AKADEMIA GÓRNICZO – HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI

KATEDRA INFORMATYKI

PRACA DYPLOMOWA MAGISTERSKA

ROZSZERZALNA PLATFORMA DO EFEKTYWNEJ, INTERAKTYWNEJ KOMUNIKACJI W HETEROGENICZNYM ŚRODOWISKU SIECIOWYM

IMIĘ I NAZWISKO: **MIROSŁAW JEDYNAK, TOMASZ MASTERNAK**

KIERUNEK STUDIÓW: **INFORMATYKA**

OPIEKUN PRACY: **DR INŻ. ŁUKASZ CZEKIERDA**

Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie i że nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

....

Spis treści

Wstęp

W ciągu ostatnich kilkunastu lat Internet rozwijał się w bardzo szybkim tempie. Z narzędzia używanego w kręgach uniwersyteckich oraz handlowych stał się popularnym, ogólnodostępnym, interaktywnym medium rywalizującym ze środkami masowego przekazu.

Już w latach sześćdziesiątych tworząc sieć ARPANET jej autorzy mieli na celu zapewnienie sprawnej komunikacji, nawet w przypadku częściowego uszkodzenia infrastruktury sieci. Późniejsze wprowadzenie powszechnej i łatwo dostępnej poczty elektronicznej było kolejnym krokiem w kierunku jeszcze szybszej komunikacji. Wysłane w ten sposób informacje mogły docierać na drugi koniec globu w czasie mierzonym w minutach, który w porównaniu do czasu przesyłania tradycyjnej poczty wydawał się wtedy bardzo krótki.

Następnym etapem w rozwoju komunikacji z wykorzystaniem Internetu było upowszechnienie się komunikatorów internetowych (ang. *Instant messaging*) w latach dziewięćdziesiątych. Aplikacje takie jak AIM (ang. *America On Line Instant Messenger*), IRC (ang. *Internet Relay Chat*), MSN (ang. *Microsoft Network*) czy Gadu-Gadu umożliwiały przesyłanie wpisanego tekstu do innych użytkowników. Opóźnienie przesyłania wiadomości w tego typu systemach mierzone w pojedynczych sekundach sprawiało wrażenie natychmiastowej komunikacji.

Pod koniec XX wieku wraz ze wzrostem możliwości technicznych i zwiększaniem przepustowości łącz, wymagania użytkowników nadal rosły. W Internecie coraz powszechniejszy stawał się statyczny obraz, dźwięk oraz obraz ruchomy. Oczekiwaniem użytkowników stała się komunikacja, dzięki której praktycznie zniknęłaby bariera odległości – komunikacja, która sprawia wrażenie rzeczywistego współdziałania. Odpowiedzią na te wymagania były łatwo dostępne systemy telekonferencyjne łączące obraz i dźwięk, w których opóźnienie przesyłania danych rzędu milisekund było niezauważalne dla użytkowników.

Ponieważ Internet jest w dzisiejszych czasach narzędziem stosowanym w codziennej pracy, systemy muszą umożliwiać zdalną współpracę, która nie ogranicza się tylko do rozmowy, ale jest również związana ze współdzielonym wirtualnym stanem, na który wpływ mają wszyscy uczestnicy – coraz bardziej popularne są wirtualne tablice, współdzielone dokumenty etc.

Celem niniejszej pracy jest przedstawienie architektury oraz implementacji platformy IGCP. Nazwa powstała z pierwszych liter słów *Interactive Group Communication Platform*. Głównymi założeniami tworzonego rozwiązania było zapewnienie interaktywnej komunikacji. Funkcjonalność ta jest coraz częściej oczekiwana przez

użytkowników, którzy wykorzystują Internet do pracy grupowej. Ponadto udostępniana w ramach platformy usługa przechowywania stanu wprowadza więcej możliwości wykorzystania w porównaniu do tradycyjnych wideokonferencji. Zachowany stan może być później wykorzystany do odtworzenia przebiegu zdarzeń oraz umożliwia lepszą optymalizację strumienia przesyłanych danych.

Omawiane rozwiązanie powstawało jako element systemu zdalnych telekonsultacji medycznych TeleDICOM [34], pełniąc w nim rolę warstwy komunikacyjnej. Dzięki elastycznej architekturze wykorzystanie platformy nie jest ograniczone wyłącznie do tego zastosowania. Rozwiązanie może być użyte w wielu innych aplikacjach opartych o zdalną współpracę.

Ze względu na charakter pracy użytkowników architektura i większość mechanizmów była projektowana z myślą o pracy w środowisku rozproszonym. Często rozwiązania, które prawidłowo zachowują się w sieciach lokalnych, w czasie użytkowania ich w sieciach rozległych sprawiają problemy. Zaprojektowanie IGCP w taki sposób, aby efektywnie działała w Internecie, było kolejnym wyzwaniem. Z tego powodu pewne decyzje projektowe zostały podjęte już we wczesnej fazie specyfikacji wymagań – dzięki temu platforma jest odporna na chwilową utratę łączności, po której następuje uruchomienie mechanizmu odtwarzania stanu odpowiedzialnego za zachowanie spójności. Klienci mogą dołączać i odłączać się w trakcie pracy pozostałych. Szczególny nacisk położony został na ograniczenie czasu odpowiedzi systemu.

Aktualnie użytkownikom coraz częściej nie wystarcza już korzystanie z Internetu za pośrednictwem zwykłych stacji roboczych w pracy czy w domu. Coraz częściej używane są urządzenia przenośne począwszy od laptopów przez urządzania typu PDA, aż po zaawansowane telefony komórkowe wykorzystujące komunikację WiFi. Z różnorodności urządzeń wynika różnorodność możliwych implementacji: od desktopowych systemów operacyjnych jak Windows czy Linux, po specjalizowane dla urządzeń przenośnych jak Symbian czy Java Mobile Edition. Aby zapewnić możliwość integracji z różnymi platformami sprzętowymi i systemowymi, IGCP korzysta ze środowiska ICE jako warstwy komunikacji – specyfikacja interfejsu niezależna od języku programowania umożliwia korzystanie z platformy w każdym ze wspieranych języków.

Struktura pracy

W pierwszym rozdziale zostaną omówione systemy pracy grupowej (ang. *groupware*) zapewniające interaktywną komunikację, przez co umożliwiają wygodną współpracę zdalnych użytkowników. Przedstawione zostaną podstawowe zagadnienia, takie jak przechowywanie stanu czy sposoby dystrybucji informacji pomiędzy uczestników. Następnie poruszone zostaną bardziej szczegółowe aspekty związane z zachowaniem

spójności danych, kontrolą przeciążenia czy wydajnością komunikacji. Przegląd rozwiązań stosowanych w systemach o podobnych wymaganiach służy przedstawieniu możliwych podejść oraz omówieniu ich wad oraz zalet.

Drugi rozdział rozpoczyna przedstawienie wymagań stawianych tworzonemu systemowi, na podstawie których w dalszej części prowadzona będzie analiza docelowego systemu. Celem rozdziału jest wybór najlepszego modelu komunikacji oraz sposobu przechowywania stanu, bazując na informacjach przedstawionych w pierwszym rozdziale oraz wymaganiach stawianych tworzonej platformie. Przedstawione wymagania podzielone zostały na techniczne, tzn. związane ze środowiskiem pracy oraz nietechniczne związane z oczekiwanymi możliwościami. Analiza rozpoczyna się od prostego abstrakcyjnego modelu spełniającego minimalne wymagania. W dalszych krokach model wzbogacany jest o kolejne cechy oraz opis ich logicznej bądź też fizycznej realizacji. W trakcie analizy przedstawiane są alternatywne rozwiązania oraz uzasadnienie podjętych przez autorów decyzji.

Trzeci rozdział przedstawia logiczną strukturę rozwiązania oraz sieć powiązań pomiędzy jego elementami. Opisywany model został opracowany zgodnie z podejściem MDA (ang. *Model Driven Architecture)* tzn. w sposób niezależny od implementacji (PIM ang. *Platform Independent Model*). Pomimo tej niezależności rozdział zawiera omówienie aspektów technicznych mających szczególny wpływ na efektywność oraz interaktywność komunikacji – najważniejsze wymagania stawiane platformie.

Kolejne dwa rozdziały przedstawiają system z punktu widzenia PSM (ang. *Platform Specific Model*), opisując rozwiązania implementacyjne. W pierwszym z nich zostaną omówione interfejsy komunikacyjne oraz ważniejsze elementy systemu związane z przetwarzaniem informacji na różnym etapie komunikacji. Objaśnione zostaną również użyte mechanizmy oraz architektura systemu. W następnym zostanie szczegółowo przedstawiona koncepcja związana z szacowaniem przepustowości łącza oraz kontrolą i zapobieganiem powstawaniu zatorów.

W końcowej części pracy zostaną zweryfikowane postawione założenia i wymagania. Na podstawie szeregu testów zostaną przedstawione wartości opisujące zachowanie i wydajność systemu. Zmiennymi w badaniach będą między innymi ilość użytkowników i charakter przesyłanych danych, a wyznaczanymi parametrami czas odpowiedzi, maksymalna przepustowość systemu czy obciążenie serwera.

Niniejsza praca jest dziełem dwóch autorów. Większy wkład w treść rozdziałów 4 i 6 miał Mirosław Jedynak, a rozdziałów 2 oraz 5 Tomasz Masternak. Pozostałe trzy (1, 3 i 7) zostały opracowane wspólnie.

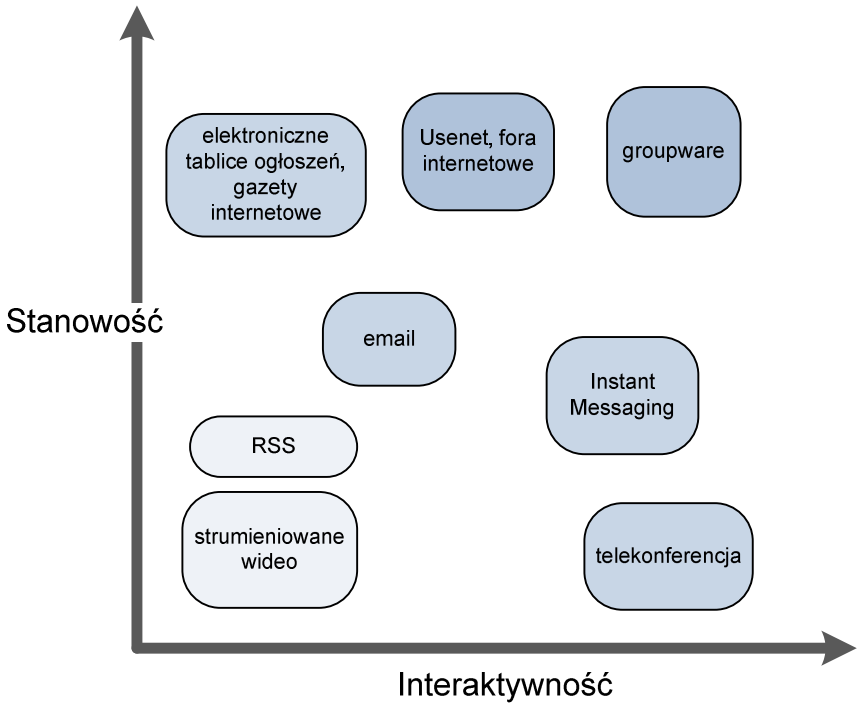
Umiejscowienie projektowanej platformy wśród systemów zdalnej komunikacji

U podstaw działania systemów takich jak Wiki, fora internetowe czy komunikatory leży wymiana informacji pomiędzy użytkownikami. Charakter komunikacji oraz gromadzenie jej historii mogą posłużyć jako cechy klasyfikujące. Każdy z powyższych systemów można rozpatrywać pod kątem interaktywności komunikacji oraz przechowywania stanu.

Przez interaktywność komunikacji rozumiemy z jednej strony przyczynowo-skutkowy związek pomiędzy wiadomościami wysyłanymi przez użytkowników, z drugiej zaś opóźnienie w dostarczaniu komunikatów. Należy zauważyć, że omawianą własność można stopniować: systemy zdalnej współpracy cechuje duża interaktywność komunikacji, natomiast wiadomości przesyłane w ramach forum internetowego charakteryzują się znacznie mniejszą. Zapewnienie interaktywnej komunikacji nie jest wymagane we wszystkich zastosowaniach – np. strumieniowanie wideo na żądanie (ang. *video on demand*). O ile krótki czasu dystrybucji wiadomości w niektórych przypadkach ma krytyczny wpływ na interaktywność (jak np. w systemach telefonii internetowej), to należy zdawać sobie sprawę, że nie zawsze tak jest. Przykładowo system poczty elektronicznej nie został zaprojektowany do pracy w takim trybie i kilkukrotne zwolnienie działania prawdopodobnie pozostałoby niezauważone przez większość użytkowników.

Systemy komunikacji można rozpatrywać także pod kątem stanowości. W skrajnej sytuacji wiadomości mogą być zupełnie ulotne (np. RSS), przeciwieństwo zaś stanowią elektroniczne tablice ogłoszeniowe przechowujące abstrakcję całej dotychczasowej komunikacji w postaci wspólnego dla wszystkich użytkowników stanu.

Podobnie jak interaktywność, zdolność do przechowywania stanu nie ma charakteru dychotomicznego. Oznacza to, że aktualnie przechowywany stan może zależeć od wszystkich dotychczas otrzymanych wiadomości, od ich części lub może nie występować wcale. Przykładowo telekonferencja, które nie jest nagrywana z wykorzystaniem mechanizmów z poza systemu nie posiada stanu, poczta elektroniczna jest przechowywana na serwerze dopóki nie zostanie pobrana przez użytkownika, natomiast wiadomości na forum przechowywane są zdecydowanie dłużej. Rysunek 1 schematycznie przedstawia powyższą klasyfikację.



Rysunek 1. Klasyfikacja systemów ze względu na stanowość i interaktywność.

Tworzony przez autorów system lokuje się w prawym górnym rogu przedstawionego diagramu, tzn. można go zaliczyć do systemów zdalnej współpracy. Z tego powodu ich koncepcja oraz typowe funkcjonalności systemów tej klasy zostaną omówione dokładniej.

Systemy zdalnej współpracy (ang. *groupware*) zostały stworzone z myślą o grupach użytkowników komunikujących się za pośrednictwem sieci komputerowej. Użytkownicy współpracują w ramach sesji obejmującej wszystkich uczestników oraz wspólne zasoby, którymi operują w celu osiągnięcia wyznaczonego przez siebie celu. Czas trwania sesji ustalany jest arbitralnie, lecz w większości przypadków system umożliwia utworzenie sesji na potrzeby współpracy oraz jej zakończenie w przypadku, gdy użytkownicy zadecydują, że powinna zostać zakończona. Dzielone zasoby mogą być różnego typu, np. tablica, na której rysując użytkownicy przedstawiają swoje pomysły (ang. *whiteboarding*), jednocześnie edytowany dokument tekstowy, wizualna reprezentacja badań lekarskich etc. Zasoby oraz ich aktualna postać określają stan sesji, który może ulegać zmianie w wyniku operacji wykonywanych przez użytkowników.

Umiejscowienie systemów zdalnej współpracy na diagramie jednoznacznie wyznacza grupę wymagań, których spełnienie jest niezbędne. Mianowicie oczekuje się, iż system będzie przechowywał stan sesji oraz zapewniał interaktywną komunikację, tzn. reakcje na zdarzenia będą znajdowały swoje odzwierciedlane w stanie sesji na tyle szybko, że będzie możliwe stworzenie iluzji rzeczywistej kooperacji.

Przechowywanie stanu oraz interaktywność, choć pomocne w celach klasyfikacyjnych, stanowią jedynie dwa spośród dużej grupy wymagań. W przypadku rzeczywistych systemów formułowane są także wymagania dotyczące efektywności, skalowalności, odporności na awarie czy wieloplatformowości. Dodatkowo możliwe jest rozszerzanie podstawowych wymagań o warunki dotyczące zachowania platformy w przypadku jednoczesnego korzystania z zasobów, późnego dołączania do sesji etc.

Oczywiście nie istnieje jedno rozwiązanie spełniające wszystkie wymagania – większość z istniejących dąży do spełnienia jedynie ich części. Należy jednocześnie pamiętać, że to wymagania kierują procesem projektowania systemu, mając wpływ na jego architekturę oraz model dystrybucji wiadomości.

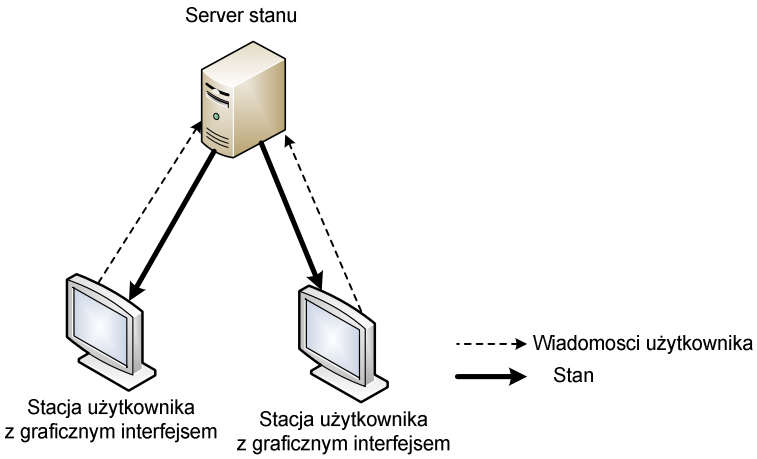
1 Przegląd mechanizmów stosowanych w systemach groupware

W poniższym rozdziale zostaną przedstawione kluczowe zagadnienia związanie z budową platform komunikacji dla potrzeb systemów zdalnej współpracy oraz dotychczasowe rozwiązania napotykanych problemów. Zostaną poruszone kwestie związane z efektywnością komunikacji, kontrolą przeciążenia oraz metodami utrzymywania spójności stanu, gdyż to właśnie te aspekty działania są najistotniejsze z punktu widzenia tworzonej platformy.

1.1 Przechowywanie stanu

Systemy ze współdzielonym stanem można klasyfikować ze względu na sposób jego przechowywania: poza aplikacjami użytkowników lub w postaci lokalnej kopii całego stanu, ewentualnie jego części w ramach każdej z nich [40].

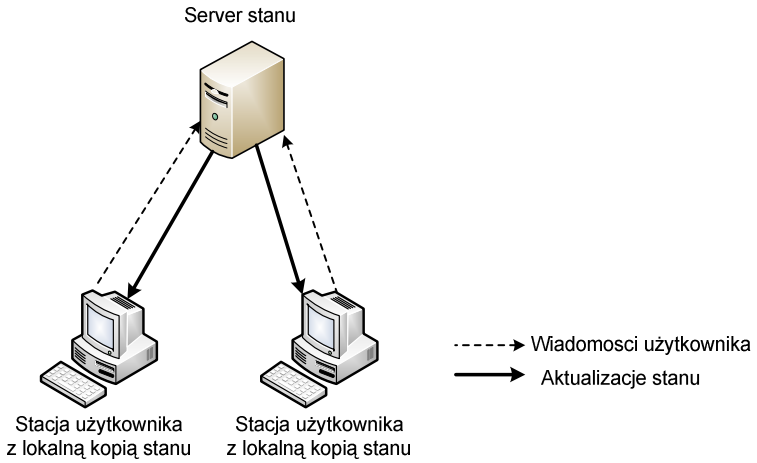
W pierwszym przypadku stan przechowywany jest poza stacjami końcowymi w dobrze znanym miejscu, najczęściej na centralnym serwerze. Użytkownicy otrzymują stan bezpośrednio przed jego prezentacją, nie przechowując jego kopii lokalnie (Rysunek 2). Do grupy tego typu systemów zaliczyć można system Microsoft NetMeeting oparty o rodzinę rekomendacji ITU [13], w których użytkownicy wysyłają zdarzenia generowane za pomocą myszki lub klawiatury, a w zamian otrzymują graficzną reprezentację części pulpitu. Na podobnej zasadzie działa system X Window – nie udostępnia on jednak możliwości jednoczesnej pracy wielu użytkownikom. Podejście polegające na przechowywaniu stanu poza aplikacjami klienckimi w dużym stopniu upraszcza problemy związane z jego współdzieleniem oraz organizacją komunikacji. Z drugiej strony brak lokalnej kopii stanu powoduje, że odpowiedzi na generowane wiadomości widoczne są dopiero po przetworzeniu ich poza stacjami końcowymi, co w znacznym stopniu zwiększa czas oczekiwania. O ile opóźnienie przetwarzania może być dopuszczalne w przypadku, kiedy środowiskiem działania systemu jest lokalna sieć komputerowa, o tyle podejście takie jest często zbyt wolne w przypadku komunikacji poprzez sieć rozległą.



Rysunek 2. Architektura z zewnętrznie przechowywanym stanem.

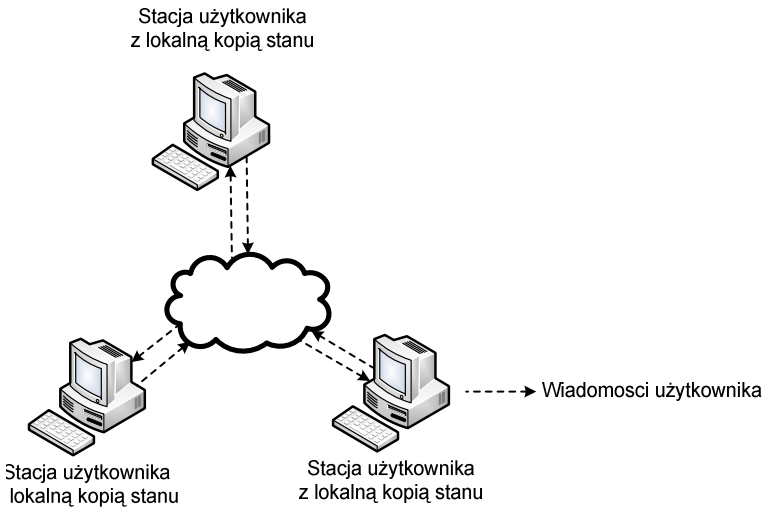
Drugą klasę systemów tworzą rozwiązania, w których stan przechowywany jest zarówno poza stanowiskami końcowymi, jak i w postaci lokalnych replik na każdej stacji końcowej (Rysunek 3). Użytkownicy wysyłają wiadomości na serwer podobnie jak w poprzednim rozwiązaniu, jednak nie muszą czekać z lokalnym przetwarzaniem zdarzeń na swojej kopii. Podobnie jak w poprzednio omówionej architekturze, stan może być również przechowywany poza stacjami końcowymi, np. na centralnym serwerze.

W przypadku systemów wymagających dużej skalowalności oraz przepustowości (wieloużytkownikowe gry sieciowe [7]), stosowane są rozwiązania polegające na zwiększeniu ilości serwerów. Ma to na celu skrócenie czasu dystrybucji komunikatów pomiędzy użytkownikami oraz zmniejszenie obciążenia pojedynczych węzłów. Pomimo swoich zalet, omawiana architektura ma również pewne wady. Istnienie wielu kopii stanu wymusza wprowadzenie mechanizmów synchronizacji w celu rozwiązywania problemów związanych z jednoczesnymi modyfikacjami tych samych zasobów.



Rysunek 3. System z lokalnymi replikami stanu.

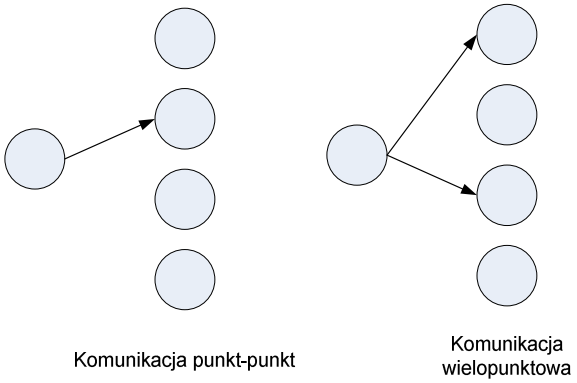
W ramach systemów z lokalną repliką warto omówić te, w których zupełnie zrezygnowano ze stanu przechowywanego zewnętrznie. Podejście takie cechuje bardzo duża skalowalność, lecz ceną, którą należy zapłacić jest zwiększona trudność realizacji komunikacji oraz utrzymywania spójności pomiędzy kopiami. Brak elementów systemu poza aplikacjami klienckimi często oznacza rezygnację z centralnego węzła komunikacji oraz globalnej wersji stanu dostępnej dla nowo dołączających użytkowników (Rysunek 4).



Rysunek 4. System ze zreplikowanym stanem bez centralnego serwera.

1.2 Dystrybucja wiadomości

W każdym systemie komunikacji dane są rozsyłane zgodnie z przyjętym modelem, określającym grupę odbiorców. W systemach zdalnej współpracy najczęściej stosowane modele to komunikacja punkt-punkt oraz wielopunktowa (Rysunek 5). W pierwszym podejściu każda wiadomość ma dokładnie jednego odbiorcę określanego podczas nadawania wiadomości. W drugim modelu wysyłane wiadomości trafiają do grupy odbiorców zainteresowanych otrzymywaniem informacji, tzn. należących do grupy docelowej.



Rysunek 5. Modele komunikacji.

Na poziomie implementacyjnym model definiowany jest w postaci protokołu komunikacyjnego. Określa on sposób zachowania nadawcy oraz odbiorcy, rodzaj przesyłanych informacji oraz sposób działania infrastruktury realizującej komunikację. W ramach protokołu definiowane są także dodatkowe cechy komunikacji związane z niezawodnością, kolejnością dostarczania komunikatów, kontrolą przeciążenia etc.

Protokoły komunikacji punkt-punkt definiowane są na wielu warstwach modelu referencyjnego OSI/ISO. Aplikacje działające w środowisku sieci rozległej nie mogą wykorzystywać specyficznych cech protokołów łącza danych (wynika to heterogenicznego charakteru struktury sieci), dlatego nie będą tutaj szerzej omawianie.

W warstwie sieciowej najczęściej wykorzystywanym protokołem komunikacji punkt-punkt jest IP (ang. *Internet Protocol*). Każda strona korzystająca z IP identyfikowana jest przez adres sieciowy. Nadawca wysyłając wiadomość umieszcza w niej adres odbiorcy, do którego jest kierowana. Za dostarczenie wiadomości do odbiorcy odpowiadają urządzenia warstwy sieciowej (ang. *router*), których działanie polega na przekazywaniu komunikatu do jednego ze swoich sąsiadów, który jest kolejnym elementem najkrótszej ścieżki prowadzącej do odbiorcy. Decyzja o wyborze sąsiada podejmowana jest na podstawie tablic routingu, tworzonych za pomocą protokołów routingu.

IP nie zapewnia niezawodności komunikacji, tzn. wiadomość wysłana przez nadawcę może nie dotrzeć do celu (ang. *best effort*). Standardowe biblioteki nie pozwalają na dostęp bezpośrednio do protokołu IP. W praktyce stosowane są protokoły warstwy wyższej, takie jak TCP (ang. *Transmission Control Protocol*), UDP (ang. *User Datagram Protocol*), SCTP (ang. *Stream Control Transmission Protocol*) etc.

Protokoły punktowe występują także w wyższych warstwach modelu referencyjnego OSI/ISO. W praktyce w systemach zdalnej komunikacji grupowej oprócz rozwiązań bazujących na protokołach warstwy sieciowej lub transportowej, najczęściej spotykane rozwiązania bazują na protokołach poziomu aplikacji.

Model komunikacji wielopunktowej ma swoje realizacje już na poziomie sieciowym w oparciu o IP (ang. *IP multicast*). Dane przesyłane w ramach komunikacji mają postać pakietu IP. Jedyna różnica w porównaniu z wersją jednopunktową polega na interpretacji adresu docelowego, który w tym przypadku oznacza grupę odbiorców. Zarządzanie przynależnością do grupy realizowane jest w warstwie sieciowej. Grupy rozgłoszeniowe, definiowane w postaci adresów sieciowych, mają charakter otwarty, tzn. aby wysłać pakiet skierowany do grupy, nie trzeba być jej członkiem. Nadawca wysyła jeden komunikat, który w miarę potrzeby zostaje zwielokrotniany przez urządzenia warstwy sieciowej i łącza danych znających semantykę przesyłanych informacji, co doprowadza do znacznego zmniejszania narzutu komunikacyjnego. Podobnie jak w przypadku punktowego IP, w celu umożliwienia komunikacji zdefiniowane zostały protokoły routingu multicastowego odpowiedzialne za wyznaczanie optymalnych tras dystrybucji wiadomości pomiędzy członkami grup. Niestety wsparcie dla komunikacji wielopunktowej IP nie jest standardową usługą urządzeń sieciowych tworzących współczesny Internet. Istnieją także problemy z integracją z mechanizmami translacji adresów sieciowych [31].

Komunikacja wielopunktowa może zostać zrealizowana za pomocą dowolnego protokołu punktowego. W takim przypadku nadawca rozsyła wiadomości do wszystkich członków grupy. Pomimo prostoty, opisane rozwiązanie wiąże się z dużym narzutem komunikacyjnym, co w przypadku systemów z większą ilością użytkowników wyklucza jego stosowanie.

Podobnie jak model punktowy, model wielopunktowej komunikacji doczekał się realizacji na poziomie aplikacyjnym. Istniejące rozwiązania [17] bazują na warstwie sieciowej dodatkowo oferując usługi niedostępne w protokole IP, jak choćby niezawodności komunikacji czy dostarczanie komunikatów w kolejności wysłania.

W przypadku realizacji w warstwie sieciowej model wielopunktowy, w porównaniu do modelu punktowego, cechuje się dużo większą skalowalnością, natomiast dużo trudniejsza w realizacji jest kontrola przeciążenia oraz kontrola przepływu. Podobnie jak w przypadku niezawodności w celu zapewnienia wyżej wymienionych usług stosowane są rozwiązania w warstwie aplikacyjnej.

Przykładem protokołu realizującego wymagane usługi w warstwie aplikacyjnej jest RTP (ang. *Realtime Transport Protocol*). RTP to protokół warstwy aplikacji często wykorzystywany w aplikacjach multimedialnych, który może działać wykorzystując w warstwie sieciowej IP multicast. Działa on w połączeniu z protokołem RTCP (ang.

Realtime Control Protocol), który udostępnia usługi kontroli przepływu oraz przeciążenia. W przypadku przeciążenia dochodzi do odrzucania pakietów, jednakże RTP nie daje możliwości kontroli nad tym, które pakiety zostaną pominięte. Protokół sam w sobie nie definiuje pojęcia stanu (podobnie jak wszystkie wymieniowe poprzednio), zaproponowano jednak jego modyfikację w postaci protokołu RTP/I [23][24]. RTP/I rozszerza RTP o pojęcie stanu, wprowadza modyfikację formatu wiadomości, wiążąc każdą wiadomość z informacjami na temat stanu. Zdefiniowane zostały także dodatkowe typy pakietów wykorzystywane do pobierania aktualnego stanu systemu.

Dotychczas opisywane rozwiązania (protokoły) definiowały komunikację jako przesyłanie komunikatów lub strumienia wiadomości pomiędzy stronami komunikacji. Na wyższym poziomie abstrakcji komunikacja może być widziana jako wywoływanie metod na współdzielonych obiektach. Nadawca inicjuje komunikację poprzez wywołanie metody na zdalnym obiekcie, natomiast odbiorca widzi jej rezultat w postaci zmiany stanu obiektu lub jest aktywnie powiadamiany. Ciekawym rozwiązaniem reprezentującym podejście obiektowe w ramach systemów zdalnej współpracy jest DreamTeam Object Manager [22]. Udostępnia on abstrakcje stanu w postaci obiektów, dając możliwość określenia, które z nich przechowywane są centralnie na serwerze, a które replikowane u wszystkich użytkowników. Warto zauważyć, iż model komunikacji nie ma wypływu na wykorzystanie zdalnych obiektów.

1.3 Zachowanie spójności

W przypadku rozproszonego systemu należy założyć, że możliwa jest równoległa praca wielu użytkowników, przez co konieczne staje się podjęcie działań mających na celu kontrolę występujących konfliktów oraz zachowanie spójności stanu. Tylko w przypadku systemów z zewnętrznie przechowywanym stanem, który jest przesyłany do klienta we fragmentach w razie takiej potrzeby, nie występuje problem utraty spójności, ponieważ klient nie posiada jego lokalnej kopii [41][17][29]. Nie wyklucza to jednak konfliktów, gdyż dwóch użytkowników może próbować jednoczesnej modyfikacji tego samego obiektu.

Istnieje kilka metod zapewniania spójności. Najprostszą z nich jest globalny jednodostęp realizowany poprzez przekazywanie między użytkownikami żetonu (ang. *token*), który jego posiadaczowi umożliwia modyfikację globalnego stanu [1][40]. Niestety w opisanym mechanizmie występują problemy związane z odłączeniem się użytkownika posiadającego żeton oraz znaczny spadek wydajności w środowiskach, w których równoległa praca jest częstym zjawiskiem.

Identycznym podejściem, lecz bardziej granularnym jest stosowanie jednodostępu na poziomie fragmentów sesji, a nie całości. Zwiększenie granularności blokad

pozytywnie wpływa na interaktywność, ponieważ stan może być modyfikowany przez wielu użytkowników jednocześnie pod warunkiem, że modyfikacje dotyczą różnych obiektów. Większa liczba obiektów, które mogą być blokowane zmniejsza również ryzyko odmowy uzyskania blokady – w przypadku rozwiązania z żetonem żądania zawsze są odrzucane, jeśli inny użytkownik modyfikuje stan sesji [25].

Trzecią, bardziej skomplikowaną techniką, jest wprowadzenie globalnej kolejności komunikatów oraz sztucznego opóźnienia ich przetwarzania. Globalna kolejność może zostać osiągnięta na podstawie znaczników czasowych wygenerowania poszczególnych operacji [19] – wykonanie operacji u odbiorców jest wstrzymywane do czasu otrzymania wszystkich poprzedników. Wprowadzanie opóźnienia dla operacji związane jest z występującym czasem transmisji tych informacji do pozostałych użytkowników. W przypadku, kiedy to opóźnienie ma wartość nie mniejszą niż połowa RTT do najdalszego (w sensie RTT) uczestnika, operacje zostaną wykonane u wszystkich w tej samej kolejności [7][36].

Najbardziej zaawansowaną techniką rozwiązywania konfliktów jest transformacja operacji. W przypadku wystąpienia konfliktu, na podstawie analizy semantyki równolegle wykonywanych operacji, generowana jest odpowiednia transformacja w celu zmiany parametrów wykonywanych operacji. Zmiany dokonywane są w taki sposób, aby wykonanie końcowej operacji dało taki efekt, jak gdyby oryginalne operacje zostały wykonane w sposób sekwencyjny na wszystkich lokalnych kopiach [33][40]. Powiedzmy, że dwie osoby jednocześnie modyfikują w edytorze tekstowym wyraz *abx*. Pierwsza z nich wstawia literę *d* po literze *a*, a druga z nich zamienia *b* na *c*. Dzięki zastosowaniu transformacji możliwe jest wykonanie operacji w różnej kolejności u każdego z użytkowników, przy czym obaj ostatecznie zobaczą tekst postaci *adcx*. W przypadku, kiedy nie jest możliwe wygenerowanie transformacji (operacje konfliktowe), obie są wycofywane. Wadą tego rozwiązania jest jego ograniczenie tylko do wybranych operacji na globalnym stanie sesji oraz skomplikowane obliczenie transformacji w przypadku wystąpienia konfliktu. Rodziną aplikacji, z której wywodzi się powyższa metoda są edytory teksu przeznaczone do pracy grupowej.

1.4 Późne przyłączanie

W niektórych modelach można wymusić rozpoczęcie sesji dopiero po dołączeniu się do niej wszystkich użytkowników, którzy będą brali w niej udział. Ze względu na małą elastyczność powyższego rozwiązania, najczęściej zezwala się na przyłączenie użytkowników do już rozpoczętej sesji (tzw. *late comming)*. Jeśli w ramach rozpoczętej wcześniej sesji został zmieniony globalny stan, musi on zostać przesłany do nowego użytkownika, aby umożliwić uczestnictwo w sesji.

Odtwarzanie stanu może być realizowane przynajmniej na dwa sposoby: przesłanie tylko aktualnego stanu obiektów lub przesłanie historii zmian dokonywanych na obiektach, co umożliwia odtworzenie aktualnego stanu. Pierwsze rozwiązanie jest często prostsze, a przesłanych danych jest mniej, ponieważ nie zawierają informacji historycznych [22]. W zależności od rozwiązania serializowane są tylko obiekty sesji lub przesyłany jest cały obraz procesu [40].

Alternatywnym rozwiązaniem jest przesyłanie historii operacji, które doprowadziły do aktualnego stanu. W takim przypadku element systemu przechowujący stan może nie być świadomy dokładnej semantyki każdej operacji, a jedynie ich kolejności – tylko odbiorca wiadomości musi wiedzieć jak je zinterpretować. Takie podejście umożliwia również tworzenie rozszerzalnych platform [1]. Użytkownicy, którzy chwilowo utracą połączenie mogą zaktualizować lokalny stan otrzymując informacje obejmujące jego zmiany, które wystąpiły w czasie ich nieobecności, co pozwala na skrócenie czasu ponownego przyłączenia. Dodatkowo w przypadku niektórych aplikacji możliwe jest przesłanie tylko wybranego fragmentu stanu aby przyśpieszyć przyłączenie – np. przesłanie tylko aktualnie widocznego fragmentu w pierwszej kolejności [1].

Powyższe rozwiązanie może być wykorzystane w systemie z centralnie przechowywanym, jak również z rozproszonym stanem, jednak w drugim przypadku jego użycie jest bardziej skomplikowane. Największym problemem, ze względu na niezerowy czas transmisji danych przez sieć, jest fakt, że w danym momencie żaden uczestnik sesji może nie posiadać globalnie aktualnego stanu [9][21][22][36][37]. W przypadku architektury zcentralizowanej, aktualny stan zawsze znajduje się na centralnym serwerze, co znacząco uprasza synchronizację [1].

