacji.

5.1 Architektura aplikacji



Rysunek 18 – Architektura aplikacji

Na etapie projektowania starano się zaproponować taką architekturę aplikacji, która z jednej strony będzie wspólnym abstrakcyjnym modelem opisu wszystkich komunikatorów, a z drugiej strony pozwoli na wygodne wykorzystanie przy zastosowaniu dowolnych interfejsów użytkownika.

W architekturze aplikacji możemy wyróżnić następujące elementy:

- **Messengers** – są to konkretne implementacje komunikatorów internetowych, realizują żądania przychodzące od interfejsu użytkownika oraz obsługują zdarzenia nadchodzące od serwera komunikatora a następnie rejestrują je we wspólnym dla wszystkich komunikatorów modelu za pomocą także wspólnego Listenera,

- **Listener** – obsługuje zdarzenia przychodzące od Messenger-a i jako jedyny ma możliwość aktualizacji modelu,
- **Model** – wspólny dla wszystkich komunikatorów model danych gromadzący wszelkie informacje potrzebne do zapewnienia pracy komunikatora takie jak: lista kontaktów, statusów, wiadomości itp.,
- **GUI** – graficzny interfejs użytkownika, umożliwia prezentację stanu aplikacji oraz wysyłania żądań do komunikatora, powinien posiadać element umożliwiający periodycznie odświeżanie elementów interfejsu użytkownika.

Biblioteka agregująca komunikatory

W architekturze aplikacji możemy wyróżnić trzy elementy: Model, Listener oraz Messenger, które mogą stanowić jedną wspólną całość i dlatego zostaną umieszczone w jednej bibliotece nazywanej w dalszej części pracy biblioteką agregującą komunikatory.

Graficzny interfejs użytkownika

Element GUI widoczny na diagramie architektury aplikacji odpowiada w rzeczywistość graficznemu interfejsowi użytkownika i wykorzystuje bezpośrednio bibliotekę agregującą komunikatory. W rozdziale czwartym, przedstawione zostały najbardziej popularne sposoby implementacji tego elementu. W tworzonej w ramach niniejszej pracy aplikacji, zostaną utworzone dwa tego typu elementy. Jednym będzie aplikacja testowa, oparta o bibliotekę graficzną Swing. Drugim, a zarazem docelowym rozwiązaniem, jest aplikacja portletowa dostępna z poziomu WWW.

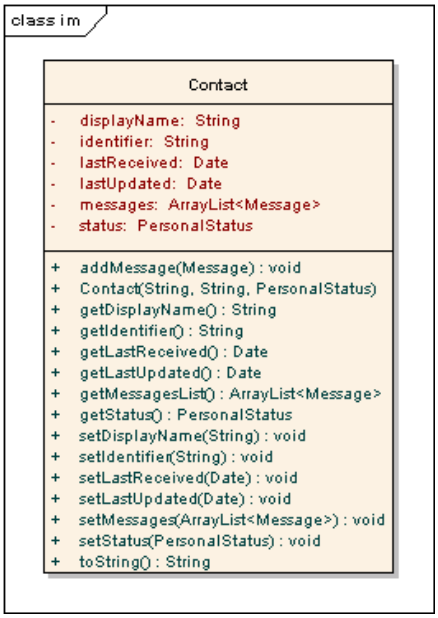
5.2 Biblioteka agregująca komunikatory

Biblioteka agregująca komunikatory jest wspólnym, abstrakcyjnym modelem opisującym różne technologicznie komunikatory internetowe. Ma za zdanie zapewnić łatwe i przejrzyste API umożliwiające obsługę wybranych protokołów komunikatorów internetowych. Zawiera w sobie następujące elementy architektury takie jak:

- Model,
- Listener,
- Messenger.

5.2.1 Model

Model został zaprojektowany tak aby zawierał w miarę wspólny dla wszystkich komunikatorów zbiór obiektów, np.: profil użytkownika, wiadomość, znajomy z listy kontaktów, status połączenia, status prywatny itp., umożliwiających reprezentowanie struktur danych wymaganych do pracy komunikatora. Każdy z elementów modelu jest w rezultacie prostym obiektem POJO, zawierającym zbiór atrybutów oraz zestaw metod do modyfikacji ich stanu tzw. „getterów" i „setterów". Przykładowa klasa modelu znajduje się na rysunku 19, a reprezentuje ona osobę z listy kontaktów, jej status oraz listę wiadomości z nią wymienionych.



Rysunek 19 – Contact - przykładowa klasa modelu

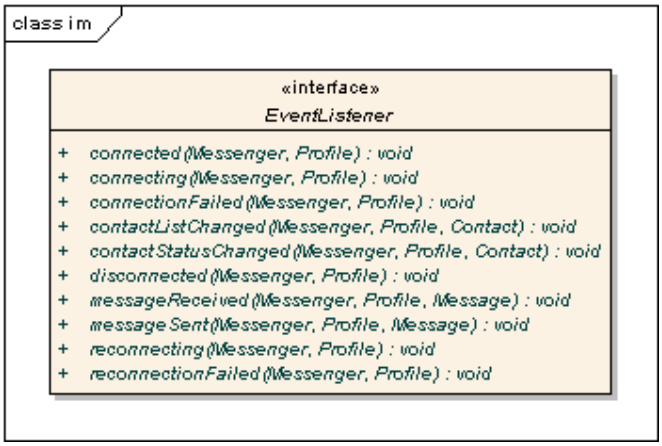
5.2.2 Listener

Listener ma za zadanie rejestrację wszystkich zdarzeń nadchodzących od komunikatora internetowego i jako jedyny może aktualizować obiekty modelu. W praktyce wygląda to tak, że każda operacja wykonana przez komunikator powinna być zarejestrowana poprzez Listenera. W momencie gdy komunikator nawiąże połączenie, odbierze wiadomość, otrzyma informację o zmianie statusu osoby z listy kontaktów itp. wołana jest odpowiednia metoda Listenera która loguje zaistniałe zdarzenie oraz aktualizuje stan modelu.

Listener odpowiada za rejestrację następujących zdarzeń:

- sygnalizuje zmianę stanu połączenia tj. połączony, rozłączony, trwa nawiązywanie połączenia, połączenie zakończyło się błędem,
- sygnalizuje nadejście nowej wiadomości,
- zmianę statusu osoby znajdującej się na liście kontaktów,
- zmianę listy kontaktów.

Biblioteka agregująca komunikatory zawiera dokładnie jedną implementację Listenera o nazwie MessengersEventListener który implementuje interfejs EventListener.



Rysunek 20 – Interfejs EventListener

Interfejs EventListener zawiera następujące metody:

- **connected** – aktualizuje status połączenia na status połączony,
- **connecting** – aktualizuje status połączenia na status trwa nawiązywanie połączenia,
- **connectionFailed** – aktualizuje status połączenia na status połączenie zakończyło się niepowodzeniem,

- **contactListChanged** – aktualizuje listę kontaktów,
- **contactStatusChanged** – aktualizuje status osoby z listy kontaktów,
- **disconnected** – aktualizuje status połączenia na rozłączony,
- **messageReceived** – aktualizuje listę wymienionych wiadomości osoby z listy kontaktów tworzy nowy alarm z informacją o nadejściu nowej wiadomości,
- **messageSent** - aktualizuje listę wymienionych wiadomości osoby z listy kontaktów,
- **reconnecting** – aktualizuje status połączenia na status próba ponownego nawiązania połączenia,
- **reconnectionFailed** – aktualizuje status połączenia na status próba ponownego nawiązania połączenia zakończyła się niepowodzeniem.

5.2.3 Messenger

Element Messenger jest implementacją konkretnego komunikatora internetowego i ma za zadanie obsługę protokołu z którego korzysta komunikator. Każda taka implementacja wykorzystuje jedną z bibliotek przeanalizowanych w rozdziale piątym oraz implementuje jej odpowiednie interfejsy. Dodatkowo każdy komunikator powinien implementować jeden wspólny interfejs o nazwie Messenger.

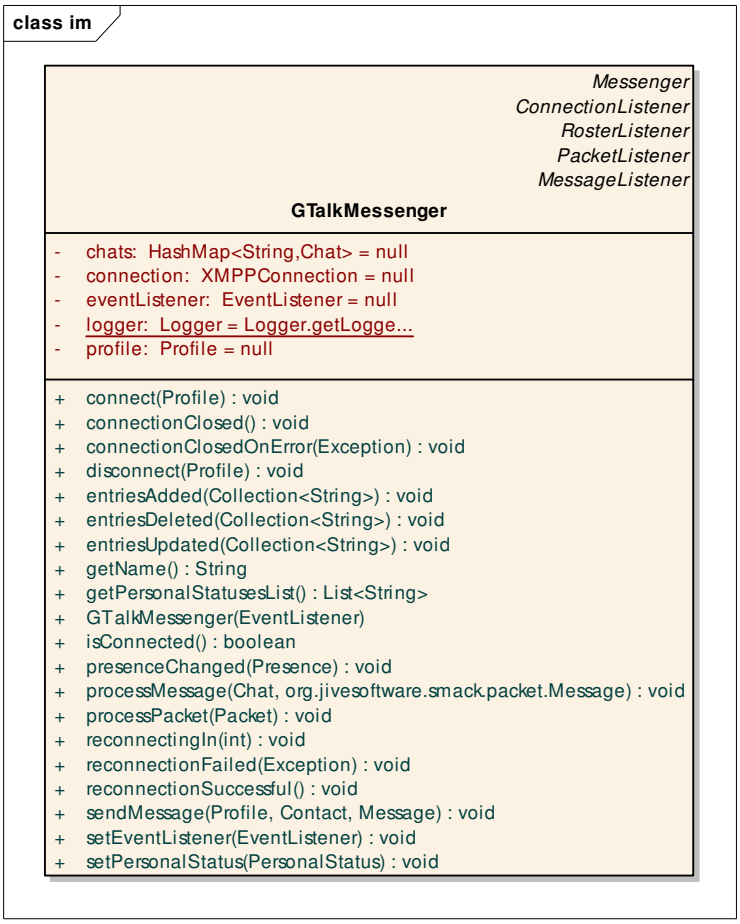
Rysunek 21 – Interfejs Messenger

Interfejs Messenger zawiera następujące metody:

- **connect** – umożliwia nawiązanie połączenia,
- **disconnect** – umożliwia rozłączenie,
- **getName** – zawraca nazwę komunikatora,
- **getPersonalStatusesList** – zwraca listę dostępnych statusów użytkownika,
- **isConnected** – zwraca informację czy komunikator jest połączony,

- **sendMessage** – umożliwia wysłanie wiadomości,
- **setEventListener** – umożliwia ustawienie konkretnej implementacji EventListenera,
- **setPersonalStatus** – umożliwia ustawienie statusu użytkownika.

