Inżynieria Oprogramowania

Studenci: Sylwia Nowak, Błażej Bobko, Jakub Cieślik, Patryk Kujawski

Prowadzący: mgr inż. M. Okulewicz

Tytuł projektu: **Computional Cluster**

Typ projektu:
System rozproszonego rozwiązywania problemów obliczeniowych zaprojektowany pod kątem DVRP.
Data oddania: 2015-01-30

Spis Treści

Wstęp

Niniejsza dokumentacja opisuje działanie systemu rozwiązującego problemy obliczeniowe w sposób współbieżny łącząc się zdalnie z wieloma węzłami obliczeniowymi o różnej specjalizacji. Ze względu na złożoność postawionego zadania zaprojektowano zestaw aplikacji oraz protokół opisujący komunikację między nimi. Jako przykładowy problem obliczeniowy postawiono szczególny przypadek problemu marszrutyzacji - DynamicVehicle Routing Problem (DVRP).

Używane określenia i skróty

- System - system rozproszonego rozwiązywania problemów obliczeniowych, przedmiot tego projektu.
- Komponent - aplikacja będąca częścią systemu (patrz: Architektura systemu).
- Warstwa obliczeniowa - część logiczna systemu odpowiedzialna za obliczenia (patrz: Architektura systemu).
- Warstwa transportowa - część logiczna systemu odpowiedzialna za: poprawne działanie komunikacji między komponentami systemu (patrz: Architektura systemu).
- Typ problemu - rodzaj problemu obliczeniowego danej klasy problemów (np. DVRP).
- DVRP - DynamicVehicle Routing Problem.
- Problem - instancja problemu.
- Instancja problemu - Typ problemu wraz z danymi wejściowymi.
- Podproblem - pojedyncze zadanie obliczeniowe powstałe po podziale problemu, wysyłane do Node'ów.
- Client - aplikacja kliencka, opisana dalej.
- CS - Communication Server, serwer komunikacyjny, opisany dalej.
- BCS - Backup Communication Server, zapasowy serwer komunikacyjny, opisany dalej.
- Server - CS lub BCS.
- Lista Serwerów - lista danych komunikacyjnych dot. CS i BCS.
- TM - Task Manager, opisany dalej.
- TSt - TaskSolver znajdujący się wewnątrz TM, opisany dalej.
- Node - Węzeł obliczeniowy, opisany dalej.
- TSn - TaskSolver znajdujący się wewnątrz Node'a, opisany dalej.
- Rejestracja - zapisanie rejestrowanych danych w strukturach wewnątrz komponentów (głównie CS).
- Awaria - niewłaściwe działanie w warstwie komunikacyjnej.

Algorytm DVRP

Algorytm DVRP jest jedną z wielu możliwości rozwinięcia problemu marszrutyzacji. Polega on na wyznaczeniu optymalnych tras przewozowych dla określonych ograniczeń zwanych *danymi początkowymi*, które przykładowo może definiować użytkownik.

Danymi początkowymi mogą być:

- Ilość środków transportu (np. samochodów czy ciężarówek) zwanych *wehikułami.* Ich liczba musi być wartością większą lub równą 1. Ponad to, każdy *wehikuł* posiada parametry, które mogą ograniczać jego możliwości (np. maksymalna pojemność).
- *Klienci* - punkty (np. współrzędne geograficzne), które muszą być obsłużone. *Klienci* posiadają *dane* (np. paczki z przesyłkami), które muszą zostać odebrane i przewiezione.
- *Koszty* (np. koszty przejazdów między wszystkimi *klientami*) - mogą być to miary kosztu czasu, utrzymania pojazdów czy amortyzacji.
- *Dana* - np. paczka, która posiada swoje cechy (np. wagę).
- Dodatkowymi parametrami mogą być np. czas zamknięcia i otwarcia magazynu czy długość możliwości czasu pracy kierowcy.

Na podstawie tak zdefiniowanych danych początkowych algorytm dokonuje obliczeń.

W naszym systemie interfejsem umożliwiającym zdefiniowanie problemu (przez określenie *danych początkowych*) jest komponent *Client*.

Osobą korzystającą z interfejsu z komponentu *Client* może być np. *dyspozytor* w firmie kurierskiej.