1.5 Zwiększanie wydajności komunikacji

Zmniejszenie ilości zajmowanego pasma komunikacji może wynikać ze stosowanego protokołu. W protokołach routingu multicastowego routery biorące udział w dystrybucji pakietów dokonują zwielokrotniania przesyłanego pakietu w razie potrzeby, tj. otrzymują pakiet na interfejsie wejściowym i powielają go na wszystkie interfejsy wyjściowe (OIL ang. *Outgoing Interface List*), które prowadzą do węzłów nasłuchujących na adresie docelowym. Korzystając z modelu IP multicast nadawca wysyła jedynie jeden pakiet, a za jego zwielokrotnienie odpowiada infrastruktura sieciowa. Cechą protokołów multicastowych jest brak zależności nadawcy od odbiorców – nadawca nie musi być świadomy jacy użytkownicy odbiorą nadawany strumień. Kolejną, wynikającą z poprzedniej, cechą jest brak możliwości kontrolowania jakości otrzymywanego strumienia u poszczególnych odbiorców – nadawca nie jest angażowany w dystrybucję, ale za to nie ma możliwości jej kontroli.

Na wyższym poziomie abstrakcji rozwiązania problemu efektywnej dystrybucji komunikatów przedstawiane są w kontekście systemów typu *publish-subscribe*. W tych systemach wszystkich użytkowników dzielimy na producentów informacji (ang. *publisher*) oraz na zainteresowanych ich otrzymywaniem, czyli konsumentów (ang. *subscriber*) (przy czym jeden użytkownik może jednocześnie pełnić obie role). Dystrybucja informacji odbywa się w ramach tematów. Typowym przykładem jest system dystrybucji informacji giełdowych, w którym odbiorcy są zainteresowani notowaniami tylko niektórych spółek.

Problem efektywnej dystrybucji komunikatów w systemach tego typu wiąże się z dopasowaniem danych do subskrypcji przed ich rozesłaniem oraz realizacją komunikacji multicast. Rozwiązanie pierwszego zagadnienia zostało przedstawione w pracach [2][3][35]. Wszystkie trzy rozwiązania opierają się na optymalizacjach filtrowania komunikatów. W pierwszym i drugim podejściu zdefiniowano język wykorzystywany przez odbiorców do opisu swoich preferencji. Wyrażenia języka przetwarzane są do postaci drzewa decyzyjnego, optymalizowanego pod kątem ilości porównań potrzebnych do podjęcia decyzji, czy odbiorca jest zainteresowany przetwarzanym komunikatem.

Trzecie rozwiązanie [35] z wcześniej przytoczonych opiera się o mechanizm agregowania subskrypcji w celu optymalizacji czasu dopasowania. W systemie definiowany jest zbiór atrybutów, za pomocą których odbiorca może określić swoje preferencje. System w momencie otrzymania subskrypcji sprawdza czy nie jest ona częścią inne już istniejącej, a pochodzącej z tego samego źródła. W takim przypadku nie jest potrzebne dodanie subskrypcji do tablicy wzorców, gdyż i tak zostanie przekazana w odpowiednie miejsce.

Oprócz optymalizacji systemów *publish-subscribe,* istotne jest wprowadzenie mechanizmów radzących sobie z chwilowymi przeciążeniami. Taka sytuacja może mieć miejsce w przypadku, kiedy wielu użytkowników próbuje w krótkim okresie czasu wysłać komunikaty. W pracy [16] zaproponowano rozwiązanie polegające na wprowadzeniu priorytetów wiadomości definiowanych przez odbiorców. W przypadku zatoru, w pierwszej kolejności przesyłane są komunikaty o najwyższym priorytecie.

Wydajność dystrybucji zdarzeń jest często omawiana w kontekście wieloużytkownikowych gier sieciowych [7]. Powszechnie stosowanym podejściem jest replikacja centralnego serwera w taki sposób, aby zmniejszyć odległości geograficzne pomiędzy nim a użytkownikami. Zdarzenie generowane przez użytkownika wysyłane na serwer najpierw trafia do pozostałych serwerów stanu (ang. *Game State Server*), które przekazują je do pozostałych użytkowników. W ten

sposób wszystkie serwery stanu mogą w sposób równoległy rozsyłać zdarzenia generowane w systemie.

1.6 Kontrola przeciążenia

Nawet w przypadku szybkiego łącza, jeśli odpowiednio wielu użytkowników generuje duży strumień danych, może dojść do jego przeciążenia – ilość wysłanych danych ze względu na fizyczne ograniczenia nie może zostać przesłana do odbiorcy. W przypadku zastosowań, gdzie interaktywność jest istotna, wystąpienie przeciążenia jest bardzo niepożądanym zjawiskiem, ponieważ odbiorca może otrzymywać dane z opóźnieniem lub o znacznie pogorszonej jakości. Mechanizm kontroli przeciążenia (ang. *congestion control*) ma na celu wykrycie takiej sytuacji oraz podjęcie działań mających na celu zapobieganie jej występowania.

W protokole TCP nadawca stwierdza przeciążenie na podstawie braku potwierdzenia otrzymania pakietu przez odbiorcę [20]. Następnie radykalnie zmniejszana jest szybkość wysyłania, a później stopniowo zwiększana (mechanizm AIMD). Działanie to ma na celu utrzymanie możliwie wysokiego tempa wysyłania danych, w którym łącze nie jest przeciążane.

W przypadku kiedy mechanizmy warstwy transportowej są niewystarczające, odpowiedzialność za kontrole przeciążenia jest delegowana do wyższych warstw. Przykładami takiego podejścia są protokół RTCP oraz opisana później metoda oparta na algorytmie RED.

W protokole RTP, często używanym w połączeniu z zawodnym protokołem warstwy transportowej, jakim jest UDP, w momencie wystąpienia przeciążania część danych jest tracona i nie dociera do odbiorcy. Z uwagi na specyfikę protokołu UDP nadawca nie może bezpośrednio dowiedzieć się o tym fakcie. W tym celu zaprojektowany został protokół RTCP, który ma za zadanie wysyłać do odbiorcy informację o jakości odbieranego sygnału (ilości utraconych pakietów). Nie istnieje natomiast mechanizm, który umożliwia nadawcy dostarczenie do odbiorcy wybranych danych w sposób niezawodny [30][28].

Innym rozwiązaniem, bazującym na algorytmie RED (ang. *random early detection*), jest *random early drop* [14][26][27]. Oryginalny algorytm polega na kontrolowaniu kolejki zdarzeń czekających na wysłanie oraz losowym odrzucaniu komunikatów, w przypadku gdy kolejka się wydłuża. W zaproponowanym rozwiązaniu prawdopodobieństwo odrzucenia komunikatu jest proporcjonalne do długości kolejki, a odrzucane komunikaty są podzielone na dwie grupy. Do pierwszej z nich należą pakiety przedawnione, tzn. takie, które przebywając zbyt długo w kolejce straciły swoją wartość informacyjną. Do drugiej grupy należą wszystkie pozostałe. Odrzucanie tego typu pakietów jest dopuszczalne, lecz w takim przypadku ze względu

na brak części istotnych informacji może dojść do utraty spójności sesji. Konieczne jest wtedy zastosowanie odpowiednich mechanizmów przywracania jej spójności. Warto nadmienić, iż algorytm losowego odrzucania jest uruchamiany jedynie w przypadku, kiedy rozmiar kolejki osiągnie z góry ustaloną długość.

Praca [6] pokazuje, że czasami analiza przesyłanych danych może pomóc w ustaleniu ich charakterystyki. Na podstawie zebranych informacji można przeprowadzić optymalizację systemu mając na uwadze jego interaktywność. Należy zwrócić uwagę, że powyższej analizy nie można uogólniać, gdyż jej wyniki są ściśle związane z konkretnym systemem. W tym przypadku [6] zwrócono uwagę na fakt, że małe pakiety były istotniejsze i wymagały mniejszego opóźnienia. Duże z kolei, najczęściej były związane z odtwarzaniem stanu dla użytkowników, którzy właśnie się dołączyli – dla nich bardziej istotna od minimalizacji opóźnienia była wielkość otrzymywanego strumienia danych.

W przypadku danych, dla których dostarczanie z małym opóźnieniem nie jest krytyczne, można zastosować mechanizmy buforowania, aby w przypadku chwilowego, większego obciążenia łącza strumień danych u odbiorcy nie ulegał gwałtownej degradacji. Dopiero w przypadku stałego pogorszenia jakości, wysyłana jest informacja zwrotna do nadawcy, co powoduje zmniejszenie wielkości strumienia kosztem jakości [8].

1.7 Przykładowe rozwiązania

W poprzednich punktach przedstawione zostały podejścia rozwiązujące konkretne problemy istotne z punktu widzenia budowy wydajnej i interaktywnej platformy komunikacyjnej. Poniżej przedstawiona została krótka charakterystyka całych systemów opartych o interaktywną komunikację.

Jedną z przykładowych aplikacji jest Multimedia Lecture Board [24] będąca rozproszoną platformą przeznaczoną do prezentacji i edycji dokumentów w ramach zdalnej konferencji (ang. *whiteboard*). W zamierzeniu twórców aplikacja powinna być wykorzystywana w połączeniu z innymi narzędziami do transmisji AV, zapewniając interaktywną komunikację użytkownikom pracującym w heterogenicznym środowisku sieciowym jakim jest Internet.

System oparto o zreplikowaną architekturę bez stanu przechowywanego zewnętrznie, tzn. każdy użytkownik przechowuje pełny stan lokalnie. Zmiany generowane lokalnie są rozgłaszane za pomocą IP multicast do wszystkich innych użytkowników. W początkowej wersji mlb wykorzystywał protokół RTP do przesyłania danych, obecnie korzysta z jego rozszerzenia tj. RTP/I.

Ciekawym przedstawicielem klasy rozproszonych systemów wirtualnej rzeczywistości (ang. *distributed virtual environment*) jest VESIR – 6 [5]. System oparty jest o protokół

IPv6 wykorzystując wiele z jego nowych możliwości niewystępujących w wersji 4. Zdecentralizowana architektura systemu oraz wykorzystanie komunikacji multicastowej pozytywnie wpływa na interaktywność komunikacji. W rozwiązaniu skorzystano także z usług QoS gwarantujących, że informacje kontrolne zostaną przesłane z najmniejszym opóźnieniem. Ciekawą cechą rozwiązania jest zastosowanie komunikacji typu anycast w celu równoważenia obciążenia, dzięki czemu system jest bardziej skalowalny.

Funkcjonalności charakterystyczne dla systemów zdalnej współpracy są często wprowadzane do już istniejących rozwiązań. Przykładem może być protokół XMPP początkowo stworzony z myślą o komunikatorach internetowych (ang. *Instant messaging*). Okazuje się, że użytkownicy komunikatorów coraz częściej oprócz wymiany informacji tekstowych potrzebują bardziej wyszukanych mechanizmów komunikacji. Dlatego w celu umożliwienia prowadzenia prostych sesji typu *whiteboard* zaproponowano rozszerzenia standardu XMPP.

W ramach rozszerzenia XEP-0113 [39] został zdefiniowany nowy typ wiadomości opisujący zmiany dokonywane w ramach sesji. Wiadomość opisuje geometryczne kształty, które mogą być rysowane w ramach sesji, a szczególny nacisk położono na minimalizację redundancji danych przesyłanych w każdym pakiecie.

2 Analiza wymagań

W poniższym rozdziale zostaną przedstawione wymagania a następnie analiza docelowego środowiska systemu. Przedstawione wymagania podzielono na techniczne tzn. związane ze środowiskiem pracy oraz nietechniczne związane z oczekiwanymi możliwościami. Analizę rozpoczyna opis prostego modelu spełniającego minimalne wymagania. W dalszych krokach model wzbogacany jest o kolejne cechy oraz opis ich logicznej, bądź też fizycznej realizacji. W trakcie analizy autorzy przedstawiają alternatywne rozwiązania oraz uzasadniają podjęte przez siebie decyzje.

W skład wymagań technicznych wchodzą:

1. Praca w heterogenicznym środowisku sieciowym – system powinien umożliwiać współpracę pomiędzy użytkownikami, którzy posiadają połączenia z siecią o różnej przepustowości, a więc powinien w odpowiedni sposób modyfikować komunikaty wysyłane do użytkowników aby uniknąć przeciążenia.
2. Praca w sieciach wykorzystujących mechanizmy translacji adresów NAT/PAT – w obecnym momencie w sieci Internet często wykorzystuje się mechanizmy translacji adresów, dlatego system powinien być w stanie pracować w takim środowisku. Zakłada się jednocześnie, iż serwer będzie posiadał publiczny adres IP, a jedynie stacje końcowe użytkowników mogą wykorzystywać wspomniane mechanizmy.
3. Praca w warunkach chwilowej nieosiągalności użytkownika – dopuszczalna jest sytuacja, w której użytkownik chwilowo traci połączenie z pozostałymi elementami systemu. Platforma powinna wykrywać takie sytuacje i umożliwiać kontynuowanie pracy po odzyskaniu połączenia.

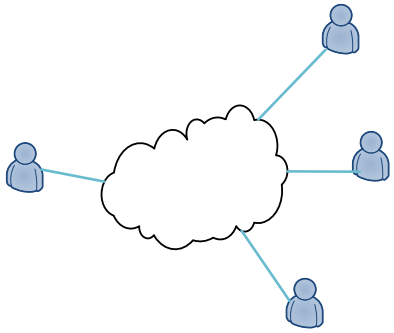
W skład wymagań nietechnicznych wchodzą:

1. Zapewnienie wysokiej interaktywności – tworzone rozwiązanie będzie wykorzystywane w systemach wymagającej dużej interaktywności, a więc powinno działać w taki sposób, aby minimalizować czas odpowiedzi systemu.
2. Wsparcie dla późnego przyłączania użytkowników do sesji – system powinien dawać możliwość dołączenia do już trwającej sesji.
3. Zarządzanie jednodostępem do obiektów sesji – system powinien zapewniać, iż dwóch lub więcej użytkowników nie będzie jednoczenie modyfikować tego samego zasobu.

4. Zachowanie spójności obiektów w czasie sesji – system powinien dawać możliwość operowania na współdzielonych obiektach w taki sposób, aby pozostawały one w spójnym stanie niezależnie od zachowania użytkowników.
5. Rozszerzalność – platforma powinna być rozszerzalna tzn. dawać możliwość dodawania kolejnych funkcjonalności w miarę potrzeby.
6. Możliwość pracy na różnych platformach – współpracujący użytkownicy powinni mieć możliwość korzystania z różnych platform sprzętowych oraz programowych.
7. Uprawnienia w ramach sesji – w ramach sesji użytkownicy powinni dysponować różnymi uprawnieniami, które mają wpływ na rodzaj operacji, które mogą wykonywać.
8. Zapewnienie bezpiecznej komunikacji – platforma powinna dawać możliwość szyfrowania komunikacji oraz uwierzytelniania stron, zmniejszając tym samym szanse powodzenia ataku na system oraz zapewniając zachowanie integralności przesyłanych informacji.

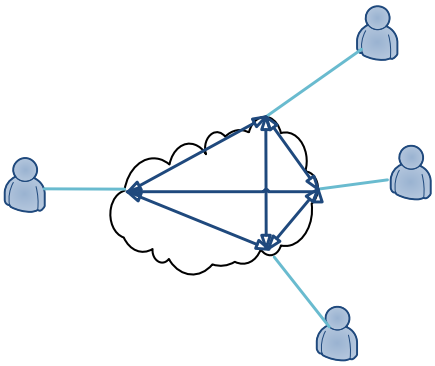
2.1 Model podstawowy

Rysunek 6 przedstawia ogólny model platformy. Użytkownicy współpracują pomiędzy sobą dokonując jednoczesnych modyfikacji współdzielonych zasobów, widocznych dla wszystkich uczestników sesji.



Rysunek 6. Model podstawowy.

Rozważania dotyczące platformy warto rozpocząć od dyskusji modelu komunikacji. Skoro wiadomości wysyłane przez jednego użytkownika powinny trafić do wszystkich pozostałych w celu poinformowania o zmianie stanu, to naturalnym modelem wydaje się komunikacja wielopunktowa (Rysunek 7).



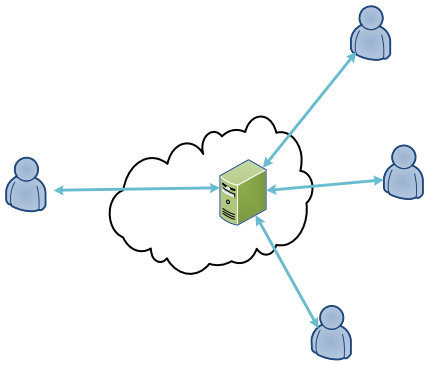
Rysunek 7. Model z grupą odbiorców.

Pomimo swoich zalet komunikacja wielopunktowa nie jest dobrym wyborem. Głównym problemem związanym z komunikacja grupową jest brak powszechnego wsparcia ze strony infrastruktury sieciowej. O ile istnieje możliwość realizacji tego modelu przy użyciu komunikacji punkt-punkt, o tyle duża efektywność osiągana jest jedynie w wyniku realizacji na poziomie warstwy sieciowej. Niestety w obecnym stanie sieć Internet nie udostępnia powszechnego wsparcia dla komunikacji multicast, co więcej istnieją problemy z integracją protokołu z mechanizmami translacji adresów sieciowych. Oczywiście komunikacja grupowa ma swoje zalety, przede wszystkim dużą skalowalność. W przypadku tworzonej platformy liczba użytkowników będzie liczona co najwyżej w dziesiątkach, a więc ta cecha modelu grupowego nie daje istotnych korzyści.

Przytoczone argumenty uzasadniają wykorzystanie modelu z centralnym punktem komunikacji. W takim przypadku w celu realizacji dystrybucji komunikatów nadawca wysyła je tylko do centralnego punktu. Rozwiązanie takie uprości sposób realizacji części klienckiej platformy oraz zapobiegnie niepotrzebnemu marnowaniu pasma w wyniku wysyłania dużej ilości informacji przez każdego z użytkowników.

Komunikacja punkt-punkt z centralnym punktem nie narzuca miejsca przechowywania stanu sesji. Możliwe są dwa rozwiązania tzn. stan przechowywany tylko na stacjach końcowych lub stan przechowywany globalnie z lokalnymi replikami u każdego klienta. Z funkcjonalnego punktu widzenia oba rozwiązania są równoważne, tj. w każdym z nich możliwa jest realizacja założonych usług, np. późnego dołączania do sesji czy utrzymywanie spójności sesji. Różnica polega na złożoności mechanizmów potrzebnych do ich realizacji. W przypadku architektury bez zewnętrznego stanu realizacja jest znacznie trudniejsza. Podobnie jak w przypadku modeli komunikacji bez centralnego punktu, przewagą architektury bez globalnego stanu jest skalowalność. Dodatkowo system taki jest bardziej odporny na awarie. W przypadku tworzonej platformy nie są to wymagania kluczowe, dlatego podjęto decyzję o zastosowaniu pierwszego z wymienionych podejść.

Przechowywanie globalnego stanu, także poza stacjami końcowymi, wiąże się z koniecznością wykorzystania centralnego serwera. W tej sytuacji IGCP przybiera postać systemu typu klient-serwer. Serwer łączy funkcje związanie z komunikacją oraz przechowywaniem stanu. Wszystkie komunikaty w systemie wysyłane są najpierw na serwer, który następnie dystrybuuje je do pozostałych użytkowników. Serwer przechowuje również aktualny stan na potrzeby późno dołączających użytkowników oraz dba o zachowanie spójności. Postać platformy na obecnym etapie analizy przedstawia Rysunek 8.



Rysunek 8. Model klient – serwer.

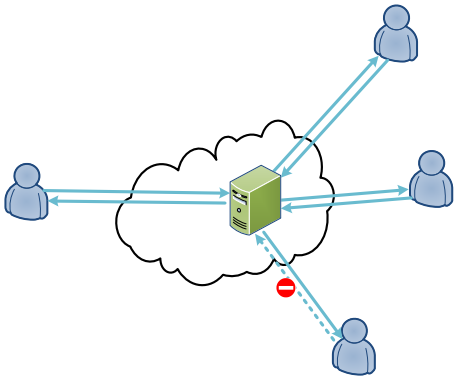
Architektura klient-serwer również nie jest pozbawiona wad. W podejściu z centralnym serwerem staje się on pojedynczym punktem awarii (ang. *single point of failure*), co w przypadku systemów, od których wymaga się maksymalnej niezawodności, może być dużym problemem. Ponadto architektura daje ograniczone możliwości skalowania wprowadzając ograniczenia na liczbę jednocześnie korzystających z systemu użytkowników. Choć warto być świadomym przytoczonych wad omawianego modelu, dotyczą one aspektów mniej istotnych z punktu widzenia tworzonego rozwiązania.

2.2 Funkcjonalne rozszerzenia modelu

Opis przedstawiony w poprzednim punkcie brał pod uwagę realizację minimalnych wymagań. Celem niniejszego punktu jest omówienie sposobu realizacji dodatkowej funkcjonalności.

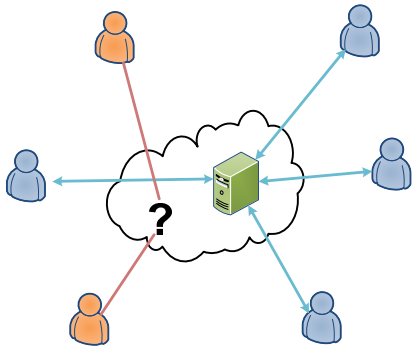
W dotychczasowych rozważaniach niejawnie założono, iż każdy uczestnik sesji może być zarówno producentem jak i konsumentem danych. W praktyce od systemów podobnego zastosowania wymaga się, aby prawo wysyłania informacji przysługiwało wybranym uczestnikom, tzn. pewni użytkownicy mogą jedynie odbierać informacje (Rysunek 9). W modelu z poprzedniego podrozdziału punkt centralny działał jako prosty reflektor, tzn. informacje otrzymywane od jednego z uczestników rozsyła do wszystkich innych.

Chęć pracy w trybie jedynie odbierania powoduje, iż taki sposób działania staje się niewystarczający. Chcielibyśmy, aby serwer na podstawie uprawnień uczestników przekazywał informacje do pozostałych (oraz modyfikował stan sesji) jedynie, jeśli pochodzą one od użytkownika o odpowiednich uprawnieniach. W naszym modelu byłoby to utrudnione chyba, że zostanie wprowadzona jednoznaczna identyfikacja użytkowników, a wszystkie informacje przesyłane w ramach sesji będą opatrzone identyfikatorem nadawcy. W takim przypadku serwer może centralnie przechowywać asocjacje pomiędzy identyfikatorem a uprawnieniami i na ich podstawie podejmować odpowiednie decyzje.



Rysunek 9. Model z uprawnieniami.

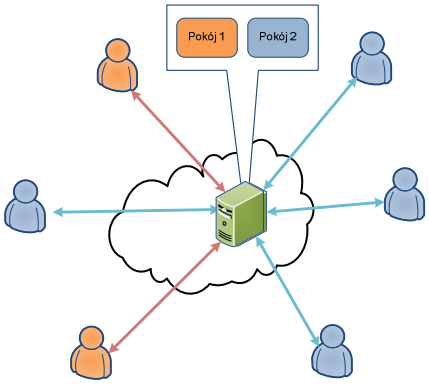
Zgodnie z tym, co wspomniane zostało wcześniej, użytkownicy pracują w ramach sesji. Należy zastanowić się nad przypadkiem, w którym jednoczenie dwie grupy wyrażają chęć rozpoczęcia nowej sesji. Należy rozstrzygnąć czy grupy powinny korzystać z tego samego serwera, czy każda z nich powinna posiadać oddzielny na własne potrzeby (Rysunek 10).



Rysunek 10. Dwie grupy użytkowników.

Tworzenie oddzielnych serwerów (lub aplikacji serwerowych) na potrzeby każdej sesji wydaje się niewygodne i nieuzasadnione, dlatego obie sesje powinny być zarządzane przez pojedynczy serwer. Zadaniem serwera staje się zatem koordynacja równolegle trwających sesji oraz poprawna klasyfikacja informacji, które są w ich ramach przesyłane. Postulat ten może zostać osiągnięty poprzez wprowadzenie pojęcia

pokoju. Pokój może zostać jednoznacznie opisany przez jego stan oraz adres. Serwer utrzymuje zbiór pokoi, a użytkownicy w celu zestawienia sesji najpierw tworzą nowy pokój, a następnie do niego dołączają (Rysunek 11).



Rysunek 11. Koncepcja pokoi.

Jednym z wymagań stawianych tworzonemu rozwiązaniu jest duża interaktywność, co w tym przypadku rozumiane jest jako możliwość jednoczesnej pracy w taki sposób, aby stworzyć iluzję współpracy w rzeczywistym świecie. Idealnym rozwiązaniem jest sytuacja, w której każdy użytkownik może w dowolnym momencie modyfikować każdy obiekt sesji. Niestety, ze względu na wymaganie utrzymania spójności, nie jest to możliwe, tzn. dwóch użytkowników nie może jednocześnie modyfikować tego samego obiektu. Jak już wspomniano w poprzednim rozdziale, istnieje wiele metod zapewniania spójności cechujących się różnym stopniem dopuszczalnej współbieżności. Z naszego punktu widzenia najodpowiedniejszy mechanizmem wydaje się blokowanie na poziomie obiektów. Z jednej strony zapewnia spójność, z drugiej umożliwia równoległą pracę.

W dotychczasowych rozważaniach nie została poruszona kwestia struktury informacji wymienianych w ramach sesji. W rzeczywistych scenariuszach informacje przysyłane przez użytkowników są logicznie powiązane. Aby umożliwić opisany wcześniej schemat blokowania, konieczne jest podzielenie sesji na logicznie niepodzielne fragmenty. IGCP będzie platformą, w ramach której sesja widziana jest jako zbiór współdzielonych obiektów. Każda operacja odzwierciedlająca modyfikację obiektu sesji jest opatrzona identyfikatorem obiektu, którego dotyczy. Użytkownik chcący modyfikować obiekt powinien wcześniej uzyskać na serwerze blokadę. Blokada to asocjacja pomiędzy identyfikatorem użytkownika, a identyfikatorem obiektu. Obiekt może być skojarzony z użytkownikiem za pomocą co najwyżej jednej blokady i w takim przypadku tylko on może modyfikować dany obiekt. Po zakończeniu komunikacji użytkownik powinien zwolnić blokadę zezwalając innym uczestnikom na dokonanie modyfikacji.

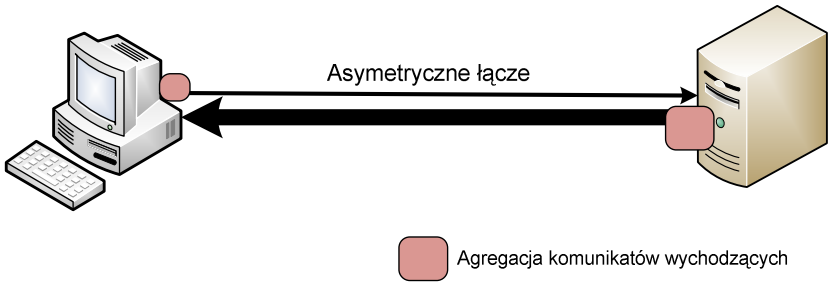
2.3 Uwarunkowania środowiska

Model koncepcyjny systemu nie może pomijać aspektów technicznych związanych ze środowiskiem jego funkcjonowania. Jednym z ważniejszych wymagań jest czas reakcji systemu na wysyłane informacje. Pożądaną cechą systemu jest to, aby uczestnicy sesji możliwe szybko otrzymywani informacje od pozostałych użytkowników.

Zanim przejdziemy do technicznych sposobów realizacji szybkiej komunikacji, należy rozważyć sposób fizycznej reprezentacji wymienianych informacji. Wyróżnia się dwa podstawowe sposoby, tj. komunikację synchroniczną, w której nadawca kończy operację wysyłania w momencie otrzymania potwierdzenia od odbiorcy oraz komunikację asynchroniczną, w której nadawca kontynuuje pracę po przekazaniu komunikatu lokalnej infrastrukturze komunikacyjnej. Z punktu widzenia interaktywności lepszym podejściem jest model asynchroniczny, gdyż użytkownik zmieniający stan nie musi czekać z wysłaniem kolejnych modyfikacji, aż otrzyma potwierdzenie odebrania poprzednich wiadomości.

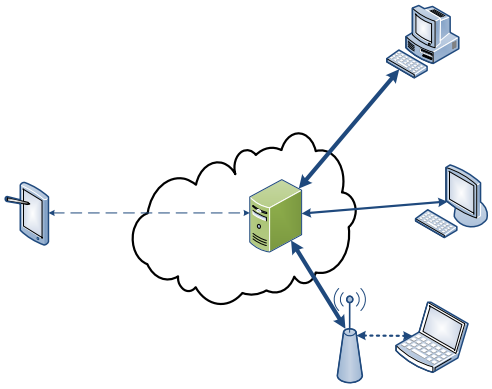
W tym przypadku naturalne podejście to wprowadzenie pojęcia komunikatu wysyłanego asynchronicznie, przechowującego przesyłane informacje. Oprócz informacji w skład komunikatu powinny wchodzić informacje opisane w poprzednich punktach, tzn. identyfikator użytkownika oraz identyfikator obiektu, którego dotyczy dany komunikat.

Rozważając problem interaktywności należy również pamiętać, że skoro użytkownicy dołączają do sesji za pośrednictwem sieci WAN, to dysponują oni różną przepustowością łącza. W takiej sytuacji niemożliwe jest zapewnienie interaktywności, jeżeli serwer będzie pracował w konfiguracji prostego reflektora. W związku z tym, że docelowe środowisko pracy ma heterogeniczny charakter, serwer powinien dokonywać odpowiednich modyfikacji informacji w celu zapewnienia odpowiednio małego opóźnienia komunikacji pomiędzy każdą parą uczestników sesji (Rysunek 12). Możliwymi modyfikacjami są pomijanie nieistotnych komunikatów, agregacja różnych komunikatów w jeden oraz odpowiednie kolejkowanie komunikatów ze względu na ich priorytet.



Rysunek 12. Agregacja na asymetrycznym łączu.

Możliwie duża cześć z tych mechanizmów powinna być niezależna od rodzaju przesyłanego komunikatu (Rysunek 13). Tym celu zdecydowano się na wprowadzenie uniwersalnego sposobu kodowania informacji tj. pary klucz-wartośc (dokładny opis w punkcie 3.5.3)



Rysunek 13. Fizyczne środowisko pracy.

Ponadto należy pamiętać o możliwych różnicach w platformach sprzętowych wykorzystywanych przez użytkowników. Ponadto system powinien umożliwiać pracę użytkownikom korzystającym z aplikacji klienckich implementowanych z wykorzystaniem różnych platform programowych.

Ważnym aspektem, na który należy zwrócić uwagę w proponowanym rozwiązaniu, jest bezpieczeństwo. Najbardziej wrażliwą na nadużycia częścią systemu jest warstwa komunikacyjna, ponieważ transmisja danych pomiędzy elementami systemu nie jest kontrolowana przez system – w klasyfikacji OSI/ISO dotyczy to warstwy sieciowej i łącza danych. Dodatkowym zagrożeniem jest fakt, że system może być wykorzystywany zarówno w sieciach lokalnych (np. korporacyjnych), jak również w rozległych (tj. Internet), w których nie ma centralnego zarządzania i monitorowania pozostałych użytkowników.

W celu zapewnienia odpowiedniego poziomu bezpieczeństwa, użytkownicy powinny przechodzić proces uwierzytelniania i autoryzacji. Dodatkowo, komunikacja pomiędzy elementami systemu powinna być przeprowadzana w ramach szyfrowanych kanałów, co zabezpieczy integralność danych, znacznie utrudniając modyfikacje przesyłanych informacji.

2.4 Podsumowanie analizy

Poniżej przedstawione zostały decyzje projektowe podjęte na podstawie wymagań stawianych systemowi.

1. Architektura klient-serwer
   IGCP będzie posiadała architekturę typu klient – serwer. Po stronie aplikacji klienckiej na rozwiązanie składać się będzie moduł sieciowy odpowiedzialny za wysyłanie do serwera komunikatów generowanych przez użytkownika.
   Stan sesji przechowywany będzie zarówno na serwerze, jak i po stronie klienta, a więc system będzie miał architekturę z centralnie przechowywanym stanem.
2. Asynchroniczna komunikacja punktowa
   Komunikaty będą przesyłane zgodnie z modelem punkt-punkt. Punktem centralnym komunikacji będzie serwer pośredniczący w wymianie informacji pomiędzy aplikacjami klienckimi.
   Komunikacja będzie miała charakter asynchroniczny w celu zwiększenia interaktywności.
3. Kontrola jednodostępu na poziomie obiektów
   System będzie zapewniał kontrolę jednodostępu na poziomie pojedynczych obiektów w celu umożliwienia równoległej pracy na rozłącznych zasobach.
4. Kodowanie klucz-wartość
   W celu zapewnienia rozszerzalności, dane przesyłane w komunikatach będą kodowane jako pary klucz-wartość, dzięki czemu będzie istniała możliwość dodawania nowych typów komunikatów bez potrzeby wprowadzania zmian po stronie serwera.
5. Stan przechowywany w postaci historii komunikacji
   Serwer będzie przechowywał aktualny stan w postaci historii komunikatów przesyłanych w ramach sesji. W przypadku chwilowego rozłączenia lub późnego dołączenia do sesji, komunikaty zostaną wykorzystane do synchronizacji oficjalnej wersji stanu z lokalną kopią użytkownika.
6. Modyfikacja strumienia komunikatów dla każdego użytkownika
   W celu umożliwienia współpracy pomiędzy użytkownikami korzystającymi z łącz o różnej przepustowości, system będzie dokonywał modyfikacji danych przesyłanych do użytkowników celu zmniejszenia ich rozmiaru.

Analiza platformy została przeprowadzona w sposób niezależny od platformy implementacji, a specyfikacja przedstawiona w kolejnym rozdziale w podejściu zgodnym z MDA. Przedstawiony model może zostać zaimplementowany z wykorzystaniem dowolnej technologii spełniającej odpowiednie wymagania. W procesie analizy poczynione zostały pewne założenia dotyczące funkcjonalności udostępnianych przez technologię komunikacyjną, wykorzystaną do implementacji. Poniżej przedstawione zostało podsumowanie tych wymagań:

1. Komunikacja synchroniczna i asynchroniczna – technologia powinna udostępniać usługi komunikacji synchronicznej i asynchronicznej.

2. Niezawodna komunikacja – platforma powinna udostępniać usługę niezawodnej komunikacji, gdyż decyzja o tym, które pakiety powinny zostać pominięte, będzie podejmowana na wyższym poziomie przez platformę komunikacyjną tworzoną przez autorów.
3. Szyfrowane kanały komunikacyjne – w celu zapewnienia wymagań związanych z bezpieczeństwem, technologia powinna udostępniać możliwość szyfrowania przesyłanych informacji.

3 Specyfikacja platformy IGCP

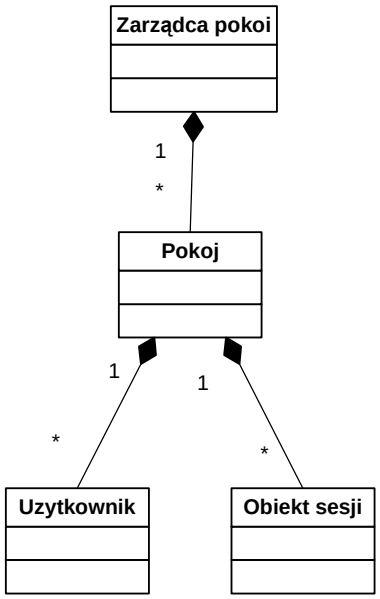
Niniejszy rozdział opisuje model koncepcyjny IGCP. Główny cel rozdziału to przedstawienie logicznej struktury rozwiązania oraz powiązań pomiędzy jego elementami. Model koncepcyjny został opracowany zgodnie z podejściem MDA (ang. *Model Driven Architecture*), tzn. w sposób niezależny od implementacji (PIM ang. *Platform Independent Model*). Pomimo niezależności od sposobu implementacji, rozdział zawiera omówienie aspektów technicznych, mających szczególny wpływ na efektywność oraz interaktywność komunikacji, będących najważniejszymi wymaganiami stawianymi platformie.

Opis modelu zaczyna się od przedstawienia podstawowych pojęć używanych do opisu systemu:

- zarządcy pokoi,
- pokoju,
- użytkownika,
- obiektów tworzących stan.

Dokładniej przedstawione zostaną strukturalne powiązania między nimi (Rysunek 14). W dalszej części omówiony zostanie również protokół komunikacji części klienckiej oraz serwerowej, stworzony z myślą o spełnieniu podstawowych wymagań stawianych platformie, tj.:

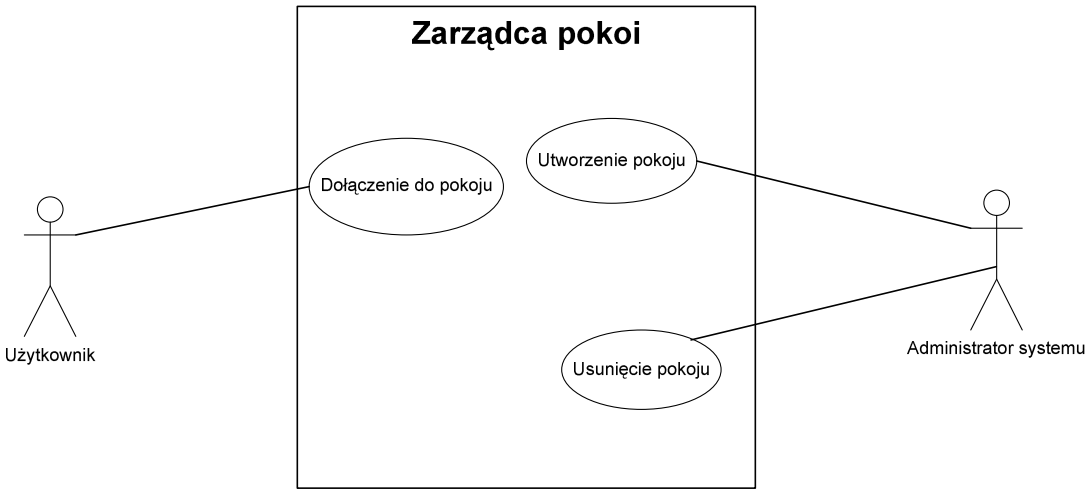
- interaktywności,
- efektywności,
- rozszerzalności.



Rysunek 14. Zarządca pokoi w strukturze obiektów na serwerze.

