Zostało zdefiniowanych 10 zadań - 2 aplikacyjne (An) i 9 infrastrukturalnych (Im). Należy zrealizować jedno wybrane zadanie aplikacyjne i jedno lub więcej zadanie infrastrukturalne; wymogiem jest także, by w wybranym zestawie znalazły się zadania (lub ich części) dotyczące (gRPC oraz (Ice lub Thrift)). Każde zadanie ma określoną maksymalną punktację stosownie do jego szacowanej trudności (nominalnie maksimum za całe laboratorium wynosi 20 pkt.). Realizacja tylko jednego zadania nie pozwoli na zaliczenie tematu. Przystępując do wyboru zadania i języka programowania należy zorientować się czy potrzebna w zadaniu funkcjonalność jest dostępna.

Zadanie A1 - "Inteligentny" dom
Aplikacja ma pozwalać na zdalne zarządzanie urządzeniami tzw. inteligentnego domu, którego wyposażeniem są różne urządzenia, np. czujniki czadu czy zdalnie sterowane lodówki, piece, kamery monitoringu z opcją PTZ, bulbulatory, itp. Każde z urządzeń może występować w kilku nieznacznie się różniących odmianach, a każda z nich w pewnej (niewielkiej) liczbie instancji. Dom ten nie oferuje obecnie możliwości budowania złożonych układów, pozwala użytkownikom jedynie na zdalne sterowanie pojedynczymi urządzeniami oraz odczytywanie ich stanu.

Dodatkowe informacje i wymagania:

- Każde z urządzeń inteligentnego domu jest reprezentowane przez obiekt/usługę strony serwerowej. Sposób jego integracji i komunikacji z rzeczywistym, sterowanym urządzeniem nie jest przedmiotem zainteresowania projektu. Urządzenia mogą działać na wielu instancjach serwerów (tj. w wielu procesach) (demonstracja: na co najmniej dwóch).
- Projektując interfejs IDL urządzeń należy używać także typów bardziej złożonych niż string czy int/long (tj. struktury, sekwencje itp.). Trzeba pamiętać o deklaracji i zgłaszaniu wyjątków lub błędów tam, gdzie to może mieć zastosowanie.
- Wystarczająca jest obsługa dwóch-trzech typów urządzeń, jeden-dwa z nich mają mieć dwa-trzy podtypy.
- Należy odwzorować podane wymagania do cech wybranej technologii w taki sposób, by jak najlepiej wykorzystać oferowane przez nią możliwości budowy takiej aplikacji i by osiągnąć jak najbardziej eleganckie rozwiązanie (gdyby żądanej funkcjonalności nie dało się wprost osiągnąć). Decyzje projektowe trzeba umieć uzasadnić.
- Zestaw urządzeń może być niezmienny w czasie życia serwera (tj. dodanie nowego urządzenia może wymagać modyfikacji kodu serwera i restartu procesu). Aplikacja kliencka może być świadoma obsługiwanych typów urządzeń w czasie kompilacji.
- Początkowy stan instancji obsługiwanego urządzenia może być zawarty w kodzie źródłowym strony serwerowej lub pliku konfiguracyjnym.
- Aplikacja kliencka ma pozwalać zademonstrować sterowanie wszystkimi urządzeniami bez konieczności restartu w celu przełączenia na inne urządzenie.
- Serwer może zapewnić funkcjonalność wylistowania nazw (identyfikatorów) aktualnie dostępnych instancji urządzeń.
- Dla chętnych: wielowątkowość strony serwerowej.
- Dla chętnych: wzbogacenie systemu o reverse-proxy (gRPC).

Technologia middleware: dowolna. Realizując komunikację w ICE należy zaimplementować poszczególne urządzenia inteligentnego domu jako osobne obiekty middleware, do których dostęp jest możliwy po podaniu jego identyfikatora Identity („Joe"). Realizując komunikację w Thrift lub gRPC należy dążyć do minimalizacji liczby instancji eksponowanych usług (ale bez ekstremizmu - lodówka i bulbulator nie mogą być opisane wspólnym interfejsem!).
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 12

Zadanie A2 - Subskrypcja na zdarzenia
Wynikiem prac ma być aplikacja klient-serwer w technologii gRPC. Klient powinien móc dokonywać subskrypcji na pewnego rodzaju zdarzenia. To, o czym mają one informować, jest w gestii Wykonawcy, np. o nadchodzącym wydarzeniu, którym jesteśmy zainteresowani ze względu na miejsce, czas, tematykę itp, o osiągnięciu określonych w żądaniu warunków pogodowych w danym miejscu, itp. Oczywiste wydaje się, że subskrypcja musi precyzyjnie określać zainteresowanie użytkownika (np. nie chcemy się subskrybować na informację o wszelkich biegach maratońskich w najbliższy weekend na całym świecie).