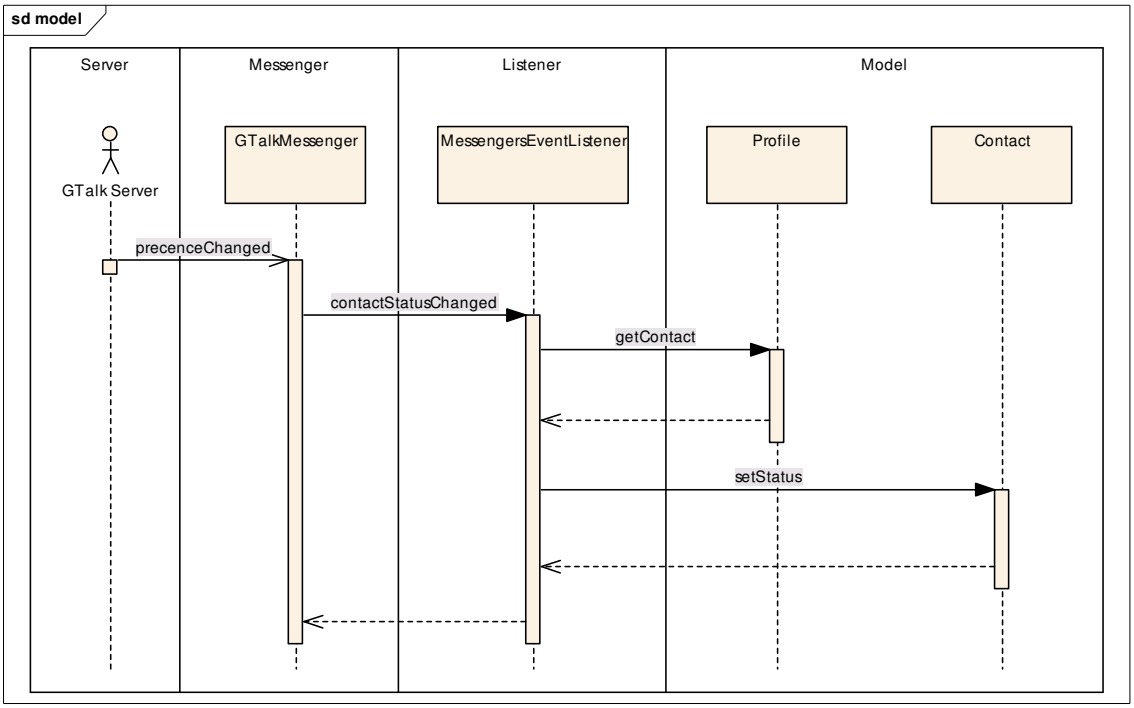
Każdy z komunikatorów odpowiada za obsługę zdarzeń nadchodzących zarówno od interfejsu użytkownika, tj. np.: nawiązanie połączenia, zmiana statusu, wysłanie wiadomości itp. oraz zdarzeń nadchodzących od serwera obsługującego ten komunikator. Zdarzenia pochodzące od interfejsu użytkownika obsługiwane są poprzez metody interfejsu Messenger zaś zdarzenia napływające od serwera poprzez metody implementujące interfejsy wybranej biblioteki obsługującej dany protokół.



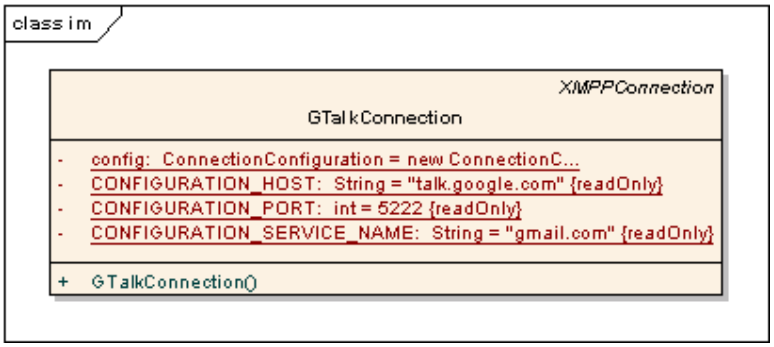
Rysunek 22 – Klasa GTalkMessenger

Jako przykład elementu Messenger może posłużyć implementacja komunikatora GoogleTalk który zbudowany jest z trzech klas: GTalkMessenger, GTalkConnection oraz GTalkUtil.

Rysunek 22 przedstawia klasę GTalkMessenger, która implementuje interfejs Messenger oraz interfejsy: ConnectionListener, RosterListener, PacketListener, MessageListener. Interfejs Messenger, jak było to wspomniane wcześniej, zawiera metody obsługujące żądania pochodzące od interfejsu użytkownika. Pozostałe interfejsy pochodzą z biblioteki Smack, umożliwiającej obsługę protokołu XMPP i rejestrują zdarzenia napływające od strony serwera. Każde takie zdarzenie jest odpowiednio interpretowane z wykorzystaniem EventListener-a aktualizowany jest stan obiektów modelu. Diagram z rysunku 23 pokazuje sposób, w jaki aktualizowany jest stan modelu po odebraniu od serwera informacji o zmianie statusu osoby z listy kontaktów.

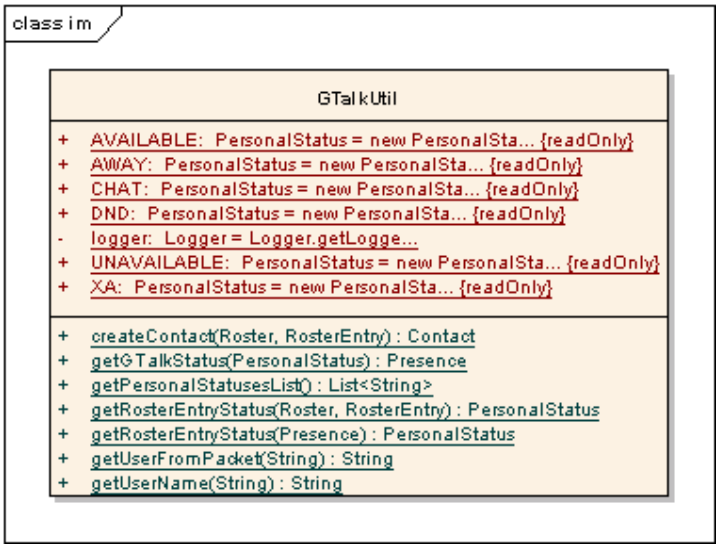


Rysunek 23 – Diagram sekwencji - aktualizacja modelu - statusu kontaktu



Rysunek 24 – Klasa GTalkConnection

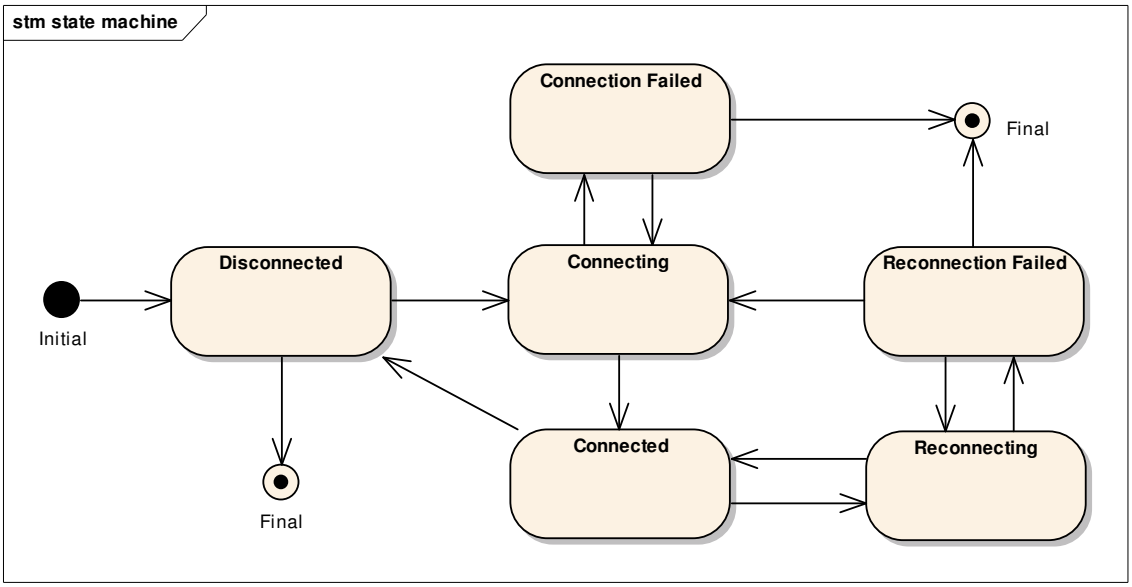
Klasa GTalkConnection pozwala na nawiązanie połączenia z serwerem GoogleTalk i jest bezpośrednio wykorzystywana w klasie GTalkMessenger. Klasa GTalkUtil odpowiedzialna jest za wszelkiego rodzaju konwersje pomiędzy strukturą danych konkretnej biblioteki, tutaj Smack do struktur danych modelu czyli np.: statusu użytkownika.