3.1 Zarządca pokoi

Użytkownicy systemu współpracują pomiędzy sobą w ramach pokoi. Za zarządzanie pokojami odpowiedzialny jest specjalizowany moduł, tj. zarządca pokoi. Do zadań modułu należy przechowywanie listy dostępnych pokoi wraz z ich lokalizacją, a więc pełni on rolę rejestru odpytywanego przez użytkowników chcących wziąć udział w sesji (Rysunek 15).

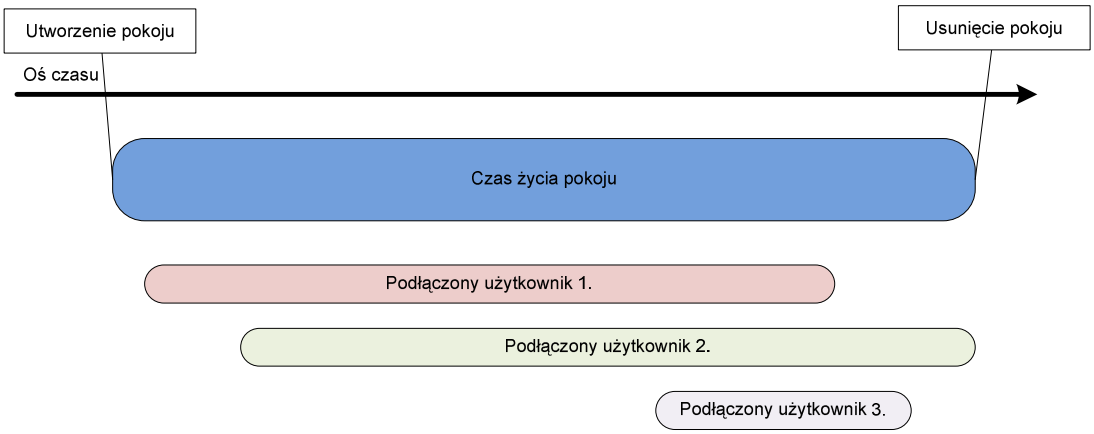


Rysunek 15. Przypadki użycia dla zarządcy pokoi.

3.1.1 Czas życia pokoju

Zarządca pokoju jest także odpowiedzialny za tworzenie oraz usuwanie pokoi – funkcjonalność ta jest dostępna dla administratora systemu. W momencie utworzenia pokoju zostaje rozpoczęta sesja. Dopóki sesja jest aktywna, do pokoju mogą dołączać kolejni użytkownicy, a uczestnicy mogą się odłączać bez wpływu

na pozostałych. Możliwe jest ustawienie czasu ważności sesji lub zamknięcie pokoju po odłączeniu ostatniego uczestnika. Czasową zależność omawianych zdarzeń przedstawia Rysunek 16.



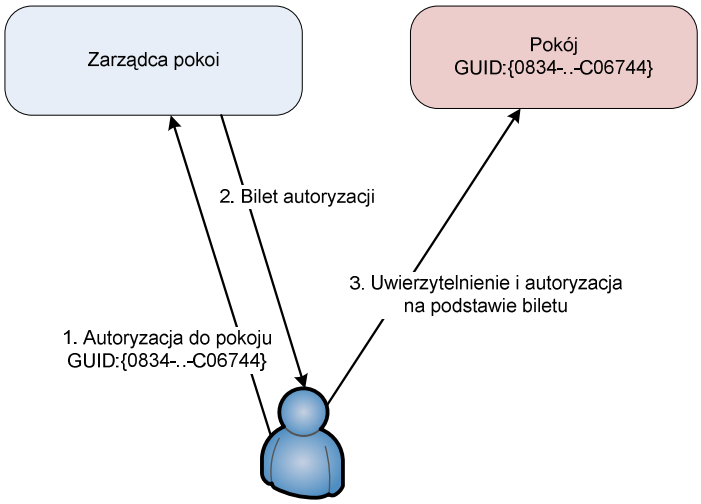
Rysunek 16. Czas życia pokoju.

3.1.2 Autoryzacja użytkowników

Zarządca pokoi pełni także istotną rolę z punktu widzenia bezpieczeństwa. W celu komunikacji z modułem, użytkownik musi najpierw przeprowadzić proces uwierzytelnienia, w ramach którego zarządca pokoi wystawia bilet świadczący o tożsamości użytkownika. Bilet jest następnie wykorzystywany w procesie autoryzacji, zarówno przez zarządcę pokoi, jak również przez sam pokój. Aby móc dołączyć do pokoju użytkownik musi wylegitymować się biletem, na podstawie którego określane są uprawnienia użytkownika w ramach sesji.

3.2 Pokój

Podstawowym pojęciem w modelu koncepcyjnym jest pokój. Pokój reprezentuje trwającą sesję, w ramach której przeprowadzana jest współpraca (koncepcja pokoju została przedstawiona w poprzednim rozdziale). Użytkownicy systemu w celu rozpoczęcia współpracy muszą dołączyć do pokoju. Jednoznaczną identyfikację pokoju zapewnia unikalny identyfikator (GUID). Identyfikator ten jest pobierany od zarządcy pokoi (Rysunek 17). Pokój składa się z użytkowników wchodzących w skład sesji oraz z zasobów tworzących współdzielony stan.



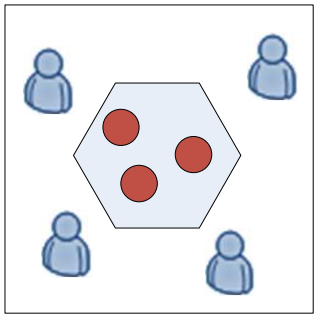
Rysunek 17. Dołączenie do wybranego pokoju.

3.2.1 Identyfikacja użytkowników

Użytkownik bierze udział w sesji za pośrednictwem aplikacji klienckiej. Każdy nowo podłączony użytkownik jest jednoznacznie identyfikowany w ramach pokoju poprzez identyfikator użytkownika (UID), który szczegółowo zostanie omówiony później.

3.2.2 Współdzielenie stanu

W ramach współpracy użytkownicy operują na współdzielonych zasobach dostępnych w postaci obiektów (Rysunek 18). Każdy z obiektów jest jednoznacznie identyfikowany poprzez odpowiedni identyfikator. Operacje możliwe do wykonania to tworzenie nowego obiektu, usunięcie, bądź modyfikacja wybranych własności już istniejącego.



Rysunek 18. Pokój złożony z użytkowników oraz stanu.

Persystencja stanu

Informacje dotyczące pokoju (użytkowników oraz stanu sesji) są przechowywane na serwerze. Dzięki posiadaniu informacji dotyczącej stanu, serwer może przesłać

do nowo dołączających użytkowników aktualny stan sesji, tzn. informacje o wszystkich operacjach dotyczących stanu, wykonanych od rozpoczęcia sesji.

Modyfikacja z jednodostępem

Powszechny dostęp do współdzielonych obiektów oraz jednoczesna praca wielu użytkowników daje możliwość równoległej modyfikacji stanu. Aby zapobiec sytuacji konfliktu, kiedy więcej niż jeden użytkownik modyfikuje ten sam obiekt, został zdefiniowany protokół wykluczania oparty na mechanizmie blokad.

Przed modyfikacją każdego obiektu użytkownik musi uzyskać wyłączną blokadę, która jest udzielana tylko wtedy, jeśli obiekt nie jest modyfikowany przez innego użytkownika. Blokady na obiekty, tak jak same obiekty, przechowywane są w ramach sesji.

3.3 Użytkownicy

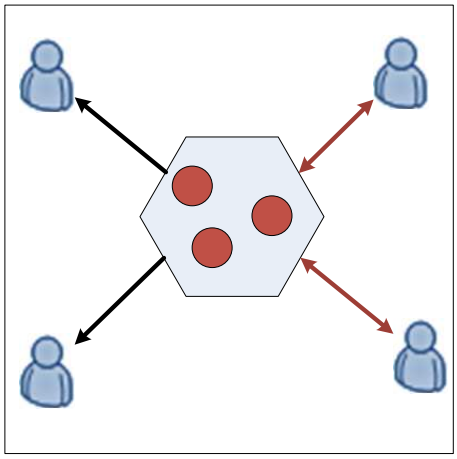
Kolejnym kluczowym elementem modelu koncepcyjnego jest użytkownik. Poniższy punkt opisuje sposób jego identyfikacji oraz mechanizm kontroli dostępu do elementów sesji.

3.3.1 Identyfikator lokalny

Oprócz podstawowych danych, takich jak imię, nazwisko, pseudonim itd., każdy użytkownik jest identyfikowany w systemie w sposób jednoznaczny za pomocą numeru UID (ang. *user identitfier*). Identyfikator ten przydzielany jest podczas pierwszego dołączenia klienta do sesji i przechowywany przez cały czas jej trwania, w celu umożliwienia identyfikacji użytkownika nawet po ponownym przyłączeniu, tzn. chwilowa utrata łączności nie ma wpływu na identyfikacje użytkownika w ramach pokoju. Identyfikator przesyłany jest wraz z każdym komunikatem, co pozwala stwierdzić, kto jest jego autorem.

3.3.2 Prawa dostępu

Komunikaty wysyłane przez użytkowników dotyczą współdzielonych obiektów. Należy jednak pamiętać, że w ramach sesji, dla każdego użytkownika definiuje się uprawnienia określające, co użytkownik może zrobić oraz czego zrobić nie jest w stanie. Uczestnik sesji może mieć prawo tylko do czytania, tzn. pobierania stanu obiektów, lub do czytania i modyfikacji. Uprawnienia przydzielane są na poziomie całej sesji, tj. do wszystkich obiektów (Rysunek 19).



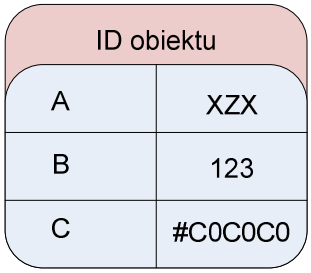
Rysunek 19. Różny poziom uprawnień do obiektów w ramach sesji.

3.4 Stan i obiekty

Użytkownicy biorący udział w sesji mogą w jej ramach operować na już utworzonych obiektach (zmieniając ich stan), dodawać nowe oraz usuwać już istniejące, o ile posiadają wymagane uprawnienia. Każdy obiekt jest skojarzony z jednym pokojem.

Rozważając sposób opisu obiektu, warto zwrócić uwagę na możliwość rozszerzania funkcjonalności systemu w przyszłości. Dodanie nowych atrybutów do obiektu nie powinno powodować błędów u klientów korzystających ze starszych wersji – w takim przypadku nowe własności powinny zostać pominięte. Przykładowo dodanie możliwości zmiany koloru tła wyświetlanego obiektu, w przypadku starszych wersji może zostać zignorowane, zaś nowsze wersje powinny odpowiednio reagować na zmiany dokonywane przez użytkownika.

W celu spełnienia wymagania rozszerzalności, obiekty na serwerze są opisywane jako lista właściwości postaci klucz-wartość. Opisanie obiektu w ten sposób nie wymaga od serwera zrozumienia semantyki przechowywanych danych oraz umożliwia użytkownikowi korzystanie tylko z tych informacji, które mogą być poprawnie zinterpretowane (Rysunek 20).



Rysunek 20. Stan obiektu zapisywany jako pary klucz-wartość.

Opis w postaci par daje możliwość definiowania właściwości w jednolity sposób, niezależnie od typu obiektu. Proste figury geometryczne, jak trójkąt, mogą być

opisane za pomocą trzech właściwości definiujących położenie wierzchołków, umożliwiając rozszerzenie opisu w przyszłości o takie informacje, jak np. kolor linii i tła. Bardziej skomplikowane obiekty również mogą zostać przedstawione w postaci listy par klucz-wartość – przykładowo wyświetlanie złożonego trójwymiarowego obiektu może być opisane przez takie właściwości jak: identyfikator wyświetlanego obiektu, położenie kamery, przybliżenie, pozycja światła itd.

3.4.1 Przechowywanie stanu

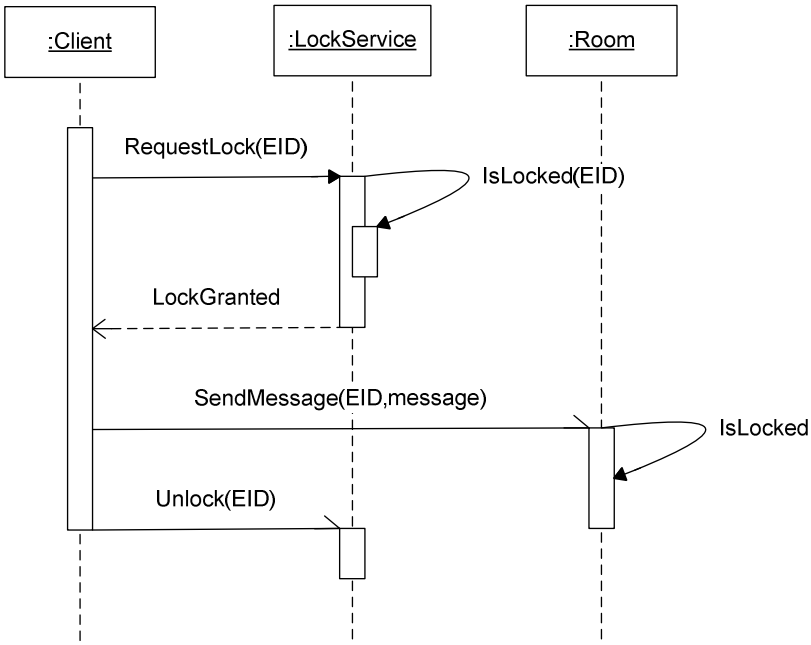
Aktualny stan obiektu jest przechowywany na serwerze w postaci kolejki komunikatów modyfikujących pary klucz-wartość, wysłanych od rozpoczęcia sesji. Każda aplikacja kliencka przechowuje ponadto kopię stanu w postaci zależnej od implementacji (najczęściej w postaci obiektowej).

3.4.2 Zapewnienie jednodostępu

Spójne współdzielenie obiektów w ramach sesji jest możliwe dzięki mechanizmowi jednodostępu. Jednodostęp będzie zrealizowany w postaci mechanizmu blokad obiektów.

Modyfikacja właściwości obiektu jest możliwa jedynie po uprzednim otrzymaniu na wyłączność prawa do modyfikacji obiektu. W przypadku niedotrzymania ustaleń protokołu spójności, tj. wysłaniu komunikatu modyfikującego pomimo braku blokady, komunikat jest ignorowany.

W systemie zostanie ponadto wprowadzony mechanizm automatycznego usuwania blokad w przypadku ich przeterminowania, np. w przypadku awarii (nieoczekiwanego zerwania połączenia) klienta, który uzyskał blokadę na wyłączność. Z każdą blokadą związany jest czas ostatniej modyfikacji (uaktualniany po otrzymaniu każdego komunikatu). W sytuacji, gdy czas ten przekroczy określoną wartość, blokada zostaje usunięta. Rysunek 21 przedstawia sekwencję działań klienta spełniającą założenia protokołu, tzn. modyfikacja obiektu poprzedzona jest uzyskaniem blokady, natomiast po dokonaniu zmian blokada jest zwalniana.



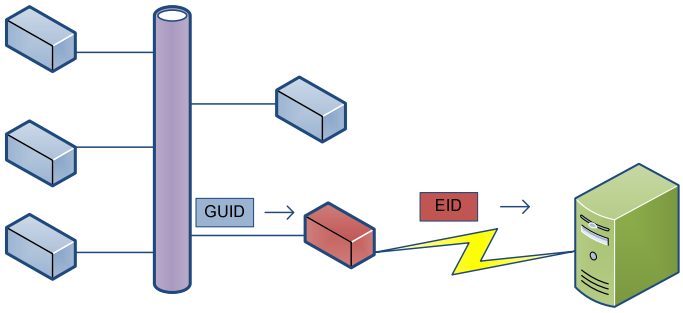
Rysunek 21. Prawidłowa sekwencja działań modyfikująca obiekt.

3.4.3 Identyfikator obiektu

Współdzielenie obiektów oraz mechanizm uprawnień wymaga wprowadzenia jednoznacznej identyfikacji obiektów. Pozwala ona na określenie właściciela obiektu oraz właściciela blokady z nim skojarzonej. Identyfikator obiektu nie zmienia się w trakcie jego życia.

Dla modułów przetwarzających ważne jest, aby nowe identyfikatory mogły być w łatwy i szybki sposób generowane. Natomiast dla modułów przesyłających dane przez sieć, istotna jest wielkość identyfikatora oraz jego unikalność w ramach całej sesji (wymóg efektywności komunikacji). Dodatkowo moduły przetwarzające nie powinny być świadome istnienia identyfikatorów globalnych w ramach sesji. W związku z tym uzasadnione wydaje się wprowadzenie dwóch typów identyfikatorów, tj. lokalnego i globalnego.

Zakres stosowania identyfikatorów lokalnych i globalnych może zostać zilustrowany na przykładzie aplikacji opartej o szynę komunikacyjną. Komunikaty wymieniane pomiędzy modułami zawierają lokalny identyfikator, natomiast w komunikacji pomiędzy modułem sieciowym a serwerem wykorzystywany jest identyfikator globalny (Rysunek 22).



Rysunek 22. Architektura modularnego klienta opartego o szynę komunikacyjną.

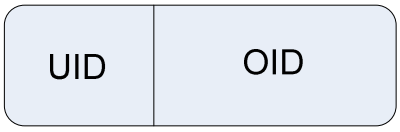
Z każdym obiektem utworzonym w systemie związany jest identyfikator globalny (unikalny w ramach całego systemu, tzn. serwera oraz wszystkich aplikacji klienckich biorących udział w sesji) oraz identyfikator lokalny (unikatowy w ramach pojedynczej aplikacji klienckiej). Z logicznego punktu widzenia oba identyfikatory, lokalny (GUID) oraz globalny (EID), spełniają tę samą funkcję, lecz stosowane są w innych fragmentach systemu.

Identyfikatory typu GUID, można w łatwy sposób generować, ale ich rozmiar jest większy i mógłby negatywnie wpłynąć na wydajność komunikacji. Identyfikatory typu EID są przyznawane w bardziej złożony sposób, ale dzięki temu posiadają mniejszy rozmiar. W ramach jednej aplikacji klienckiej istnieje jednoznaczne odwzorowanie pomiędzy dwoma typami identyfikatorów.

Każda operacja związana z obiektem zawiera informacje o jego identyfikatorze. W zależności od obszaru, w którym komunikat jest przesyłany, jest to EID lub GUID.

Globalny identyfikator - EID

W obszarze serwera oraz w komunikacji pomiędzy serwerem a aplikacjami klienckimi do identyfikacji obiektów wykorzystywany jest EID (ang. *extended identity*). Identyfikator EID składa się z dwóch części, tj. UID oraz OID (ang. *object identity*). OID przypisywany jest lokalnie w ramach każdej aplikacji klienckiej. Składowa UID zapewnia, że obiekty utworzone przez innych klientów nigdy nie będą identyczne (zawsze będą się różniły UID) – Rysunek 23.



Rysunek 23. EID jako połączenie UID i OID.

Jeśli aplikacja pracuje chwilowo bez połączenia do centralnego serwera, UID ma wartość sprzed zerwania połączenia lub jest niezdefiniowany, jeśli pierwsze połączenie nie zostało jeszcze nawiązane.

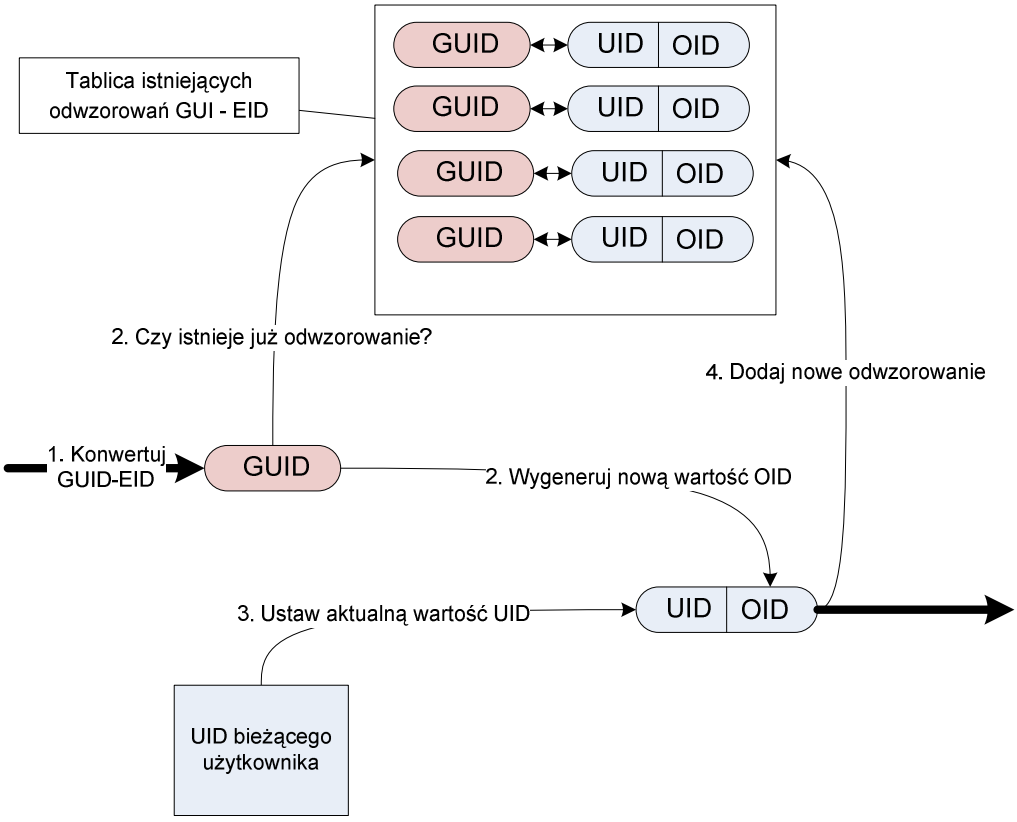
Lokalny identyfikator – GUID

W wewnętrznej komunikacji pomiędzy modułami aplikacji klienckiej wykorzystywany jest identyfikator typu GUID. Wartość identyfikatora przypisywana jest przez moduł tworzący nowy obiekt.

Odwzorowanie EID – GUID

Pomiędzy opisanymi powyżej dwoma typami identyfikatorów istnieje jednoznaczne odwzorowanie. Za przeprowadzenie odwzorowania odpowiedzialny jest moduł sieciowy aplikacji klienckiej. Proces odwzorowania przeprowadzany jest przed przesłaniem komunikatu przez sieć, jak również po otrzymywaniu komunikatu z serwera. W przypadku pracy aplikacji klienckiej bez modułu sieciowego, odwzorowanie nie jest przeprowadzane.

Moduł sieciowy realizuje swoje zadanie przechowując zbiór asocjacji pomiędzy dwoma typami identyfikatorów. W przypadku napotkania nowego identyfikatora, na podstawie identyfikatora jednego typu, tworzony jest odpowiadający mu identyfikator drugiego typu. Kiedy nowy identyfikator pochodzi z komunikatu sieciowego, generowany jest nowy GUID. W przypadku identyfikatora typu GUID pochodzącego z modułu wewnętrznego, przydzielany jest kolejny wolny OID (inkrementacja ostatniej wartości) oraz dołączany do UID przydzielonego w procesie dołączania do pokoju. Schematycznie przedstawia to Rysunek 24. Odwzorowanie EID na GUID przebiega w odwrotnej kolejności.



Rysunek 24. Odwzorowanie GUID na EID.

Dwóch różnych klientów będzie ten sam obiekt na serwerze (to samo EID) odwzorowywało lokalnie na dwa różne GUID. Wartość GUID nie jest nigdy wysyłana poza lokalna instancję aplikacji klienta.

Wprowadzenie dwóch typów identyfikatorów w dużym stopniu ułatwia integrację platformy z już istniejącymi aplikacjami. Dzięki możliwości odwzorowywania lokalnych identyfikatorów (w ogólności mogących mieć dowolną postać) na identyfikator globalny, użycie platformy nie wymusza zmiany sposobu identyfikacji obiektów w istniejącym rozwiązaniu. Powoduje to, że rozszerzanie istniejących systemów o możliwość zdalnej współpracy jest procesem mało inwazyjnym, gdyż nie wymaga zmian w sposobie identyfikacji obiektów lokalnych.

3.5 Protokół komunikacyjny

Przesyłanie komunikatów od użytkownika do serwera i w drugą stronę, jest jedynym sposobem na modyfikację stanu sesji oraz powiadomienie o modyfikacji dokonanej przez innego użytkownika. Wszystkie komunikaty, za wyjątkiem żądania dołączenia do pokoju, muszą być przesyłane w obrębie istniejącej sesji.

Wyróżnić można dwa typy komunikatów: wiadomości, które dotyczą tworzenia, usuwania lub modyfikacji obiektu na serwerze, oraz komunikaty kontrolne, których wymiana wymagana jest w celu prawidłowego działania serwera sesji (np. zapewnienia jednodostępu).

3.5.1 Wiadomości

Wiadomość dotycząca obiektu

Wiadomość zawsze dotyczy istniejącego obiektu sesji, z wyjątkiem wiadomości tworzącej obiekt. Nie przewiduje się wiadomości, która mogłaby modyfikować właściwości kilku obiektów jednocześnie – konieczne jest wysłanie żądania modyfikacji każdego obiektu osobno. Wiadomość posiada pewien zestaw parametrów, identyfikujących jej nadawcę oraz obiekt, który modyfikuje.

Przechowując komunikaty dotyczące danych obiektów, można odtworzyć historię ich modyfikacji, umożliwiając w ten sposób przesłanie stanu do później dołączających użytkowników.

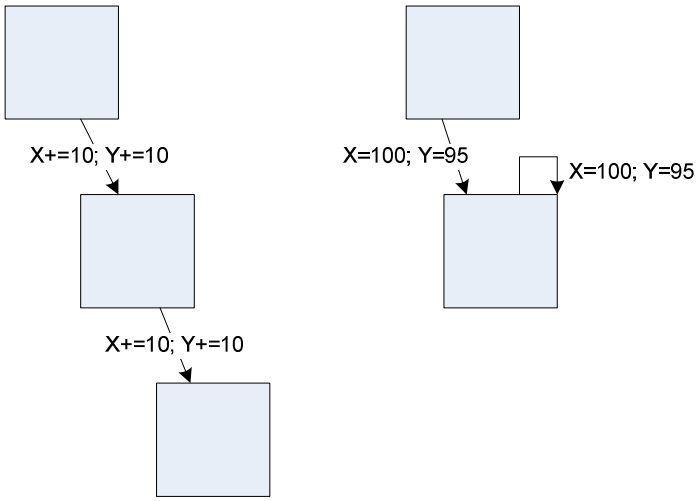
Bezwzględne wartości

Aby umożliwić pomijanie komunikatów, które bezpośrednio po sobie zmieniają tę samą właściwość obiektu, zostało wprowadzone założenie o przesyłaniu bezwzględnych właściwości modyfikujących obiekt. Przykładowo, w przypadku dwukrotnego przesłania wiadomości zmieniającej położenie obiektu, pierwsza wiadomość może zostać pominięta. Niespełnienie tego wymagania (np. przesłanie względnych wartości przesunięcia obiektu), uniemożliwiłoby agregację wiadomości

i wpłynęłoby negatywnie na wydajność komunikacji. Ma to szczególne znaczenie przy odtwarzaniu stanu, kiedy do użytkownika może być wysyłanych wiele wiadomości dotyczących modyfikacji obiektu od początku sesji.

Dodatkowo spełnienie tego wymagania powoduje, że każda wiadomość jest idempotentnym komunikatem, tzn. dozwolone jest jej wielokrotne wysłanie, a wielokrotne odebranie tej samej wiadomości nie wpływa na stan serwera. Możliwe jest ponawianie wysyłania w przypadku błędu bez obawy o duplikację wiadomości.

Przykładowo, przesyłając informację o przesunięciu obiektu, należy podawać bezwzględne wartości nowego położenia. Przesyłanie względnego przesunięcia może powodować niespójność, którą przedstawia rysunek 25. Przedstawia on sytuację, w której doszło do zduplikowania jednego z pakietów. W przypadku przesyłania względnych wartości, spowoduje to przesunięcie obiektu na dwa razy większą odległość, natomiast w przypadku wartości bezwzględnych, ponowne odebranie pakietu nie będzie miało niepożądanych skutków.



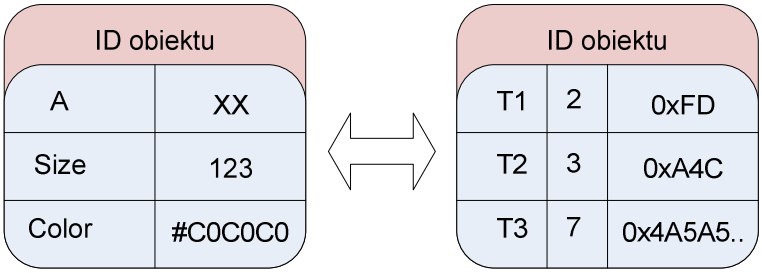
Rysunek 25. Przesyłanie względnych i bezwzględnych wartości modyfikujących położenie obiektu, przy zduplikowanym drugim pakiecie.

Kodowanie klucz-wartość

Jak zostało wcześniej wspomniane, obiekty istniejące w ramach sesji opisywane są jako zbiór par klucz-wartość. Umożliwia to uniknięcie powiązania pomiędzy implementacją systemu a semantyką obiektów sesji – takie podejście nie wymaga od serwera rozumienia treści przetwarzanych komunikatów. Umożliwia to niezależny rozwój części klienckiej bez konieczności aktualizacji serwera, w pozytywny sposób wpływając na rozszerzalności tworzonej platformy.

Naturalnym sposobem przesyłania informacji dotyczącej obiektu zapisanego w ten sposób wydaje się być tablica TLV (ang. *type length value*), która składa się z trójek: typ, wartość, długość. Zapisanie wartości w sposób bezwzględny oraz uwzględnienie typów, umożliwia ich efektywne przetwarzanie (Rysunek 26). Kodowaniem

i dekodowaniem wartości zapisanych w TLV zajmują się serializatory i deserializatory opisane w dalszej części rozdziału.



Rysunek 26. Kodowanie w TLV właściwości obiektu.

Typy wiadomości

W celu rozróżnienia operacji wykonywanych na obiektach sesji, został wprowadzony mechanizm typizacji wiadomości. Typ wiadomości określany jest na podstawie tematu przesyłanego w ramach aplikacji klienckiej.

Istnieje globalne powiązanie pomiędzy tematem a typem wiadomości, dzięki czemu każdy klient wie, jaki typ wiadomości został przydzielony danemu tematowi. Tworzeniem asocjacji typ-temat zajmuje się serwer: w momencie pierwszego zapytania o typ dla danego tematu, zwracany jest pierwszy wolny identyfikator typu. W przypadku kolejnych zapytań o ten sam temat, zwracana jest wcześniej przydzielona wartość powiązana z danym tematem.

Temat wiadomości oraz typ zostały wprowadzone z tego samego powodu co lokalne i globalne identyfikatory, tzn. temat wiadomości (w postaci łańcucha znaków) jest większy od typu, więc wprowadzone odwzorowanie pozytywnie wypływa na wydajność komunikacji. Ponadto, z punktu widzenia modułów aplikacji klienckiej, wygodniejsze jest posługiwanie się nazwani operacji w postaci łańcuchów znaków niż wartościami numerycznymi.

Numery sekwencyjne w ramach połączenia

W ramach połączenia pomiędzy modułem sieciowym i serwerem, komunikaty wysyłane są w sposób asynchroniczny. Ponieważ od stosowanego środowiska middleware nie wymaga się dostarczania komunikatów w kolejności, może zaistnieć sytuacja, kiedy np. dwa następujące po sobie komunikaty dotyczące modyfikacji obiektu zostaną odebrane w odwrotnej kolejności niż zostały nadane.

Aby uniknąć takiej sytuacji wysyłane komunikaty są znakowane numerem sekwencyjnym. Numery sekwencyjne nadawane są w sposób rosnący, dzięki czemu odbiorca komunikatów może szeregować komunikaty względem kolejności ich wysłania.

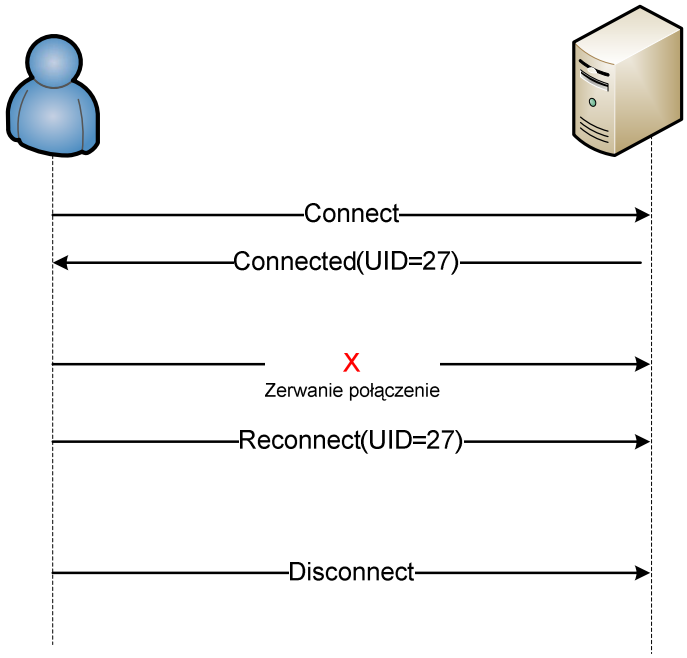
W momencie zerwania połączenia i jego ponownego nawiązania, numeracja zostaje rozpoczęta od początku. Oznacza to, że w ramach jednej sesji z jednym użytkownikiem mogą zostać wygenerowane komunikaty o takich samych numerach sekwencyjnych tylko w przypadku, kiedy komunikacja odbywała się w ramach dwóch połączeń (pierwsze zostało zerwane).

3.5.2 Komunikaty kontrolne

Komunikaty kontrolne, w przeciwieństwie do wiadomości, nie modyfikują obiektów. Wprowadzenie ich jest związane z koniecznością zarządzania sesją.

Dołączanie i odłączanie

Pierwszymi omawianymi komunikatami są komunikaty dołączania do i odłączania od pokoju, służące do zarządzania użytkownikami, którzy korzystają z sesji (Rysunek 27).



Rysunek 27. Schemat komunikacji pomiędzy użytkownikiem a serwerem w czasie dołączania i odłączania.

Komunikat żądania dołączenia musi zostać wysłany przed rozpoczęciem korzystania z sesji. W momencie nawiązania połączenia z serwerem, wysłana jest informacja, do którego pokoju (sesji) użytkownik chce się dołączyć. Następnie ustanawiane jest dwustronne połączenie między konkretnym pokojem a użytkownikiem. Po zakończeniu operacji dołączania, użytkownikowi nadawany jest omówiony już unikalny identyfikator (UID). Procedura dołączania do sesji musi być również przeprowadzona w przypadku dołączenia po chwilowym zerwaniu połączenia, jednak w tym przypadku nie jest nadawany nowy identyfikator, lecz wykorzystywany wcześniej przypisany.

Wysłanie komunikatu odłączenia od sesji ma na celu poinformowanie pozostałych użytkowników o niedostępności użytkownika. W przypadku, kiedy użytkownik nie wyśle komunikatu odłączenia od sesji, a połączenie zostanie zerwane, po pewnym ustalonym czasie serwer wygeneruje komunikat odłączenia od sesji w celu powiadomienia pozostałych użytkowników. Po wysłaniu komunikatu odłączenia, w przypadku ponownego połączenia, nadawany jest inny identyfikator użytkownika (UID), ponieważ wszystkie zasoby związane z poprzednim identyfikatorem zostały wcześniej zwolnione.

Zakładanie blokady(Lock)

Komunikat żądania blokady wysyłany jest w sposób synchroniczny, tj. wywołujący czeka na komunikat zwrotny z serwera. Wprowadzenie synchronicznego, centralnie zarządzanego mechanizmu blokad jest konieczne ze względu na możliwość równoległej pracy użytkowników.

W pakiecie żądania założenia blokady na dany obiekt, wysyłany jest identyfikator obiektu, którego żądanie dotyczy. Jeśli na serwerze nie została założona blokada na ten obiekt przez innego użytkownika, serwer zakłada blokadę, której właścicielem jest nadawca komunikatu. W przeciwnym przypadku zwraca informację, że inny użytkownik już założył blokadę na obiekt.

W przypadku braku możliwości nawiązania połączenia z serwerem lub w przypadku przekroczenia czasu odpowiedzi, również jest zwracana informacja o braku uzyskania blokady.

Zdejmowanie blokady(Unlock)

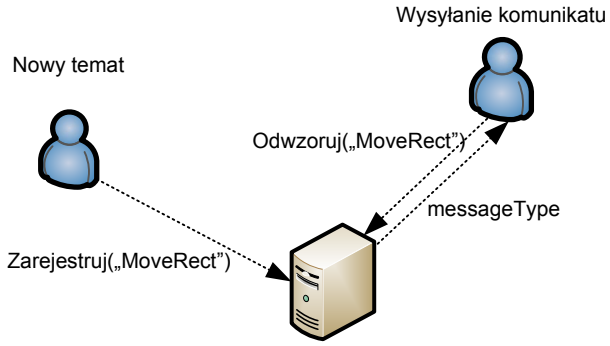
Komunikat zdjęcia blokady wysyłany jest w momencie, kiedy użytkownik kończy wysyłanie komunikatów edytujących dany obiekt. Taki komunikat może zostać wysłany również w sposób niejawny przez system, kiedy przez określony czas nie są lokalnie wykonywane żadne modyfikacje obiektu – nie są wysyłane komunikaty z nim związane. W przypadku utraty połączenia przed zwolnieniem blokady, za zwolnienie zasobów odpowiedzialne są odpowiednie mechanizmy na serwerze. Szczegółowo zostanie to opisane w rozdziale poświęconym zarządzaniu połączeniem.

W przypadku, kiedy użytkownik wysyła komunikat zwolnienia blokady dla obiektu, do którego blokady nie posiadał, komunikat jest ignorowany.

Semantyka komunikatu zwalniania blokady wymaga, aby dotarł on po ostatnim pakiecie modyfikującym obiekt. W celu spełnienia tego wymagania jest on wysyłany kanałem asynchronicznym, ponieważ w ramach jednego kanału kolejność dostarczania komunikatów dotyczących tego samego obiektu jest zapewniona w sposób programowy.