Zadaniem *dyspozytora* jest między innymi ciągła analiza dostępności *wehikułów* i przekazywanie *kierowcom wehikułów* kolejnego miejsca docelowego, będącego następną lokalizacją do odebrania/dostarczenia *danej* (paczki). *Dyspozytor* jest również osobą, która dodaje nowych *klientów.* Dla jeszcze raz zdefiniowanej instancji problemu wysyła ponownie problem do obliczenia. Po zakończeniu wykonywania algorytmu *dyspozytor* otrzymuje nową listę przydziałów. Taki schemat powtarzany jest przez cały dzień pracy *dyspozytora*.

Algorytm generując trasy dla *wehikułów* dodatkowo może sprawdzać, którzy *klienci* zostali już obsłużeni, czy moment zgłoszenia nowego *klienta* nie przekroczył *czasu cut-off* (zdefiniowanego czasu odcięcia decydującego np. o przeniesieniu *klienta* do listy *klientów* na dzień następny) lub czy *wehikuły* mogą przyjąć jeszcze *dane* (paczki).

Rys historyczny

Pierwsze informacje dotyczące rozwinięcia problemu marszrutyzacji w postaci dynamicznej zawdzięczamy Wilson'owi i Colvin'owi, którzy prowadzili badania nad problemem DARP (dial-a-ride problem) w roku 1977. Zakładali oni istnienie jednego wehikułu, którego nowe punkty docelowe mogły pojawiać się w sposób dynamiczny.

Algorytm wielokrotnie w późniejszym czasie ulepszany doczekał się swoich lat świetności w momencie globalnego wprowadzenia usług GPS i GIS.

Algorytm DVRP okazuje się być niezwykle pomocny przy zarządzaniu operacyjnym flotą środków transportu.

Zaletami płynącymi z wykorzystywania tego algorytmu jest przykładowo redukcja kosztów, polepszenie jakości usług świadczonym klientom (np. w spedycji towarów) czy redukcja zanieczyszczeń emitowanych do środowiska (wynikająca np. z redukcji emisji spalin pojazdów kurierskich).

Architektura Systemu

Z myślą o przejrzystości opisu działania poszczególnych części systemu podzieliliśmy go na dwie warstwy logiczne: warstwę transportową odpowiedzialną za synchronizację danych potrzebnych do prawidłowego działania systemu oraz warstwę obliczeniową, której zadaniem jest rozwiązanie postawionego problemu poprzez podzielenie go na podproblemy, rozwiązanie ich, a następnie połączenie ich w ogólne rozwiązanie.

Komponenty systemu

Communication Server (CS) - aplikacja odpowiadająca za pośredniczenie w komunikacji pomiędzy wszystkimi pozostałymi komponentami systemu.

- Warstwa Transportowa - pośredniczy w synchronizacji danych potrzebnych do prawidłowego działania komunikacji między wszystkimi komponentami.
- Warstwa Obliczeniowa - przechowuje i synchronizuje ze wszystkimi BCS'amitymczasowe informacje dotyczące problemów i podproblemów oraz ich rozwiązań (tak długo jak będą potrzebne).

Backup Communication Server (BCS) - czeka w gotowości na wypadek gdyby CS przestał funkcjonować by przejąć jego zadania. Synchronizuje dane z CS by poprawnie go zastąpić.

- Warstwa Transportowa - w przypadku otrzymania komunikatu (od dowolnego komponentu systemu), który powinien zostać zaadresowany do CS, przekierowuje go do CS.
- Warstwa Obliczeniowa - brak zadań.

Task Manager (TM) - aplikacja inicjująca i zakańczająca rozwiązywanie zadań.

- Warstwa Transportowa - pobiera dane potrzebne do połączenia z CS oraz w przypadku awarii, z którymś BCS.
- Warstwa Obliczeniowa - na podstawie typu zadania inicjalizuje wyspecjalizowany TaskSolver (TSt), który na podstawie dostępnej ilości węzłów obliczeniowych (Node) dzieli zadanie na podproblemy do rozwiązania przez TaskSolvery (TSn). Nastepnie po otrzymaniu rozwiązań podproblemów łączy je w jedno rozwiązanie udostępniane Clientowi.