Rysunek 25 – Klasa GTalkUtil

5.2.4 Model stanów komunikatora

Diagram na rysunku 26 przedstawia możliwe stany w jakich może znajdować się komunikator.



Rysunek 26 – Diagram stanów komunikatora

Wyjściowym stanem komunikatora jest stan Disconnected. W stanie tym komunikator jest rozłączony z serwerem. W momencie nawiązywania połączenia z serwerem komunikator posiada stan Connecting. Gdy połączenie zostanie nawiązane w sposób prawidłowy, komunikator przejdzie w stan Connected, zaś w momencie gdy nie uda się nawiązać połączenia osiągnie stan Connection Failed. Podczas pracy komunikatora może zajść sytuacja w której komunikator utraci połączenie z serwerem wtedy wchodzi w stan Reconnecting próbując odzyskać połączenie. Po udanym odzyskaniu połączenia komunikator osiąga stan Connected, zaś w momencie nie udanej próby stan Reconnection Failed.

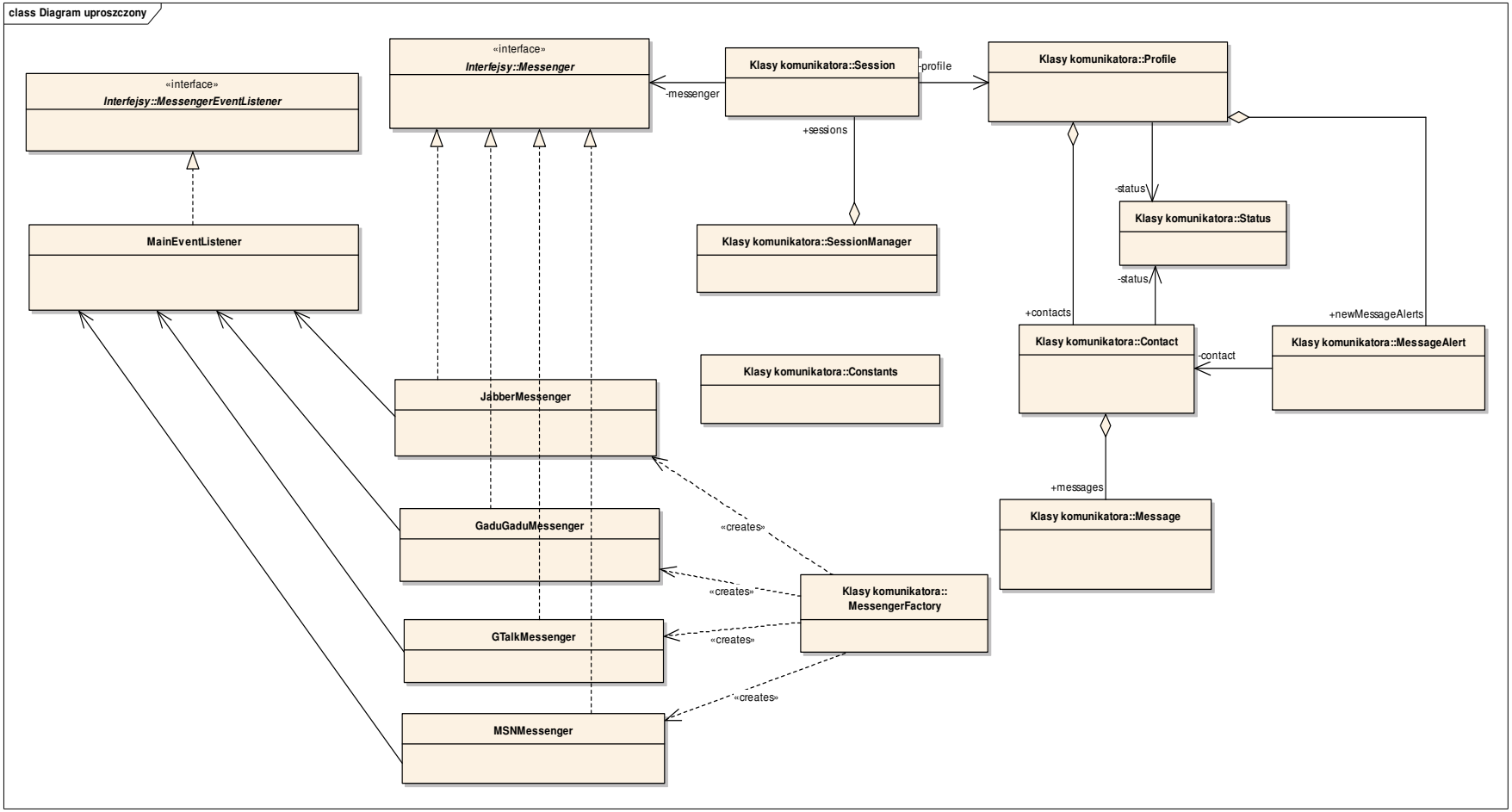
5.2.5 Pakiety

Biblioteka agregująca komunikatory zbudowana jest z następujących pakietów:

- **im** – zawiera klasy Constans, MessengerFactory oraz Session,
- **im.model** - pakiet zawierający podstawowe klasy modelu takie jak ConnectonStatus, PersonalStatus, Profile, Contact, Message oraz MessageAlert,
- **im.interfaces** - zawiera interfejsy Messenger oraz EventListener,
- **im.messengers** - obecnie zawiera wspólną dla wszystkich komunikatorów implementację interfejsu EventListenera - MessengersEventListener,
- **im.messengers.gtalk** - zawiera klasy potrzebne do obsługi GoogleTalk, tj. GTalkConnection umożliwiająca nawiązanie połączenia, GTalkMessenger główna klasa obsługująca komunikator GTalk (implementuje interfejs Messenger) oraz GTalkUtil gdzie znajdują się statyczne metody pomocnicze,
- **im.messengers.jabber** - zawiera klasy potrzebne do obsługi komunikatora Jabber, tj. JabberMessenger - główna klasa obsługująca Jabbera (implementuje interfejs Messenger) oraz JabberUtil gdzie znajdują się statyczne metody pomocnicze,
- **im.messengers.gg** - zawiera klasy potrzebne do obsługi komunikatora GaduGadu, tj. GGMessenger główna klasa obsługująca Gadu-Gadu (implementuje interfejs Messenger) oraz GGUtil gdzie znajdują się statyczne metody pomocnicze,
- **im.messengers.msn** – zawiera klasy potrzebne do obsługi komunikatora Windows Live Messenger takie jak MSNMessenger główna klasa obsługująca ten komunikator (implementuje interfejs Messenger) oraz MSNUtil, gdzie znajdują się statyczne metody pomocnicze.

5.2.6 Uproszczony diagram klas

Rysunek 26 przedstawia uproszczony diagram klas biblioteki, agregującej komunikatory, mający na celu pokazanie zależności pomiędzy klasami.



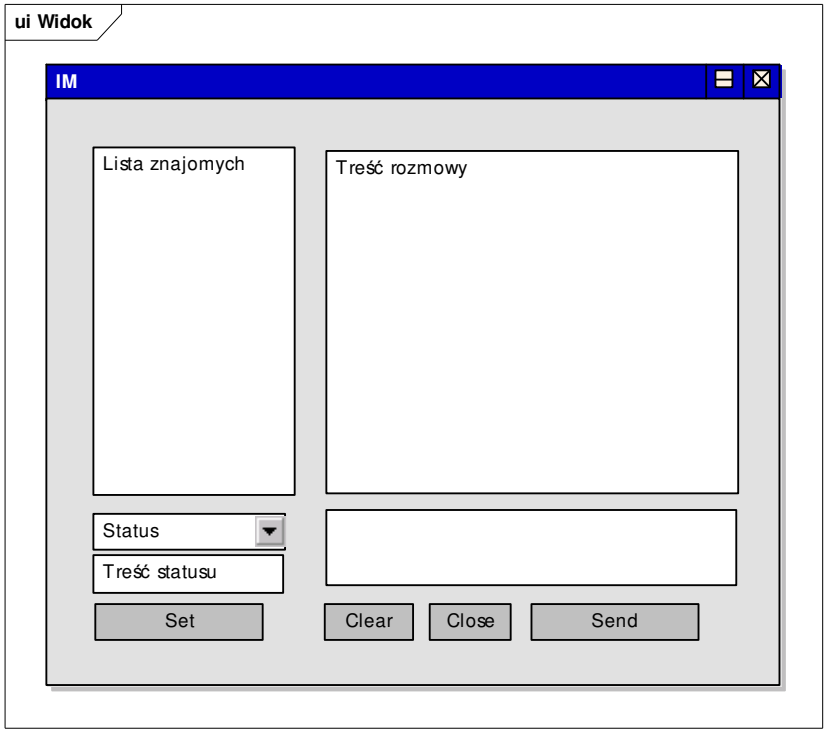
Rysunek 27 – Uproszczony diagram klas biblioteki agregującej komunikatory

5.3 Klient dostępny z poziomu WWW

Klient dostępny z poziomu WWW jest elementem architektury aplikacji oznaczonym na diagramie z rysunku 18 jako GUI i jest jednym z kilku możliwych sposobów implementacji analizowanych w rozdziale czwartym.