Odwzorowanie tematu i typu wiadomości

W celu umożliwienia centralnego zarządzania tematami wiadomości i odwzorowaniami na typ wiadomości, zostały wprowadzone dodatkowe komunikaty: rejestracja nowego tematu, żądanie odwzorowania typu wiadomości na temat oraz tematu na typ wiadomości. Odwzorowanie to jest przechowywane na serwerze i jest specyficzne dla każdego pokoju.

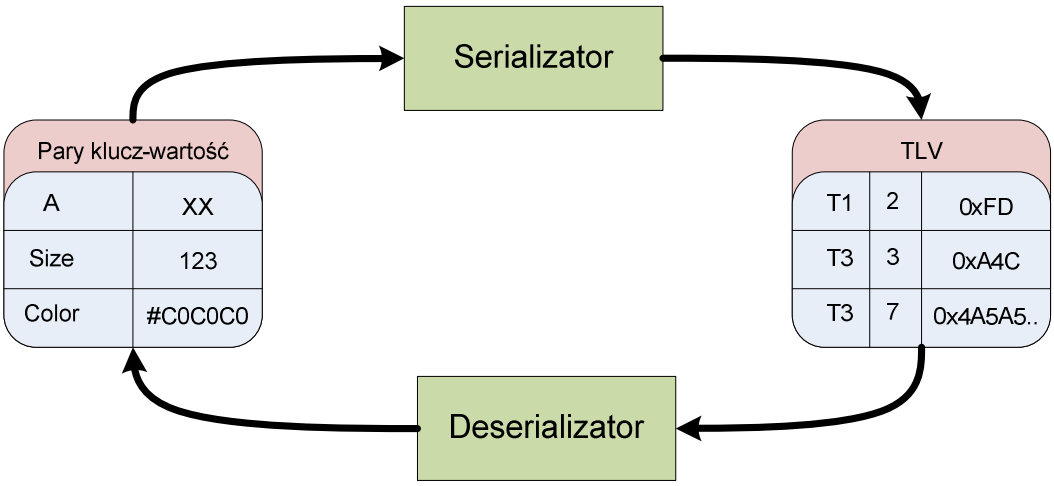


Rysunek 28. Odwzorowanie tematu na typ wiadomości.

Zanim klient będzie mógł wysyłać wiadomości dotyczące jakiegoś tematu, musi go wcześniej zarejestrować. W czasie rejestracji serwer przydziela tematowi unikatowy typ wiadomości. Po rejestracji każdy klient, który chce wysłać wiadomość na zadany temat pyta serwer o odwzorowanie tematu na typ wiadomości (Rysunek 28).

3.5.3 Serializacja i deserializacja

Zadaniem serializacji jest konwersja z formatu wykorzystywanego w aplikacji klienckiej do uniwersalnego formatu TLV omówionego wcześniej. Deserializacja ma miejsce w momencie odebrania wiadomości z serwera – deserializator zmienia ją na format rozpoznawalny przez aplikację (Rysunek 29). W celu poprawnego przesłania pomiędzy aplikacjami klienckimi, z każdym typem wiadomości powinna być jednoznacznie skojarzona para serializator-deserializator.



Rysunek 29. Wykorzystanie serializatorów i deserializatorów do konwersji typów.

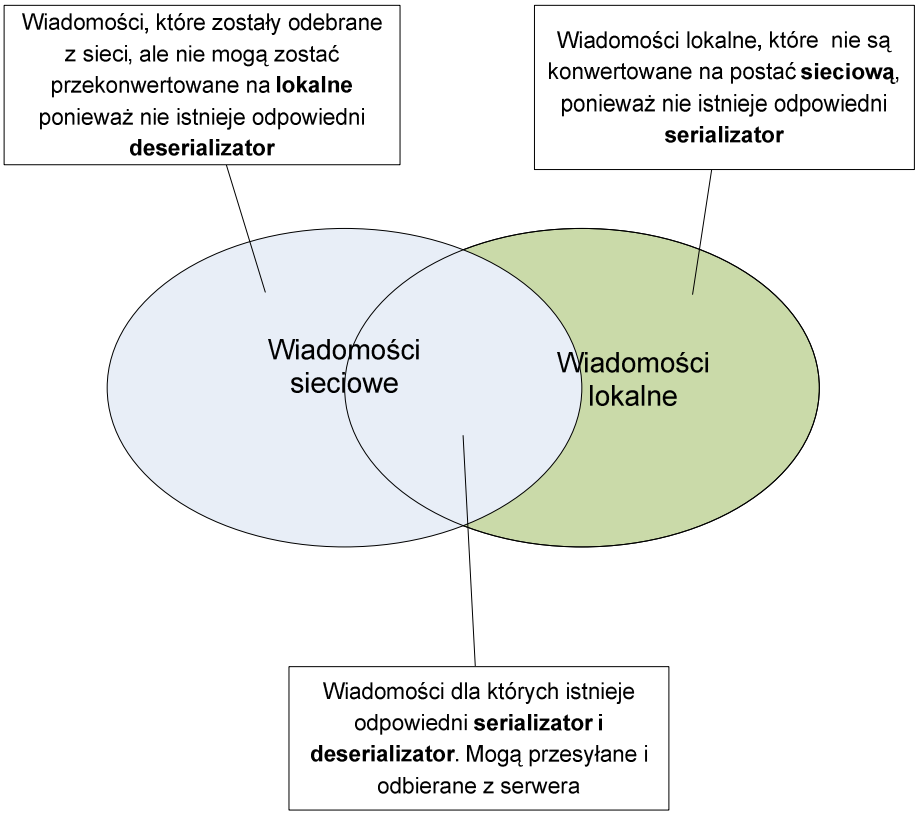
Rejestracja serializatorów i deserializatorów

Zarządzaniem serializatorami i deserializatorami zajmuje się odpowiednie repozytorium (każda aplikacja kliencka posiada swoją instancję). W trakcie uruchamiania aplikacji klienckiej, do lokalnego repozytorium serializatorów dodawane są wszystkie znane serializatory i odpowiadające im deserializatory. Repozytorium wykorzystywane jest procesie zmiany wiadomości z formatu aplikacji na TLV i odwrotnie.

Wiadomości lokalne i sieciowe

Podobnie jak w przypadku dwóch typy identyfikatorów (EID i GUID), zostały wprowadzone dwa typy wiadomości: lokalne i sieciowe. Umożliwia to łatwe przetwarzanie lokalne oraz wydajne przesyłanie przez sieć. Możliwość konwersji jednego typu wiadomości na drugi powoduje, że prezentowane rozwiązanie może zostać wprowadzone, jako moduł do dowolnej aplikacji, dla której zostanie utworzony konwerter wiadomości.

Wiadomości lokalne przesyłane są w obrębie aplikacji klienckiej. Te spośród nich, dla których istnieją serializatory i deserializatory, mogą być tłumaczone do postaci wiadomości sieciowej, a następnie wysyłane. Jeśli dla otrzymanej z serwera wiadomości sieciowej nie istnieje deserializator, to nie będzie ona mogła zostać przetłumaczona na wiadomość lokalną – zostanie przetworzona tylko lokalnie. Zależności pomiędzy wiadomościami sieciowymi a lokalnymi pokazuje rysunek 30.



Rysunek 30. Zależność między wiadomościami sieciowymi a lokalnymi.

Dzięki wprowadzeniu dwóch typów wiadomości, część kliencka tworzonej platformy może zostać dodana do już istniejącej aplikacji jako dodatkowy moduł. Ponadto, przedstawione powyżej podejście umożliwia korzystanie z platformy w nowo tworzonych rozwiązaniach, nie narzucając decyzji projektowych dotyczących architektury aplikacji, która ją wykorzystuje.

Wybór serializatora

W sytuacji, kiedy wiadomość trafia do modułu sieciowego aplikacji klienckiej, wyszukiwany jest serializator zarejestrowany dla danego tematu wiadomości. W przypadku, kiedy taki serializator nie istnieje, wiadomość uznawana jest za lokalną i nie jest przesyłana na serwer. W przeciwnym wypadku, serializator zamienia dane zawarte w wiadomości na tablicę TLV.

Wybór deserializatora odbywa się w analogiczny sposób – po otrzymaniu wiadomości na podstawie typu wiadomości określany jest wymagany deserializator, który zmienia wiadomość na postać rozumianą przez aplikację. Jeśli taki deserializator nie istnieje, zakłada się, że aplikacja nie potrafi interpretować konkretnych wiadomości – są one ignorowane przez klienta.

Bezstanowość serializatora

Ze względu na sposób działania IGCP oraz mając na uwadze wydajność komunikacji, serializator nie powinien posiadać stanu. Oznacza to, że serializacja takiej samej

wiadomości zawsze powinna zwrócić ten sam rezultat bez względu na poprzednio serializowane wiadomości.

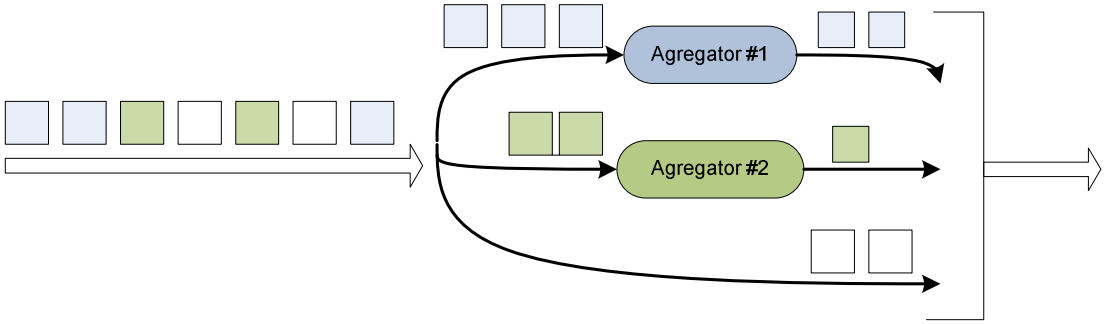
Takie samo założenie dotyczy deserializatorów, których działanie również nie powinno zależeć od kolejności przetwarzanych komunikatów. Zachowanie takie jest wymagane np. w przypadku, kiedy agregator (odpowiedzialny za zmniejszenie rozmiaru danych) pominie niektóre wiadomości – nie zostaną one wysłane z powodu obciążenia łącza.

3.5.4 Agregacja

Jak zostało powiedziane we wstępie do rozdziału, istotnym aspektem projektowanego systemu jest wymaganie związane z interaktywnością. Rozwijana platforma powinna w możliwie krótkim czasie (do 150 ms. [12]) przetwarzać i dostarczać wiadomości generowane przez użytkowników. W tym celu wprowadzony został mechanizm agregacji, który może w sposób stratny doprowadzać do zmniejszenia rozmiaru lub ilości przesyłanych komunikatów. Należy zwrócić uwagę na fakt, że bezpośrednim celem agregacji nie jest zmniejszenie ilości wysyłanych danych, ale zmniejszenie czasu oczekiwania odbiorcy na wysłane dane. Oznacza to, że jeśli czas potrzebny na agregację i wysłanie mniejszej ilości danych jest większy niż czas potrzebny na przesłanie niezagregowanych danych, to agregacja nie powinna być wykorzystywana.

Proces agregacji przeprowadzany jest na zbiorze wiadomości dotyczących tego samego obiektu. Warto zauważyć, iż proces ten jest stratny w przeciwieństwie do procesu kompresji, w którym dane przed ponownym odczytaniem poddawane są procesowi dekompresji. Ponadto moduły odpowiedzialne za przeprowadzanie agregacji działają w sposób bezstanowy, a więc nie mogą korzystać z informacji historycznych, jak niektóre metody kompresji stosowane w protokołach komunikacji internetowej, np. kompresja nagłówka TCP czy RTP.

Agregator na podstawie informacji o semantyce przesyłanych komunikatów dokonuje transformacji, dlatego konieczne jest wprowadzenie powiązania między typem wiadomości a konkretnym typem agregatora. Agregatory, podobnie jak serializatory i deserializatory, są lokalnie dodawane do repozytorium w czasie uruchamiania aplikacji. Podobne repozytorium utrzymywane jest przez serwer. Istnienie powiązania pomiędzy typem wiadomości a agregatorem oznacza, że możliwa jest agregacja wiadomości. W przypadku, kiedy istnieje tylko para <serializator, deserializator> (tzn. agregator nie został zarejestrowany), wiadomości wysyłane są bez dokonywania agregacji. Warto podkreślić, że sposób agregacji jest ściśle związany z semantyką przesyłanych wiadomości, dlatego wprowadzenie standardowego agregatora zdolnego do obsługi wszystkich typów wiadomości jest zadaniem trudnym lub wręcz niemożliwym. Zasadę działania agregatorów przedstawia rysunek 31.



Rysunek 31. Agregacja dla zarejestrowanych z wykorzystaniem dedykowanych agregatorów.

Parametry agregacji

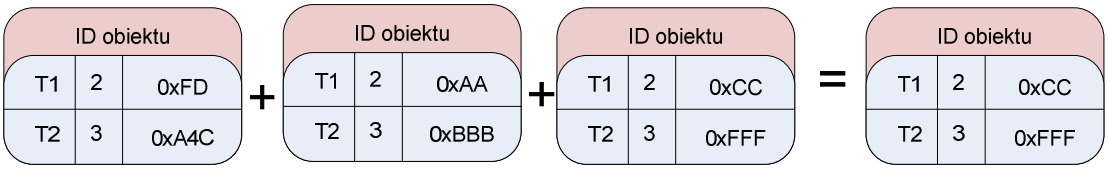
Aby móc stosować różne stopnie agregacji, został wprowadzony parametr określający stan łącza. Takie podejście umożliwia dostosowanie ilości i jakości przesyłanych danych (np. obrazu) w zależności od zmieniających się warunków.

Przykładowo, w przypadku nieobciążonego łącza o dużej przepustowości nie jest konieczna agregacja, ponieważ wprowadza niepotrzebne opóźnienie związane z czasem potrzebnym na jej przeprowadzenie. Jednak w przypadku, kiedy to samo łącze zostaje obciążone (np. przez inną aplikację), parametry agregacji powinny ulec zmianie, aby nie zwiększać opóźnienia związanego z ograniczoną przepustowością sieci.

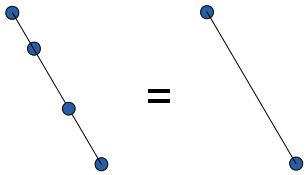
W celu unifikacji obsługi agregatorów, wprowadzony został interfejs, który musi być implementowany przez każdy agregator. Agregator otrzymuje następujące dane:

- tablice TLV,
- wymagany stopień agregacji.

Na podstawie tablic TLV danych, dokonywana jest agregacja komunikatów. Sposób agregacji zależy od zaimplementowanego algorytmu. Może to być zarówno prosty algorytm usuwający następujące po sobie komunikaty związane z tą samą właściwością (np. dwukrotna zmiana koloru – Rysunek 32), jak również bardziej złożony, usuwający współliniowe punkty w przypadku, gdy modyfikowany obiekt jest linią (Rysunek 33).

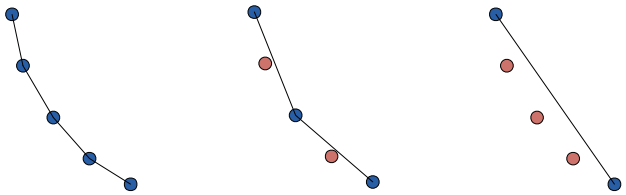


Rysunek 32. Prosta agregacja polegająca na zapisaniu ostatnich wartości z ciągu.



Rysunek 33. Bardziej złożona agregacja polegająca na usuwaniu współliniowych punktów.

Wartość wymaganego stopnia agregacji określa, jak silna agregacja powinna zostać przeprowadzona. Stopień agregacji określany jest przez liczbę z przedziału od 1 do 10, gdzie 1 oznacza najsłabszą agregację (najwięcej informacji pozostawionych), a 10 najsilniejszą. Przykładowo, agregując krzywą z różnymi stopniami kompresji można usuwać większą liczbę punktów, które są prawie współliniowe (Rysunek 34).



Rysunek 34. Stratna agregacja krzywej dla różnych wartości stopnia agregacji. Punkty zaznaczone na czerwono zostały odrzucone w wyniku agregacji.

Nie istnieje bezpośredni związek pomiędzy stopniem agregacji a stanem łącza, ponieważ stopień agregacji zależy również od ilości danych wysyłanych, które mogą dotyczyć innych obiektów. W przypadku, kiedy równolegle wysłane są komunikaty na temat różnych obiektów, mimo łącza dobrej jakości, wymagany stopień kompresji może zostać zwiększony.

Podobnie jak w przypadku serializatorów, zakłada się, że agregator nie posiada stanu. Takie zachowanie jest wymagane, ponieważ w przypadku np. zerwania połączenia, te same wiadomości mogłyby być agregowane powtórnie. Uwzględnienie takiej sytuacji w agregatorze utrudniłoby jego implementację, nie wpływając znacząco na poprawę jakości agregacji.

3.5.5 Zarządzanie połączeniem

Praca w heterogenicznym środowisku nakłada na system wymóg pracy w warunkach chwilowej utraty łączności, często występujących w sieciach mobilnych. Brak komunikacji, który nie trwa zbyt długo, nie powinien być przeszkodą w dalszym uczestnictwie w sesji. W celu spełnienia tego wymagania system został zaopatrzony w odpowiednie mechanizmy przedstawione poniżej.

Pinger

Pomiędzy aplikacją kliencką a serwerem, w ustalonych z góry odstępach czasu, przesyłane są komunikaty kontrolne nieprzenoszące żadnych informacji (ang. *ping*). Celem komunikacji jest możliwie wczesne stwierdzenie zerwania połączenia. Reakcją na zaistniały problem jest uruchomienie mechanizmu ponownego zestawienia połączenia.

Brak powyższego rozwiązania pozwalałby na zaistnienie sytuacji, w której informacje nie docierałyby do aplikacji klienckiej, a ta byłaby tego nieświadoma – sytuacja, w której klient biernie słucha a sam nie wysyła żadnych informacji.

Rozłączenie użytkownika

Z powodu awaryjności sieci może zaistnieć sytuacja, w której dojdzie do czasowego zerwania połączenia pomiędzy klientem a serwerem. W przypadku wystąpienia takiego problemu, IGCP przeprowadza próbę ponownego zestawienia połączenia.

Wykrycie zerwania połączenia może zostać stwierdzone w wyniku przeterminowania czasu potwierdzenia, związanego z wysłanym komunikatem. W przypadku, kiedy nie są wysyłane komunikaty, działa opisany wyżej mechanizm *ping.*

Zerwanie połączenia może doprowadzić do nieprawidłowego działania mechanizmu zapewniania jednodostępu – blokady posiadane przez aplikacje kliencką, z którą serwer utracił połączenie, mogą nigdy nie być przez nią zwolnione, uniemożliwiając modyfikacje części obiektów przez pozostałych uczestników sesji. W celu rozwiązania powyższego problemu zostały wprowadzone czasy przedawnienia blokad. Jeżeli właściciel blokady przez odpowiednio długi okres czasu nie dokonał żadnej modyfikacji obiektu, jego blokada zostaje usunięta nawet, jeżeli nie została ona przez niego zwolniona w sposób jawny.

Ponowne przyłączenie

Kontynuacja pracy w przypadku zerwania połączenia, wiąże się z koniecznością ponownego dołączenia do sesji. Fakt pracy bez połączenia z serwerem wymusza potrzebę synchronizacji lokalnego stanu sesji oraz stanu przechowywanego na serwerze.

W trybie pracy z modułem sieciowym jedynym przypadkiem, w którym może dojść do przeprowadzenia sprzecznych modyfikacji tego samego obiektu przez odłączonego klienta oraz jednego z klientów uczestniczącego w sesji, jest sytuacja, kiedy w momencie odłączenia od sesji użytkownik posiadał blokadę tego obiektu.

Bazując na założeniu, że wszystkie komunikaty przenoszą bezwzględne wartości modyfikowanych atrybutów obiektu, w celu synchronizacji sesji, użytkownik po ponownym dołączeniu otrzymuje wszystkie komunikaty kontrolne, wysłane w ramach

pokoju od momentu jego odłączenia oraz wszystkie wiadomości wysłane od początku sesji dotyczące obiektów, których blokady posiadał w momencie odłączenia.

3.5.6 Bezpieczeństwo

Omówienie aspektów platformy związanych z bezpieczeństwem warto rozpocząć od przedstawienia możliwych zagrożeń. Rodzaje ataków sieciowych można podzielić następująco:

- przerwanie komunikacji,
- podsłuchiwanie komunikacji,
- modyfikacja komunikacji,
- podrobienie komunikacji.

Często spotykaną techniką jest metoda MITM (ang. *Man In The Middle*), w której agresor staje się pośrednikiem w procesie komunikacji pomiędzy nadawcą i odbiorcą. W systemach zdalnej współpracy przykładowy atak może mieć miejsce, kiedy osoba próbująca naruszyć bezpieczeństwo systemu, spowoduje niedocieranie informacji, takich jak głos, do pozostałych uczestników lub przeciwnie – spowoduje dostarczanie informacji, których uczestnik nie wysyłał lub nie chciał wysyłać.

W przedstawianej platformie kwestie bezpieczeństwa powinny być uwzględnione zarówno w części serwerowej, jak również w aplikacji klienckiej, z której korzystają użytkownicy. W celu zapewnienia odpowiedniego poziomu bezpieczeństwa konieczne jest, aby system umożliwiał:

- uwierzytelnianie serwera – klient będzie miał pewność, że nawiązuje komunikację z serwerem,
- uwierzytelnianie klienta – serwer wie, kto wysłał dane, co uniemożliwia podszywanie się pod innego użytkownika,
- sprawdzanie spójności danych – co zapobiega modyfikacji przesyłanych danych pomiędzy klientem a serwerem,
- szyfrowanie przesyłanych informacji – nawet w przypadku podsłuchania strumienia danych, odczytanie informacji wymaga złamania algorytmu szyfrującego.

Należy jednak pamiętać, że wprowadzenie każdego z wyżej wymienionych mechanizmów powoduje zwiększenie zapotrzebowania na zasoby. Szyfrowanie komunikacji wymaga dodatkowych obliczeń przed wysłaniem i po odebraniu informacji, a zaszyfrowane informacje mają większy rozmiar niż oryginalne. Sprawdzanie spójności danych związane jest z dołączaniem pewnej sumy kontrolnej, co ma niekorzystny wpływ na ogólny czas potrzebny na dostarczenie wiadomości. Uwierzytelnianie klienta i serwera z wykorzystaniem SSL i certyfikatów, wymaga

odpowiednich prac administracyjnych związanych z przyznawaniem i zarządzaniem certyfikatami.

Mając na uwadze wady i zalety przytoczonych rozwiązań, system powinien umożliwiać wybór stosowanych mechanizmów w zależności od zastosowania.

4 Implementacja platformy

Poniższy rozdział przedstawia podstawowe informacje, związane ze sposobem implementacji platformy. Rozdział rozpoczyna omówienie podjętych decyzji związanych z możliwymi technologiami implementacji platformy. W kolejnych częściach przedstawione zostały szczegóły techniczne związane z warstwą komunikacyjną, implementacją agregatorów stosowanych w procesie zmniejszania ilości danych przesyłanych do użytkowników, ze sposobem serializacji przesyłanych wiadomości oraz mechanizmami zapewnienia jednodostępu. Rozdział kończy omówienie wewnętrznej struktury części serwerowej oraz krótka charakterystyka jego głównych modułów funkcjonalnych.

4.1 Technologie implementacji

Zanim przedstawione zostaną możliwe do wykorzystania technologie komunikacyjne, warto omówić platformę programową, na której będą realizowane wszystkie fragmenty systemu, gdyż to ona wyznacza możliwe rozwiązania. Aby implementowany system cechowała łatwa rozszerzalność oraz możliwość integracji z innymi systemami, technologia wykorzystana do jego stworzenia powinna odznaczać się dostateczną dojrzałością. Wykorzystanie takiego rozwiązania gwarantuje możliwość wyboru formy komunikacji spośród dużego grona kandydatów, odpowiednie wsparcie ze strony twórców oraz zapewnia praktyczną przydatności technologii. Zdaniem autorów jedną z takich technologii jest platforma .Net firmy Microsoft wraz z obiektowym językiem C#. Głównymi składnikami platformy jest środowisko uruchomieniowe CLR (ang. *Common Language Runtime*), zestaw klas bazowych FCL (ang. *Framework Class Library*) oraz zestaw kompilatorów do języków wspieranych przez platformę.

Platforma .Net w warstwie komunikacyjnej udostępnia następujące rozwiązania:

- interfejs gniazd – strumieniowa lub datagramowa komunikacja oparta o protokoły TCP lub UDP,
- .Net Remoting – zdalne wywoływanie metod i komunikację w modelu rozproszonych obiektów,
- Windows Comunication Foundation (WCF) – biblioteka udostępniająca bogaty zestaw mechanizmów komunikacyjnych, opartych o specyfikację kontraktów dotyczących udostępnianych metod oraz przesyłanych danych, mogąca wykorzystywać do komunikacji protokół TCP lub HTTP.

Niezależnie od możliwości udostępnianych przez platformę, istnieją inne technologie niezwiązane z konkretną platformą lecz dostępne na platformie .Net. Dwie z nich

godne uwagi to CORBA [11] (ang. *Common Object Request Broker Architecture*) oraz ICE (ang. *Internet Communication Engine*) [10].

Z punktu widzenia otwartości systemu, na największą uwagę zasługują dwa ostatnie rozwiązania. Wynika to z faktu, iż dają one możliwość specyfikowania interfejsów komunikacyjnych w sposób niezależny od platformy, w postaci języków opisu interfejsu, tzw. IDL (ang. *Interface Definition Language*). Podobną cechę posiada komunikacja oparta o serwisy internetowe (ang. *Web Service*), udostępniana przez technologie WCF. Jednak z powodu mniejszej wydajności (badania prowadzone przez autorów) została ona odrzucona. W celu wyboru technologii najodpowiedniejszej dla tworzonego systemu, warto porównać dwa wymienione wcześniej rozwiązania pod kątem najważniejszych wymagań:

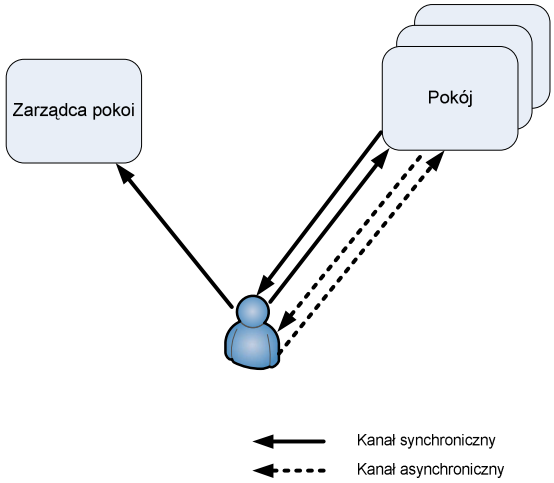
- Wydajność – z badań przeprowadzonych przez autorów wynika, że pod względem wydajności, ICE wypada lepiej niż CORBA, zarówno jeśli chodzi o opóźnienie przesyłanych komunikatów, jak i pasmo udostępniane w warstwie aplikacyjnej. Specyfikacja GIOP, protokołu wykorzystywanego do komunikacji w CORBA, definiuje asynchroniczne wywoływanie metod, ale nie wszystkie implementacje je wspierają. ICE posiada pełne wsparcie dla asynchronicznie wywoływanych metod,
- Wieloplatformowość – zarówno jedno, jak i drugie rozwiązanie umożliwia implementację interfejsów komunikacyjnych w wielu językach (choć w przypadku CORBA większość komercyjnych rozwiązań wspiera jedynie języki C++ oraz Java). Niestety nie istnieje pełne wsparcie dla CORBA dla platformy .Net, lecz jedynie częściowa implementacja specyfikacji [11]. ICE wspiera wiele języków programowania (C++, Java, Python, Ruby, PHP, C#),
- Heterogeniczne środowisko sieciowe – w celu umożliwienia komunikacji, w heterogenicznym środowisku sieciowym, jakim jest Internet, technologia wymiany wiadomości powinna umożliwiać komunikację pomiędzy stronami rozdzielonymi mechanizmami translacji adresów sieciowych (ang. *Network Address Translation*). Nie wszystkie istniejące implementacje standardu CORBA umożliwiają komunikację w takim środowisku, w przeciwieństwie do ICE, który posiada wbudowane mechanizmy pozwalające na pracę w takiej konfiguracji.

Z przytoczonych powyżej argumentów wyraźnie wynika, że ICE jest technologią najlepiej spełniającą wymagania stawiane tworzonemu systemowi.

4.2 Kanały komunikacji

IGCP korzysta z dwóch typów kanałów komunikacji, tj. synchronicznego oraz asynchronicznego. W celu zapewnienia komunikacji w przypadku stosowania mechanizmów translacji adresów sieciowych, zwrotne kanały (od serwera do klienta)

wykorzystują to samo połączenie, z którego korzysta klient przesyłając dane na serwer. Typy komunikatów wysyłanych przez dany kanał uwarunkowane są sposobem działania systemu. Rysunek 35 przedstawia wykorzystywane kanały komunikacji.



Rysunek 35. Wykorzystywane kanały komunikacji.

W zależności od konfiguracji, kanały komunikacji mogą dostarczać:

- możliwość szyfrowania komunikacji z wykorzystaniem SSL,
- wykorzystanie certyfikatów serwera do uwierzytelnienia przed klientem,
- wykorzystanie certyfikatów klienta do uwierzytelnienia przed serwerem.

4.2.1 Kanał komunikacji asynchronicznej

Wprowadzenie asynchronicznej komunikacji nie wymaga od nadawcy oczekiwania na potwierdzenie wykonania operacji przez odbiorcę. Dzięki takiemu rozwiązaniu, RTT (ang. *round trip time*) nie jest ograniczeniem dla wydajności.

Z tego powodu wiadomości modyfikujące obiekty są wysyłane tym typem kanału. Podobnie ma się sytuacja z innymi komunikatami, dla których nie jest ważny komunikat zwrotny, a jedynie zapewnienie, że zostaną one dostarczone do serwera – informacja o zwolnieniu blokady oraz dołączenie i odłączenie z pokoju.

Kanałem asynchronicznym, z powodu możliwości wystąpienia tzw. *race condition*, czyli sytuacji, kiedy system może nie działać poprawnie z powodu równoległego działania dwóch procesów, wysyłane są również żądania przydzielenia blokady obiektu. Mimo użycia asynchronicznego kanału komunikacji, operacje z punktu widzenia pozostałych modułów systemu wykonywane są w sposób synchroniczny, gdzie jako rezultat wykonania metody zwracana jest informacja o powodzeniu operacji.

4.2.2 Kanał komunikacji synchronicznej

W komunikatach synchronicznych przesyłane są informacje, dla których interaktywność jest mniej istotna od wygody użycia. Są to informacje związane z badaniem przepustowości łącza, dołączaniem do pokoju oraz komunikat, w których zarządca pokoi przesyła bilet uwierzytelniania. Podobnie ma się sytuacja z komunikatem ponownego przyłączenia do pokoju, kiedy to serwer przesyła klientowi informacje niezbędne do synchronizacji stanu jego sesji.

4.3 Interfejs komunikacyjny

W technologii ICE interfejsy komunikacyjne opisywane są w języku SLICE (*Specification Language for ICE*) niezależnym od platformy implementacji. SLICE definiuje kontrakt, który opisuje typy oraz interfejsy obiektów używanych w komunikacji pomiędzy klientem a serwerem. Opis ten jest niezależny od języka implementacji, dlatego nie jest istotne, aby klient i serwer korzystał z tego samego języka. Translator języka SLICE tłumaczy niezależną od języka definicję w specyficzne dla języka konstrukcje. Aktualnie ICE wspiera odwzorowanie do takich języków jak: C++, Java, C#, Python, Ruby i PHP.

W kolejnych podrozdziałach zostanie omówiony format wiadomości oraz definicje interfejsów, z których korzysta IGCP.

4.3.1 Format pakietu

Zanim zostanie omówiony format pakietów, rozumiany jako wiadomość lub komunikat kontrolny, konieczne jest wprowadzenie definicji typów pomocniczych (Listing 1), użytych przy definiowaniu formatu wiadomości. Zdefiniowane typy pomocnicze to:

- *ByteSeq* – sekwencja bajtów o zmiennej długości,
- *Tlv* – struktura definiująca wprowadzone wcześniej pojęcie struktury TLV (ang. *type, length, value*). Ponieważ ICE dostarcza informacji o długości przesyłanej tablicy, pole *length* może zostać pominięte. Pola tej struktury to:
  - *type* – typ przesyłanych danych w postaci liczby,
  - *value* – wartość w postaci zdefiniowanej powyżej sekwencji bajtów,
- *TlvSeq* – sekwencja struktur *TLV* używana do opisania właściwości obiektu,
- *TimeStampOpt* – opcjonalny parametr znacznika czasowego dołączany do pakietu. ICE nie udostępnia bezpośrednio możliwości definiowania argumentów opcjonalnych w metodach, stąd implementacja w postaci sekwencji (brak parametru przedstawiany jako pusta sekwencja).

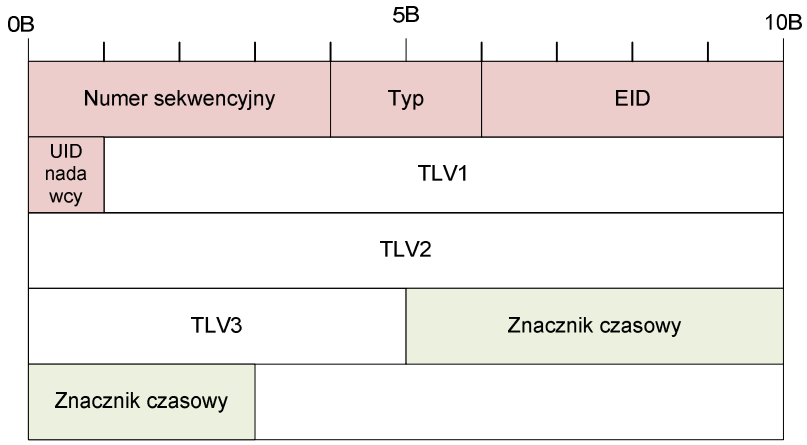
```
struct Tlv
{
   byte type;
   ByteSeq value;
};

sequence<Tlv> TlvSeq;

sequence<long> TimestampOpt;
```

Listing 1. Definicja typów pomocniczych w formacie SLICE.

W wiadomości można wyróżnić trzy grupy elementów:

- nagłówek – część pakietu, która zawsze występuje i zawiera informacje kontrolne, takie jak np. id użytkownika wysyłającego wiadomość,
- dane – część pakietu, która zawiera informacje o zmianach w danym obiekcie,
- dane opcjonalne – informacje, które mogą zostać dołączone do pakietu, lecz których brak nie wpłynie na poprawność przetworzenia pakietu.

Graficznie przedstawia to rysunek 36.



Rysunek 36. Graficzna reprezentacja formatu wiadomości: obowiązkowe pola (kolor czerwony), zmienna ilość elementów TLV o różnej długości oraz opcjonalny znacznik czasowy.

Kolejno zdefiniowane elementy wiadomości(Listing 2):

- *sn* – numer sekwencyjny wiadomości. Nadawany jako kolejno rosnące liczby jest unikatowy w ramach połączenia. Na jego podstawie zapewniana jest, w sposób programowy, kolejność dostarczenia pakietów, zgodna z kolejnością ich wysłania,
- *messageType* – typ wiadomości, który został utworzony na podstawie tematu wiadomości lokalnej,

- *eid* – identyfikator obiektu w postaci EID, powstały przez połączenie identyfikatora użytkownika (UID) i obiektu (OID),
- *uID* – identyfikator użytkownika, który wygenerował wiadomość,
- *tlvs* – tablica zmiennej długości zawierająca elementy TLV i opisująca, które właściwości obiektu mają zostać ustawione,
- *timestamp* – opcjonalny znacznik czasowy wysłania pakietu. Na jego podstawie, o ile został dołączony do komunikatu, może zostać zidentyfikowane wystąpienie zatoru.

```
struct Message
{
   int sn;
   short messageType;
   int eid;
   byte uID;
   TlvSeq tlvs;
   TimestampOpt timestamp;
};
```

Listing 2. Definicja wiadomości w formacie SLICE.

4.3.2 Interfejs zarządcy pokoi

Interfejs zarządcy pokoi (Listing 3) udostępnia jedną metodę (*Connect)* umożliwiającą podłączenie się do wybranego pokoju. Metoda ta zwraca informacje potrzebne do zlokalizowania pokoju. Przyjmowane argumenty to:

- *roomGuid* – jednoznaczny identyfikator pokoju, do którego użytkownik chce się dołączyć,
- *version* – wersja używanego protokołu. Rozpoznanie wersji protokołu używanego przez klienta, umożliwia odrzucenie połączenia w przypadku niezgodności wersji lub pracę w trybie zgodności z wcześniejszymi wersjami. Taka decyzja może zostać podjęta na podstawie zmian wprowadzanych w kolejnych wersjach protokołu,
- *secure* – wartość logiczna oznaczająca gotowość do dostępu do pokoju poprzez szyfrowany kanał,
- *compressed* – wartość logiczna oznaczająca gotowość do transmisji z wykorzystaniem kompresji. Sposób kompresji jest specyficzny dla ICE i jest niezależny od omówionej wcześniej agregacji,
- *out synRoomRef* – zwracany parametr zawierający informację o interfejsie pokoju, przez który to interfejs dane przesyłane są w sposób synchroniczny,
- *out asynRoomRef* – zwracany parametr zawierający informację o interfejsie pokoju, przez który to interfejs dane przesyłane są w sposób asynchroniczny.

Metoda może zgłosić następujące wyjątki:

- *InvalidProtocolVersionException* – serwer nie obsługuje wersji protokołu zadeklarowanej przez klienta,
- *AuthorizationException* – klient nie ma uprawnień do pobrania informacji na temat lokalizacji pokoju o podanym GUID,
- *AuthenticationException* – dane uwierzytelniające klienta nie są poprawne.

```
interface IIceRoomManager
{
   void Connect(
      string roomGuid,
      byte version,
      bool secure,
      bool compressed,
      out IIceSynchronousRoom* synRoomRef,
      out IIceAsynchronousRoom* asynRoomRef
   )
   throws InvalidProtocolVersionException,
          AuthorizationException,
          AuthenticationException;
};
```

Listing 3. Interfejs zarządcy pokoi zapisany w formacie SLICE.