Computational Node (CN)- aplikacja węzła obliczeniowego.

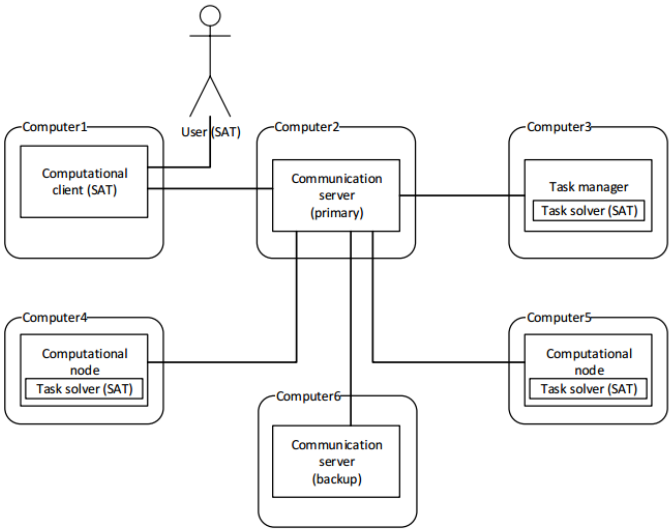
- Warstwa Transportowa - pobiera dane potrzebne do połączenia z CS oraz w przypadku awarii, z którymś BCS.
- Warstwa Obliczeniowa - inicjalizuje wyspecjalizowany TaskSolver (TSn), który rozwiązuje przesyłane mu podproblemy.

Computatnional Client (CC) - aplikacja kliencka.

- Warstwa Transportowa - pobiera dane potrzebne do połączenia z CS oraz w przypadku awarii, z którymś BCS.
- Warstwa Obliczeniowa - pośredniczy między użytkownikiem a resztą systemu. Pozwala na wyspecyfikowanie problemu (rodzaj problemu, dane wejściowe), który następnie przesyła do TM (za pośrednictwem CS) oraz otrzymuje i wyświetla wynik.

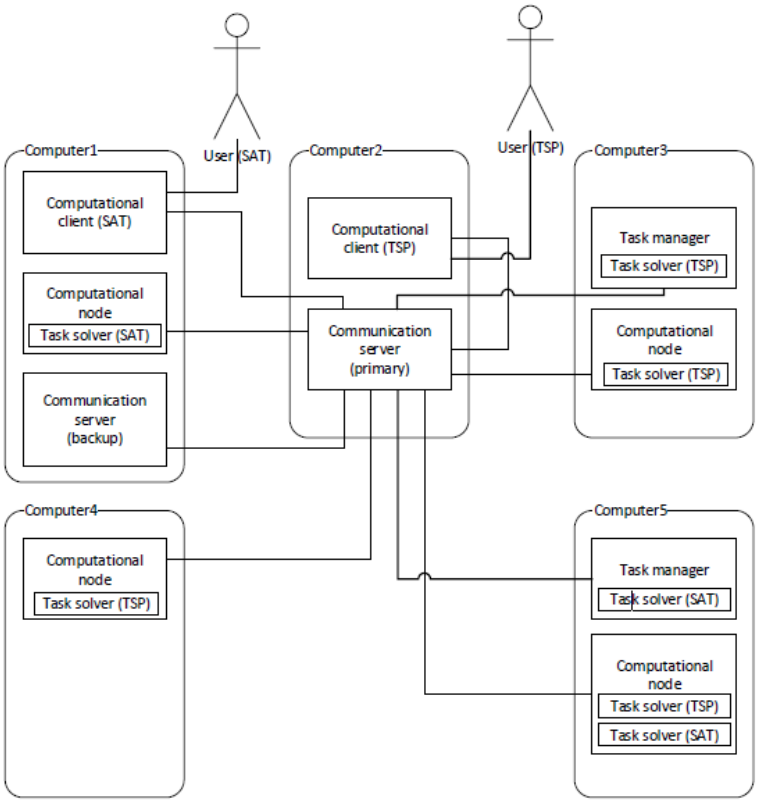
Tryby pracy systemu w zależności od ilości komponentów

Przypadek trywialny



W tym przypadku system pracuje sprawnie przy minimalnej ilości komponentów potrzebnych do prawidłowej funkcjonalności - uruchomiony jest 1 CS, 1 CC, 1 TM i więcej niż 1 CN. Dodatkowo pracuje w nim 1 BCS, ale z opisanej później architektury systemu wyniknie, że system jest między innymi skalowalny do dowolnej (w granicach możliwości sieci) ilości BCS'ów.