Dodatkowe informacje i wymagania:

- Na pojedyncze zdarzenie może się zasubskrybować wielu odbiorców naraz.
- Może istnieć wiele niezależnych subskrypcji (tj. np. na wiele różnych instancji spotkań).

- Projektując protokół komunikacji pomiędzy stronami należy odpowiednio wykorzystać mechanizm strumieniowania (*stream*) - niedopuszczalny jest *polling*.
- Wiadomości mogą nadchodzić z różnymi odstępami czasowymi (w rzeczywistości nawet bardzo długimi), jednak na potrzeby demonstracji rozwiązania należy przyjąć interwał rzędu pojedynczych sekund.
- W definicji wiadomości przesyłanych do klienta należy wykorzystać pola liczbowe, *enum*, *string*, *message* - wraz z co najmniej jednym modyfikatorem *repeated*. Etap subskrypcji powinien w jakiś sposób precyzować, które powiadomienia danej usługi (spośród wszystkich) są dla odbiorcy interesujące (np. obejmować wskazanie miasta, którego warunki pogodowe nas interesują) i dany odbiorca powinien otrzymywać wyłącznie interesujące go powiadomienia.
- Dobrze widziana będzie możliwość odsubskrybowania się z notyfikacji o wcześniej zasubskrybowanych zdarzeniach bez konieczności odłączania się klienta (ale odłączenie się musi pociągać za sobą zakończenie subskrypcji).
- Dla uproszczenia realizacji zadania można (nie trzeba) pominąć funkcjonalność samego tworzenia instancji wydarzeń lub miejsc, których dotyczy subskrypcja i notyfikacja - może to być zawarte w pliku konfiguracyjnym, a nawet kodzie źródłowym strony serwerowej. Treść wysyłanych zdarzeń może być wynikiem działania bardzo prostego generatora.
- W realizacji należy zadbać o odporność komunikacji na błędy sieciowe (które można symulować czasowym gwałtownym wyłączeniem klienta lub serwera lub włączeniem zapory sieciowej). Ustanie przerwy w łączności sieciowej musi pozwolić na ponowne ustanowienie komunikacji bez konieczności restartu procesów. Wiadomości przeznaczone do dostarczenia dla odbiorcy powinny być buforowane przez serwer do czasu ponownego ustanowienia łączności. Rozwiązanie musi być także "NAT-friendly" (tj. uwzględniać rozważane na laboratorium sytuacje związane z translacją adresów, w tym podtrzymywaniem aktywności w kanale komunikacyjnym) - co należy umieć udowodnić np. analizując komunikację sieciową.

Technologia middleware: gRPC
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 12

Zadanie I1 - Wywołanie dynamiczne
Celem zadania jest demonstracja działania wywołania dynamicznego po stronie klienta middleware. Wywołanie dynamiczne to takie, w którym nie jest wymagana znajomość interfejsu zdalnego obiektu lub usługi w czasie kompilacji, lecz jedynie w czasie wykonania (w zadaniu: klient ma nie mieć dołączonych żadnych klas/bibliotek stub będących wynikiem kompilacji IDL). Wywołania mają być zrealizowane dla kilku (co najmniej trzech) różnych operacji/procedur używających przynajmniej w jednym przypadku nietrywialnych struktur danych (np. listy (sekwencji) struktur) i sposobu komunikacji (gRPC: wywołanie strumieniowe). Nie trzeba tworzyć żadnego formatu opisującego żądania użytkownika ani parsera jego żądań - wystarczy zawrzeć to wywołanie "na sztywno" w kodzie źródłowym, co najwyżej z konsoli parametryzując szczegóły danych. Jako bazę można wykorzystać projekt z zajęć. Trzeba przemyśleć i umieć przedyskutować przydatność takiego podejścia w budowie aplikacji rozproszonych
ICE: Dynamic Invocation https://doc.zeroc.com/ice/3.7/client-server-features/dynamic-ice/dynamic-invocation-and-dispatch. Odnośnie ograniczeń warto spojrzeć tu: https://doc.zeroc.com/ice/3.7/client-server-features/dynamic-ice/streaming-interfaces
gRPC: Dopuszczalne (rekomendowane?) jest użycie usługi refleksji. Równorzędnymi funkcjonalnie (w stosunku do stworzonego klienta) narzędziami są grpcurl oraz Postman - należy umieć zademonstrować ich działanie w czasie oddawania zadania.