4.3.3 Synchroniczny interfejs pokoju

Listing 4 przedstawia interfejs pokoju, przez który dane przesyłane są w sposób synchroniczny. Metody w tym interfejsie można podzielić na cztery grupy:

- Związane z przyłączaniem do pokoju: *JoinToRoom, RejoinToRoom, AttachReceiver,*
- Związane z tematami i typami wiadomości: *RegisterSubject*, *SubjectToMessageType*, *MessageTypeToSubject,*
- Związane z badaniem przepustowości łącza: *MarkStartSending*, *SendData,*
- Związane z wykrywaniem zatorów: *SendCongestionNotification.*

Każda metoda w omawianym interfejsie może zgłosić wyjątek:

- *AuthorizationException* – wyjątek zgłaszany w momencie, kiedy klient nie ma uprawnień do wykonania metody.

Kolejno omówione zostaną zdefiniowane w nim metody.

JoinToRoom – metoda wywoływana przy pierwszym dołączeniu do pokoju. Użytkownikowi identyfikowanemu za pomocą nazwy, przydzielony zostaje unikatowy w ramach pokoju identyfikator użytkownika (UID). Parametry metody to:

- *userName* – nazwa użytkownika, który dołącza się do pokoju,
- *out userId* – nadany unikatowy w ramach pokoju identyfikator użytkownika.

Metoda może zgłosić wyjątek:

- *RoomFullException* – wyjątek zgłaszany w sytuacji, kiedy do pokoju dołączyła się już maksymalna możliwa liczba uczestników i nie ma miejsca dla nowo dołączającego klienta.

RejoinToRoom – metoda wywoływana przy ponownym przyłączaniu się użytkownika do pokoju. Wywołanie tej metody ma miejsce w przypadku chwilowej utraty łączności pomiędzy klientem a serwerem. Parametry tej metody to:

- *userName* – nazwa użytkownika, który dołącza się do pokoju,
- *userId* – nadany podczas pierwszego połączenia unikatowy w ramach pokoju identyfikator użytkownika,
- *sn* – numer sekwencyjny ostatnio otrzymanego pakietu. Pozwala na wznowienie transmisji pakietów od tego pakietu.

AttachReceiver – po przyłączeniu do pokoju (metoda *JoinToRoom* lub *RejoinToRoom*), użytkownik podłącza swój synchroniczny interfejs, na którym będzie odbierał komunikaty wysyłane przez serwer. Parametry tej metody to:

- *userId* – nadany w czasie pierwszego przyłączenia, unikatowy w ramach pokoju identyfikator użytkownika,
- *ident* – identyfikator użytkownika nadany przez ICE'a – umożliwia zidentyfikowanie interfejsu wywołań zwrotnych klienta.

RegisterSubject – metoda rejestrująca na serwerze nowy temat dla przesyłanych wiadomości. Jeśli na serwerze istnieje już taki temat, wywołanie metody jest ignorowane. Parametry metody to:

- *subject* – rejestrowany temat w postaci ciągu znaków,
- *isVolatile* – wartość logiczna oznaczająca, czy wiadomości z danym tematem są ulotne, tj. dostarczenie kliku wiadomości tego samego typu równocześnie jest równoważne dostarczeniu ostatniej (np. pozycja wskaźnika myszy).

SubjectToMessageType – metoda umożliwiająca klientom przyłączonym do danego pokoju otrzymanie jednoznacznego odwzorowania tematu wiadomości na typ wiadomości. Używana przez klienta podczas serializacji lokalnej wiadomości do postaci TLV. Zwracana wartość to odwzorowany typ w postaci liczby. Parametry tej metody to:

- *subject* – temat w postaci ciągu znaków, dla którego ma zostać zwrócony typ.

MessageTypeToSubject – metoda mająca zastosowanie w sytuacji, kiedy konieczne jest uzyskanie odwzorowania typu wiadomości (liczby) na temat. Metoda jest symetryczna do wyżej opisanej metody *SubjectToMessageType,* tzn. *X = SubjectToMessageType(MessageTypeToSubject (X)).* Zwracana wartość to temat w postaci ciągu znaków odpowiadający podanemu typowi wiadomości.

- *messageType* – typ wiadomości jako liczba, dla którego ma zostać zwrócony temat.

MarkStartSending – metoda używana podczas badania przepustowości łącza. Wywołanie tej metody jest dla odbiorcy informacją, że bezpośrednio po niej zostanie wysłany ciąg danych, którego czas transmisji będzie mierzony. Przyjmowane argumenty to:

- *userId* – nadany identyfikator użytkownika, który rozpoczyna nadawanie.

SendData – druga metoda wykorzystywana podczas mierzenia przepustowości łącza. Wywołanie tej metody powoduje transmisję tablicy bajtów, której czas transmisji określa przepustowość łącza. Parametry metody to:

- *userId* – identyfikator użytkownika rozpoczynającego wysyłanie danych,
- *data* – tablica bajtów o różnej długości, której czas transmisji jest mierzony.

Metoda ta może zwrócić trzy różne typy odpowiedzi:

- wartość > 0 – oznacza zmierzoną przepustowość łącza w bitach na sekundę,
- -1 – oznacza konieczność wysłania takiej samej ilości danych ponownie, aby umożliwić dokładniejsze oszacowanie,
- -2 – oznacza konieczność wysłania większej ilości danych – zmierzony czas był za mały do oszacowania przepustowości łącza.

SendCongestionNotification – metoda używana do powiadomienia nadawcy o zbyt dużym tempie wysyłania pakietów, które prowadzi do powstania zatorów. Odbiorca powinien zareagować zwiększając stopień agregacji wysyłanych pakietów.

- *userId* – nadany identyfikator użytkownika, dla którego powstał zator,
- *packetSequenceNumber* – numer sekwencyjny pakietu, dla którego został wykryty zator. Informacja o pakiecie zapobiega kaskadowemu zwiększaniu stopnia agregacji w wyniku chwilowego zatoru.

```
interface IIceSynchronousRoom
{
   void JoinToRoom(string userName, out byte userId)
            throws RoomFullException,
                   AuthorizationException;
   void RejoinToRoom(string userName, byte userId, int ssn)
            throws AuthorizationException;
   void AttachReceiver(byte userId, Ice::Identity ident)
            throws AuthorizationException;
   void RegisterSubject(string subject, bool isVolatile)
            throws AuthorizationException;
   short SubjectToMessageType(string subject)
            throws AuthorizationException;
   string MessageTypeToSubject(short messageType)
            throws AuthorizationException;
   void MarkStartSending(byte userId)
            throws AuthorizationException;
   int SendData(byte UserId, ByteSeq data)
            throws AuthorizationException;
   void SendCongestionNotification(byte userId,
                                   int packetSequenceNumber)
            throws AuthorizationException;
};
```

Listing 4. Synchroniczny interfejs pokoju zapisany w formacie SLICE.

4.3.4 Asynchroniczny interfejs pokoju

Listing 5 przedstawia interfejs pokoju, przez który dane przesyłane są w sposób asynchroniczny. Kolejno omówione zostaną zdefiniowane w nim metody.

AttachReceiver – metoda analogiczna jak *IIceSynchronousRoom.AttachReceiver*, ale mająca zastosowanie w przypadku dołączania interfejsu wywołań zwrotnych dla kanału asynchronicznego. Parametry tej metody to:

- *userId* – nadany w czasie pierwszego przyłączenia unikatowy w ramach pokoju identyfikator użytkownika,
- *ident* – identyfikator użytkownika nadany przez ICE'a – umożliwia zidentyfikowanie interfejsu wywołań zwrotnych klienta.

Metoda może zgłosić wyjątek *AuthorizationException* w podobnych okolicznościach, jak dla każdej metody interfejsu synchronicznego.

SendMessages – metoda wykorzystywana do wysyłania wiadomości na serwer. Parametry tej metody to:

- *messages* – tablica wiadomości, które mają zostać dostarczone do pokoju. Format wiadomości został opisany wcześniej.

Metoda ta może zgłaszać poniżej wymienione wyjątki. Jeśli metoda zgłosiła wyjątek, to żadna z wiadomości nie została przetworzona na serwerze – konieczna jest retransmisja wszystkich wiadomości.

- *NoValidCallbackException* – złapanie takiego wyjątku informuje klienta o konieczności ponownego nawiązania połączenia z serwerem i utworzenia aktywnych wywołań zwrotnych. Taka sytuacja ma miejsce w przypadku, kiedy serwer utracił połączenie zwrotne do klienta,
- *ObjectLockedException* – złapanie takiego wyjątku informuje klienta o braku blokady na obiekt, który jest modyfikowany przez jedną z wiadomości. Reakcją klienta powinno być uzyskanie blokady i ponowna transmisja wszystkich wiadomości.

Ping – metoda wykorzystywana podczas badania aktywności łącza. Wystąpienie błędu w czasie wywołania tej metody oznacza brak aktywnego połączenia z serwerem. Metoda nie przyjmuje żadnych argumentów.

```
["ami"]
interface IIceAsynchronousRoom
{
   MessageSequence AttachReceiver(byte userId,
                                  Ice::Identity ident)
            throws AuthorizationException;

   void SendMessages(MessageSequence messages)
            throws RestablishConnectionRequiredException,
                   ObjectLockedException;

   void Ping();
};
```

Listing 5. Asynchroniczny interfejs pokoju w formacie SLICE.

4.3.5 Synchroniczny interfejs klienta

Listing 6 przedstawia interfejs klienta, przez który dane z serwera przesyłane są w sposób synchroniczny. Interfejs ten jest funkcjonalnie podobny do interfejsu pokoju, jednak nie są przesyłane w argumentach informacje o identyfikatorze użytkownika. Nie jest to konieczne, ponieważ nadawcą zawsze jest serwer, którego nie trzeba jednoznacznie identyfikować. Kolejno omówione zostaną zdefiniowane w nim metody.

MarkStartSending – metoda analogiczna do *ISynchronousRoom.MarkStartSending*. Używana jest podczas badania przepustowości łącza od serwera do klienta. Wywołanie tej metody jest dla odbiorcy informacją, że bezpośrednio po niej zostanie wysłany ciąg danych, którego czas transmisji będzie mierzony. Metoda nie przyjmuje argumentów.

SendData – metoda analogiczna do *ISynchronousRoom.SendData. W*ykorzystywana podczas mierzenia przepustowości łącza od serwera do klienta. Wywołanie tej metody powoduje transmisję tablicy bajtów, której czas transmisji określa przepustowość łącza. Parametry metody to:

- *data* – tablica bajtów o różnej długości, której czas transmisji jest mierzony.

Metoda ta może zwrócić trzy różne typy odpowiedzi:

- wartość > 0 – oznacza zmierzoną przepustowość łącza w bitach na sekundę,
- -1 – oznacza konieczność wysłania takiej samej ilości danych ponownie, aby umożliwić dokładniejsze oszacowanie,
- -2 – oznacza konieczność wysłania większej ilości danych – zmierzony czas był za mały do oszacowania przepustowości łącza.

```
interface IIceSynchronousClient
{
   void MarkStartSending();
   int SendData(ByteSeq data);
};
```

Listing 6. Synchroniczny interfejs klienta.

4.3.6 Asynchroniczny interfejs klienta

Listing 7 przedstawia interfejs klienta, przez który dane z serwera przesyłane są w sposób asynchroniczny. Interfejs ten jest funkcjonalnie podobny do interfejsu pokoju, jednak nie są przesyłane w argumentach informacje o identyfikatorze użytkownika. Nie jest to konieczne, ponieważ nadawcą zawsze jest serwer, którego nie trzeba jednoznacznie identyfikować. Dodatkowo znajdują się w nim informacje o dołączaniu i odłączaniu użytkowników w ramach pokoju.

W interfejsie tym można wyróżnić trzy grupy funkcjonalne:

- Przesyłanie wiadomości: *SendMessages,*
- Informacje o aktywności użytkowników: *UserJoined, UserDisjoined,*
- Zarządzanie połączeniem: *SendCongestionNotification, Ping.*

Kolejno omówione zostaną zdefiniowane w nim metody.

SendMessages – metoda analogiczna jak *IAsynchronousRoom.SendMessages,* ale bez parametru identyfikującego użytkownika. Wykorzystywana do wysyłania wiadomości od serwera do klienta. Parametry tej metody to:

- *messages* – tablica wiadomości, które mają zostać dostarczone do pokoju. Format wiadomości został opisany wcześniej.

UserJoined – informuje odbiorcę o fakcie dołączenia nowego użytkownika do pokoju. Parametry tej metody to:

- *userName* – tekstowa nazwa użytkownika, który dołączył do pokoju,
- *userId* – nadany, unikatowy w ramach pokoju, identyfikator użytkownika, który dołączył do pokoju.

UserDisjoined – informuje odbiorcę o fakcie odłączenia się użytkownika od pokoju. Parametry tej metody to:

- *userId* – nadany, unikatowy w ramach pokoju, identyfikator użytkownika, który odłączył się od pokoju.

SendCongestionNotification – metoda analogiczna jak *IAsynchronousRoom.Send-CongestionNotification,* ale bez parametru identyfikującego użytkownika. Używana do powiadomienia nadawcy o zbyt dużym tempie wysyłania pakietów, które prowadzi do powstania zatorów. Odbiorca powinien zareagować zwiększając stopień agregacji wysyłanych pakietów.

- *packetSequenceNumber* – numer sekwencyjny pakietu, dla którego został wykryty zator. Informacja o pakiecie zapobiega kaskadowemu zwiększaniu stopnia agregacji w wyniku chwilowego zatoru.

Ping – metoda wykorzystywana podczas badania aktywności łącza. Wystąpienie błędu w czasie wywołania tej metody oznacza brak aktywnego połączenia z serwerem. Metoda nie przyjmuje żadnych argumentów.

```
["ami"]
interface IIceAsynchronousClient
{
      void SendMessages(MessageSequence messages);
      void UserJoined(string userName, byte userID);
      void UserDisjoined(byte userID);
      void SendCongestionNotification(int packetSequenceNumber);
      void Ping();
};
```

Listing 7. Asynchroniczny interfejs klienta w formacie SLICE.

4.4 Serializacja i deserializacja

Serializacja i deserializacja to operacje wykonywane po stronie klienckiej po otrzymaniu wiadomości z serwera lub podczas wysyłania ich do sieci. Serializatory przechowywane są w repozytorium, z którego pobierane są w miarę potrzeby. Każdy moduł odpowiedzialny za serializację oraz deserializację implementuje interfejs (Listing 8).

```
interface ISerializer

{

   TlvHolder[] Serialize(EventArgs msg);

   EventArgs Deserialize(TlvHolder tlvs);

   bool IsVolatile {get; }

}
```

Listing 8. Interfejs serializatora w C#.

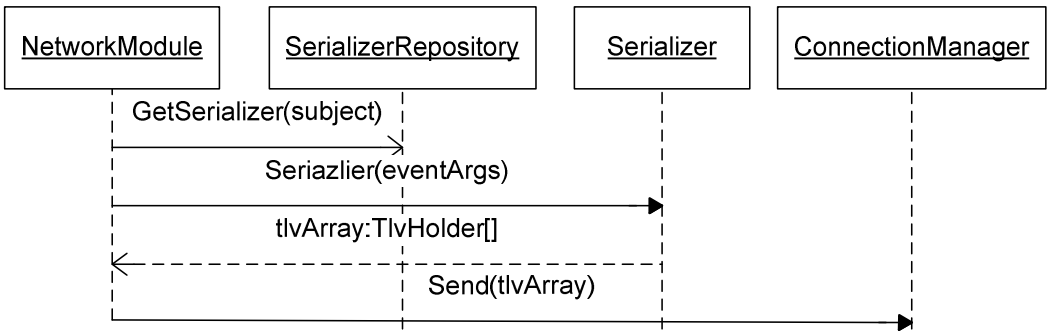
Znaczenie poszczególnych metod omówiono poniżej.

Serialize – metoda odpowiedzialna za przeprowadzanie konwersji lokalnej dla aplikacji klienckiej postaci obiektów do postaci ogólnej, tzn. TLV.

- *msg* – lokalna reprezentacja operacji dokonywanych na obiektach,
- *TlvHolder[]* – tablica TLV reprezentująca atrybuty obiektu będącego parametrem.

Deserialize – metoda odwrotna do serializacji, konwertująca tablicę TLV do lokalnej reprezentacji obiektu.

IsVolatile – właściwość określająca ulotność typu wiadomości, dla której stworzony został serializator. Ulotność wiadomości oznacza, że serwer powinien w stanie przechowywać jedynie ostatnią wiadomość tego typu dotyczącą każdego obiektu, np. w przypadku wskaźnika wystarczy pamiętać jedynie ostatnią pozycję.



Rysunek 37. Serializacja wysyłanych danych.

Diagram sekwencji opisujący proces serializacji przedstawia rysunek 37. Moduł sieciowy przed wysłaniem wiadomości wyszukuje w repozytorium odpowiedni dla danego tematu serializator. Po otrzymaniu instancji serializatora, wykorzystuje go do zamiany lokalnej reprezentacji obiektu na tablicę TLV.

4.5 Agregacja

W celu umożliwienia agregacji pakietów, w systemie musi zostać zarejestrowany odpowiedni agregator. Agregator jest klasą implementującą interfejs agregatora (Listing 9).

```
interface IAggregator
 {
      TlvAggregationMessage[] Aggregate(
            TlvAggregationMessage[] messages, int  agregationLevel);
 }
```

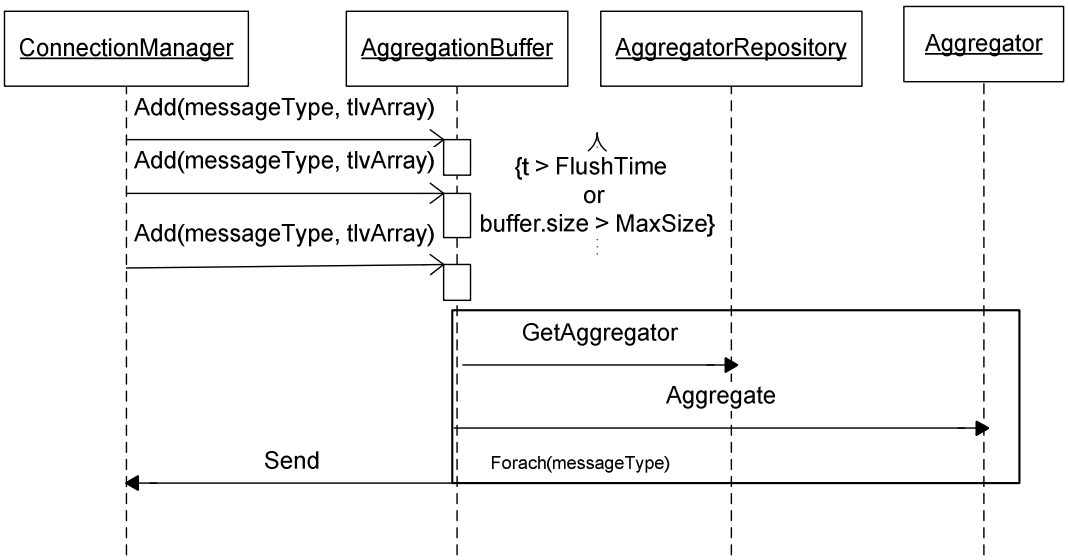
Listing 9. Interfejs agregatora w C#.

Interfejs agregatora posiada jedną metodę:

Aggregate – metoda odpowiedzialna za przeprowadzanie agregacji otrzymywanych wiadomości.

- *messages* – tablica TLV dotycząca jednego obiektu,
- *aggregationLevel* – liczba całkowita określająca poziom agregacji.

Zarówno po stronie serwera, jak i po stronie klienta, agregacją pakietów czekających na wysłanie zajmują się te same struktury (Rysunek 38).



Rysunek 38. Schemat działania modułu agregacji.

Podstawowym elementem modułu agregacji jest bufor, do którego trafiają tablice TLV przeznaczone do agregacji. Proces agregacji jest rozpoczynany w momencie, gdy rozmiar bufora przekroczy maksymalną wartość lub gdy zostanie przekroczony maksymalny czas pomiędzy kolejnymi agregacjami. Bufor korzysta z instancji agregatorów zarejestrowanych w repozytorium. Dla każdego typu wiadomości bufor odpytuje o odpowiedni agregator, a po przeprowadzeniu całego procesu agregacji, przekazuje wynik do obiektu odpowiedzialnego ze przesyłanie pakietów przez sieć.

4.6 Szeregowanie wiadomości

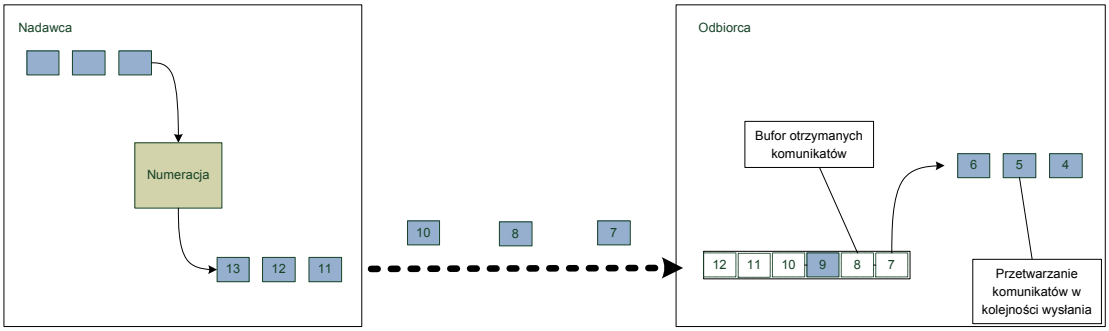
Jak zostało wcześniej powiedziane, od warstwy komunikacyjnej (ang. *middlware*) nie wymaga się dostarczania wiadomości w kolejności. Takie podejście powoduje, że w przypadku przetwarzania wiadomości na serwerze, z wykorzystaniem wielu wątków, mogą wystąpić warunki powodujące *race condition*. Wykorzystanie tylko jednego wątku na serwerze wyeliminowałoby ten problem, ale znacząco wpłynęłoby też na wydajność.

Rozwiązaniem tej niedogodności jest wprowadzenie programowej kontroli dostarczania komunikatów: w ramach jednego kanału komunikacyjnego wiadomości na serwerze przetwarzane są w kolejności, w której zostały wysłane. Ponieważ szeregowanie ma miejsce tylko w ramach wiadomości od jednego klienta, nadal możliwa jest równoległa obsługa wielu użytkowników.

Mechanizm szeregowania składa się z dwóch modułów: po stronie nadawcy i po stronie odbiorcy. Po stronie nadawcy wszystkie wysyłane wiadomości są znakowane kolejnymi liczbami. W trakcie wysyłania lub odbierania w warstwie komunikacyjnej,

wiadomości mogą zmienić kolejność i zamienione dotrzeć do modułu po stronie odbiorcy. Tam są one umieszczane w buforze, z którego są pobierane dopiero w momencie, kiedy utworzony zostanie ciągły blok uszeregowanych wiadomości (Rysunek 39).

Aby zapewnić szeregowanie wiadomości w obu kierunkach: od serwera do klienta i od klienta do serwera, zarówno klient, jak i serwer posiadają moduł nadawczy numerujący wysłane wiadomości oraz moduł odbiorczy, który szereguje odebrane. Ponadto, w celu uproszczenia implementacji, po zerwaniu połączenia numeracja wiadomości rozpoczyna się od początku.



Rysunek 39. Szeregowanie wiadomości po stronie odbiorcy w kolejności wysyłania.

4.7 Przykład implementacji serializatora i agregatora

W celu lepszego zobrazowania mechanizmów serializacji i agregacji, w poniższym punkcie przedstawiona zostanie implementacja prostych modułów serializacji oraz agregacji przesyłanych wiadomości.

Zadaniem tworzonych modułów będzie obsługa wiadomości dotyczących wskaźnika multimedialnego. Jest to mechanizm często stosowany w aplikacjach zdalnej współpracy o charakterze interaktywnym. Za jego pomocą użytkownicy mogą pokazywać wybrane elementy graficzne pozostałym użytkownikom. W przypadku wskaźnika stanem jest para liczb określających aktualne położenie na ekranie. Warto również dodać, że ważne jest jedynie aktualne położenie wskaźnika, a historia stanu jest mało istotna.

Implementację funkcji *Serialize* oraz *Deserialize* z interfejsu *ISerializator* przedstawia listing 10. W metodzie odpowiedzialnej za serializację, na podstawie wewnętrznej reprezentacji obiektu (obiekt typu *EventArgs*), dokonywana jest konwersja do tablicy TLV. Na podstawie atrybutów x oraz y, tworzone są dwa wpisy określające położenie w pionie i poziomie, których wartością jest tablica bajtów przedstawiająca dwie liczby typu zmiennoprzecinkowego. W metodzie desierializującej dokonywana jest

konwersja odwrotna, tzn. tworzony jest obiekt wewnętrznej reprezentacji obiektu (*PointerEventArgs*), którego atrybuty przyjmują wartości zapisane wcześniej przez serializator w tablicy TLV.

```
TlvHolder[] Serialize(EventArgs msg)
{
      List<TlvHolder> tlvs = new List<TlvHolder>();

      TlvHolder tlv;
      tlv = new TlvHolder("X", BitConverter.GetBytes(msg.x));
      tlvs.Add(tlv);

      TlvHolder tlv;
      tlv = new TlvHolder("Y", BitConverter.GetBytes(msg.y));
      tlvs.Add(tlv);

      return tlvs.ToArray();
}

EventArgs Deserialize(TlvHolder tlvs)
{
   EventArgs result = new PointerEventArgs();
   foreach (TlvHolder t in tlvHolders)
   {
      if (t.Type ==  “X")
         result.x = BitConverter.ToInt(t.Value,0);
      if (t.Type == "Y")
         result.y = BitConverter.ToInt(t.Value,0);
   }
   return result;
}
```

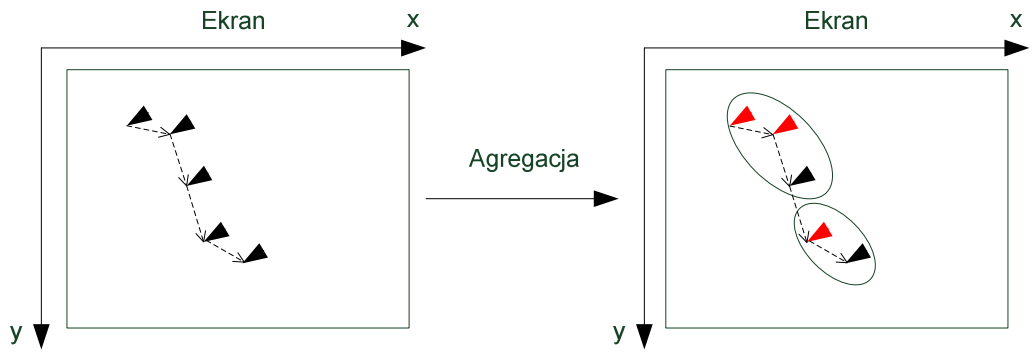
Listing 10. Implementacja serializatora obiektu wskaźnika multimedialnego.

W celu zmniejszenia wielkości strumienia przesyłanych danych dla wiadomości dotyczących wskaźnika powinien zostać utworzony agregator. Podstawą do przeprowadzenia agregacji tego typu wiadomości jest fakt, że liczy się jedynie ostatnia pozycja wskaźnika w agregowanej grupie pakietów, a więc proces agregacji może polegać na odrzuceniu wszystkich oprócz ostatniego (Listing 11).

```
TlvAggregationMessage[] Agregate(TlvAggregationMessage[] messages,
                                 int  agregationLevel)
{
   TlvAggregationMessage[] returnMessages =
            new TlvAggregationMessage[1]
   returnMessages = messages[messages.Length – 1];
   return returnMessages;
}
```

Listing 11. Implementacja agregatora wskaźnika multimedialnego.

Rysunek 40 przedstawia ideę działania agregatora. Pięć pozycji wskaźnika zostało poddanych agregacji w dwóch turach, najpierw pierwsze trzy, a potem kolejne dwa. Z każdej grupy odrzucone zostały wszystkie pozycje oprócz ostatniej. Po prawej stronie rysunku przedstawiony został rezultat agregacji (pozycje zaznaczone na czerwono nie zostaną przesłane).



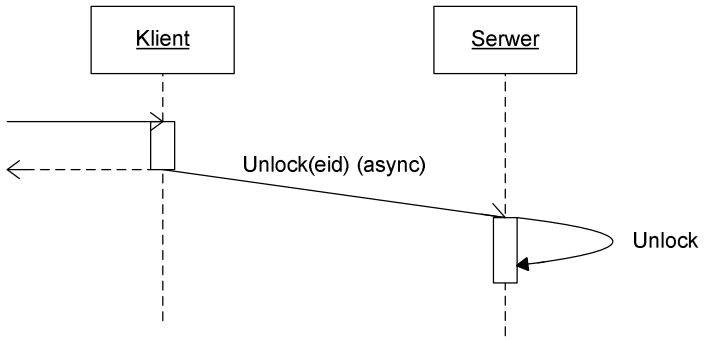
Rysunek 40 Agregacja wskaźnika multimedialnego

4.8 Mechanizm zapewniania jednodostępu

Jednodostęp do obiektów jest zapewniany poprzez wymuszenie zakładania blokad na obiekt przez użytkownika, który chce go zmodyfikować. Koncepcja przedstawiona została w 3.4, natomiast w tym rozdziale zostaną przedstawione szczegóły implementacyjne tego mechanizmu: najpierw schemat komunikacji pomiędzy klientem a serwerem, a następnie część kliencka i serwerowa.

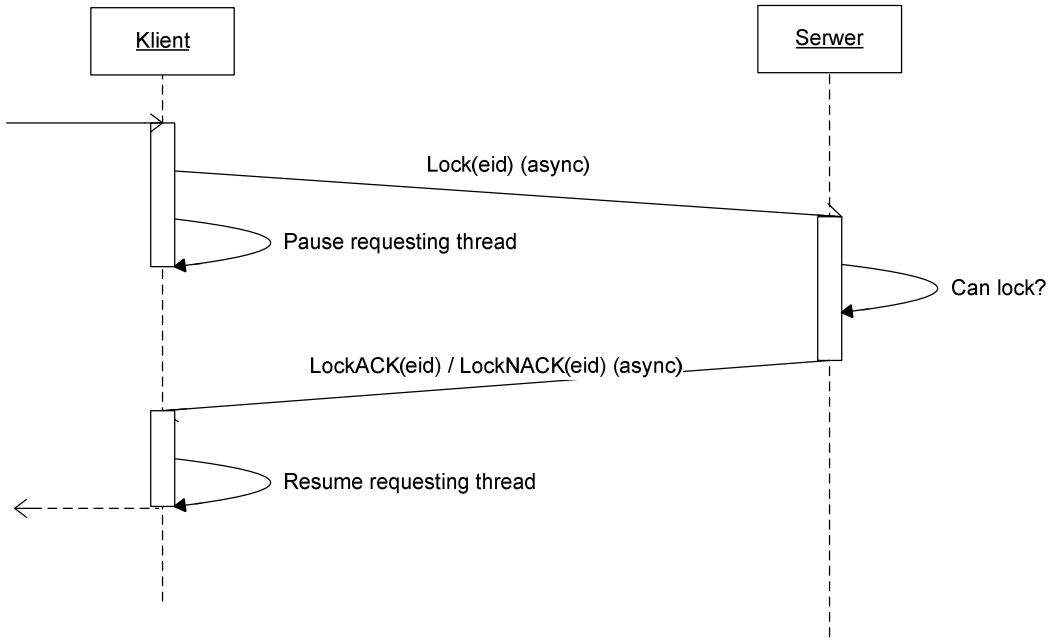
Wiadomości dotyczące modyfikacji obiektu wysyłane są kanałem asynchronicznym, co powoduje, że komunikaty kontrolne dotyczące blokad na te obiekty również muszą być wysyłane tym kanałem. Gdyby odstąpić od tego założenia, mogłaby zdarzyć się sytuacja, kiedy komunikat zdejmujący blokadę dotarłby do serwera wcześniej niż ostatnia wiadomość modyfikująca obiekt wysłana drugim kanałem – kolejność dostarczania komunikatów zapewniona jest programowo tylko w ramach jednego kanału.

Komunikat zdejmowania blokady z obiektu jest dla serwera informacją, że obiekt nie będzie już modyfikowany. Klient nie oczekuje od serwera żadnej informacji zwrotnej (Rysunek 41), dlatego wykorzystanie kanału asynchronicznego nie wprowadza istotnych zmian w implementacji.



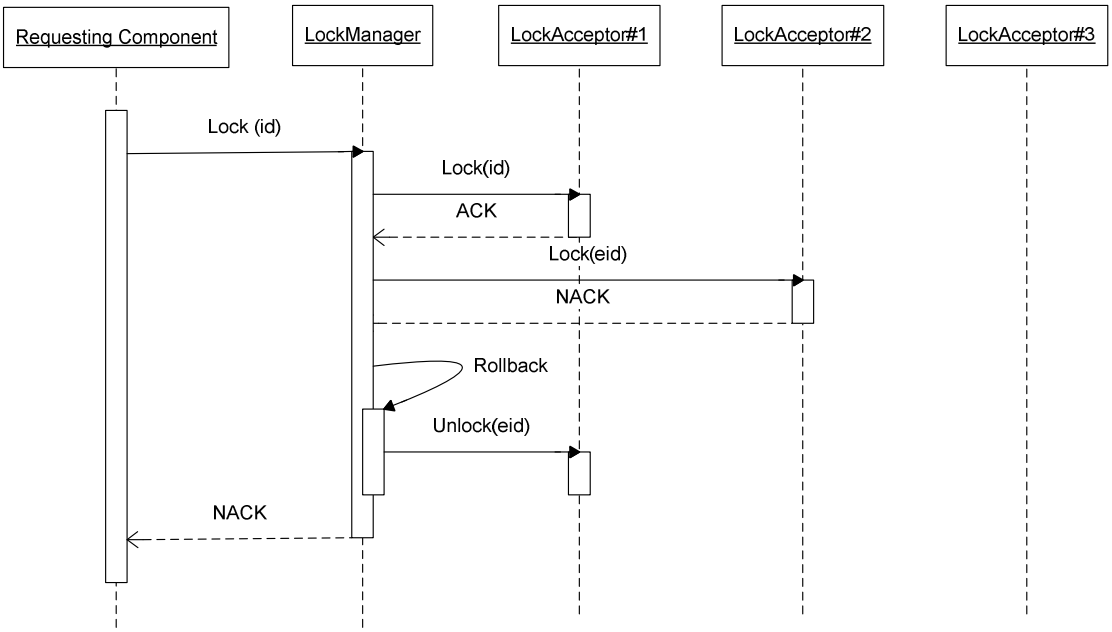
Rysunek 41. Komunikat zdjęcia blokady wysłany przez kanał asynchroniczny.

Inaczej wygląda sytuacja z żądaniem założenia blokady na obiekt, gdzie klient oczekuje informacji o powodzeniu operacji. Asynchronicznie wysłany komunikat nie może zwracać informacji z serwera, dlatego konieczne jest wprowadzanie dodatkowego mechanizmu: w momencie wysłania asynchronicznego żądania na serwer, klient dodaje do lokalnej listy informację o żądaniu, na które nie uzyskał jeszcze od serwera odpowiedzi. Na kliencie blokowany jest wątek, który oczekuje na informację o powodzeniu operacji. Po otrzymaniu żądania, serwer sprawdza, czy możliwe jest uzyskanie blokady i wysyła nowy asynchroniczny komunikat łączem zwrotnym do klienta. Wtedy klient odblokowuje czekający wątek. Dzięki takiemu podejściu, udostępniony został synchroniczny interfejs programistyczny, który ukrywa sposób przesyłania komunikatów w sposób asynchroniczny (Rysunek 42).



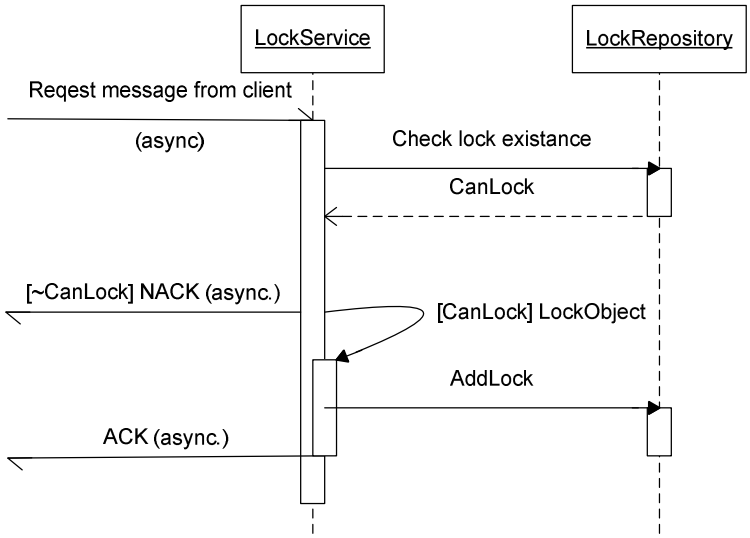
Rysunek 42. Założenie blokady na obiekt z wykorzystaniem kanałów asynchronicznych.

Aby zapewnić rozszerzalność platformy również w aspekcie zarządzania jednodostępem, po stronie klienta wprowadzony został mechanizm kolektywnej akceptacji żądań założenia blokad na obiekt. Komponent, który chce zarządzać jednodostępem musi implementować interfejs *ILockAcceptor*. W momencie, kiedy jeden z komponentów aplikacji zgłosi do zarządcy blokad (*LockManager*) potrzebę uzyskania blokady, wysłana jest informacja do wszystkich zarejestrowanych komponentów implementujących *ILockAcceptor*. Dopiero po uzyskaniu potwierdzenia od wszystkich, komponent wysyłający żądanie otrzymuje informację o powodzeniu operacji. W przypadku, kiedy jeden z komponentów odrzuci żądanie, konieczne jest uruchomienie procedury odtwarzania (ang. *rollback*). IGCP jest komponentem, który implementuje *ILockAcceptor*, umożliwiając zarządzanie jednodostępem do obiektów w ramach sesji. Przykładową sekwencję operacji, w wyniku której żądanie zostało odrzucone, przedstawia rysunek 42.