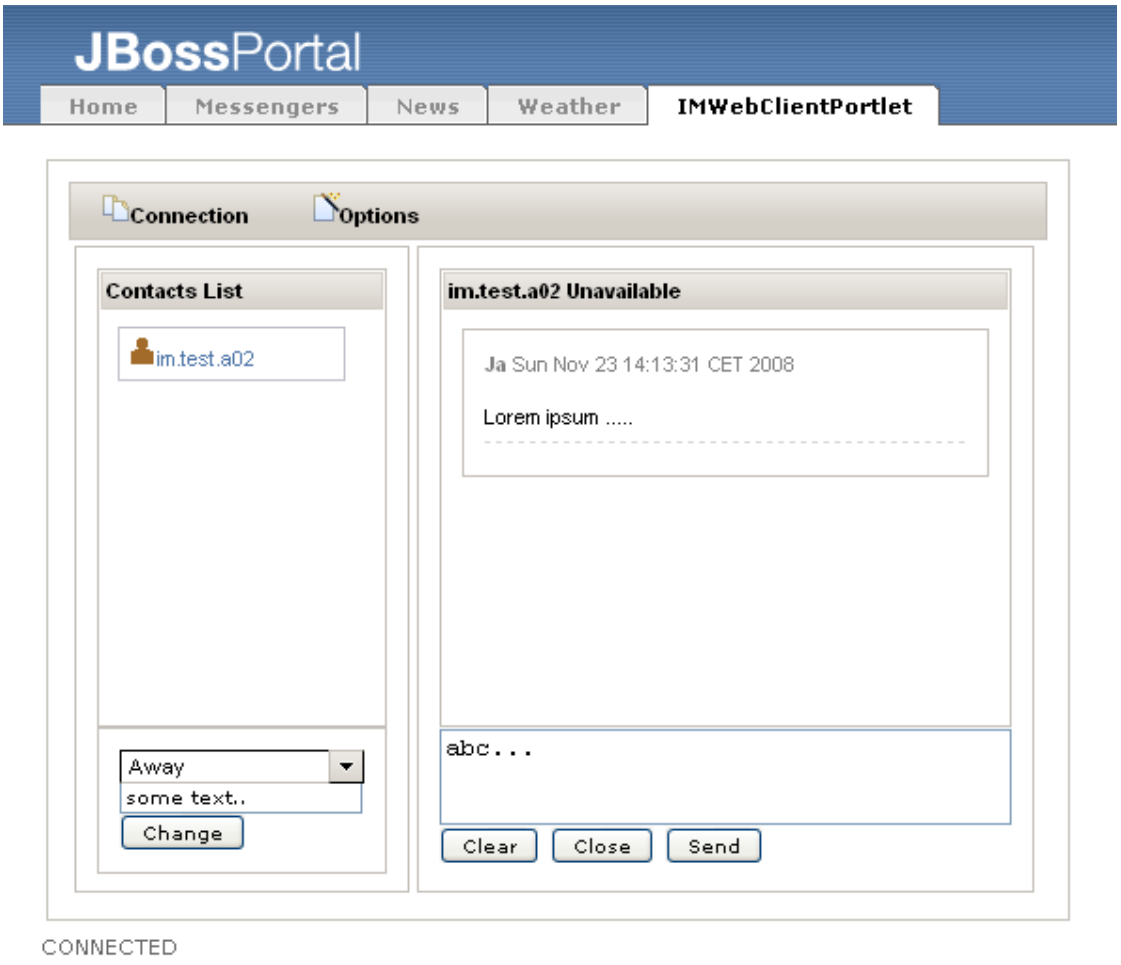
5.3.1 Interfejs użytkownika

Interfejs użytkownika powinien zapewnić wszystkie elementy umożliwiające wygodne prowadzenie rozmowy. Nie powinien także odbiegać wyglądem od tego typu aplikacji co mogłoby zniechęcić użytkownika. W skład interfejsu wchodzą następujące elementy: lista znajomych, okno chatu zawierające treść rozmowy, panel wprowadzania tekstu oraz guziki umożliwiające zamknięcie okna, wyczyszczenie treści rozmowy oraz wysłanie wiadomości. Pod listą znajomych znajdują się elementy umożliwiające zmianę statusu użytkownika.



Rysunek 28 – Zarys interfejsu użytkownika

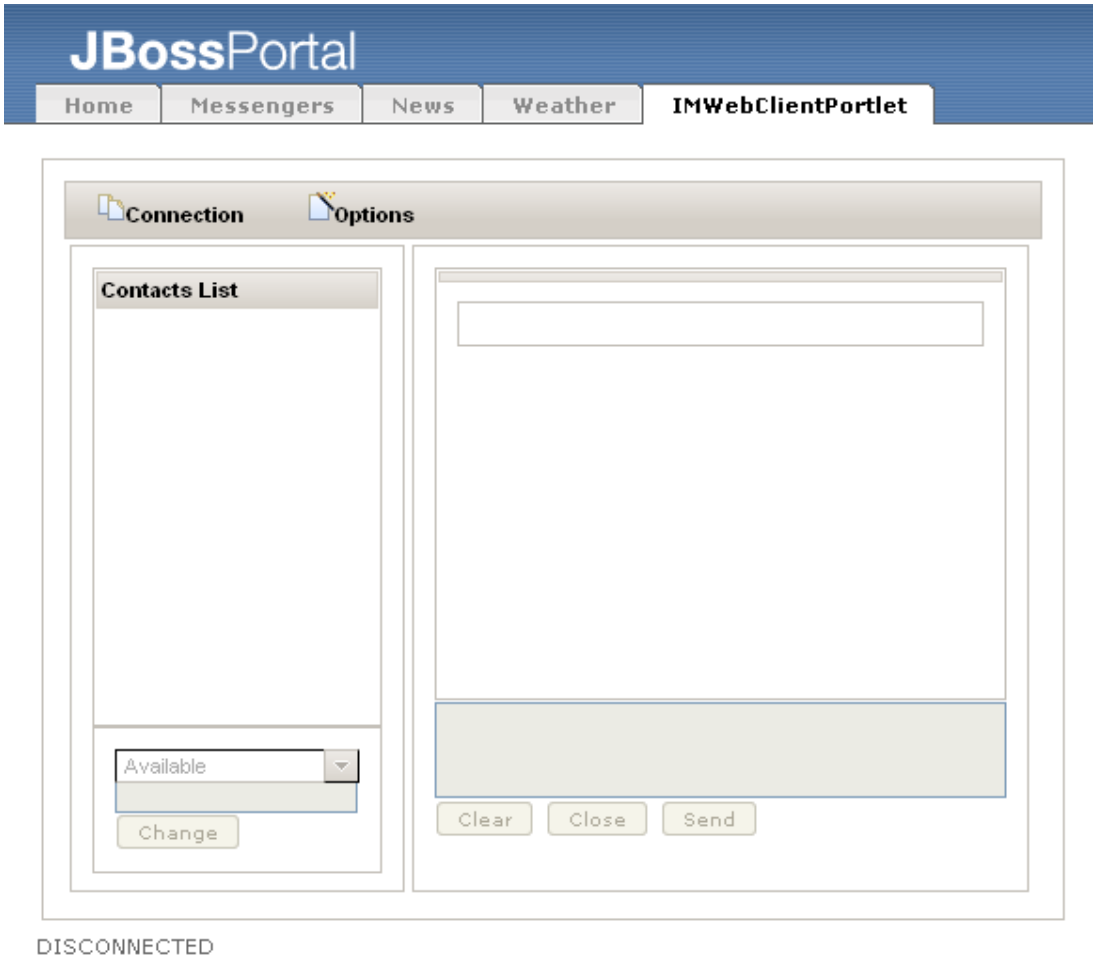
5.3.2 Portlet



Rysunek 29 – Portlet - status połączony (Connected)