Technologia middleware: Ice albo gRPC
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 8

Zadanie I2 - Efektywne zarządzanie serwantami
Celem zadania jest demonstracja (na bardzo prostym przykładzie) mechanizmu zarządzania serwantami technologii Ice. Zadanie powinno mieć postać bardzo prostej aplikacji klient-serwer, w której strona serwerowa obsługuje wiele obiektów Ice. Obiekty middleware występujące w aplikacji są dwojakiego typu: część powinna być zrealizowana przy pomocy dedykowanego dla każdego z nich serwanta, druga część ma korzystać ze współdzielonego dla nich wszystkich serwanta. Zarządzanie serwantami ma być efektywne, tj. dla dedykowanych serwantów, taki serwant jest instancjonowany dopiero w momencie pierwszego zapotrzebowania na niego i może pozostać w pamięci serwera do końca działania procesu lub po pewnym czasie być poddany ewikcji.
Interfejs IDL obiektu może być superprosty, choć musi implikować konkretny sposób realizacji serwanta (co należy umieć uzasadnić).
Aplikacja kliencka powinna jedynie umożliwić zademonstrowanie funkcjonalności serwera. Logi na konsoli po stronie serwera muszą pozwolić się zorientować, na którym obiekcie i na którym serwancie zostało wywołane żądanie i kiedy nastąpiło instancjonowanie serwanta.
W zadaniu trzeba korzystać bezpośrednio z mechanizmów zarządzania serwantami oferowanego przez technologię, a nie własnych, zbliżonych mechanizmów (w szczególności dla dedykowanych serwantów należy wykorzystywać istniejącą tablicę ASM). Każdy obiekt middleware musi być „osiągany" przez klienta przez podanie jego identyfikatora (Identity) - najlepiej jako parametr podawany w czasie działania aplikacji.
Należy zwrócić uwagę na działanie metod checkedCast oraz uncheckedCast – serwant musi być tworzony dopiero po wywołaniu operacji z biznesowego interfejsu obiektu (tj. IDL), na rzecz którego działa.

Rozszerzeniem funkcjonalności systemu (+2 pkt) może być specjalizowany ewiktor usuwający z pamięci RAM najmniej potrzebne (np. najdawniej używane) serwanty (wraz z zachowaniem ich stanu, np. do pliku) w razie przekroczenia zdefiniowanej, maksymalnej liczby serwantów i przywracający zapisany stan serwanta w razie ponownego zapotrzebowania na niego.

Technologia middleware: Ice
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 8

Zadanie I3 - System *middleware* o dużej skali
Celem zadania jest przygotowanie infrastruktury middleware pozwalającej na na obsługę bardzo dużego ruchu (o skali przewyższającej możliwości jednej maszyny/procesu). W tym celu należy zadbać co najmniej o odpowiednie, przeźroczyste dla klientów routowanie i równoważenie ruchu wśród dostępnych w danym czasie usług, wielowątkowość przetwarzania, dla chętnych - mechanizmy *backpressure*. Należy przemyśleć i zrealizować różne strategie równoważenia obciążenia.
Środowisko demonstracyjne może oczywiście działać na pojedynczym komputerze, ale liczba instancji serwerów powinna być spora (co najmniej pięć), eksponowane usługi powinny być opisane przynajmniej dwoma interfejsami IDL, a liczba ich instancji nie musi być ekstremalnie duża). Kod aplikacji klienckiej powinien pozwolić sprawnie przetestować działanie systemu.