Rysunek 43. Sekwencja wywołań dla sytuacji, w której LockAcceptor#2 odrzuca żądanie założenia blokady.

Po stronie serwera w momencie odebrania żądania założenia blokady sprawdzane jest, czy inny użytkownik nie posiada już blokady. Jeśli tak nie jest, to zakładana jest blokada, a następnie wysyłana jest informacja zwrotna. Ponieważ obiekty posiadają unikalny identyfikator w ramach sesji (OID), rozwiązaniem jest utrzymywanie na serwerze tablicy asocjacyjnej, indeksowanej identyfikatorami obiektu. Istnienie wartości o zadanym kluczu świadczy o założonej blokadzie na obiekt. Każda wartość w tabeli ma postać pary: użytkownik, dla którego została założona blokada oraz czas jej założenia. Czas ten jest wykorzystywany do okresowego ściągania blokad z obiektów, które przez określony czas nie zostały odblokowane. Działanie opisanego mechanizmu przedstawia rysunek 43.



Rysunek 44. Mechanizm zakładania blokad po stronie serwera.

4.9 Architektura serwera

Część serwerowa tworzonej platformy oparta została o architekturę warstwową. Wydzielenie logicznie odrębnych fragmentów oraz dokładne ustalenie zadań, które powinny spełniać, ma pozytywny wpływ na łatwość modyfikacji oraz usuwanie błędów odnajdywanych w czasie testowania.

Rysunek 45. Architektura części serwerowej.

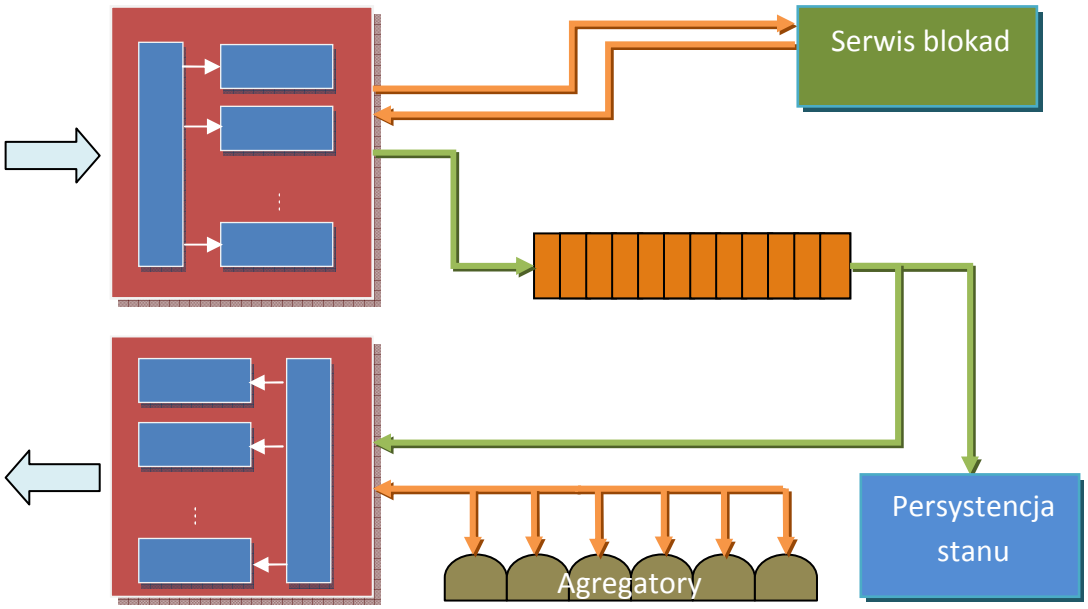
W architekturze serwera wyróżnić można trzy warstwy – Rysunek 45. Każdy pakiet docierający do części serwerowej lub przez nią generowany, przechodzi przez wszystkie warstwy – od najniższej do najwyższej w przypadku docierających i w przeciwnym kierunku w przypadku generowanych przez serwer.

Każda z warstw definiuje interfejs, poprzez który udostępnia usługi warstwom wyższym (Tabela 1). Dzięki takiej konstrukcji, każda warstwa zależna jest jedynie od warstwy bezpośrednio pod nią. Warto zauważyć, że z usług warstwy logiki mogą korzystać aplikacje administracyjne wykorzystywane przez osoby zarządzające platformą, dlatego na liście usług znalazły się funkcje związane także z tym aspektem.

Warstwa	Udostępniane usługi
Logiki	• zarządzanie cyklem życia pokoju, • uwierzytelnianie i autoryzacja użytkownika, • przechowywanie stanu, • zarządzanie jednodostępem.
Zarządzania połączeniem	• niezawodne dostarczanie wysłanych wiadomości, • monitorowanie stanu łącza, • szeregowanie wiadomości dochodzących, • demultipleksacja nadchodzących wiadomości, • agregacja pakietów wychodzących.
Middleware	• interfejs komunikacyjny, • marshaling/unmarshaling, • synchroniczna i asynchroniczna komunikacja.

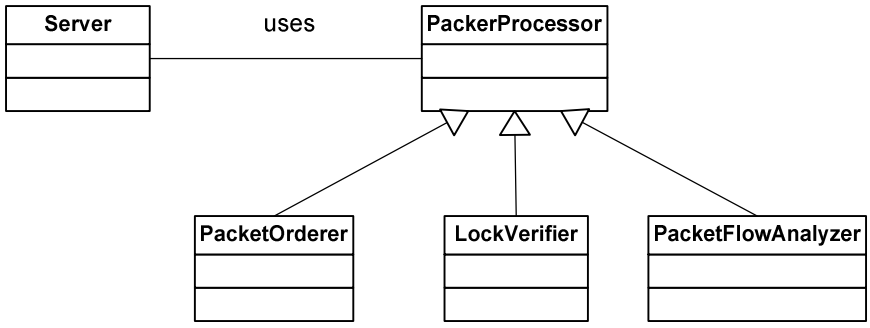
Tabela 1. Usługi warstw części serwerowej.

Każdy komunikat docierający do serwera jest przetwarzany, a następnie przesyłany do wszystkich uczestników sesji poza autorem w celu propagacji operacji, którą reprezentuje (pakiety kontrolne nie są propagowane). Schemat przetwarzania pakietów po stronie serwera przedstawiony został na rysunku 46.



Rysunek 46. Przetwarzanie pakietu po stronie serwera.

Wiele z usług, udostępnianych przez warstwę zarządzania połączeniem oraz logiki, można przedstawić w postaci ciągu operacji dokonywanych na otrzymywanych przez serwer pakietach. Przykładowo, po otrzymaniu pakietu uruchamiany jest moduł analizy przepływu, weryfikacji blokad oraz szeregowania otrzymanych pakietów.

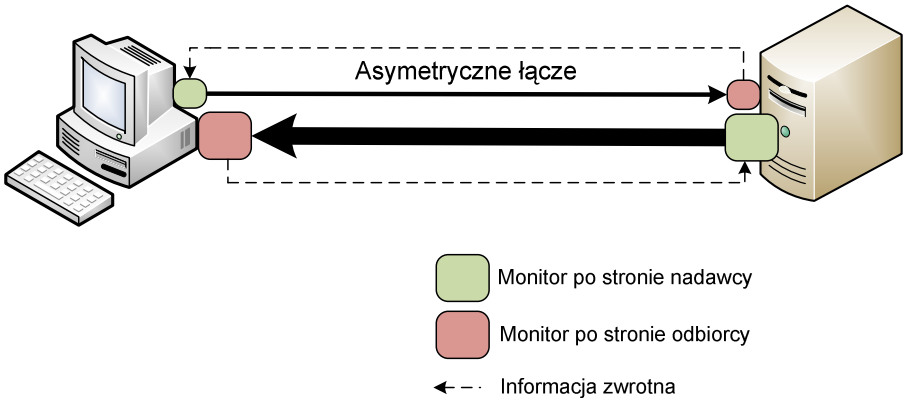


Rysunek 47. Moduł przetwarzania pakietów.

W ramach części serwerowej został zdefiniowany interfejs modułu przetwarzającego pakiet, który jest implementowany przez konkretne realizacje. Struktura rozwiązania została zrealizowana zgodnie z wzorcem projektowym „*Chain of Responsibility*". Rysunek 47 przedstawia schemat rozwiązania.

5 Szacowanie przepustowości łącza

Praca w środowisku heterogenicznym nakłada wymagania, które muszą zostać spełnione przez tworzoną platformę. Użytkownicy korzystający z IGCP dysponują łączami różnej przepustowości i dlatego dane, wysyłane do konkretnego użytkownika, muszą być osobno przetwarzane. Dodatkowo stan łącza może okresowo zmieniać się ze względu na możliwość wykorzystania wielu aplikacji lub pracę innych użytkowników sieci. Dokładny opis modyfikacji, którym poddawane są rozsyłane komunikaty został przedstawiony w analizie (2.3). Warto jedynie przypomnieć, że ich wynik jest zależny od dostępnego pasma na łączu, które może być również asymetryczne. Poglądowo zostało to przedstawione na rysunku 48.



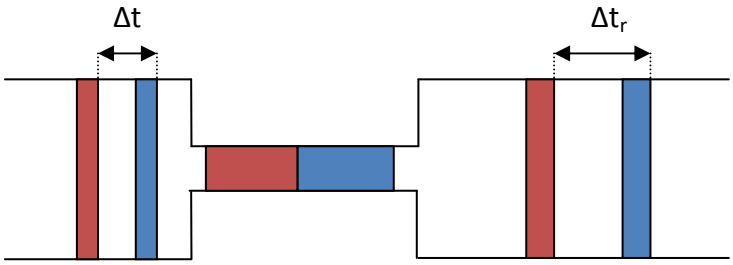
Rysunek 48. Szacowanie przepustowości łącza asymetrycznego.

5.1 Mechanizmy szacowania dostępnego pasma

W niniejszym podrozdziale zostaną przedstawioną dwie metody szacowanie przepustowości łącza. Pierwsza, oparta na badaniu różnic czasowych, umożliwia wykrycie zatorów. Druga, oparta o wysycanie łącza, pozwala na oszacowanie wartości przepustowości.

5.1.1 Pomiary oparte na badaniu różnic czasowych

Problem badania dostępnego pasma, mając duże znaczenie praktyczne, doczekał się wielu rozwiązań w postaci narzędzi dokonujących testowych pomiarów, np. c*probe* [4], *pipechar* [18], *Delphi* [32], *pathload* [15]. Spośród wymienionych narzędzi najlepsze rezultaty w przypadku sieci o skomplikowanej strukturze daje ostatnie z nich [15].



Rysunek 49. Zator w sieci.

Przytoczone rozwiązanie opiera się o następującą obserwację: jeżeli czas pomiędzy odebraniem kolejnych pakietów jest większy, niż czas pomiędzy nadawaniem kolejnych pakietów, to znaczy, że na ścieżce wystąpił zator. Przez zator rozumiemy sytuację, w której pośredniczące urządzenia sieciowe nie są w stanie wysłać danych z intensywnością równą szybkości ich otrzymywania (Rysunek 49). Jeśli taki stan utrzymywałby się dłużej, to doszłoby do przepełnienia buforów i gubienia pakietów.

Na przedstawionym rysunku, ścieżka pomiędzy nadawcą a odbiorcą składa się trzech łącz, z czego pierwsze i trzecie mają tę samą przepustowość, natomiast środkowe mniejszą. Jeżeli nadawca wysyła pakiety z dostatecznie dużą szybkością, tzn. czas pomiędzy początkiem wysyłania kolejnych pakietów jest odpowiednio mały (oznaczony jako Δt_s), to środkowe łącze może nie zdążyć z wysłaniem pierwszego pakietu zanim dotrze do niego drugi. W takim przypadku początki kolejnych pakietów zostaną rozsunięte w czasie. Nawet jeśli kolejne łącze posiada większą przepustowość, to różnica w czasie mierzona przez odbiorcę (oznaczona jako Δt_s) będzie większa niż różnica w czasach nadawania.

Wykrywanie zatoru w sieci można zatem sprowadzić do badania warunku $\Delta t_r > \Delta t_s$, jednak nie jest on wystarczający do zbadania przepustowości. Aby poznać szukany parametr ścieżki, można skorzystać z algorytmu wyszukiwania połówkowego, tzn. przechowywać aktualne oszacowanie w postaci dwóch liczb ograniczających rzeczywistą wartość z góry i z dołu. W każdym kroku wyliczamy średnią z aktualnych granic i wysyłamy serię pakietów. Jeżeli doszło do zatoru, średnia jest nowym maksimum, w przeciwnym przypadku minimum.

Autorzy omawianego rozwiązania wzięli pod uwagę wiele innych aspektów, mających wpływ na pomiary. Nie będą one jednak dokładnie omawiane, ponieważ w znacznej mierze zależą od aspektów technicznych, które nie mają wpływu na koncepcję. Szczegółowy opis implementacji narzędzia *pathload* można znaleźć w [15].

Podobne podejście zostało wykorzystane w projekcie PITTSA [38], gdzie przepustowość sieci badano na potrzeby usług mobilnych. Rozwiązanie to jest godne

uwagi, ponieważ jego implementacji dokonano w języku C#, który jest również wykorzystywany przez autorów.

Zaletą opisanego w tym podrozdziale podejścia jest niewielka ilość danych, które są wysyłane w celu badania łącza – komunikaty są powiększane tylko o rozmiar równy wielkości znacznika czasowego, a informacja zwrotna jest wysyłana dopiero w momencie wystąpienia zatoru. Wadą tego podejścia jest brak informacji o zmieniającym się stanie łącza, jeśli użytkownik nie wysyła komunikatów – zareagowanie na zmiany możliwe jest dopiero w chwili ich ponownego wysyłania.

5.1.2 Pomiary wysycające łącze

Drugim podejściem do problemu badania łącza jest metoda oparta o jego wysycenie. Sposób dokonywanego pomiaru wykorzystuje TCP, ponieważ zakłada się, że protokół udostępnia komunikację niezawodną.

Dokonywanie pomiaru polega na odmierzeniu czasu, jaki zajmuje wysłanie pakietu o określonej wielkości. Podejście takie opiera się na założeniu, że podobna sytuacja będzie miała miejsce w przypadku, gdy nadawca dokona zbyt słabej agregacji wysyłanych komunikatów i całkowicie wykorzysta dostępne pasmo.

Zaletą takiego podejścia jest fakt, że symulowane jest takie obciążenie łącza, które wpływa na inne strumienie danych, które z niego korzystają, co dobrze odwzorowuje rzeczywistą sytuację. Przykładowo, w protokole TCP inne strumienie ulegną zmniejszeniu na skutek działania algorytmu AIMD, dostosowując się do aktualnego ruchu w sieci. Wadą jest duże wykorzystywanie łącza na przesyłanie danych, które nie niosą wartości informacyjnej.

5.2 Wymagania modułu wykrywania zatorów

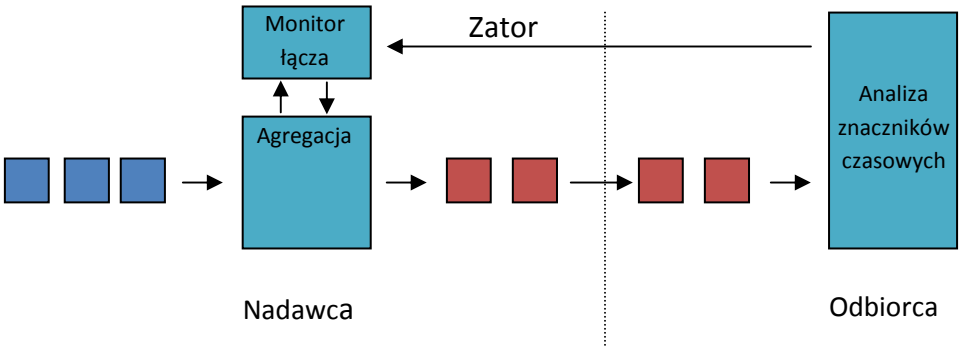
Przytoczone w poprzednim punkcie rozwiązania wydają się przydatne z punktu widzenia platformy. Udostępniają one prosty sposób pomiaru dostępnego pasma na poziomie aplikacyjnym oraz dają dość dobre rezultaty [14]. Co więcej, z punktu widzenia tworzonego systemu, istotne jest wykrycie zatoru, a wiedza o dokładnej przepustowości lub wielkości dostępnego pasma nie jest wymagana.

Należy jednocześnie pamiętać, że omówione rozwiązanie musi zostać zmodyfikowane w celu dostosowania do stawianych wymagań. Dostępna przepustowość łącza może się zmieniać w trakcie trwania sesji, zatem powinna być monitorowana na bieżąco. Z drugiej zaś strony, nie jest pożądana sytuacja, w której komunikaty są wysyłane jedynie w celu dokonania pomiaru. Widać więc, że pomiary powinny być dokonywane periodycznie, przy czym nie powinny one niepotrzebnie wykorzystywać pasma w nieuzasadniony sposób.

Sposób realizacji części systemu odpowiedzialnej za agregację pakietów oraz monitorowanie łącza w celu wykrycia zatorów, został omówiony w kolejnym punkcie.

5.3 Moduł wykrywania zatorów

Rysunek 50 przedstawia schematyczną budowę modułu odpowiedzialnego za wykrywanie zatorów: mechanizm pętli sprzężenia zwrotnego wykrywający zatory oraz dynamicznie modyfikujący stopień agregacji wysyłanych pakietów.



Rysunek 50. Moduł wykrywania zatorów.

Po stronie nadawcy dwa najważniejsze moduły to: moduł agregacji oraz monitor stanu łącza. Każdy pakiet wysyłany od nadawcy przechodzi przez agregator, którego zadanie polega na agregowaniu pakietów, w celu dostosowania ilości przesyłanych danych do przepustowości łącza pomiędzy nim a odbiorcą. Parametrem procesu agregacji jest stopień agregacji, za którego wyznaczenie odpowiedzialny jest monitor stanu łącza. Dodatkowo, do każdego pakietu opuszczającego nadawcę, dołączany jest znacznik czasowy wykorzystywany w procesie wykrywania zatorów. Odbiorca, po otrzymaniu kolejnych pakietów, dokonuje odpowiedniej analizy, a w sytuacji, kiedy dojdzie do wykrycia zatoru powiadamia o tym monitor łącza strony nadającej.

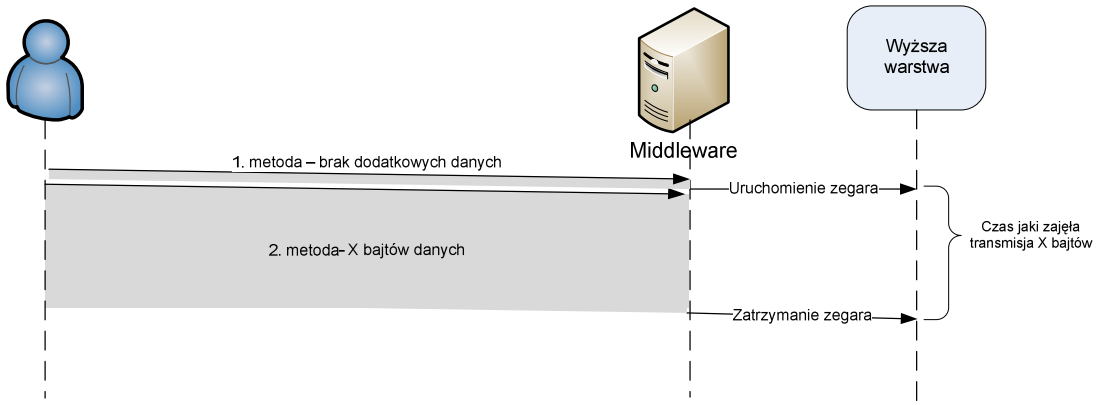
Następnie moduł monitorujący łącze odpowiednio modyfikuje aktualny stopień agregacji. W ramach tworzonej platformy stopień agregacji przyjmuje wartości z przedziału od 1 do 10, przy czym wartość 1 oznacza, że pakiety zostaną przesłane w niezmodyfikowanej postaci, natomiast 10 oznacza maksymalną agregację. Po wystąpieniu zatoru, wartość parametru agregacji zwiększana jest o 1 (o ile nie osiągnęła już maksymalnej wartości) – jest to jedyny moment, kiedy może ulec powiększeniu. Jeżeli aktualny stopień agregacji jest różny od 1 i po pewnym czasie nie doszło do zatoru, zostaje on zmniejszony o 1.

5.4 Badanie przepustowości łącza

Początkowa wartość parametru agregacji ustalana jest na podstawie badania wysycającego. Dokonywane jest ono raz dla każdego użytkownika podczas

pierwszego dołączenia do sesji. Dzięki takiemu podejściu, monitor łącza ma dobre oszacowanie początkowe, które może następnie być modyfikowane w trakcie wykrycia zatorów. Badanie wysycające jest procesem inwazyjnym i degraduje strumień danych, jednak jest dokonywane tylko raz, więc nie ma dużego wpływu na pracę użytkowników.

Większość środowisk middleware nie udostępnia mechanizmu bezpośredniego dostępu do strumienia danych, komunikując się z warstwami wyższymi z wykorzystaniem wywołania metod, jako abstrakcji przesyłania danych. Z tego powodu nie jest możliwe zmierzenie czasu, jaki upłynął od początku odbierania dużej wiadomości do jej końca, ponieważ cała operacja widziana jest jako atomowe wywołanie metody z przesłanymi argumentami. Rozwiązaniem w takiej sytuacji jest w pierwszej kolejności wywołanie metody, która nie powoduje transmisji dużej ilości danych, aby poinformować odbiorcę, że powinien uruchomić wewnętrzny zegar. Następnie wywoływana jest metoda, której argumentem jest duża tablica bajtów o zadanym rozmiarze, co powoduje zatrzymanie zegara. Na podstawie odczytu zegara i informacji o przesłanych danych w drugim wywołaniu, możliwe jest oszacowanie przepustowości łącza. Zasadę działanie przedstawia rysunek 51.

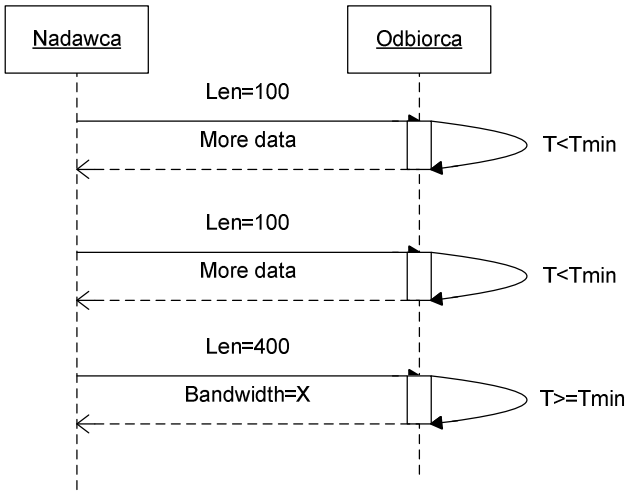


Rysunek 51. Szacowanie przepustowości łącza w oparciu o wywołanie dwóch metod.

Aby badanie było dokładne czas transmisji musi być odpowiednio długi – wtedy na wynik nie wpływa istotnie stan urządzeń sieciowych (np. wypełnienie buforów) albo specyficzne cechy protokołu ujawniające się na początku połączenia (np. nawiązywanie połączenie lub powolny start w TCP). Z tego punktu widzenia czas transmisji powinien być jak najdłuższy, jednak im dłuższy czas badania, tym dłużej łącze pozostaje zablokowane dla transmisji innych typów danych.

W celu rozwiązania tego problemu został wprowadzony mechanizm adaptacyjnego badania łącza, który rozpoczyna badanie przepustowości wykorzystując małą wielkość paczki danych. Jeśli czas tego badania był za mały, zwielokrotniana jest wielkość paczki, a następnie dane są transmitowane przez sieć i czas jest ponownie mierzony. Proces ten kończy się w momencie, kiedy czas pojedynczego badania

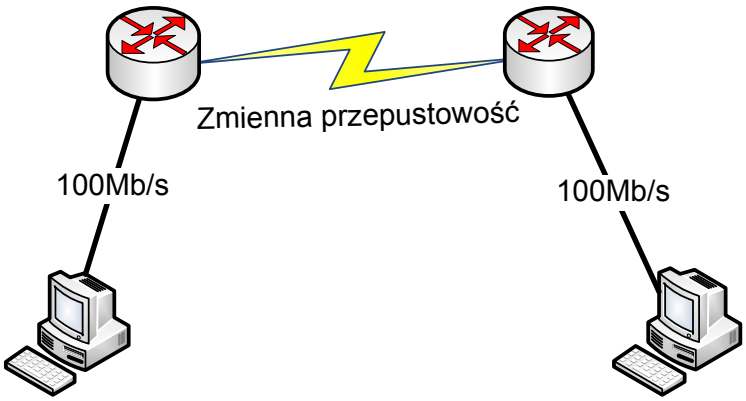
przekroczy pewną wartość (Rysunek 52). W wyniku pomiarów, których wyniki są przedstawione w następnym podrozdziale, zostało ustalone, że czas pomiaru powinien być nie mniejszy niż 0,4 s. Zwielokrotnianie rozmiaru paczki przy ponownej transmisji, prowadzi do wykładniczego wzrostu rozmiaru paczki, co dla mnożnika równego dwa, oznacza, że całkowity czas wszystkich badań będzie od dwóch do czterech razy większy, niż najkrótszy akceptowalny czas pojedynczego badania.



Rysunek 52. Zwiększanie rozmiaru wysyłanych danych, aż do uzyskania odpowiedniego czasu transmisji.

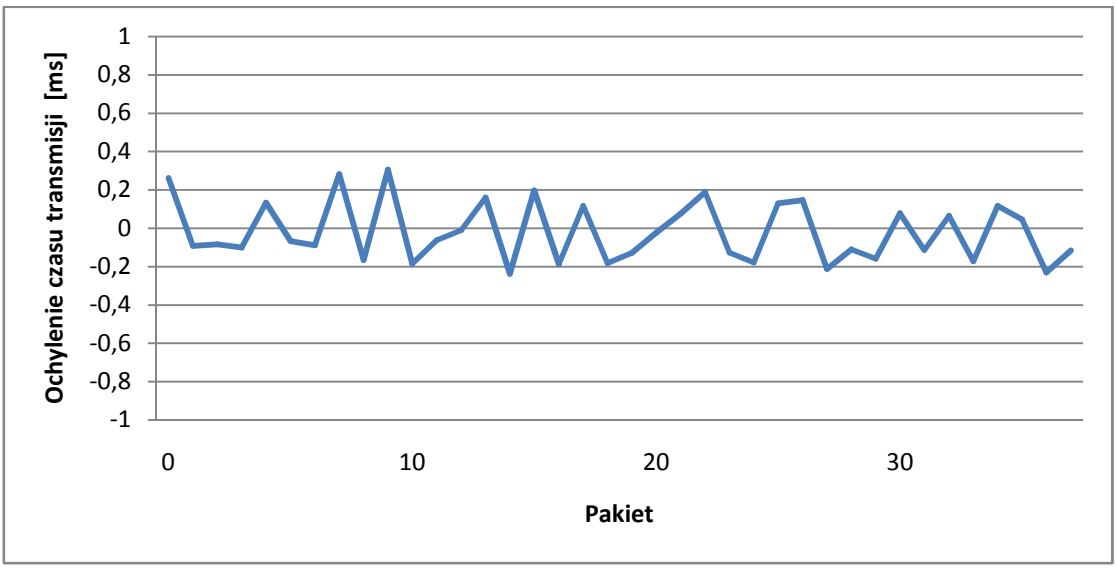
5.5 Testy i analiza wyników

Testy przedstawionych wyżej rozwiązań zostały przeprowadzone w sieci, której topologię przedstawia rysunek 53. Dzięki połączeniu dwóch routerów, możliwa była symulacja łącza o przepustowościach, które są wykorzystywane w środowiskach sieci WAN. Należy zwrócić jednak uwagę na fakt, że w testowej sieci nie występował żaden ruch generowany przez innych użytkowników – w przypadku wykorzystania w rzeczywistym środowisku może to wpłynąć na dokładność pomiarów.



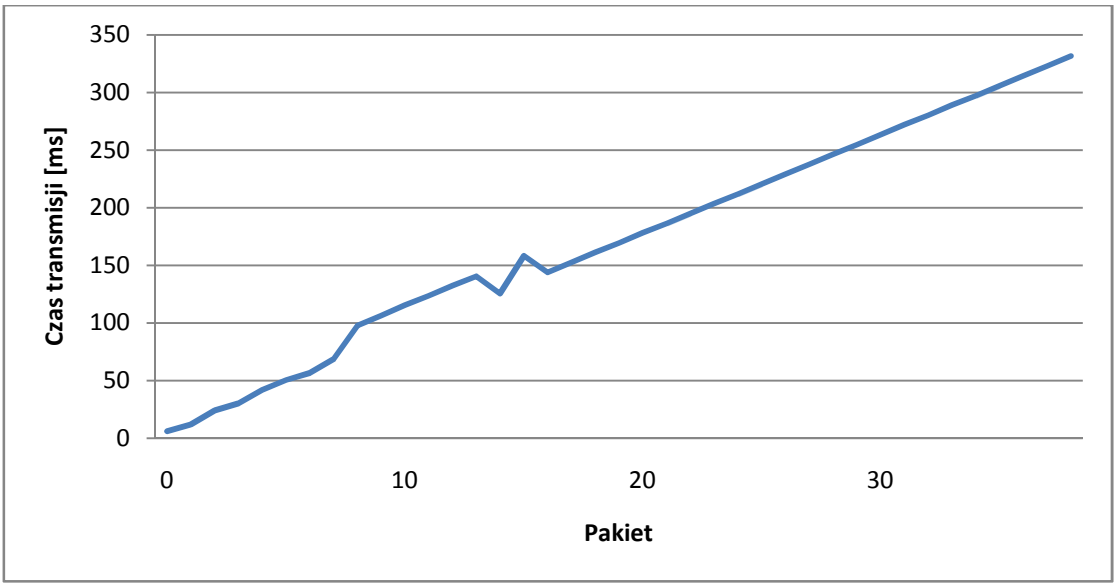
Rysunek 53. Sieć, na której dokonywano pomiarów.

Rysunek 54 przedstawia zależność opóźnia transmisji kolejnych pakietów względem średniej, w przypadku, kiedy wykorzystywane było 450 z 500 kb/s dostępnego łącza. Można zauważyć, że różnice czasu transmisji w kolejnych pakietach nie przekraczają 0,6 ms. – wartości między ok. -0,2 ms. a ok. -0,3 ms., co pozwala uważać, że pomiar oparty o opóźnienie transmisji będzie wystarczająco dokładny.



Rysunek 54. Odchylenie od średniej czasu transmisji pakietów przy wykorzystaniu 450 kb/s z 500 kb/s dostępnych.

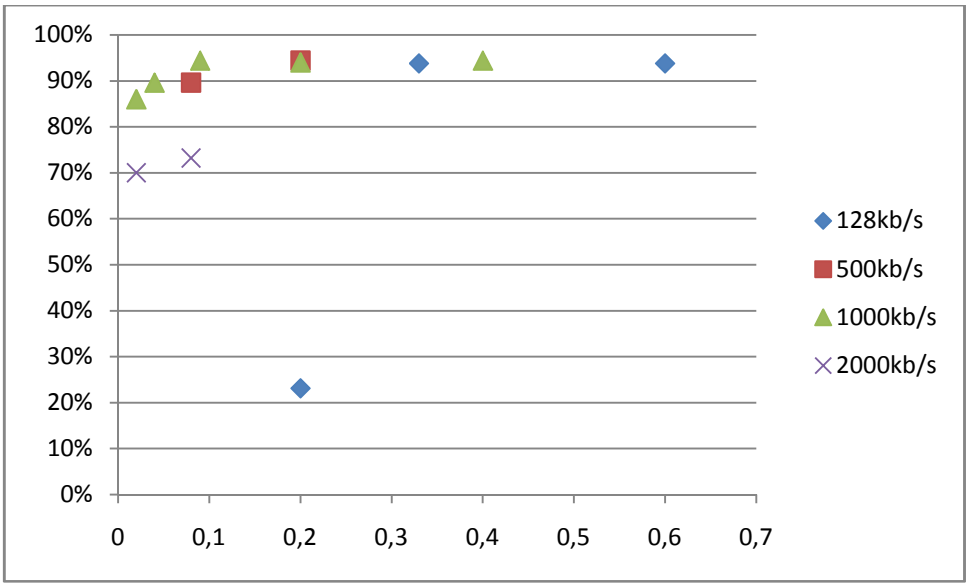
W przypadku, kiedy strumień wysyłanych danych jest większy od przepustowości łącza, wyraźnie widać rosnący czas transmisji (Rysunek 55). Wykrycie takiego trendu świadczy o powstaniu zatoru i skutkuje wysłaniem informacji do nadawcy, w celu zmniejszenia wielkości wysyłanego strumienia. To z kolei powoduje zlikwidowanie zatoru i dostarczanie wiadomości z mniejszym opóźnieniem.



Rysunek 55. Czas transmisji pakietów przy wykorzystaniu 550 kb/s z 500 kb/s dostępnych.

W przypadku wysycającego badania łącza, analiza w testowym środowisku miała na celu znalezienie takiej wielkości pakietu, którego przesłanie umożliwia dokładne zbadanie łącza. Dla pakietów zbyt małych czas transmisji był na tyle krótki, że nie pozwalał na dokładne szacowanie. Z kolei dla zbyt dużych pakietów, użytkownik niepotrzebnie musiałby czekać na ukończenie procesu badania łącza.

Rysunek 56 przedstawia zależność dokładności badania łącza w zależności od czasu, jaki zajmowała transmisja. Badania zostały przeprowadzone dla łączy o różnej przepustowości (od 128 kb/s do 2000 kb/s), aby dobrana wartość była odpowiednia dla użytkowników korzystających z łączy o różnych parametrach. Na podstawie analizy przedstawionego wykresu można zauważyć, że już dla pakietów, które są przesyłane dłużej niż 0.3 sek., dokładność jest większa od 90%. Ponieważ pomiar łącza wykonywany jest jednorazowo, czas pomiaru mniejszy od sekundy nie powinien być uciążliwy dla użytkownika.



Rysunek 56. Zależność wierności oszacowania przepustowości łącza [%] od czasu transmisji pakietu [sek.] dla różnych łącza.

6 Potencjalne obszary zastosowań

Celem niniejszego rozdziału jest przedstawienie możliwości wykorzystania opisywanej platformy w różnych zastosowaniach. Najpierw zostaną opisane aspekty, które wpływają na łatwość rozszerzania i dostosowywania do docelowego użycia. Następnie zostanie przedstawiony opis aplikacji umożliwiającej przeprowadzanie konsultacji telemedycznych, która wykorzystuje IGCP do realizacji zdalnych sesji. Na zakończenie pojawi się dokładniejszy opis kroków, które muszą zostać podjęte w celu rozszerzenia systemu o nową funkcjonalność.

6.1 Cechy wspierające rozszerzalność

Od początku tworzenia systemu jednym z wymagań była jego łatwa rozszerzalność i adaptowalność do różnych obszarów zastosowań. Mając to na uwadze zostały wprowadzone:

- Dwa typy wiadomości: lokalne i sieciowe – odseparowanie warstwy komunikacji pomiędzy modułami aplikacji od warstwy przesyłania przez sieć, daje możliwość niezależnych zmian w każdej z warstw, umożliwiając wykorzystanie platformy IGCP w dowolnych aplikacjach opartych o model wewnętrznej szyny komunikatów. Można wprowadzić zmianę kodowania danych przesyłanych przez sieć, która nie będzie zauważona przez pozostałe moduły.
- Rozszerzalny mechanizm zakładania blokad – za zarządzanie blokadami odpowiada *LockManager*, w którym mogą rejestrować się moduły, które chcą mieć wpływ na przebieg operacji przydzielania blokad. IGCP jest jednym z takich modułów, jednak łatwo wyobrazić sobie możliwość istnienia innych modułów o podobnej funkcjonalności. Wprowadzenie tych modułów jest niezależne od działania IGCP.
- Tematy wiadomości – wykorzystanie modelu używanego w systemach *publish-subscribe*, tj. rozgłaszania wiadomości, które są opisane przez temat, daje dodatkową możliwość wprowadzania nowych funkcji do aplikacji oraz w przejrzysty sposób oddziela nadawcę od odbiorcy – nadawca nie musi wiedzieć, kto odbiera, a kto nadaje wiadomości. Dodanie nowego tematu również nie wymaga zmian w IGCP.
- Niezależność serwera od semantyki przesyłanych wiadomości – wprowadzone zostało kodowanie z wykorzystaniem par klucz-wartość z bezwzględnymi wartościami poszczególnych właściwości, dzięki czemu możliwe jest przetwarzanie wiadomości bez potrzeby rozumienia ich znaczenia. Dodatkowo, możliwe jest wprowadzanie nowego rodzaju wiadomości lub

zmiana semantyki już istniejącej bez potrzeby zmian na serwerze – wystarczy, żeby klienci rozumieli wiadomości, które otrzymują.

- Możliwość rozszerzania serwera – zainstalowanie na serwerze modułów, które mogą rozróżniać różne rodzaje wiadomości, przyczynia się do zwiększenia efektywności komunikacji – możliwe jest np. agregowanie pakietów dotyczących punktów w linii, również na serwerze, przed wysłaniem do klienta. Należy jednak zaznaczyć, że jest to opcjonalne, a w przypadku braku takiego modułu wiadomości nadal będą przetwarzane w standardowy sposób.
- Wersjonowanie protokołu – możliwe jest wprowadzanie nowych wersji protokołu, które będą posiadały zmodyfikowaną funkcjonalność. Serwer, na podstawie wersji protokołu używanej przez klienta, może zachowywać się w różny sposób. W szczególnym przypadku może poinformować go o braku odpowiedniej wersji i odrzucić próbę połączenia.
- Możliwości korzystania z platformy w różnych środowiskach – wykorzystanie technologii ICE jako platformy umożliwia tworzenie klientów w oparciu o dowolny, wspierany język. Aktualnie są to: C++, Java, C#, Python, Ruby oraz PHP.

6.2 Możliwe zastosowania platformy

IGCP można wykorzystać m.in. w systemach opartych o koncepcje dzielonej tablicy, często wykorzystywanej w systemach telekonsultacyjnych. System taki mógłby znaleźć zastosowanie w medycynie. Wówczas w ramach sesji byłyby omawiane poszczególne badania, a obiektami sesji mogłyby być wykonane zdjęcia, dopiski lekarzy oraz wyniki pracy przyrządów pomiarowych. Zmiany wprowadzone na jednym ze stanowisk natychmiast rozsyłane byłyby do pozostałych lekarzy, co zapewniałoby szybką i sprawą możliwość komunikacji, nawet między odległymi ośrodkami.