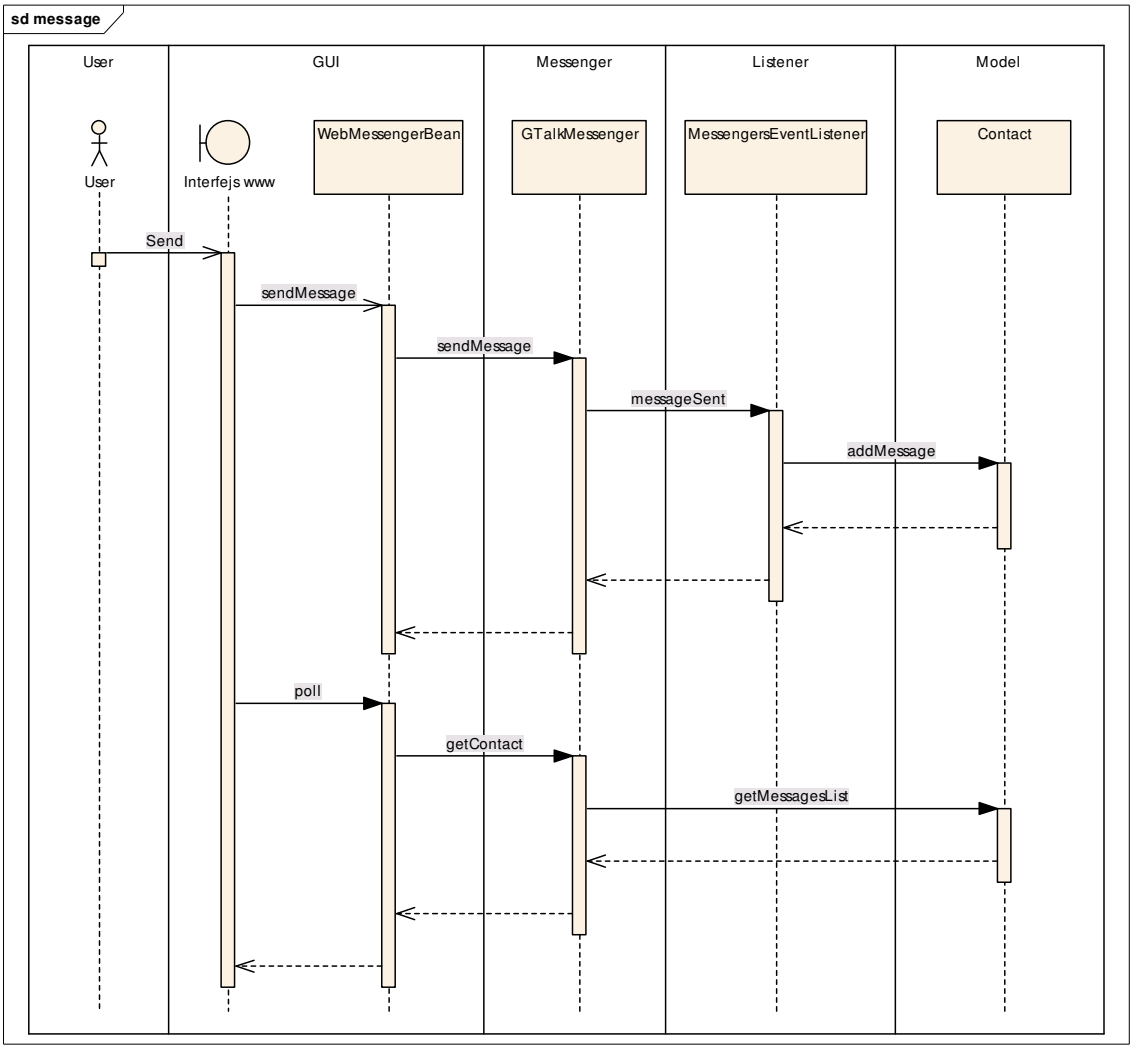
Utworzona w ramach niniejszej pracy aplikacja jest w standardową aplikacją opartą o JSF, wzbogaconą jedynie o pewne interaktywne elementy interfejsu i dzięki wykorzystaniu JBoss Portlet Bridge osadzona w kontenerze portletów jakim jest JBoss Portal. Zgodnie z ideą JSF, aplikacja zbudowana jest z kilku widoków, czyli plików w formacie xhtml. Plik profile.xhtml umożliwia wybór i konfigurację połączenia komunikatora internetowego, chat.xhtml reprezentuje okno chatu z którego użytkownik może wysyłać wiadomości do znajomej osoby. Kolejnym plikiem jest contact.xhtml, który odpowiada za wyświetlenie listy znajomych oraz elementów umożliwiających zmianę statusu osobistego użytkownika. Plik menu.xhtml zawiera menu z którego użytkownik może wybrać opcję umożliwiającą nawiązanie lub

rozwiązanie połączenia a także zmianę obecnej konfiguracji. Dodatkowym plikiem jest log.xhtml, który pokazuje żądania płynące z warstwy prezentacji w formie żądań AJAX i jest on wykorzystywany tylko w fazie implementacji. Wszystkie wymienione widoki, poza log.xhtml, stanowią jedną całość i połączone zostały za pomocą taga <ui:include> występującego w faceletach. Na każdym z widoków znajdują się odpowiednie tagi JSF, RichFaces oraz Ajax4jsf, dzięki którym aplikacja jest w pełni interaktywna. Pliki widoków przejmują całą logikę odpowiedzialną za komunikację z użytkownikiem. Pod warstwą prezentacji znajduje się WebMessengerBean, który jest odpowiedzialny za obsługę wszelkich żądań płynących od użytkownika - warstwy prezentacji. Bean ten wykorzystuje bezpośrednio bibliotekę agregującą komunikatory. Rysunki 29, 30 oraz 32 pokazują aplikację umieszczoną na stronie JBoss Portalu.



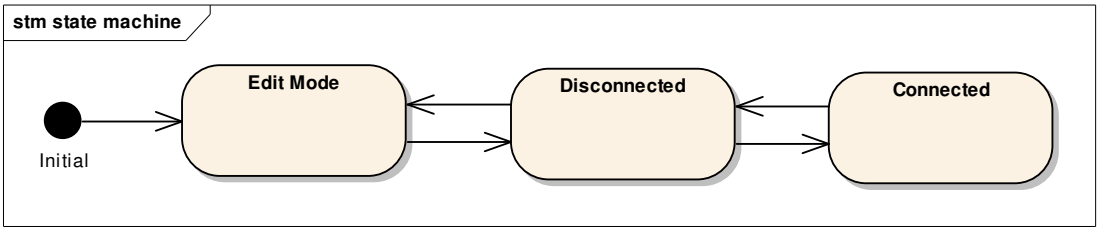
Rysunek 30 – Portlet - status rozłączony (Disconnected)

Diagram sekwencji przedstawiony na rysunku 31 pokazuje sposób, w jaki wszystkie elementy architektury czyli GUI, Messenger, Listener oraz Model współpracują ze sobą tworząc jedną aplikację dostępną z poziomu przeglądarki internetowej. Po kliknięciu przez użytkownika w element interfejsu odpowiedzialny za wysłanie wiadomości, wywoływane są kolejno odpowiednie metody poszczególnych warstw odpowiedzialne za wysłanie wiadomości oraz zaktualizowanie stanu modelu o wysłaną wiadomość. W dolnej części diagramu pokazany jest sposób, w jaki element interfejsu odpowiedzialny za aktualizację danych prezentowanych użytkownikowi odpytuje niższe warstwy, w celu uzyskania najświeższych danych. Wszystkie wywołania metod następują z wykorzystaniem żądań AJAX, stąd aplikacja jest w pełni interaktywna.



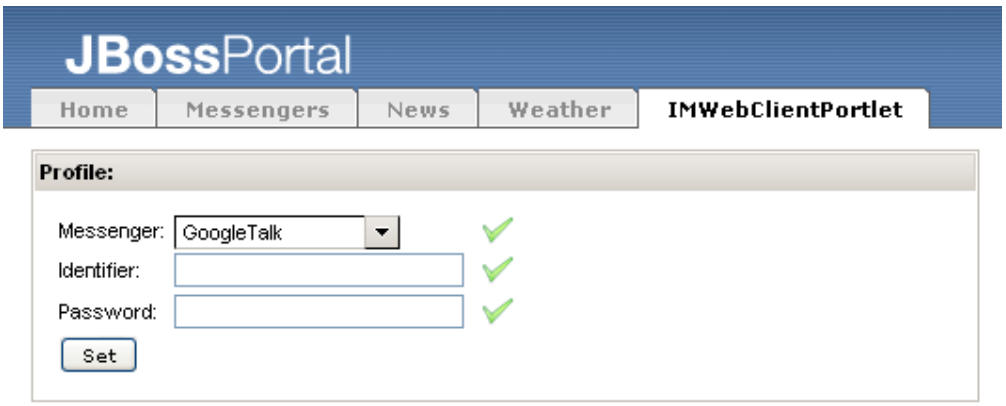
Rysunek 31 – Diagram sekwencji - wysłanie wiadomości

Aplikacja może przyjmować trzy stany, w których poszczególne elementy interfejsu użytkownika mogą być zablokowane oraz nie odświeżane. Dostępne stany widoczne są na rysunku 32.