Przypadek zaawansowany

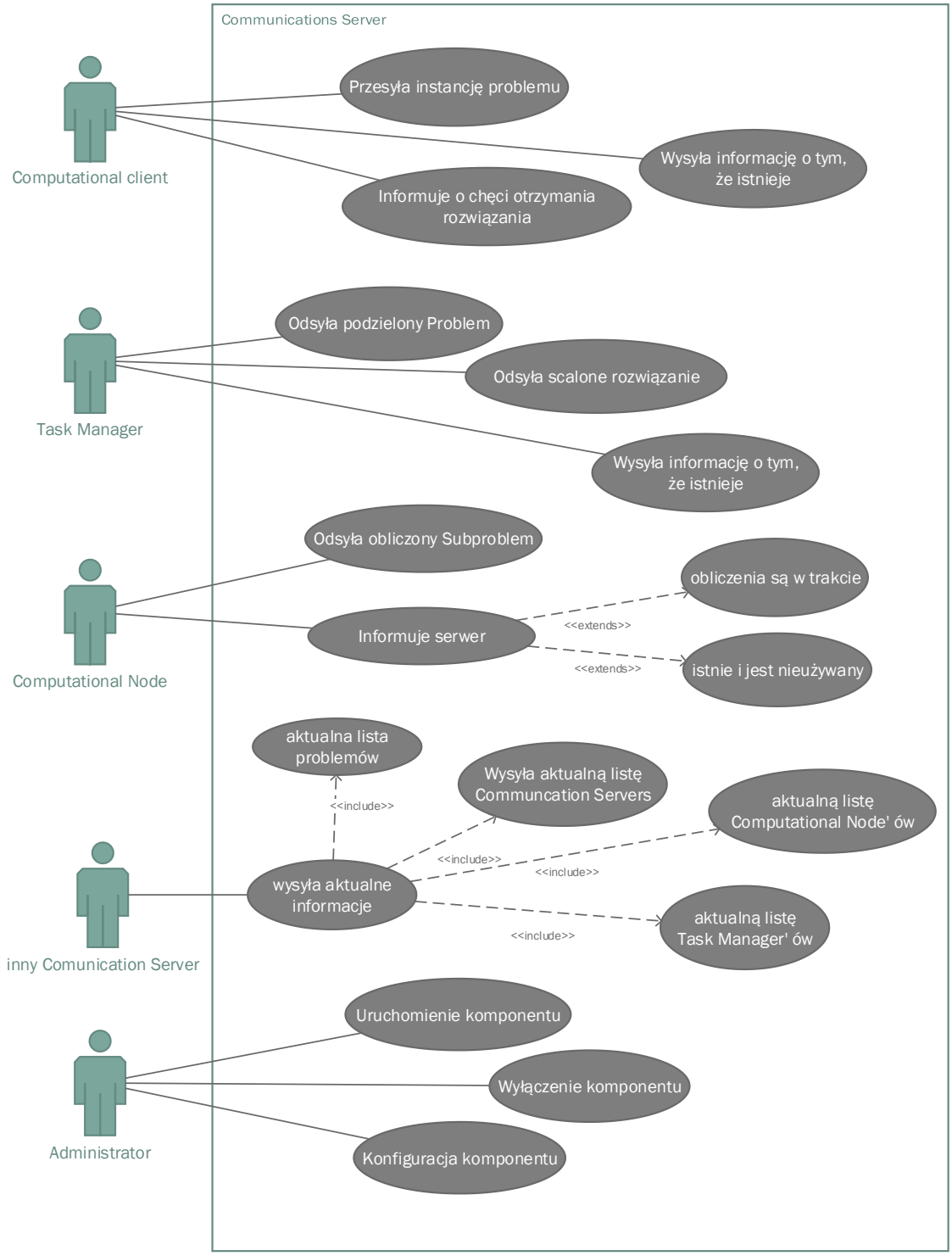


W systemie istnieje wiele komponentów różnego typu. Przypadek ten jest wiele bardziej skomplikowany. Przedstawione rozwiązanie umożliwia dowolną skalowalność systemu.