Demonstracja zadania: Omówienie interfejsów, omówienie infrastruktury, demonstracja działania systemu i przedstawienie wniosków (np. jak są obsługiwane wywołania strumieniowe w gRPC)?

gRPC: reverse-proxy (np. nginx).
Ice: IceGrid (https://doc.zeroc.com/ice/3.7/ice-services/icegrid)

Technologia middleware: Ice albo gRPC
Języki programowania: dwa różne
Maksymalna punktacja: 8

Zadanie I4- gRPC-Web
Celem zadania jest demonstracja aplikacji klient-serwer zrealizowanej w technologii gRPC, gdzie aplikacja kliencka działa w środowisku przeglądarki WWW wykorzystując gRPC-Web. Ważnym elementem zadania jest dokonanie oceny aplikacji pod kątem wydajności i ograniczeń komunikacji. Wynikiem prac powinien też być krótki raport podsumowujący najciekawsze aspekty realizacji.
Interfejs udostępnianej usługi musi używać nietrywialne typy danych i oferować co najmniej jedno wywołanie strumieniowe.
Demonstrowane zadanie nie może być wierną kopią rozwiązań znalezionych w Internecie lub w dokumentacji technologii.
Koniecznym elementem systemu jest reverse-proxy (envoy (sprawdzony) lub np. caddy (niesprawdzony)). Najprostszym sposobem jego uruchomienia jest środowisko Docker.

Technologia middleware: gRPC
Języki programowania: usługa (serwer) może być zaimplementowany w dowolnym języku programowania
Maksymalna punktacja: 9

Zadanie I5 - Bezpieczeństwo komunikacji *middleware*
Prezentowane w czasie laboratorium projekty intencjonalnie pomijały aspekty bezpieczeństwa, które są przecież bardzo ważne w produkcyjnym systemie, zwłaszcza działającym w sieci publicznej. Celem zadania jest implementacja kryptograficznych zabezpieczeń (TLS) w każdej z trzech technologii prezentowanych na zajęciach – bazą mogą być projekty prezentowane na zajęciach.

W czasie prezentacji trzeba umieć wykazać, że prowadzona komunikacja jest faktycznie zabezpieczona (analiza ruchu sieciowego, certyfikaty itp.).

Technologia middleware: Ice i Thrift i gRPC
Języki programowania: w przynajmniej jednej technologii middleware dwa różne
Maksymalna punktacja: 8

Zadanie I6 - Porównanie omówionych technologii middleware i usług wykorzystujących API REST oraz usług GraphQL
Celem zadania jest porównanie sposobu komunikacji stosowanego w omawianych technologiach middleware oraz usługach wykorzystujących wzorzec REST i usługach korzystających z GraphQL. Należy wziąć pod uwagę a) dostępne wzorce komunikacji (np. komunikacja strumieniowa w gRPC czy połączenia dwukierunkowe w ICE i ich realizowalność w jakiś sposób w każdym z rozwiązań), b) efektywność komunikacji pod względem ilości przesyłanych danych liczoną na poziomie L7, tj. ilość danych przesyłanych przez TCP/UDP, c) czas zdalnego wywołania, d) inne czynniki stanowiące o zaletach poszczególnych rozwiązań w stosunku do innych.

Eksperymenty muszą być prowadzone w porównywalnych warunkach (rozmieszczenie klienta i serwera, podobny interfejs i zbiór danych), ew. trzeba uwzględnić występujące różnice (np. serwer działający lokalnie i w Internecie, kryptograficzne zabezpieczenia komunikacji i jego brak).

Badania mogą pomijać pewne wymienione aspekty pod warunkiem dokładniejszej analizy innych (dopuszczana jest własna inwencja).

Wynikiem prac powinien być zwarty i treściwy raport zawierający m.in. warunki eksperymentu, konkretne osiągnięte wyniki liczbowe i konkluzje. W czasie demonstracji zadania należy umieć przedyskutować najważniejsze tezy opracowania oraz wykonać podstawowe testy.

Technologia middleware: (Ice lub Thrift lub gRPC) oraz (REST i GraphQL)
Języki programowania: wystarczy jeden
Maksymalna punktacja: 10

Zadanie I7 - IceRPC
IceRPC (https://zeroc.com/icerpc) jest nowym, jeszcze niedokończonym projektem o interesującej funkcjonalności. Ciekawa jest możliwość użycia różnych języków IDL (slice i protobuf) oraz wykorzystanie protokołu transportowego QUIC. Celem zadania jest zbudowanie prostej aplikacji klient-serwer, przeanalizowanie sposobu prowadzenia komunikacji oraz przedstawienie wniosków z ewaluacji tej technologii middleware. Pożądane jest porównanie wydajności komunikacyjnej ze zbliżonymi funkcjonalnie aplikacjami Ice oraz gRPC.