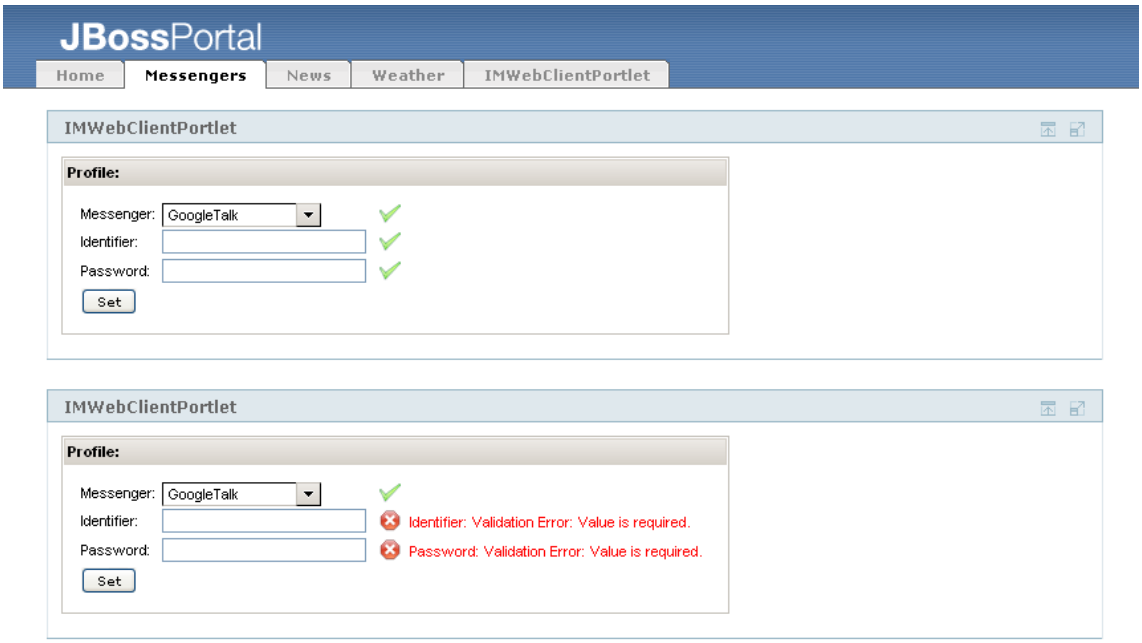
Rysunek 32 – Diagram stanów aplikacji

Wejściowym stanem aplikacji jest tryb edycji profilu (Edit Mode) przedstawiony na rysunku 33, w którym użytkownik ma możliwość wyboru oraz konfiguracji komunikatora. Po poprawnym skonfigurowaniu profilu aplikacja przechodzi w stan rozłączony (Disconnected), który przedstawia rysunek 30. Aplikacja będąca w tym stanie posiada zablokowane odpowiednie elementy interfejsu takie jak okno chat-u, listę kontaktów oraz okno zmiany statusu. Jedynym elementem interfejsu, jaki jest w tym stanie odświeżany jest element odpowiedzialny za prezentacje statusu połączenia. W momencie nawiązania przez komunikator połączenia aplikacja wchodzi w stan połączony (Connected) przedstawiony na rysunku 29 w którym wszystkie elementy interfejsu są dostępne oraz odpowiednio odświeżane.



Rysunek 33 – Tryb edycji profilu (Edit mode)

Z uwagi na to, iż aplikacja jest portletem, może być ona umieszczana w dowolnym miejscu JBoss Portalu, a nawet mogą istnieć dwie lub więcej instancji aplikacji na jednej stronie, które bez problemów mogą się ze sobą komunikować. Przykład takiej konfiguracji przedstawia rysunek 34.



Rysunek 34 – Dwie instancje aplikacji

5.4 Struktura kodu aplikacji

Struktura kodu aplikacji utworzona jest z wykorzystaniem rozwiązań Mavena [30]. Maven to narzędzie pozwalające na automatyzację budowy oprogramowania na platformę Java. Plik określający sposób budowy aplikacji nosi nazwę pom.xml i znajduje się w katalogu głównym każdego z projektów. Jest to dokument w formacie xml opisujący sposób budowania projektu, a także zawierający informację o zależnościach - wykorzystywanych bibliotekach.

Tworzona w ramach niniejszej pracy struktura kodu aplikacji została podzielona na trzy moduły, z których każdy generuje oddzielny artefakt – plik wynikowy. Po zbudowaniu aplikacji otrzymujemy następującą listę tych plików:

- biblioteka agregująca komunikatory – biblioteka w formie pliku jar
- klient Swing – aplikacja testowa w formie pliku jar
- klient WWW – portlet w formie pliku war

6. Weryfikacja rozwiązania i testy

Rozdział ten zawiera opis sposobu weryfikacji i testowania tworzonej w ramach niniejszej pracy aplikacji.

6.1 Testy jednostkowe

Z uwagi na fakt, iż każda oddzielna implementacja komunikatora korzysta z tego samego modelu danych oraz jednego wspólnego Listenera utworzono odpowiednie testy jednostkowe mające na celu weryfikację tworzonego kodu.

Opracowano następujące testy jednostkowe zawarte w pakietach:

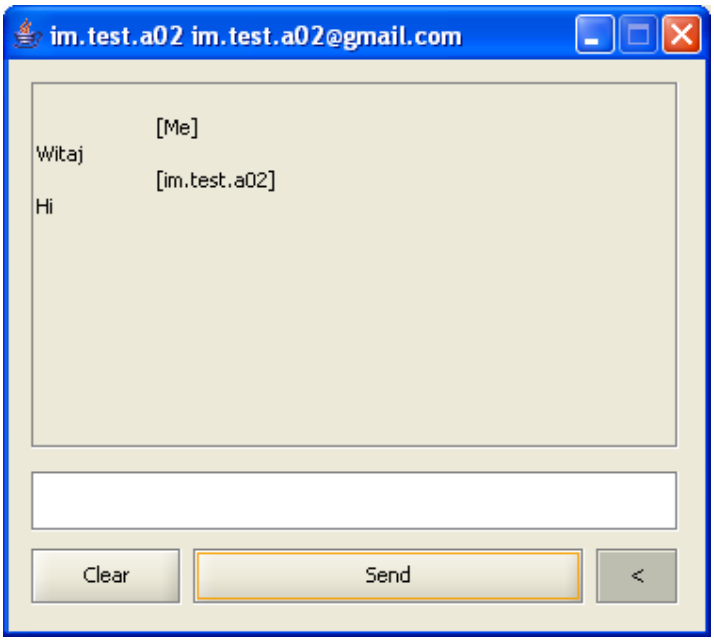
- **im.test** – zawiera testy obiektów MessengerFactory oraz Session,
- **im.test.model** – obejmuje testy obiektów Profile, Contact, Message, MessageAlert oraz PersonalStatus,
- **im.test.messengers** – zawiera test MessengersEventListenera ,
- **im.test.messengers.gg** – zawiera test komunikatora Gadu-Gadu,
- **im.test.messengers.gtalk** – zawiera test komunikatora GoogleTalk,
- **im.test.messengers.jabber** - zawiera test komunikatora Jabber,
- **im.test.messengers.msn** - zawiera test komunikatora Windows Live Messenger.

6.2 Aplikacja testowa

W celu dodatkowej weryfikacji poprawności tworzonej biblioteki powstała aplikacja testowa, oparta o bibliotekę graficzną Swing. Dzięki niej łatwiejsze staje się odtwarzania sytuacji, w których występują rożne niespodziewane zachowania powodujące błędy. Rysunki 32 oraz 33 przedstawiają aplikacje testową. Aplikacja testowa implementuje część architektury odpowiedzialną za GUI. Zbudowana jest z trzech klas:

- **MessengerGUI** – główna klasa sterująca
- **ChatWindow** – klasa reprezentująca okno chatu
- **RefreshMessengerState** – klasa odpowiedzialna za aktualizowanie stanu interfejsu dostępnego dla użytkownika

Rysunek 35 – Aplikacja testowa



Rysunek 36 – Aplikacja testowa, okno chat-u

7. Podsumowanie

Celem niniejszej pracy, było stworzenie interaktywnej aplikacji dostępnej z poziomu przeglądarki internetowej, umożliwiającej komunikację z wybranymi, najbardziej popularnymi protokołami komunikatorów internetowych. Aplikacja internetowa, która została wykonana pozwala na korzystanie z najbardziej popularnych komunikatorów internetowych takich jak Gadu-Gadu, GoogleTalk, Jabber oraz Windows Live Messenger i jest ona nazwana w pracy dyplomowej wieloprotokołowym komunikatorem internetowym. Rozwiązanie to ma na celu przeniesienie stosowanych powszechnie komunikatorów z poziomu systemu operacyjnego na poziom przeglądarki internetowej. Założony cel został osiągnięty. Stworzona aplikacja jest portletem umieszczonym w systemie JBoss Portal, a dzięki wykorzystaniu technologii AJAX jest ona w pełni interaktywna.

Wieloprotokołowy komunikator internetowy – aplikacja przedstawiona w przedmiotowej pracy dyplomowej, może być rozwijana poprzez wprowadzanie personalizacji użytkowników, archiwizacji prowadzonych rozmów oraz integracji z LDAP[2]. Użytkownik logując się do portalu byłby automatycznie logowany do wybranego komunikatora działającego wewnątrz firmy np.: Jabber. Ponadto, użytkownik mógłby komunikować się tylko z wybranymi osobami w zależności od potrzeb wynikających np.: ze struktury organizacyjnej firmy lub stanowiska jakie zajmuje. W przypadku obsługi klientów zewnętrznych, istnieje możliwość wprowadzenia archiwizacji oraz łatwego przeglądania prowadzonych przez pracownika rozmów. Wykonana aplikacja nie zamyka możliwości przedstawionych rozwiązań, a jest otwarta na dalsze rozwijanie i udoskonalanie.

