

**Zostało zdefiniowanych 11 zadań - 2 aplikacyjne (A.n) i 9 infrastrukturalnych (I.m). Należy zrealizować jedno wybrane zadanie aplikacyjne i jedno lub więcej zadanie infrastrukturalne; dodatkowym wymogiem jest, 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.**

#### **Zadanie A.1 - "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 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 urządzeniem nie jest przedmiotem zainteresowania projektu. Urządzenia mogą działać na wielu instancjach serwerów (demonstracja: na co najmniej dwóch).
- Projektując interfejs urządzeń należy używać także typów bardziej złożonych niż string czy int/long. 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 mogą 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 powinna pozwalać zademonstrować sterowanie różnymi 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 A.2 - 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.

##### 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.
- 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 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).

Technologia middleware: gRPC

Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)

Maksymalna punktacja: 12

### **Zadanie I.1 - Efektywność serializacji i czasu trwania wywołań technologii *middleware***

Celem zadania jest stworzenie prostej aplikacji klient-serwer i przeprowadzenie eksperymentów porównujących wydajność czasową (mierzoną przez czas wywołania synchronicznego blokującego) i efektywność komunikacyjną (wyrażoną w liczbie bajtów na poziomie L7) serializacji oferowanej przez Ice, Thrift i gRPC dla kilku (trzech) jakościowo różnych struktur danych i wielkości samych danych (tj. np. sekwencja o długości 1 i 100 elementów). Struktury danych definiowane przy pomocy różnych - z konieczności - IDL powinny być możliwie zbliżone do siebie. Dla uproszczenia realizacji wszystkie operacje/procedury mogą zwracać wartość pustą (void). Wyniki pomiarów należy przedstawiać po wyeliminowaniu wartości odstających (outlier). Eksperymenty powinny być przeprowadzone z wykorzystaniem interfejsu loopback oraz sieciowego (Ethernet/WiFi) - tj. na dwóch różnych komputerach.

Demonstracja zadania: Omówienie środowiska testowego, zamieszczonego zwięzłego raportu z warunkami eksperymentów i stosownymi wykresami oraz przedstawienie najważniejszych wniosków. W czasie prezentacji nie można się ograniczyć do suchego przedstawienia osiągniętych wyników, trzeba umieć je uzasadnić i „obronić”.

Technologie middleware: Ice i Thrift i gRPC (trzy)

Języki programowania: jeden (dla zmniejszenia liczby wymiarów problemu)

Maksymalna punktacja: 9

### **Zadanie I.2 - 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. Wywołania powinny być zrealizowane dla kilku (trzech) różnych operacji/procedur używających przynajmniej w jednym przypadku nietrywialnych struktur danych (np. listy (sekwencji) struktur). Nie trzeba tworzyć żadnego formatu opisującego żądanie 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>

gRPC: „dynamic grpc”, „reflection”, grpcurl

Technologia middleware: Ice albo gRPC

Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)

Maksymalna punktacja: 6

### **Zadanie I.3 - 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, np. dla dedykowanych serwantów, taki serwant jest instancjonowany dopiero w momencie pierwszego zapotrzebowania na niego.

Interfejs IDL obiektu może być superprosty, choć ma implikować konkretny sposób realizacji serwanta (co należy umieć uzasadnić).

Aplikacja kliencka powinna jedynie umożliwić zademonstrowanie funkcjonalności serwera. Logi na konsoli po stronie serwera powinny 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 dotyczy to wykorzystania tablicy ASM). Każdy obiekt middleware musi być „osiągalny” przez klienta przez podanie jego identyfikatora (Identity).

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) w razie przekroczenia zdefiniowanej, maksymalnej liczby serwantów.

Technologia middleware: Ice

Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)

Maksymalna punktacja: 7

#### Zadanie I.4 - Opcjonalne pola struktur danych i argumenty wywołania *middleware*

Celem zadania jest analiza zagadnienia definiowania opcjonalnych pól w strukturach danych IDL i wywołaniach *middleware* i demonstracja działania na prostych przykładach.

Demonstracja zadania: Omówienie interfejsów, omówienie zamieszczonego zwięzłego raportu podsumowującego wyniki sposobu serializacji danych (np. podgląd komunikacji w wireshark) i przedstawienie wniosków.

Ice: Optional Values (<https://doc.zeroc.com/ice/3.7/best-practices/optional-values>)

Technologia middleware: dwie lub trzy spośród technologii Ice i Thrift i gRPC

Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera) przynajmniej dla jednej technologii middleware

Maksymalna punktacja: 8

#### Zadanie I.5.- 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 powinien 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.

Technologia middleware: gRPC

Języki programowania: wystarczy jeden

Maksymalna punktacja: 10

#### Zadanie I.6 - 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 I.7 - 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 wzorce 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 prezentacji zadania należy umieć przedyskutować najważniejsze tezy opracowania oraz wykonać podstawowe testy.

Technologia middleware: Ice lub Thrift oraz gRPC (oraz REST i GraphQL)

Języki programowania: wystarczy jeden

Maksymalna punktacja: 10

#### Zadanie I.8 - 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\\_2](https://docs.icerpc.dev/getting-started/supported-platforms/icerpc-csharp-0_2)

Technologia middleware: iceRPC

Maksymalna punktacja: 10

#### Zadanie I.9 - Własny pomysł studenta

Celem zadania jest zbadanie i przedstawienie wybranych cech jednej lub 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.
- Aplikacja kliencka powinna mieć postać tekstową (z wyjątkiem zadania I5) 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.
- nieznanomość zasad działania aplikacji 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 15 minut.
- Termin nadesłania zadania **dla wszystkich grup**: 6 maja 2024, godz. 12:00.
- 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 niesamodzielnosc wykonania.

Ostatnia modyfikacja: wtorek, 16 kwietnia 2024, 09:44



Platforma obsługiwana przez:  
[Centrum e-Learningu i Innowacyjnej Dydaktyki AGH](#)  
[Centrum Rozwiązań Informatycznych AGH](#)

Pobierz aplikację mobilną



---

Wybierz język