Warto jednak zauważyć, że wykorzystanie IGCP nie ogranicza się do tego typu systemów. Można go użyć także w przypadku procesorów tekstu, gdzie wielu użytkowników edytuje jednocześnie ten sam dokument. Edytowany dokument może być przechowywany na serwerze jako stan sesji, gdzie użytkownicy przesyłają tylko swoje modyfikacje, które byłby natychmiast obserwowane przez pozostałych.

Innym zastosowaniem może być wspólna praca nad projektami typu CAD (*Computer-Aided Design*) - wielu projektantów pracuje wtedy nad wspólnym projektem. Na serwerze przechowywane są poszczególne elementy wchodzące w skład projektu, a dzięki mechanizmowi zakładania blokad na obiekty, tylko jeden z użytkowników może modyfikować dany element, przez co unika się niespójności w tworzonym modelu, nawet podczas równoległej pracy.

Można także wyobrazić sobie możliwość zastosowania platformy w systemach zdalnego sterowania urządzeniami. Przykładowo pracownicy, monitorujący obiekt lub uczestników imprezy masowej, mieliby możliwość zdalnego sterowania kamerami niezależnie od siebie. Platforma przechowywałaby informacje na temat aktualnego położenia kamery oraz zarządzałaby dostępem w taki sposób, aby najwyżej jedna osoba mogła w danym momencie kontrolować urządzenie.

6.3 Koncepcja systemu TeleDICOM

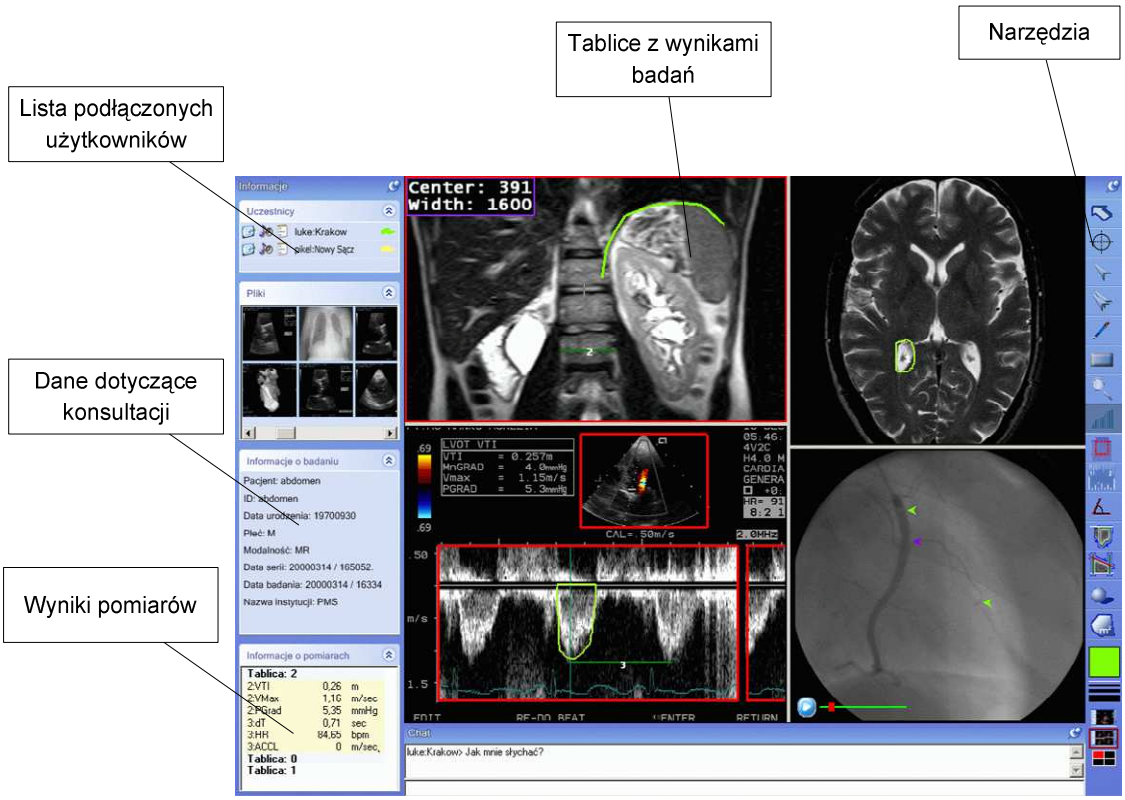
Platforma IGCP została wykorzystana w projekcie TeleDICOM [34]. TeleDICOM jest rozproszoną aplikacją umożliwiającą grupie konsultantów wspólną interaktywną pracę nad dokumentacją medyczną w postaci plików graficznych, przedstawiających wyniki badań pacjentów. Wszelkie akcje wykonane przez każdego z uczestników takiej sesji są natychmiast rozsyłane do pozostałych uczestników. Mają oni, między innymi, możliwość powiększania i animowania konsultowanych dokumentów, wyróżniania ich fragmentów, używania narzędzi pomiarowych oraz wskaźników. Podstawowymi dokumentami używanymi podczas konsultacji w aplikacji TeleDICOM są pliki w formacie DICOM, choć możliwe jest także dołączenie plików w formatach ogólnego przeznaczenia, jak np. JPEG, AVI lub GIF. TeleDICOM zapewnia ponadto uczestnikom konsultacji zintegrowaną komunikację głosową oraz możliwość wymiany komunikatów tekstowych. Środowisko to stanowi połączenie aplikacji wyświetlającej dokumenty medyczne ze środowiskiem pozwalającym na współdzielenie stanowiska pracy i systemu telekonferencyjnego.

Przed rozpoczęciem konsultacji przygotowywane są materiały, które będą dostępne dla jej uczestników. Następnym krokiem jest przyłączenie się uczestników do zaplanowanej sesji i przeprowadzenie konsultacji on-line.

Z punktu widzenia uczestnika aplikacja składa się z (Rysunek 57):

- tablic, na których przestawiane są omawiane badania,
- narzędzi medycznych, umożliwiających przeprowadzanie pomiarów na istniejących badaniach, jak np. tętna czy przepływu,
- narzędzi ułatwiających przeprowadzanie zdalnych konsultacji, jak np. interaktywny wskaźnik,
- części informacyjnej zawierającej m.in. dane pacjenta, szczegóły dotyczące badania oraz uczestników sesji.

Uczestnik ma możliwość zmiany wyświetlanych badań na poszczególnych tablicach, a modyfikacje te są odwzorowywane u wszystkich uczestników sesji. Także wyniki wykonywanych w trakcie sesji pomiarów są natychmiast przesyłane do pozostałych.



Rysunek 57. Elementy aplikacji widziane przez uczestnika konsultacji w aplikacji TeleDICOM.

Z punktu widzenia wewnętrznej architektury aplikacji klienta, można wyróżnić następujące elementy:

- szyna komunikacyjna, po której przesyłane są komunikaty pomiędzy modułami,
- shell – graficzny szkielet aplikacji, w którym zostały wyróżnione obszary na narzędzia, tablice oraz na część informacyjną,
- menedżer grafiki – udostępnia programistyczny interfejs dostępu do operacji wyświetlania obiektów z wykorzystaniem DirectX,
- zarządca tablic – odpowiedzialny za kontrolowanie wyświetlanych badań na tablicach,
- zarządca sesji – odpowiedzialny za poprawne zestawienie komunikacji interaktywnej z wykorzystaniem IGCP, pobranie odpowiednich plików z badaniami przez wszystkich uczestników oraz zestawienie kanału komunikacji głosowej,
- inne, które ze względu na budowę modułową oraz wykorzystanie szyny komunikacyjnej mogą być w łatwy sposób dodawane.

6.3.1 Mechanizm rozszerzeń

Procedura dodawania nowych funkcji będzie przebiegać we wszystkich przypadkach w bardzo podobny sposób. Poniżej zostały przedstawione kroki potrzebne do tego, aby rozszerzyć zakres możliwości aplikacji:

- Stworzenie nowego modułu programowego (ang. *assembly*) oraz implementacja interfejsu IPlugin, w postaci klasy, której zadaniem będzie inicjalizacja tworzonego modułu. Standardowym elementem procesu inicjalizacji jest dokonanie subskrybcji na tematy, którymi moduł jest zainteresowany,
- Przystosowanie do pracy sieciowej:
  - stworzenie modułu programowego adaptera sieciowego,
  - implementacja interfejsów serializatora (*ISerializator*) oraz agregatora (*IAggregator*),
  - implementacja interfejsu i rejestracja serializatora i deserializatora w momencie inicjalizacji modułu,
- Dodanie nowego modułu do listy aktywnych w pliku konfiguracyjnym aplikacji.

6.3.2 Synchronizacja widoków z wykorzystaniem IGCP

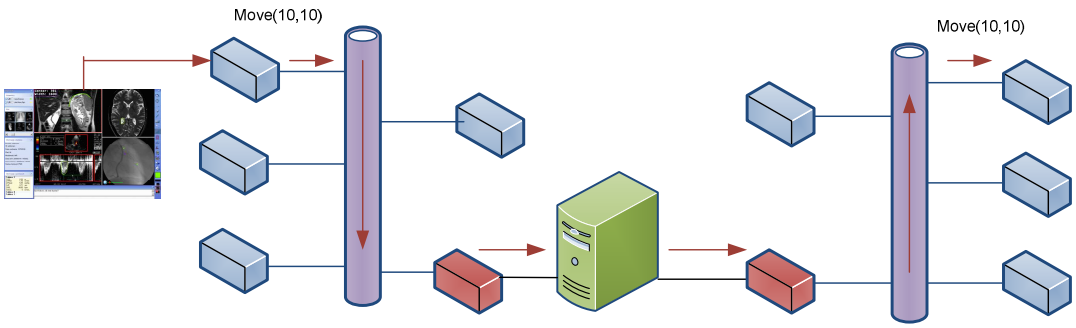
W celu lepszego zrozumienia działania aplikacji TeleDICOM w oparciu o tworzoną platformę, warto rozpatrzyć rolę jednego z modułów funkcjonalnych udostępnianych przez tę aplikację.

Zgodnie z tym, co zostało powiedziane, współpraca między lekarzami w czasie telekonsultacji polega na omawianiu badań lekarskich oraz dokonywaniu pomiarów za pomocą specjalistycznych narzędzi. Podstawową funkcją wymaganą w takim trybie pracy jest możliwość kontrolowania, który fragment badania jest aktualnie oglądany. Każdy uczestnik ma możliwość zmiany aktualnie oglądanej części w miarę potrzeby.

Udostępnienie omawianej funkcjonalności wymagało wprowadzenia dodatkowych obiektów do współdzielonego stanu oraz zdefiniowanie operacji, które na min operują. Widoczny fragment badania jest przechowywany w stanie sesji w postaci obiektu „widok”, którego atrybutami są współrzędne położenia w płaszczyźnie x-y-z. Zbiór możliwych operacji wynika bezpośrednio z atrybutów obiektu i należą do niego operacje przesunięcia oraz powiększenia i oddalenia.

Fragment aplikacji odpowiedzialny za dostarczenie omawianej funkcjonalności został wprowadzony w postaci nowego modułu. Dodatkowo zdefiniowane zostały implementacje serializatora oraz agregatora, dzięki czemu wiadomości dotyczące obiektu mogą być przesyłane przez sieć.

Rysunek 58 przestawia sekwencję przesyłanych wiadomości. Pierwsza jest generowana, gdy użytkownik dokona modyfikacji zmiany widoku przy pomocy graficznego interfejsu użytkownika. W takim przypadku kontroler (w sensie wzorca projektowego MVC), wchodzący w skład modułu, przechwytuje zdarzenie generowane przez interfejs i zamienia je na komunikat zmiany widoku. Przed rozesłaniem komunikatu, kontroler prosi serwis zarządzania jednodostępem o blokadę obiektu „widok". Jednym z modułów udzielających pozwolenia na przydzielenie blokady jest moduł sieciowy, który wyraża zgodę na blokadę, po uprzednim otrzymaniu zgody od serwera. Komunikat modyfikacji widoku jest odbierany przez moduł sieciowy oraz przez moduł zarządcy widoku, z którego pierwotnie pochodził. Moduł sieciowy, dzięki udostępnieniu serializatora, przesyła komunikat na serwer. Ten z kolei przekazuje go do wszystkich uczestników sesji. W ramach każdej z aplikacji klienckich, komunikat jest poddawany deserializacji oraz rozgłoszeniu docierając do lokalnego modułu zarządzania widokiem.



Rysunek 58. Moduł zarządzania widokiem. Wymiana wiadomości.

W momencie, gdy moduł zarządzający widokiem odbierze wiadomość modyfikującą, generuje on wiadomość skierowaną do menedżera grafiki, prosząc go o uchwyt do urządzenia wideo. Po otrzymaniu uchwytu, dokonuje odpowiednich zmian kamery odpowiadających modyfikacji widoku.

6.3.3 Dalsze rozszerzenia

Mechanizm definiowania obiektów w ramach stanu oraz definiowania nowych komunikatów daje możliwość dalszej rozbudowy aplikacji. Inne rozszerzenia aplikacji mogą polegać na udostępnieniu takich możliwości jak:

- rysowanie figur geometrycznych oraz ciągłej linii w celu oznaczania szczególnie interesujących fragmentów badania,
- wskazywanie wybranych obszarów za pomocą wskaźnika multimedialnego,
- korzystanie z wyspecjalizowanych metod pomiarowych, takich jak np. pomiar przepływu,
- wykonywanie skomplikowanych pomiarów z zakresu echokardiografii, np. PVLVT, RVWD, LVd, LVd2c, LVd4c

7 Weryfikacja działania platformy

W niniejszym rozdziale zostaną przedstawione testy, którym poddany został system w celu weryfikacji jego działania. System oceniany był pod kątem spełnienia:

- wymagań funkcjonalnych,
- wymagań niefunkcjonalnych.

Weryfikacja wymagań funkcjonalnych najczęściej jest możliwa, poprzez wykonanie określonych scenariuszy użycia i sprawdzenie poprawności wyników. W naszym przypadku testami funkcjonalnymi może być sprawdzenie, czy zachowana jest kolejność otrzymywania komunikatów albo czy odbierane dane są takie same, jak wysyłane. Tego rodzaju testy zostały określone na podstawie wymagań funkcjonalnych i zostały zaimplementowane jako testy jednostkowe, które po każdej zmianie w kodzie były uruchamiane – zapewniało to ciągły i powtarzalny proces testowania.

Dużo trudniejsze do określenia i weryfikacji są wymagania niefunkcjonalne, szczególnie jeśli dotyczą wydajności. Często stosowanym podejściem jest testowanie systemu pod kątem wymagań niefunkcjonalnych, jeśli przeszedł pomyślnie testy wymagań funkcjonalnych. W przypadku niespełnienia wymagań niefunkcjonalnych, konieczna jest taka modyfikacja, która skutkuje pozytywnym zakończeniem fazy testów.

Poniżej przedstawiony zostanie szereg testów, którym poddana została platforma IGCP. Są to m.in.:

- narzut wykorzystanych technologii komunikacyjnych,
- określenie optymalnego okresu opróżnienia bufora agregacji,
- zależność czasu transmisji od ilość użytkowników,
- zależność czasu transmisji od przepustowości łącza,
- przepustowość systemu,
- czas reakcji modułu kontroli przeciążeń,
- parametry przykładowego agregatora.

Testy były przeprowadzane na standardowym komputerze stacjonarnym wyposażonym w dwurdzeniowy procesor 2GHz oraz 2GB pamięci RAM.

7.1 Narzut wykorzystanych technologii komunikacyjnych

Wszystkie informacje przesyłane w ramach wykorzystywanej platformy poddawane są enkapsulacji w pakiety warstwy transportowej oraz sieciowej modelu OSI/ISO.

Narzuty w komunikacji związane są także ze sposobem serializacji danych, poprzez warstwę middleware (ICE) wykorzystywaną przez omawianą platformę.

Celem poniższego punktu jest oszacowanie narzutu związanego z procesem enkapsulacji dla danych wysyłanych podczas pracy z typową aplikacją zdalnej współpracy.

Autorzy pominęli dane związane z komunikacją na poziomie warstwy łącza danych, ponieważ w środowisku sieci heterogenicznej, w ramach której ma pracować platforma, nie można zakładać stosowania identycznej technologii w ramach wszystkich segmentów sieci.

Warstwa	Rozmiar nagłówka
IP	20 B
TCP	20 B
ICE	45 B
IGCP	19 B

Tabela 2. Narzut enkapsulacji dla pojedynczego pakietu.

Tabela 2 przedstawia narzuty związane z przesłaniem pojedynczego pakietu pochodzące od poszczególnych warstw oraz od samej platformy. Łączny rozmiar wszystkich nagłówków wynosi 104 bajtów.

W każdym przesyłanym pakiecie dane istotne z punktu widzenia aplikacji przesyłającej wiadomości kodowane są w postaci TLV. Dla każdej trójki TLV typ wiadomości przechowywany jest w zmiennej typu *byte*, podobnie jak długość, natomiast rozmiar pola *value* zależy od przesyłanych danych.

Do oszacowania narzutu związanego z enkapsulacją można wykorzystać pakiet opisujący fragment łamanej. W standardowej konfiguracji platformy IGCP (okres opróżnienia bufora ustawiony na 80 ms.) w jednym pakiecie przesyłana jest informacja średnio o czterech punktach. Każdy z nich opisuje lista dwóch rekordów TLV. Trójki opisujące punkt przenoszą informację o: współrzędnej x, współrzędnej y. Pozostałe informacje związane z kolorem, stylem czy też grubością linii są informacjami wspólnymi dla wszystkich punktów, przesyłanymi w wiadomości tworzącej linię, dlatego też nie są brane pod uwagę w przedstawionym oszacowaniu.

Obie współrzędne mogą być zakodowane za pomocą czterech bajtów. Wykorzystanie kodowania, w którym zbiór punktów zapisywany jest jako tablica kolejnych współrzędnych (na przemian x i y), umożliwia wykorzystanie jednego pola TLV do kodowania większej liczby punktów. W ten sposób 4 punkty mogą być zapisane jako

16 bajtów danych, 2 bajty do zapisania długości tablicy z danymi oraz 1 bajt opisujące typ, co daje w sumie 19 bajtów. Ostatecznie można stwierdzić, że dla przesyłanych pakietów narzut enkapsulacji wynosi około 570%. Tak duża wartość wynika z bardzo efektywnego sposobu kodowania przesyłanych danych, gdzie dane zajmują 15%, oraz z charakteru przesyłanych danych – interaktywna komunikacja charakteryzuje się przesyłaniem małych porcji danych z zachowaniem minimalnego opóźnienia. Ponadto narzut związany z nagłówkami wynika z rozszerzalności platformy – została zaprojektowana do pracy na różnych platformach (dzięki wykorzystaniu ICE) oraz z różnymi typami danych (dzięki wprowadzeniu nagłówka IGCP).

Warto również wspomnieć jak kosztowne (w sensie rozmiaru) jest wprowadzenie dodatkowych własności, opisujących cechy rysowanej linii np. grubości czy dodatkowej współrzędnej w przypadku rysowania w przestrzeni trójwymiarowej. Koszt wprowadzenia każdej właściwości to rozmiar typu opisującego tę własność powiększony o 3 bajty. Widać więc, że przyjęty sposób kodowania wiadomości, choć wiąże się z pewnym kosztem, to umożliwia rozszerzanie własności obiektów bez wprowadzania dużych narzutów.

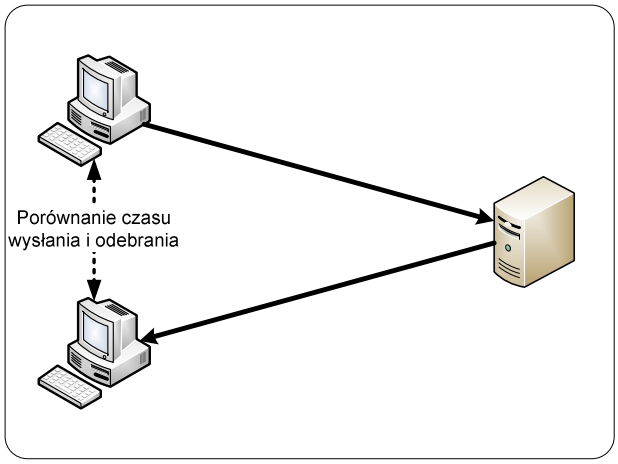
7.2 Optymalny okres opróżnienia bufora agregacji

Pierwszy test, który został przeprowadzony miał na celu określenie optymalnego czasu, po którym bufor agregacji powinien zostać opróżniony, a dane w nim zgromadzone zagregowane. Im większa byłaby ta wartość, tym więcej danych otrzymałby agregator w pojedynczym przebiegu, co umożliwiłoby uzyskanie lepszego stopnia kompresji. Przykładowo, kompresując linię, otrzymując dane po dwa punkty, bezstanowy agregator nie może w żaden sposób zmniejszyć strumienia danych. Jednak zwiększanie liczby punktów dostarczanych do agregatora, zwiększa szansę, że zostanie zmniejszony strumień danych, np. poprzez usunięcie punktów współliniowych.

Wadą zwiększania okresu opróżnienia bufora jest zmniejszenie interaktywności – im dłuższy czas przechowywania danych w buforze, tym później zostaną one wysłane. Dodatkowo zwiększenie ilości danych przekazywanych agregatorowi, powoduje większy nakład na obliczenia związane z kompresją, co dodatkowo zwiększa opóźnienie wysłania. Średnio opóźnienie wysłania wyniesie około połowy wartości okresu opróżnienia bufora

Testowanie odbywało się na jednym fizycznym komputerze, na którym uruchomiony był jeden proces serwera oraz dwa procesy klienta (Rysunek 59). Nadawca wysyłał określoną porcję danych do serwera, który przesyłał ją do drugiego klienta. Następnie porównywane były czasy wysłania i otrzymania komunikatu, na podstawie których określany był czas transmisji, czyli czas jaki upłynie od wysłania wiadomości do jej odebrania.

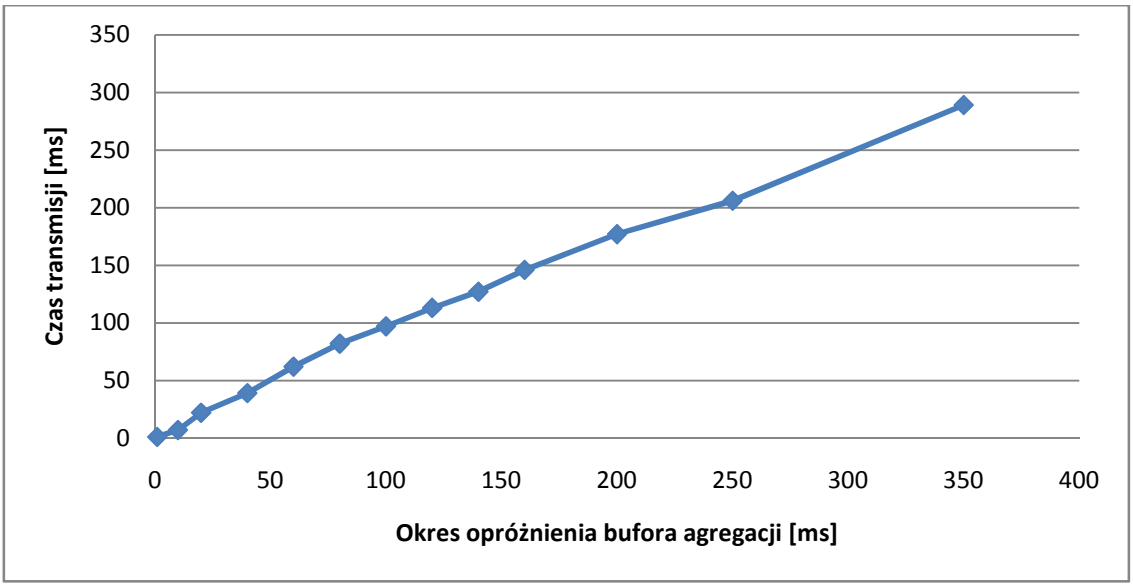
Uruchomienie wszystkich trzech procesów powodowało, że do mierzonego czasu nie były doliczane narzuty związane z komunikacją przez fizyczną sieć – liczone było tylko opóźnienie, powstałe w czasie przetwarzania wiadomości na kliencie i serwerze, oraz czas potrzebny na serializację i deserializację. Ponadto, nie było konieczności synchronizacji zegarów pomiędzy klientami w celu określenia czasu przesyłania, ponieważ oba procesy korzystały z tego samego. Aby zapewnić poprawność wyników w sytuacji chwilowego obciążenia maszyny testowej, każdy komunikat transmitowany był 100 razy, a wartość czasu transmisji uśredniana.



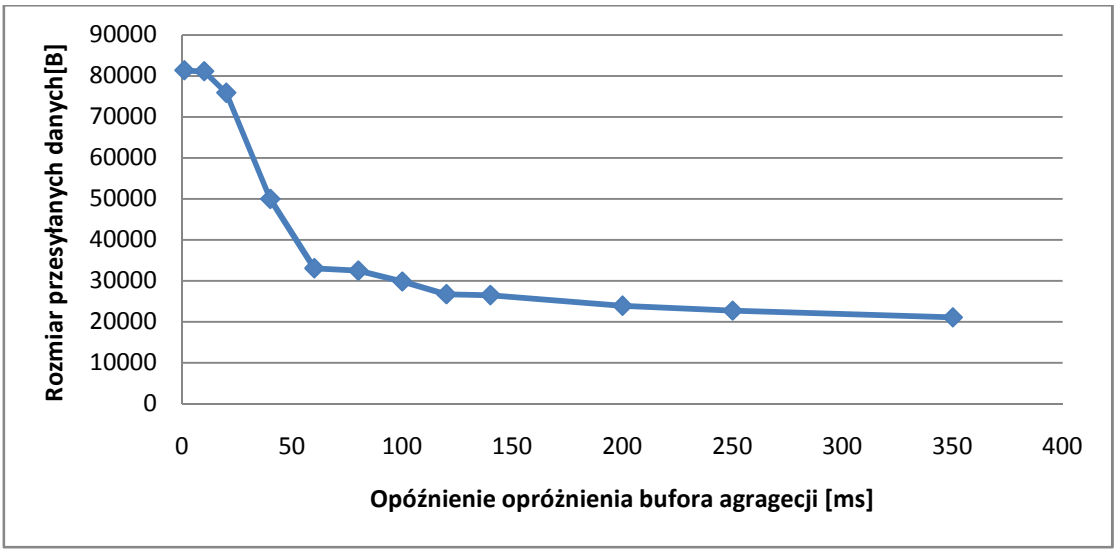
Rysunek 59. Środowisko testowe.

Drugim parametrem, określanym w czasie testów był rozmiar danych po kompresji. Zwiększanie okresu opróżniania bufora powodowało polepszenie agregacji kosztem czasu transmisji. Agregacja wykonywana była na przykładowej linii, która odpowiada wykorzystaniu platformy w typowym środowisku do pracy grupowej.

Poniżej zostały przedstawione wyniki testów: zależność czasu transmisji oraz rozmiaru przesyłanych danych od okresu opróżnienia bufora agregacji. Zgodnie z przypuszczeniami, zwiększanie wartości okresu opróżniania zwiększało czas transmisji, przy czym zależność ta była w przybliżeniu liniowa. W przypadku ilości danych po agregacji, zauważyć można wyraźny spadek w okolicach ok. 60 ms. oraz późniejszą stabilizację trendu. Oznacza to, że dla większych wartości opóźnienia, kompresja nie przynosiła rezultatów proporcjonalnych do rosnącego opóźnienia.



Rysunek 60. Zależność pomiędzy czasem transmisji a wartością okresu opróżnienia bufora agregacji.



Rysunek 61. Zależność pomiędzy ilością danych po kompresji a wartością opóźnienia opróżnienia bufora agregacji.

Na podstawie analizy tych danych, mając na uwadze zapewnienie interaktywnej i efektywnej komunikacji, wartość opóźnienia została ustalona na 80 ms. Zapewni to, wg autorów, skuteczną kompresję zachowując jednocześnie akceptowalny czas transmisji – dla testowych danych stopień kompresji wynosił ok. 2,6, a czas transmisji ok. 89 ms.

Warto zwrócić uwagę, że zgodnie ze specyfikacją IT-UT G.144 [12], czas transmisji w jedną stronę nie powinien przekraczać 150 ms., jeśli system ma zapewniać wysoką interaktywność. Wynika z tego, że dla ustalonej wartości opóźnienia opróżnienia bufora (89 ms.), możliwe jest wykorzystanie sieci wprowadzającej dodatkowe opóźnienie nie większe niż 61 ms.

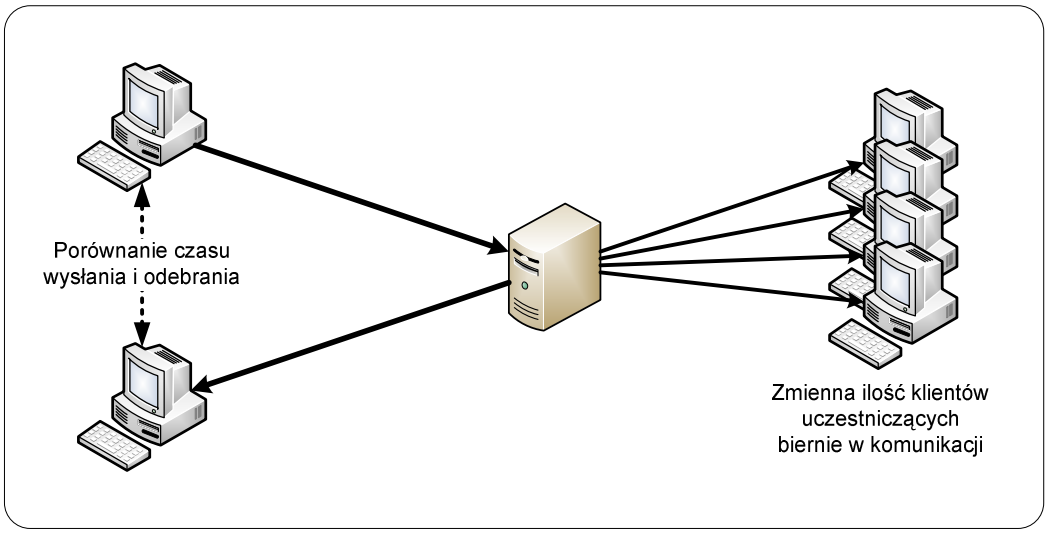
Mniej rygorystyczna norma wprowadzona przez G.114, zezwala na opóźnienie nie większe niż 400 ms., co umożliwia wykorzystanie IGCP w sieciach z opóźnieniem nawet 311 ms.

Ważną cechą platformy jest możliwość konfiguracji opóźnienie opróżnienia bufora. Dzięki temu, w szybkich sieciach lokalnych, gdzie agregacja prawdopodobnie nie będzie potrzebna, można zrezygnować z dodatkowego opóźnienia opróżnienia bufora agregacji, zwiększając w ten sposób interaktywność. Z drugiej strony, dla sieci bezprzewodowych o dużym opóźnieniu (RTT) i małej przepustowości, możliwe jest zwiększenie wartości opóźnienia opróżnienia bufora. Dodatkowy czas na agregacje może wtedy skutkować mniejszą ilością przesyłanych danych, co wpłynie na pozytywnie na komfort pracy, przy stosunkowo niedużym zwiększeniu opóźnienia przesyłanych danych.

7.3 Czas transmisji a liczba użytkowników

Drugi test miał na celu określenie zależności pomiędzy czasem transmisji komunikatów pomiędzy użytkownikami a liczbą dołączonych użytkowników. Idealną zależnością byłby stały czas transmisji niezależny od liczby połączeń, który umożliwiałby tworzenie dowolnie dużych sesji pomiędzy użytkownikami bez straty interaktywności.

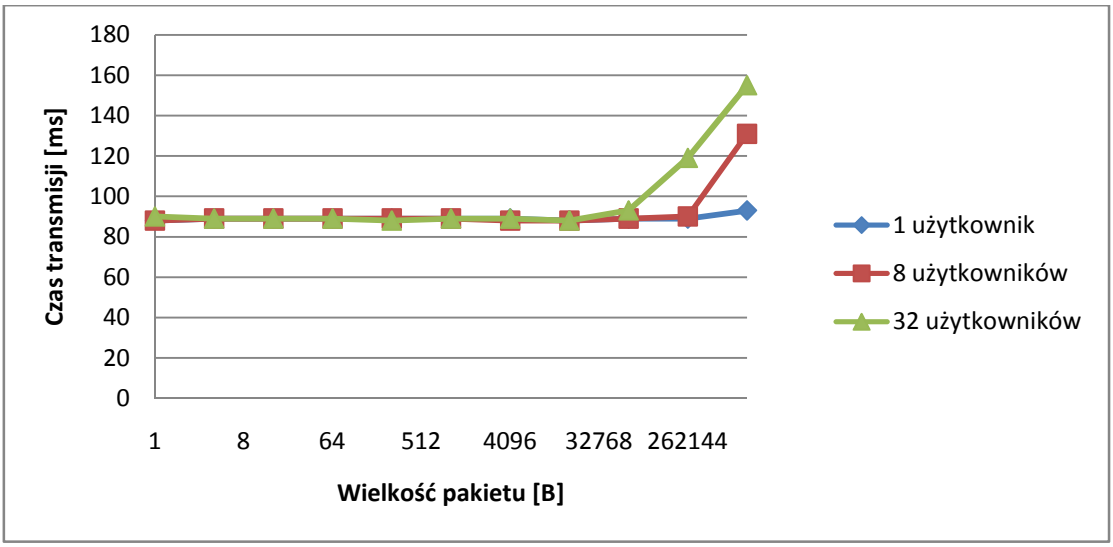
Rysunek 62 przedstawia środowisko testowe – podobnie jak w poprzednim teście proces serwera oraz klientów zostały uruchomione na jednym komputerze. Oprócz dwóch klientów, którzy porównywali czas transmisji wiadomości uruchamiana była zmienna liczna klientów, który tylko odbierali wiadomości. Ich rolą w tym teście było symulowanie zwiększonej ilości komunikacji, która miała miejsce pomiędzy klientami a serwerami. Ponieważ komunikacja pomiędzy procesami nie korzystała z fizycznej sieci, przedstawione poniżej wartości nie uwzględniają narzutów na taką komunikację. Dzięki temu wyniki testów nie są zaburzone przez warunki panujące w sieci, a których wartość może ulegać zmianie w zależności od środowiska wykorzystania.



Rysunek 62. Środowisko testowe ze zmienną liczbą biernie nasłuchujących klientów.

Wyniki testów (Rysunek 63) przedstawiają zależność czasu transmisji od wielkości komunikatu dla różnej liczby użytkowników. Wielkość komunikatu, przedstawiona w skali logarytmicznej, przyjmowała wartości od 1 B do 1 MB (2^{20} B). Do tych wartości nie są wliczane wielkości nagłówków wiadomości. Testowanie odbywało się dla liczby użytkowników równej 1, 8 oraz 32, a dla każdej wielkości komunikatu pomiar przeprowadzany było 100 razy, a otrzymane wartości uśredniane. Dzięki temu procedura testowa była odporna na zaburzenia wyników spowodowane chwilowym obciążeniem maszyny testowej.

Na podstawie analizy danych, można zauważyć, że dla komunikatów mniejszych niż ok. 2^{15} B (32 KB) zwiększona do 32 liczba użytkowników nie wpływała na czas transmisji. Warto zwrócić uwagę, że dane interaktywne, dla których czas transmisji jest kluczowy najczęściej są dużo mniejsze niż 32 KB (np. dane o pozycji kursora), a takie są dostarczane ze opóźnieniem niezależnym od liczny użytkowników. Dane, których rozmiar jest większy (np. tworzenie złożonych obiektów), często nie wymagają natychmiastowego dostarczenia i zwiększenie opóźnienia może pozostać niezauważone przez użytkownika. Ponadto grupy, które będą współpracowały, zazwyczaj nie będą przekraczały podanej liczby uczestników.

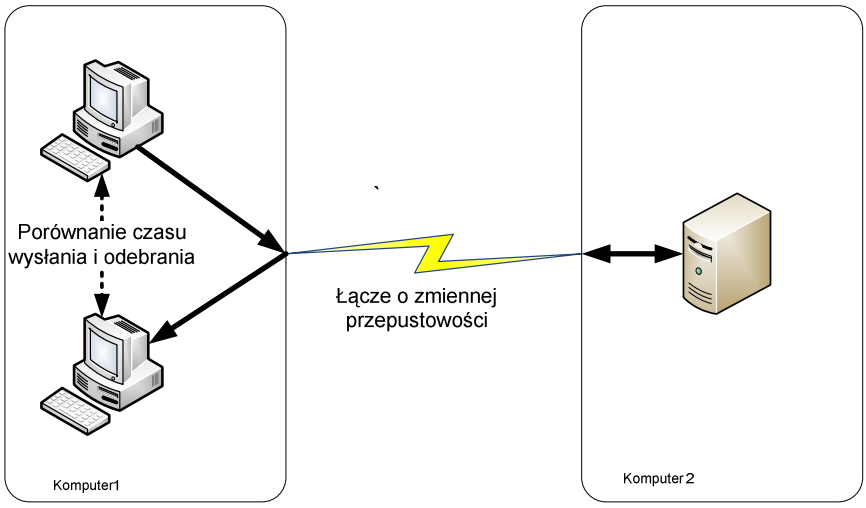


Rysunek 63. Zależność czasu transmisji od wielkości komunikatu dla różnej liczby użytkowników.