8. Literatura

[1] John W. Rittinghouse, James F. Ransome - „IM Instant Messaging Security" – Elsevier Digital Press 2005

[2] Karol Krysiak – „Sieci komputerowe" – Wydanie II Helion 2005

[3] M. Day, J. Rosenberg, H. Sugano - RFC 2778 - http://www.ietf.org/rfc/rfc2778.txt

[4] M. Day, S. Aggarwal, G. Mohr, J. Vincent - RFC 2779 – http://www.ietf.org/rfc/rfc2779.txt

[5] CTSS - http://www.multicians.org/thvv/7094.html

[6] Multics - http://www.multicians.org/

[7] Model komunikacji peer-to-peer – http://www.idea-group.com/downloads/excerpts/Subramanian01.pdf

[8] talk - http://en.wikipedia.org/wiki/Talk_(software)

[9] Architektura klient-serwer - http://www.sei.cmu.edu/str/descriptions/clientserver_body.html

[10] Komunikator talker – http://en.wikipedia.org/wiki/Talker

[11] Komunikator IRC - http://www.irc.org/

[12] mIRC klient IRC dla Windows – http://www.aliendownload.com/files/screens/2228_mirc.gif

[13] Usługa sieciowa Quantum Link – http://upload.wikimedia.org/wikipedia/en/a/a1/Qlink-mainmenu.png

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler – RFC 3261 – "SIP: Session Initiation Protocol" - Czerwiec 2002 - http://tools.ietf.org/html/rfc3261

[15] Simple Working Group - http://www.ietf.org/html.charters/simple-charter.html

[16] P. Saint-Andre - RFC 3920 - "Extensible Messaging and Presence Protocol (XMPP): Core" - Jabber Software Foundation, Październik 2004 – http://tools.ietf.org/html/rfc3920

[17] P. Saint-Andre - RFC 3921 - "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence" - Jabber Software Foundation, Październik 2004 - http://tools.ietf.org/html/rfc3921

[18] P. Saint-Andre - RFC 3922 - "Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)" - Jabber Software Foundation, Październik 2004 - http://tools.ietf.org/html/rfc3922

[19] P. Saint-Andre - RFC 3923 - "End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)" - Jabber Software Foundation, Październik 2004 - http://tools.ietf.org/html/rfc3923

[20] Vademecum teleinformatyka tom III - IDG Poland S.A. 2004

[21] Vademecum teleinformatyka tom I - IDG Poland S.A. 1999

[22] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson – RFC 1889 – "RTP: A Transport Protocol for Real-Time Applications", Styczeń 1996 http://tools.ietf.org/html/rfc1889

[23] B. Campbell, R. Mahy, C. Jennings - RFC 4975 – http://www.rfc-editor.org/rfc/rfc4975.txt

[24] JBoss Portal - http://www.jboss.org/jbossportal/

[25] Java Web Start – http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp

[26] JSR 168: Portlet Specification - http://www.jcp.org/en/jsr/detail?id=168

[27] JSR 286: Portlet Specification 2.0 - http://www.jcp.org/en/jsr/detail?id=286

[28] JSR 252: JavaServer Faces 1.2 - http://www.jcp.org/en/jsr/detail?id=252

[29] JSR 301: Portlet Bridge Specification for JavaServerTM Faces – http://jcp.org/en/jsr/detail?id=301

[30] Apache Maven - http://maven.apache.org/

Dodatek A

Instalacja i konfiguracja

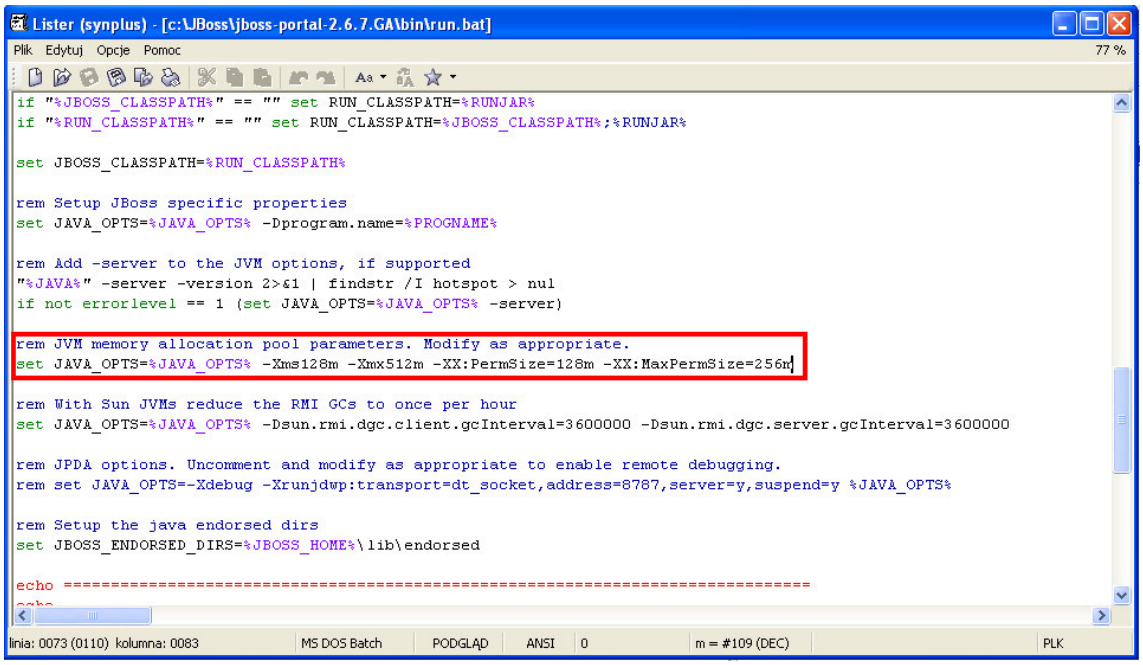
Do uruchomienia aplikacji wymagany jest JBoss Portal w wersji 2.6.7 GA, który można pobrać z witryny firmy JBoss znajdującej się pod adresem:
http://www.jboss.org/jbossportal/download/index.html

Istnieją dwie dystrybucje portalu: jedna wraz z serwerem aplikacji, natomiast druga bez serwera, w przypadku gdy takowy serwer został zainstalowany oddzielnie.

Przed uruchomieniem serwera aplikacji należy zadbać aby w skrypcie uruchomieniowym czyli pliku run.bat lub run.sh w zależności od systemu operacyjnego w którym uruchamiany jest serwer zwiększyć przydział pamięci perm. Aby tego dokonać należy podmienić stosowną linijkę na:

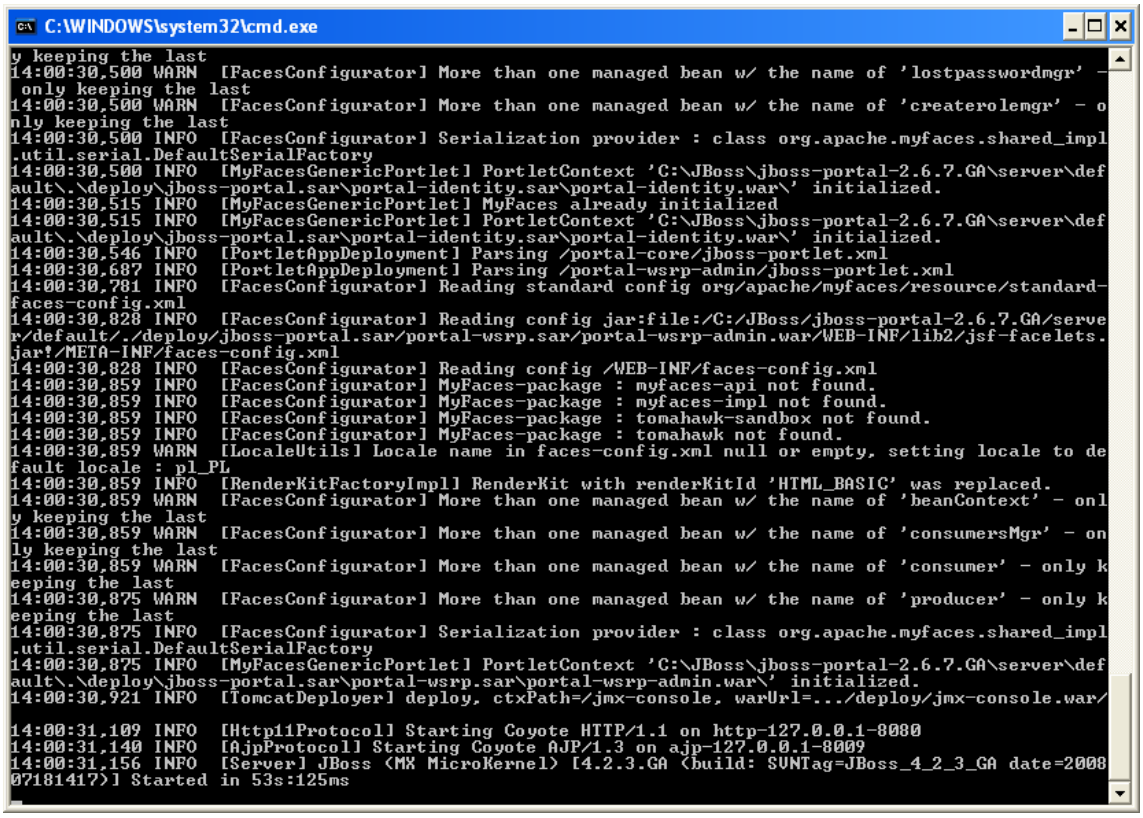
rem JVM memory allocation pool parameters. Modify as appropriate.
set JAVA_OPTS=%JAVA_OPTS% -Xms128m -Xmx512m -XX:PermSize=128m -XX:MaxPermSize=256m

co przedstawia rysunek 37.



Rysunek 37 – Konfiguracja JBoss Portal

Następnie za pomocą tak skonfigurowanego pliku należy uruchomić serwer co przedstawia rysunek 38.