Uwaga: technologia jest obecnie obsługiwana w mocno ograniczonej liczbie systemów i języków: https://docs.icerpc.dev/getting-started/supported-platforms/icerpc-csharp-0_4

Technologia middleware: iceRPC
Maksymalna punktacja: 10

Zadanie I8 - Własny pomysł studenta
Celem zadania jest zbadanie i przedstawienie wybranych cech jednej ub większej liczby technologii middleware. Pomysł musi być przedstawiony Prowadzącemu (luke@agh.edu.pl) przed zamieszczeniem zadania zaakceptowany przez niego.

Technologia middleware: do ustalenia
Demonstracja zadania: do ustalenia
Maksymalna punktacja: 10

Uwagi wspólne:

- Interfejsy IDL powinny być proste, ale zaprojektowane w sposób dojrzały (odpowiednie typy proste, właściwe wykorzystanie typów złożonych), w zadaniach aplikacyjnych dodatkowo uwzględniając możliwość wystąpienia różnego rodzaju błędów. Tam gdzie to możliwe i uzasadnione należy wykorzystać dziedziczenie interfejsów IDL.
- Działanie aplikacji może (ale nie musi) być demonstrowane na jednej maszynie.
- Kod źródłowy zadania powinien być demonstrowany w IDE a dodatkowe elementy (np. raporty z testów) przy pomocy oprogramowania pozwalającego na wygodne i szybkie zapoznanie się z nimi.
- Aktywność poszczególnych elementów aplikacji należy odpowiednio logować (wystarczy na konsolę) by móc sprawnie ocenić poprawność jej działania. Demonstracja może (na życzenie odbierającego) obejmować także analizę komunikacji sieciowej.
- Aplikacja kliencka powinna mieć postać tekstową (z wyjątkiem zadania I4) i może być minimalistyczna, lecz musi pozwalać na przetestowanie funkcjonalności aplikacji szybko i na różny sposób (musi więc być przynajmniej w części interaktywna).
- Pliki generowane (stub, skeleton, itp.) powinny się znajdować w osobnym katalogu niż kod źródłowy klienta i serwera. Pliki stanowiące wynik kompilacji (.class, .o itp) powinny być w osobnych katalogach niż pliki źródłowe.

Sposób oceniania:

Wykonanie tylko jednego z zadań **nie pozwoli** na uzyskanie zaliczenia zadania.

Sposób wykonania zadania będzie miał zasadniczy wpływ na ocenę. W szczególności:

- niestarannie przygotowany interfejs IDL: -2 pkt.
- niestarannie napisany kod (m.in. zła obsługa wyjątków, błędy działania w czasie demonstracji): -3 pkt.
- brak aplikacji w więcej niż jednym języku programowania (gdy wymagany): -3 pkt.
- brak wymaganej funkcjonalności lub realizacja funkcjonalności w sposób niezgodny z wytycznymi: -8 pkt.
- nieznajomość zasad działania aplikacji w zakresie omówionym na laboratoriach, w szczególności w zakresie zastosowanych mechanizmów: -10 pkt,
- dodatkowa funkcjonalność: +3 pkt.

Punktacja dotyczy sytuacji ekstremalnych - całkowitego braku pewnego mechanizmu albo pełnej i poprawnej implementacji - możliwe jest przyznanie części punktów (lub punktów karnych).

Pozostałe uwagi:

- Zadanie trzeba prezentować sprawnie, będzie na to 14 minut.
- Termin nadesłania zadania **dla wszystkich grup**: 28 kwietnia 2025, godz. 13:00.
- Prezentowane **muszą** być **dokładnie** te zadania, które zostały zamieszczone na moodle, tj. nie są dopuszczalne żadne późniejsze poprawki.
- Konieczne jest dołączenie do zadania oświadczenia o samodzielnym jego wykonaniu. Wykonanie zadania wspólnie z innym studentem, a tym bardziej kopiowanie fragmentu lub całości zadania (także z rozwiązań z ubiegłych lat) będzie traktowane jako wykonanie **niesamodzielnie**. Natomiast konsultowanie się z innymi studentami **jak** zrealizować poszczególne funkcjonalności czy **wzorowanie** się na przykładach dostępnych w Internecie (a w szczególności w dokumentacji technologii) nie jest oczywiście traktowane jako niesamodzielność wykonania.

Ostatnia modyfikacja: czwartek, 17 kwietnia 2025, 21:45

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE
AGH

Platforma obsługiwana przez:
Centrum e-Learningu i Innowacyjnej Dydaktyki AGH
Centrum Rozwiązań Informatycznych AGH

Pobierz aplikację mobilną

Wybierz język