7.4 Czas transmisji a przepustowość łącza

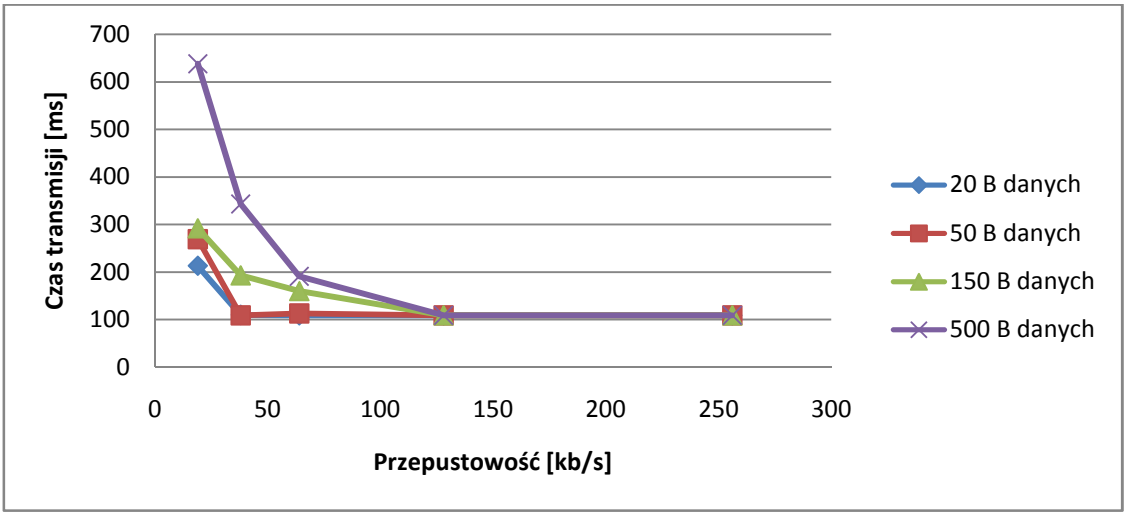
Zadaniem trzeciego testu było określenia zależności czasu transmisji różnej wielkości komunikatów od przepustowości łącza. Ponieważ czas transmisji zależy m.in. od ilości danych które należy przesłać, oczekuje się zmniejszania czasu transmisji wraz ze wzrostem przepustowości lub spadkiem ilości przesyłanych danych. Dodatkowe opóźnienie mogą być generowane w czasie przetwarzania komunikatów przez system.

Środowisko testowe (Rysunek 64) składało się z dwóch komputerów. Na jedynym z nich uruchomiony był serwer, a na drugim dwie instancje aplikacji klienckiej. Jedna z tych instancji wysyłała komunikaty, dla których po odebraniu był liczony czas transmisji. Komunikacja pomiędzy wykorzystanymi komputerami odbywała się z wykorzystaniem łącza o zmiennej przepustowości, dzięki czemu możliwa była symulacja pracy w różnych sieciach. Testowanie odbywało się na łączu o przepustowości od 15 kb/s do 250 kb/s z wykorzystaniem komunikatów, dla których pole danych miało rozmiar od 20 do 500 bajtów. Tak dobrana wielkość komunikatów odpowiada ilości danych, które najczęściej powinny być przesyłane są z zachowaniem interaktywności, zaś badana przepustowość jest mniejsza niż ta, której oczekuje się od docelowego użytkownika systemu. Dla potrzeb testów wyłączony został mechanizm agregacji.



Rysunek 64. Środowisko testowe z łączem o zmiennej przepustowości.

W wyniku przeprowadzonych testów można stwierdzić, że już dla przepustowości większej od 120 kb/s komunikaty z danymi o rozmiarze do 500 B przesyłane są w praktycznie takim samym czasie. Dla bardzo małych przepustowości (15 kb/s) czas transmisji 500 B jest istotnie większy niż 20 B, jednak nie wynika to ze sposobu implementacji platformy, ale z ograniczeń jakie nakłada warstwa sieciowa – wykorzystywane łącze było wysycane.



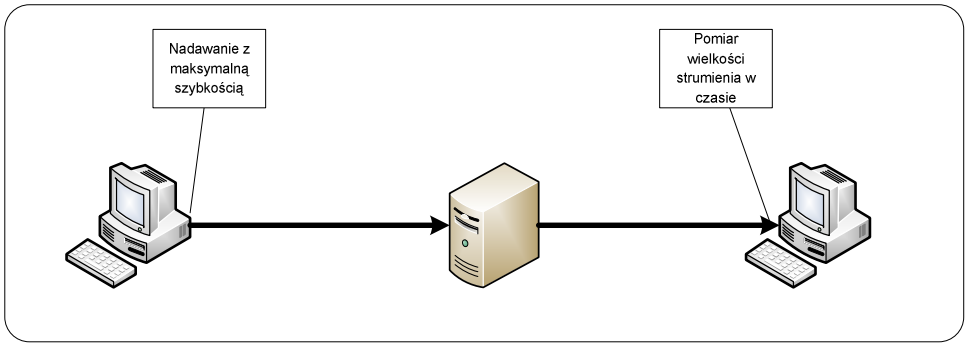
Rysunek 65. Zależność czasu transmisji komunikatów różnej wielkości od przepustowości łącza.

7.5 Przepustowość systemu

Celem tego testu był pomiar ilości danych, które w jednostce czasu mogą zostać przetworzone przez system – pomiar przepustowości systemu. Im większa byłaby ta wartość, tym większą ilość aktywnych użytkowników mógłby obsługiwać system. Dodatkowo, zbadanie pod kątem przepustowości elementów środowiska, w którym uruchomiana jest platforma, pozwala na wykrycie wąskich gardeł. Przykładowo

bezcelowe jest poprawianie parametrów sprzętowych serwera, jeśli wąskim gardłem jest sieć i przeciwnie – w przypadku, kiedy wąskim gardłem jest szybkość pracy serwera, lepsza infrastruktura sieciowa nie poprawi sytuacji.

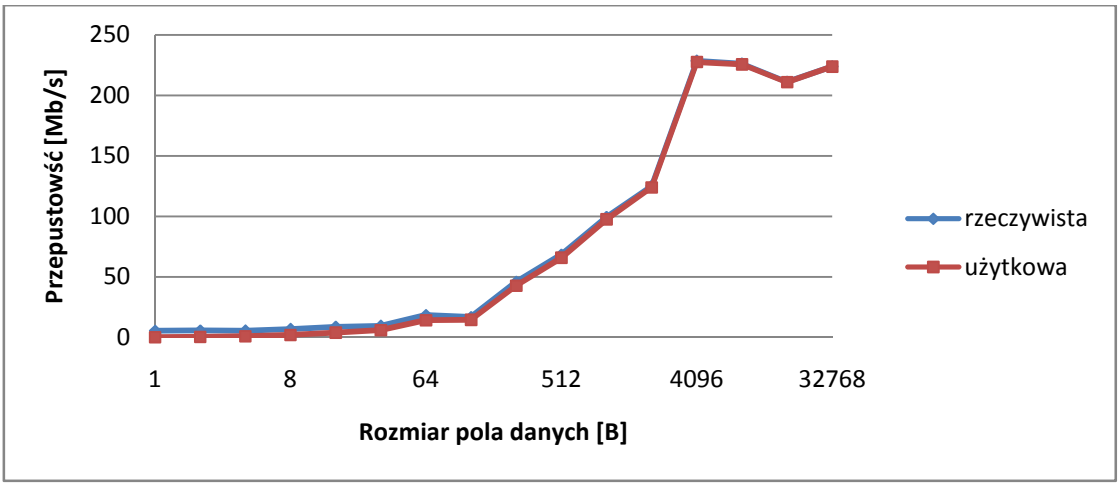
Środowisko testowe (Rysunek 67) składało się z dwóch aplikacji klienckich oraz serwera uruchomionych na jednym fizycznym komputerze. Jeden z klientów wysyłał dane do serwera z maksymalną szybkością. Dane te były następnie przekazywane do drugiego klienta, na którym odbywał się pomiar wielkości strumienia przychodzących danych. Na tej podstawie, dla różnych wielkości komunikatów, wyznaczane były dwa parametry systemu: przepustowość użytkowa oraz przepustowość rzeczywista. Przepustowość użytkowa wyznaczana była jako iloraz wielkości pola danych odebranych komunikatów do czasu. Przy obliczaniu przepustowości rzeczywistej ilość odebranych danych liczona była jako suma wielkości pola danych i nagłówków.



Rysunek 66. Środowisko testowe do pomiaru przepustowości systemu.

Rysunek 67 przedstawia wyniki pomiarów przepustowości rzeczywistej i użytkowej. Przepustowość rzeczywista dla każdej wielkości pola danych komunikatu była większa niż użytkowa, ponieważ ilość danych użyta przy jej obliczaniu była większa, a czas pomiaru taki sam. Warto również zauważyć, że wraz ze wzrostem rozmiaru komunikatu, różnica między przepustowości rzeczywistą a użytkową malała, co wynikało ze zmniejszającego się stosunku rozmiaru nagłówka do wielkości pola danych.

Istotnym spostrzeżeniem jest fakt, że wraz ze wzrostem rozmiaru komunikatu zwiększała się również przepustowość systemu. Wynikało to z tego, że nakład obliczeń potrzebnych do przetworzenia komunikatu, jest w dużej mierze niezależny od wielkości pola danych – serwer nie interpretuje pola danych i dlatego jego wielkość wpływa tylko na czas serializacji i zapotrzebowanie na pamięć na serwerze.



Rysunek 67. Przepustowość rzeczywista i użytkowa w zależności od rozmiaru pola danych w komunikacie.

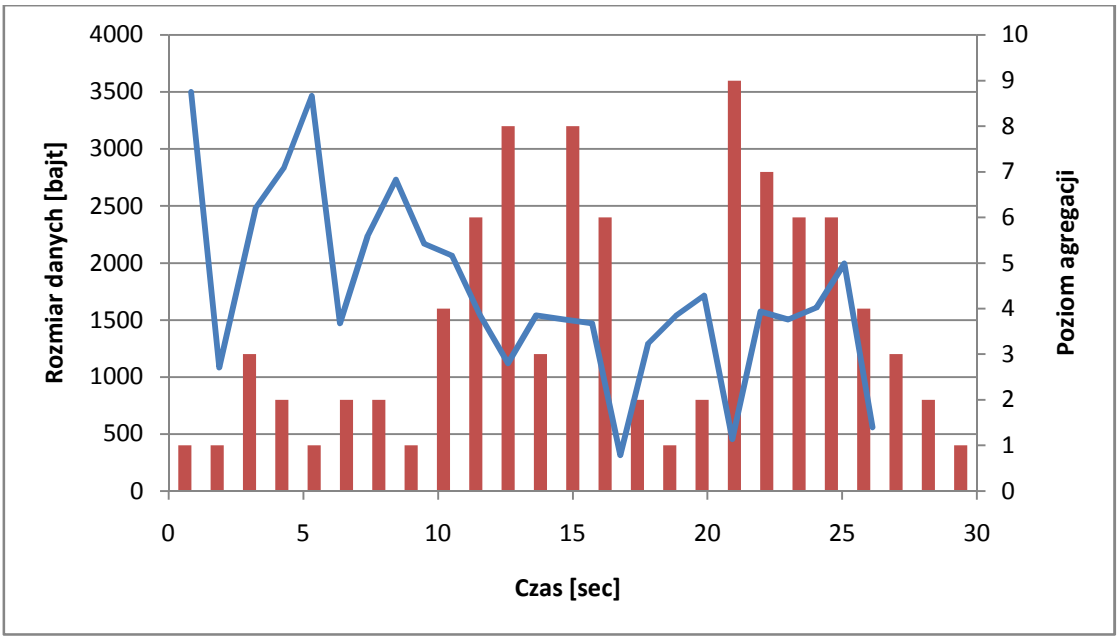
Maksymalna przepustowość systemu osiągana jest dla pakietów o rozmiarze powyżej 4 KB i wynosi ok. 220 Mb/s, z czego wynika, że w przypadku wykorzystania sieci o przepustowości 100 Mb/s dla wykorzystywanych danych testowych, wąskim gardłem nie byłby serwer. Dla danych o charakterze interaktywnym, gdzie wielkość komunikatu liczona jest w setkach bajtów, przepustowość systemu to ok. 20-50 Mb/s, co jest wartością większą niż powszechnie wykorzystywane łącza internetowe, z myślą o których projektowana była platforma IGCP.

7.6 Czas reakcji modułu kontroli przeciążeń

Duży wpływ na jakość pracy użytkowników ma sposób działania modułu kontroli przeciążeń. Moduł ten odpowiada za dynamiczne określanie aktualnej przepustowości pomiędzy stroną nadającą i odbierająca wiadomości. Informacje te wykorzystywane są przez moduł agregacji w celu odpowiedniego dopasowania ilości danych wysyłanych do odbiorcy.

Celem przeprowadzonego testu było oszacowanie czasu reakcji modułu kontroli przeciążenia w przypadku wystąpienia zatoru. Środowisko testowe składało się z jednego nadawcy oraz serwera. Użytkownik systemu będący nadawcą korzystał z łącza o przepustowości (48 kb/s), która nie była dostateczna, przez co w przypadku dużej aktywności użytkownika dochodziło do zatorów.

Test polegał na przesłaniu wcześniej zarejestrowanych danych odpowiadających sposobowi korzystania z systemu (rysowaniu łamanej) oraz śledzeniu ilości bajtów wysyłanych w jednostce czasu przez użytkownika oraz aktualnego poziomu agregacji wyznaczonego przez moduł kontroli przeciążeń. Rysunek 68 przedstawia zmiany stopnia agregacji oraz ilość wysyłanych danych w trakcie trwania testu. Oś odciętych przedstawia czas od rozpoczęcia komunikacji.



Rysunek 68. Zależność stopnia agregacji oraz ilość wysyłanych danych od czasu.

Widać, że w pierwszych 10 sekundach komunikacji przesłanych zostało więcej danych niż w późniejszym okresie. Poziom agregacji początkowo nie ulegał zmianie (1-10 sekunda), dopiero potem uległ zwiększeniu z chwilowymi spadkami wartości.

Na podstawie przedstawionych danych można wnioskować, iż czas reakcji modułu kontroli przeciążenia wynosi kilkanaście sekund, co w praktyce nie powinno powodować uciążliwości związanych z dużym opóźnieniem dostarczanych komunikatów.

Warto również wspomnieć, iż testy zostały przeprowadzone na stosunkowo wolnym łączu, odpowiadającym szybkością połączeniu opartemu na modemie analogowym, które w praktyce są coraz mniej spotykane, a mimo to przez cały czas trwania testu nie został osiągnięty maksymalny stopień agregacji.

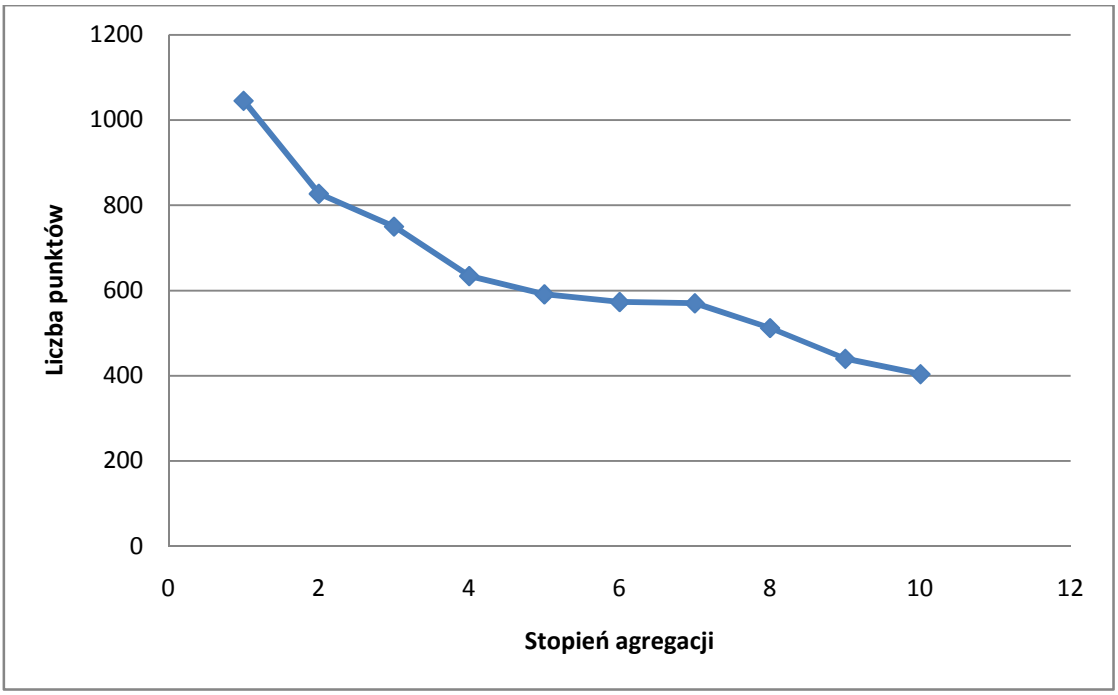
7.7 Parametry przykładowego agregatora

W poniższym punkcie przedstawione zostaną pomiary dokonane w trakcie testów przykładowego agregatora. Badany agregator został stworzony jako moduł odpowiedzialny za agregowanie linii rysowanych przez użytkowników systemu TeleDICOM.

Zgodnie z interfejsem omówionym w rozdziale dotyczącym implementacji agregator na wejściu otrzymuje tablicę złożoną w TLV kodujących kolejne punkty linii oraz stopień agregacji, określający jak bardzo mają zostać zagregowane przesyłane dane.

Należy podkreślić, iż testy zostały przeprowadzone w docelowym środowisku działania tzn. agregator nie dostał na wejściu wszystkich danych opisujących całą linię. Moduł agregatora był uruchamiany wielokrotnie, co odpowiadało sposobowi działania bufora agregacji odpowiedzialnego za sterowanie tym procesem.

Dane wykorzystane do przeprowadzenia testu opisywały krzywą łamaną złożoną z 1042 punktów. Rysunek 69 przedstawia wyniki przeprowadzonych pomiarów.



Rysunek 69. Zależność liczby punktów od stopnia agregacji dla przykładowej linii.

Z przedstawionych danych wynika, że w przypadku maksymalnego stopnia agregacji, strumień danych zmniejszany jest ok. 60%, a więc w przypadku linii użytkownicy dysponujący połączeniem o przepustowości o połowę mniejszym niż strumień generowanych danych są oglądać proces rysowania linii bez opóźnień.

Warto także zauważyć, że liczba punktów po agregacji maleje w sposób proporcjonalny do stopnia agregacji, a pomiędzy kolejnymi stopniami nie występują znaczne skoki w ilości otrzymywanych punktów (wyjątkiem są jedynie wartości 5, 6 i 7).

8 Podsumowanie

W niniejszej pracy została przedstawiona koncepcja stworzonej przez autorów platformy komunikacyjnej. Głównym celem rozwiązania miało być zapewnienie mechanizmów interaktywnej i efektywnej komunikacji pomiędzy współpracującymi użytkownikami pracującymi w środowisku heterogenicznym.

Rozdział 7 udowodnił praktyczną przydatność platformy. Dzięki możliwości łatwego rozszerzania, może ona zostać wykorzystana do budowy aplikacji służących wielu zastosowaniom tj. konsultacjom telemedycznym, równoczesnemu redagowaniu tekstu, wspólnemu projektowaniu modeli CAD itd.

Badania weryfikujące parametry platformy (rozdział 8) pokazują także, że zostały spełnione główne wymagania techniczne wynikające z heterogenicznego środowiska pracy oraz docelowego sposobu wykorzystania. Dzięki mechanizmom adaptacji istnieje możliwość wykorzystania platformy w sytuacji, gdy użytkownicy korzystają z łączy komunikacyjnych o różnej przepustowości, a mały narzut związany z przetwarzaniem strumienia danych umożliwia stworzenie iluzji rzeczywistej współpracy.

Z technicznego punktu widzenia, system został stworzony zgodnie z nowoczesnymi metodami projektowania aplikacji rozproszonych, zapewniając możliwie mały koszt rozwijania platformy o nowe funkcje oraz integracji z innymi systemami. Zastosowanie nowoczesnych technologii komunikacyjnych (w tym technologii ICE) sprawiło, iż system stał się rozwiązaniem otwartym, a poszczególne jego elementy mogą być w razie potrzeby implementowane z wykorzystaniem różnych platform sprzętowych i programowych.

O użyteczności platformy świadczy jej praktyczne zastosowanie w projekcie TeleDICOM. Współpracując z modułami zarządzania sesjami umożliwia przeprowadzanie zdalnych konsultacji pomiędzy lekarzami w różnych ośrodkach. Dzięki zastosowaniu mechanizmów umożliwiających stosunkowo łatwą integrację z istniejącymi elementami aplikacji, wykorzystanie IGCP nie wymagało zmian w jej architekturze. Z kolei odporność na chwilowe awarie sieci oraz mechanizmy adaptacji do przepustowości łącza, umożliwiają pracę w różnorodnych środowiskach, gdzie nie można gwarantować jakości połączenia.

Istnieje wiele możliwych rozszerzeń systemu, które mogą zostać wprowadzone w miarę jego dalszego rozwoju. Jedną z możliwości jest wprowadzenie mechanizmów węzłów pośredniczących (ang. *proxy*) w celu zwiększenia efektywności wykorzystania pasma, w przypadku, gdy duża grupa użytkowników połączona jest z centralnym serwerem za pomocą jednego łącza. Dodatkowo system można wyposażyć w moduł

persystencji stanu odpowiedzialny za jego przechowywanie w pamięci zewnętrznej. Przydatnym rozwinięciem mógłby być także mechanizm nagrywania trwającej sesji dający możliwość późniejszego odtworzenia w sposób przypominający oglądanie obrazu wideo.

9 Bibliografia

[1]. Ch. Bouras, A. Gkamas , “Internet Protocols for Synchronous Distance Learning”, 2000

[2]. A. Campailla, S.Chaki, E. Clarke, “Efficient Filtering in Publish-Subscribe Systems using Binary Decision Diagrams”, 2001

[3]. F. Cao, J. P. Singh, “Efficient Event Routing in Content-based Publish-Subscribe Service Networks”, 2004

[4]. R. Carter, M. Crovella, " Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks”, 1996

[5]. M. Eraslan, N. Georganas, J. Gallardo, D. Makrakis, “A Scalable Network Architecture for Distributed Virtual Environments with Dynamic QoS over IPv6”, 2003

[6]. W. Feng, F. Chang, W. Feng, J. Walpole, “Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server”, 2002

[7]. S. Ferretti , C. Palazzi, M. Roccetti, G. Pau, M. Gerla , “Buscar el Levante por el Poniente: In Search of Fairness Through Interactivity in Massively Multiplayer Online Games”

[8]. M. Fiedler, L. Isaksson, S. Chevul, J. Karlsson, P. Lindberg , “Measurements and Analysis of Application-Perceived Throughput via Mobile Links”, 2005

[9]. W. Geyer, J. Vogel, M. Mauve, “Efficient State Initialization Mechanisms for Shared Whiteboards and Distributed Applications”, 2000

[10]. ICE: http://www.zeroc.com/

[11]. IIOP-NET: http://iiop-net.sourceforge.net

[12]. ITU-T, G.114, Series G: One-way transmission time

[13]. ITU-T, T.120, Data protocols for multimedia conferencing

[14]. M. Jain, C. Dovrolis, “End-to-End Available Bandwidth: Measurement methodology, Dynamics, and Relation with TCP Throughput”, 2002

[15]. M. Jain, C. Dovrolis. “Pathload: A Measurement Tool for End-to-End Available Bandwidth”, 2002.

[16]. Z. Jerzak, C. Fetzer, “Handling Overload in Publish/Subscribe Systems”

[17]. JGroups Communication Toolkit, http://www.jgroups.org/javagroupsnew/docs/ overview.html

[18]. G. Jin, G. Yang, B. Crowley, D. Agarwal, “Network Characterization Service”, 2001

[19]. L. Lamport, “Time, clocks, and the ordering of events in a distributed system”, 1978

[20]. J. Liebeherr, M. Zarki, “Mastering networks. An Internet Lab Manual”, Chapter 10, 2003.

[21]. S. Lukosch, “Transparent Latecomer Support for Synchronous Groupware”, 2003

[22]. S. Lukosch, C. Unger , “Flexible Management of Shared Groupware Objects”, 2000

[23]. M. Mauve, V. Hilt, C. Kuhmünch, W. Effelsberg, “RTP/I – Towards A Common Application Level Protocol For Distributed Interactive Media”, 2001

[24]. Multimedia Lecture Board, http://www.informatik.uni-mannheim.de/pi4/projects/mlb/

[25]. R. E. Newman-Wolfe , M. L. Webb , M. Montes, “Implicit locking in the ensemble concurrent object-oriented graphics editor”, 1992

[26]. C. Palazzi, S. Ferretti, S. Cacciaguerra, M. Roccetti, “Interactivity-Loss Avoidance in Event Delivery Synchronization for Mirrored Game Architectures”

[27]. C. Palazzi, S. Ferretti, S. Cacciaguerra, M. Roccetti, “On Maintaining Interactivity in Event Delivery Synchronization for Mirrored Game Architectures”

[28]. C. Perkins, J. Crowcroft, ”Notes of the use of RTP for shared workspace applications”

[29]. Real VNC, http://www.realvnc.com

[30]. RFC 3550: H. Schulzrinne, S. Casner, R. Frederick , V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”

[31]. RFC 5135: D. Wing, T. Eckert, “IP Multicast Requirements for a Network Address Translator (NAT) and a Network Address Port Translator (NAPT)”

[32]. V. J. Ribeiro, M. Coates, R. H. Riedi, S. Sarvotham, R. G. Baraniuk, “Multifractal cross traffic estimation”, 2000

[33]. C. Sun, C. Ellis, “Operational transformation in real-time group editors: issues, algorithms and achievements”, 1998

[34]. TeleDICOM, http://www.teledicom.pl

[35]. P. Triantafillou, A. Economides, “Subscription Summarization: A New Paradigm for Efficient Publish/Subscribe Systems”, 2004

[36]. J. Vogel, M. Mauve, “Consistency control for distributed interactive media”, 2001

[37]. J. Vogel, M. Mauve, V. Hilt, W. Effelsberg, “Late join algorithms for distributed interactive application”, 2003

[38]. K. Wac, P. Arlos, M. Fiedler, “Accuracy Evaluation of Application-Level Performance Measurements”, 2007

[39]. XEP-0113: Simple Whiteboarding, www.xmpp.org/extensions/xep-0113.html

[40]. Q. Xia, “Leveraging single user application for multi-user collaboration”, 2006

[41]. X-window, http://www.x.org/wiki/

A. Interfejs komunikacyjny zdefiniowany w SLICE

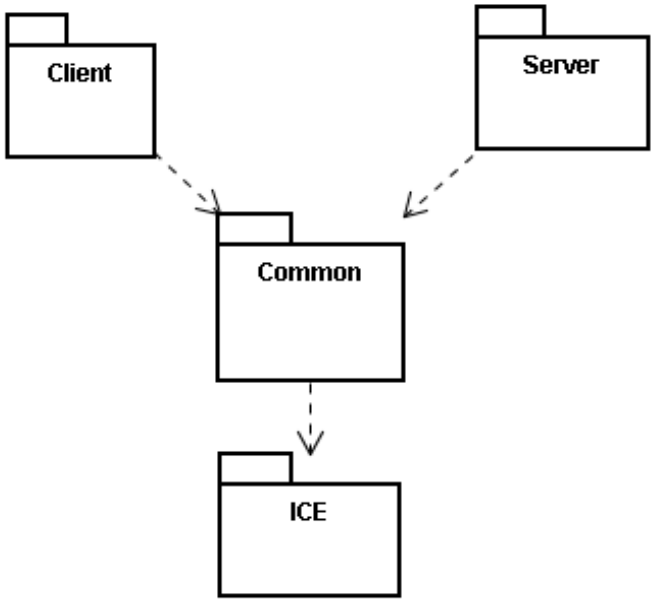
```
#include <Identity.ice>
#include <BuiltinSequences.ice>

module Net{
   module Common{
      module NetworkLayer{
         module Implementation{
            struct Tlv
            {
               byte type;
               Ice::ByteSeq value;
            };

            struct Eid
            {
               byte uid;
               short oid;
            };

            sequence<Tlv> TlvSeq;

            sequence<long> TimestampOpt;

            struct Message
            {
               byte uID;
               int sn;
               short messageType;
               int eid;
               TlvSeq tlvs;
               TimestampOpt timestamp;
            };

            sequence<Message> MessageSequence;

            exception RoomFullException {};
            exception AuthorizationException {};

            interface IIceSynchronousRoom
            {
```

```
      void JoinToRoom(string userName, out byte userId)
         throws RoomFullException,
               AuthorizationException;

      void RejoinToRoom(string userName, byte userId,
            int ssn)
         throws AuthorizationException;

      void AttachReceiver( byte userId,
            Ice::Identity ident)
         throws AuthorizationException;

      void RegisterSubject(string subject,
            bool isVolatile)
         throws AuthorizationException;

      short SubjectToMessageType(string subject)
         throws AuthorizationException;

      string MessageTypeToSubject(short messageType)
         throws AuthorizationException;

      void MarkStartSending(byte userId)
         throws AuthorizationException;

      int SendData(byte UserId, Ice::ByteSeq data)
         throws AuthorizationException;
   };

   exception NoValidCallbackException {};
   exception ObjectLockedException{};

   ["ami"]
   interface IIceAsynchronousRoom
   {
      MessageSequence AttachReceiver(byte userId,
               Ice::Identity ident)
         throws AuthorizationException;

      void SendMessages(MessageSequence messages)
         throws NoValidCallbackException,
               ObjectLockedException;

      void Ping();

      void SendCongestionNotification(byte userId,
         int packetSequenceNumber)
```

```
                throws AuthorizationException;
            };

            exception InvalidProtocolVersionException{};
            exception AuthenticationException{};

            interface IIceRoomManager
            {
               void Connect(
                  string roomGuid,
                  byte version,
                  bool secure,
                  bool compressed,
                  out IIceSynchronousRoom* synRoomRef,
                  out IIceAsynchronousRoom* asynRoomRef
               ) throws InvalidProtocolVersionException,
                        AuthorizationException,
                        AuthenticationException;
            };

            interface IIceSynchronousClient
            {
               void MarkStartSending();

               int SendData(Ice::ByteSeq data);
            };


            ["ami"]
            interface IIceAsynchronousClient
            {
               void SendMessages(MessageSequence messages);

               void SendCongestionNotification(
                  int packetSequenceNumber);

               void UserJoined(string userName, byte userUID);

               void UserDisjoined(byte userUID);

               void Ping();
            };
         };
      };
   };
};
```

B. Struktura projektu

Strukturę pakietów projektu platformy przedstawia rysunek 70.



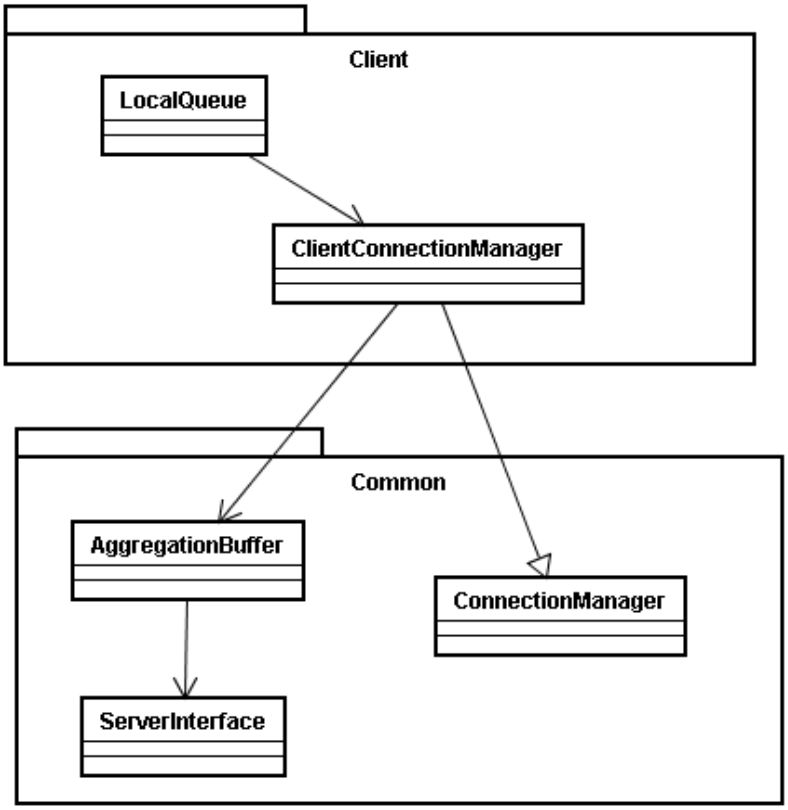
Rysunek 70. Struktura pakietów projektu platformy.

W skład projektu wchodzą 4 podstawowe pakiety:

- ICE – moduł middleware. Odpowiedzialny za udostępnianie mechanizmów komunikacji wymaganych do implementacji modelu koncepcyjnego. Umożliwia korzystanie z kanałów asynchronicznych i synchronicznych, przeprowadzanie serializację i deserializację danych.
- Common – pakiet zawierający wspólne elementy wykorzystywane po stronie klienta i serwera. W tym pakiecie znajdują się klasy bazowe (wsparcie wielokrotnego wykorzystania kodu) służące do implementacji takich mechanizmów jak: jednodostęp, serialiacja, agregacja, szeregowanie pakietów, badanie przepustowości itd.
- Client – główny pakiet części klienckiej. Odpowiada za implementację klienckiej części protokołu oraz integrację z aplikacją w ramach, której wykorzystywana jest platforma,
- Server – główny pakiet części serwerowej. Odpowiada za implementację serwerowej części protokołu komunikacyjnego oraz udostępnia funkcje administracyjne wykorzystywane w procesie zarządzania pokojami.

Implementacje klienta i serwera są od siebie niezależna, a jedynym elementem wspólnym jest definicja interfejsu komunikacyjnego znajdująca się w pakiecie *Common.* Oba pakiety tj.: *Server* i *Client* w równym stopniu wykorzystują klasy bazowe w celu przyspieszenia implementacji oraz zmniejszenia duplikacji kodu.

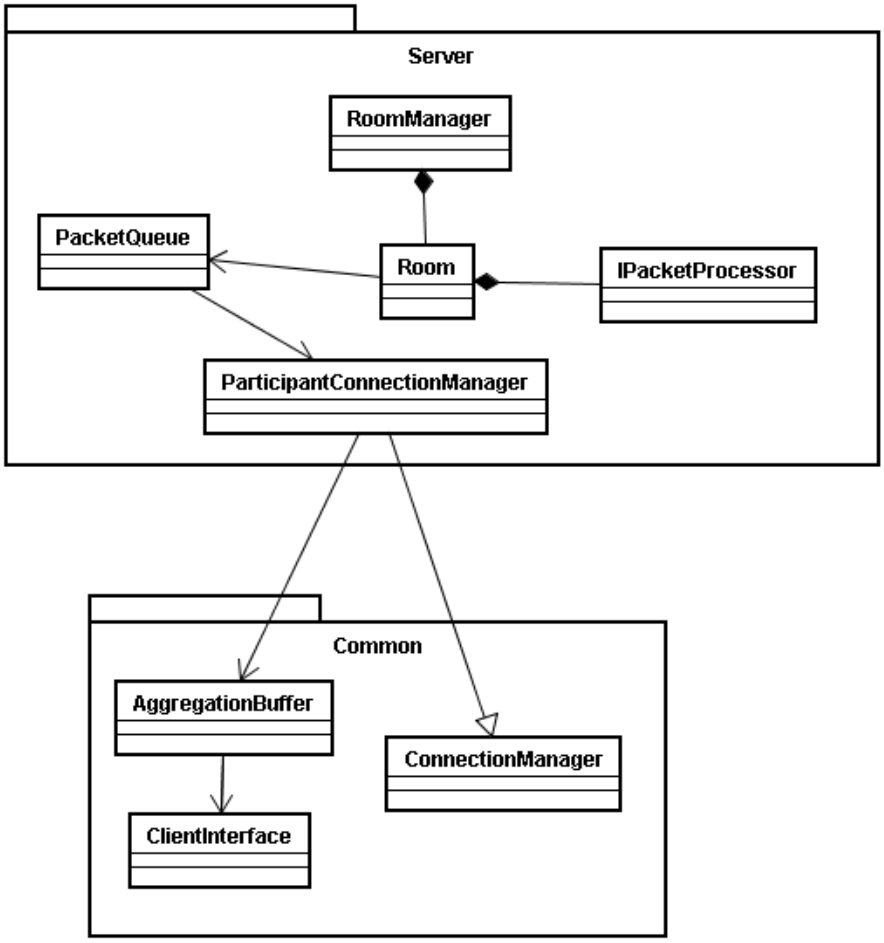
Implementacja części klienckiej jest znacznie prostsza od części serwerowej. Struktura pakietu jest przedstawiona na rysunku 71.



Rysunek 71. Struktura pakietu Client.

Podstawowymi elementami części klienckiej są: kolejka komunikatów czekająca na wysłanie oraz moduł odpowiadający za zarządzanie połączeniem z serwerem. Elementy platformy odpowiedzialne za agregacje pakietów są na tyle uniwersalne, że zostały umieszczone w pakiecie *Common.*

Część serwerowa ma bardziej skomplikowaną strukturę, co obrazuje rysunek 72. Podobnie ja w części klienckiej występują elementy odpowiedzialne za kolejkowanie pakietów oraz zarządzanie połączeniem. Ponadto ważnymi składnikami pakietu są klasy reprezentujące pokój oraz zarządcę pokoi. Na diagramie przedstawiony został także interfejs *IPacketProcessor* wykorzystywany do realizacji wejściowego potoku przetwarzania po stronie serwera.



Rysunek 72. Struktura pakietu Server.

W trakcie implementacji autorzy korzystali z następujących narzędzi:

- Microsoft Visual Studio 2008 – zintegrowane środowisko programistyczne,
- Microsoft Team Foundation Server 2008 – repozytorium kodu źródłowego,
- RhinoMocks – biblioteka wspierająca tworzenie testów jednostkowych,
- Ants Profiler – narzędzie profilujące wykorzystane do badania problemów z wydajnością.

Powyższy opis ma charakter bardzo ogólny. Przedstawiony został jedynie ogólny zarys podziału projektu ze względu na funkcje implementowane w każdym z pakietów. W celu zobrazowania rozmiaru projektu, warto przedstawić kilka liczb charakteryzujących rozmiar oraz stopień złożoności implementacji. Rozwiązanie składa się z 250 klas, 10 000 linii kodu źródłowego, 56 testów jednostkowych i integracyjnych, oraz 8 podprojektów. Warto zauważyć, że podstawowe mechanizmy komunikacyjne zrealizowane zostały przy użyciu technologii ICE, a podane liczby odnoszą się jedynie do implementacji logiki działania opisanej w niniejszej pracy.