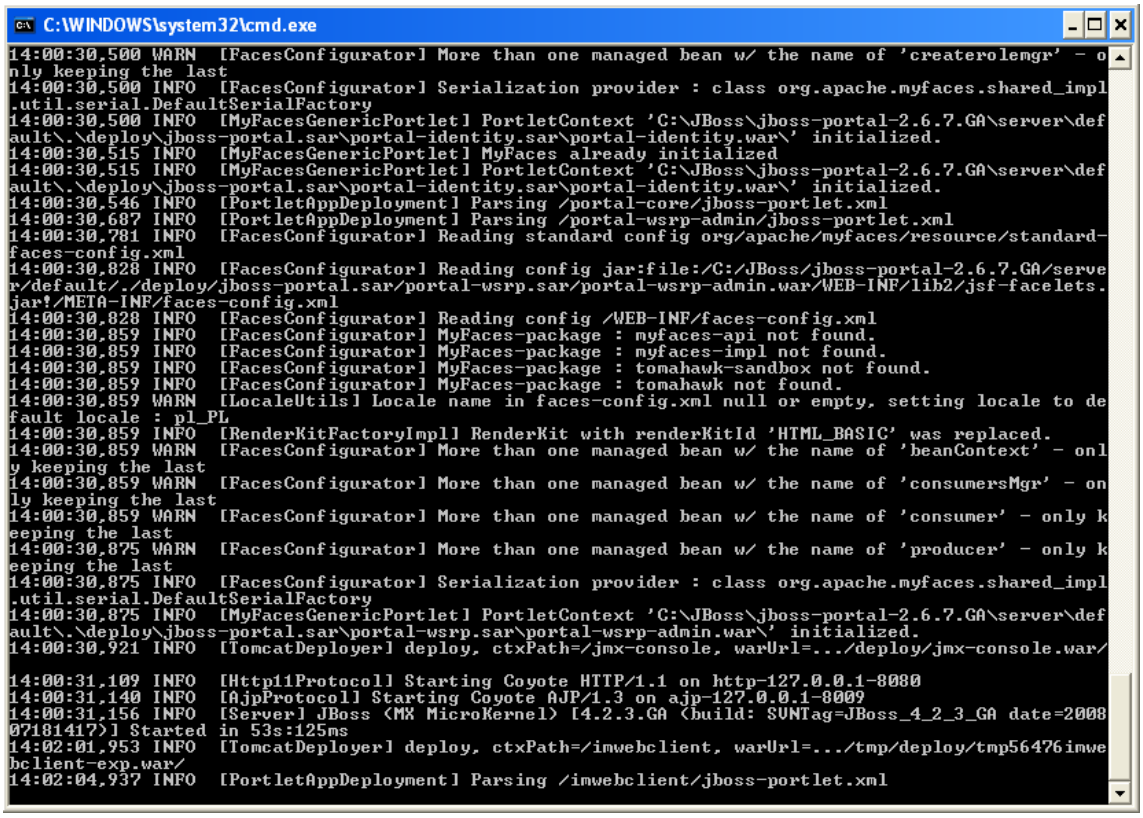


Rysunek 38 – Start serwera JBoss

Po uruchomieniu serwera należy przegrać plik imwebclient.war do katalogu deploy serwera. Po poprawnym zainicjowaniu aplikacji, w logach serwera powinien ukazać się następujący wpis:

20:14:10,734 INFO [TomcatDeployer] deploy, ctxPath=/imwebclient,
warUrl=.../tmp/deploy/tmp52106imwebclient-exp.war/
20:14:14,593 INFO [PortletAppDeployment] Parsing /imwebclient/jboss-portlet.xml

Dokładniej zilustrowany na rysunku 39.



Rysunek 39 – Deploy aplikacji

Jeśli nie zarejestrowano żadnych wyjątków można uruchomić portal za pomocą przeglądarki wpisując adres http://localhost:8080/portal lub bezpośrednio odwołać się do portletu pod adresem: http://localhost:8080/portal/portal/default/IMWebClientPortlet

Kolejne instancje aplikacji można umieszczać w dowolnym miejscu portalu w sąsiedztwie innych portletów.

Konta testowe

GoogleTalk

Lp.	Login	Hasło
1	im.test.a01@gmail.com	qweqwea01
2	im.test.a02@gmail.com	qweqwea02

Gadu-Gadu

Lp.	UIN	Hasło
1	11346110	qweqwe
2	13410480	qweqwe

Jabber

Lp.	Login	Hasło
1	im.test.a01@jabberpl.org	qweqwea01
2	im.test.a02@jabberpl.org	qweqwea02

Windows Live Messenger

Lp.	Login	Hasło
1	im.test.01@live.co.uk	wlm123
2	im.test.02@live.co.uk	wlm123

Słownik akronimów

AJAX - (ang. **A**synchronous **J**avaScript **a**nd **X**ML) – Asynchroniczny JavaScript i XML, technika tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się bez przeładowywania całego dokumentu.

CTSS – (ang. **C**ompatible **T**ime-**S**haring **S**ystem)**,** jeden z pierwszych systemów operacyjnych z podziałem czasu opracowany na MIT (ang. Massachusetts Institute of Technology) i pierwszy raz zademonstrowany w 1961 roku.

Java Web Start – jest to framework pozwalający na uruchomienie aplikacji na platformie Java bezpośrednio z Internetu za pomocą przeglądarki internetowej.

Klient – Serwer – model komunikacji w sieci komputerowej. W architekturze tej mamy do czynienia z rozdzieleniem funkcjonalności pomiędzy serwer – dostawcę usług, a klientów – odbiorców usług. Klienci w celu zaspokojenia swoich potrzeb komunikują się za pomocą żądań wysyłanych do serwera.

LDAP – (ang. Lightweight Directory Access Protocol) - jest to protokół przeznaczony do dostępu do usług katalogowych. Pracuje w oparciu o protokół TCP/IP stąd jest bardzo szybki. Przechowywane dane grupowane są w strukturze przypominającej drzewo katalogów. Każdy obiekt jest jednoznacznie identyfikowany poprzez swoje położenie w drzewie.

MSRP - (ang. The Message Session Relay Protocol) jest to protokół odpowiadający za transmisję serii powiązanych ze sobą wiadomości w kontekście sesji. Sesja wiadomości jest traktowana jak każdy innym strumień danych i ustanawiana jest poprzez dowolny protokół zestawiania sesji Np.: SIP.

Multics – (ang. **Mult**iplexed **I**nformation and **C**omputing **S**ervice) - system operacyjny z podziałem czasu, jest kontynuacją prac nad systemem CTSS spowodowanych bardzo dużą popularnością tego typu systemów w tym czasie i dostosowywaniem ich do nowo powstałych maszyn.

Peer-to-Peer (P2P) - (ang. peer-to-peer - równy z równym) - model komunikacji w sieci komputerowej, który gwarantuje obydwu stronom równorzędne prawa (w przeciwieństwie do modelu klient-serwer). W sieciach P2P każdy komputer może jednocześnie pełnić zarówno funkcję klienta, jak i serwera. W najpopularniejszej implementacji modelu P2P, jaką są programy do wymiany plików w Internecie każdy węzeł sieci (czyli komputer użytkownika) odgrywa rolę serwera przyjmując połączenia od innych użytkowników danej sieci, jak i klienta, łącząc się i pobierając dane z innych maszyn działających w tej samej sieci. Wymiana danych jest zawsze prowadzona bez pośrednictwa centralnego serwera. Sieć P2P charakteryzuje się także płynną strukturą, która zmienia się w zależności od tego, jakie komputery są w niej aktualnie zalogowane.

Portlet - jest to niezależny komponent nadający się do umieszczenia na stronie www. Portlet jest umieszczany w kontenerze portletów, który agreguje zawartość prezentowanej strony. Celem portletów jest stworzenie programu, który będzie uniezależniony od kontenera na którym będzie uruchamiany co stwarza możliwość jego wielokrotnego użycia.

PSTN - (ang. Public Switched Telephone Network) publiczna komutowana sieć telefoniczna. Początkowo wykorzystywała technologie analogowe, obecnie prawie w całości zbudowana w oparciu o technologie cyfrowe.

SIMPLE – (ang. SIP for IM and Presence Leveraging Extensions) - jest to protokół zapewniający dwie podstawowe usługi komunikatorów internetowych czyli usługę wysyłania wiadomości oraz usługę obecności. Powstał w oparciu o protokół SIP.

SIP – (ang. Session Initiation Protocol) - jest protokołem służącym do inicjowania sesji pomiędzy punktami końcowymi sieci IP. Został zaproponowany przez organizację IETF i jest obecnie dominującym protokołem sygnalizacyjnym dla sieci VoIP.

Swing - biblioteka graficzna używana w języku programowania Java, upubliczniona w lipcu roku 1997. Jest nowszą, ulepszoną wersją biblioteki AWT.

VoIP - (ang. Voice over Internet Protocol) technologia cyfrowa umożliwiająca przesyłanie dźwięków mowy za pomocą łączy internetowych lub dedykowanych sieci wykorzystujących protokół IP, popularnie nazywana "telefonią internetową". Dane przesyłane są przy użyciu

protokołu IP, co pozwala wykluczyć niepotrzebne "połączenie ciągłe" i Np.: wymianę informacji gdy rozmówcy milczą.

XMPP – (ang. Extensible Messaging and Presence Protocol) – protokół bazujący na języku XML umożliwiający przesyłanie w czasie rzeczywistym wiadomości oraz statusu.