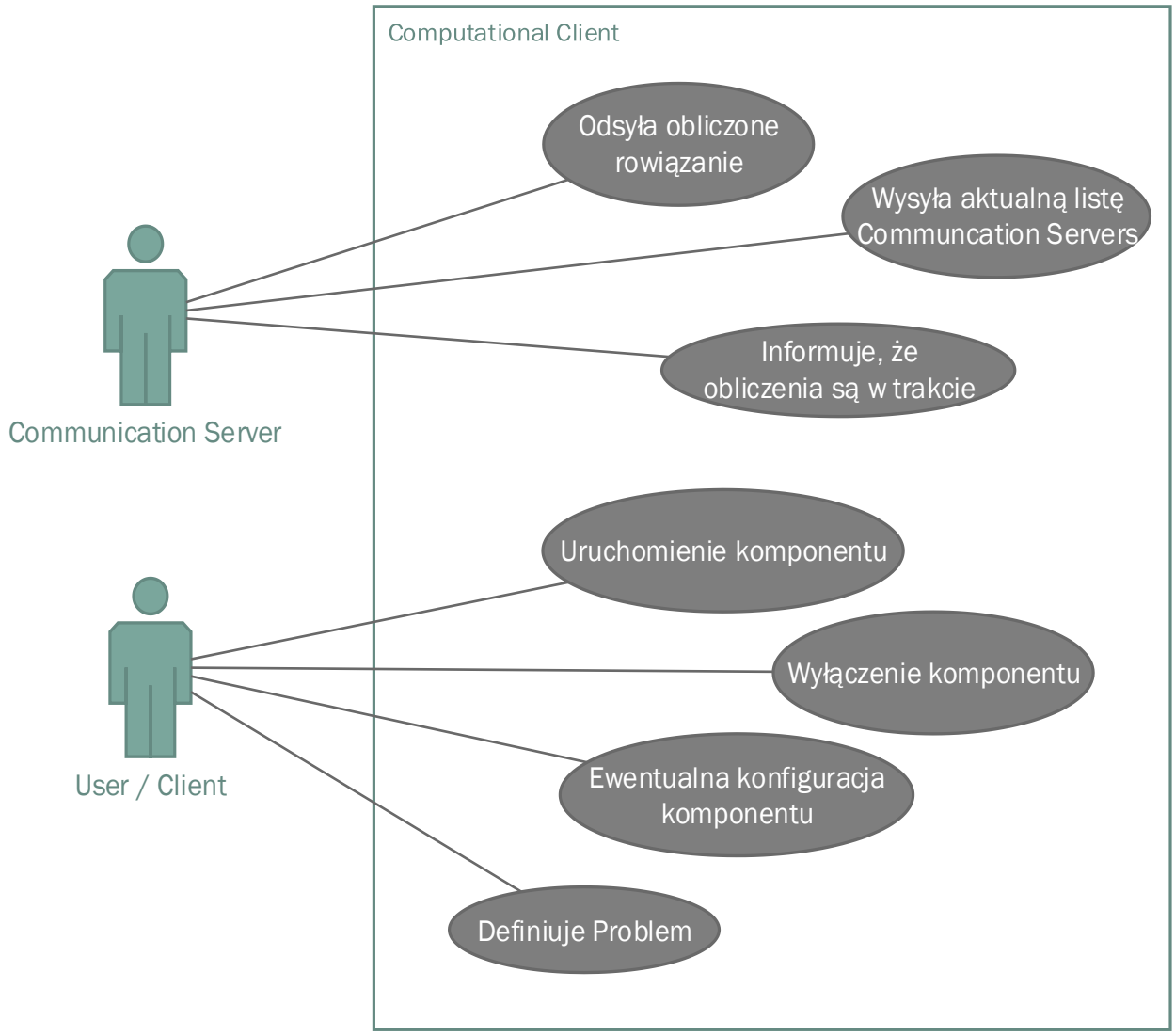
Zagadnienia związane z pracą systemu

Przypadki użycia programów

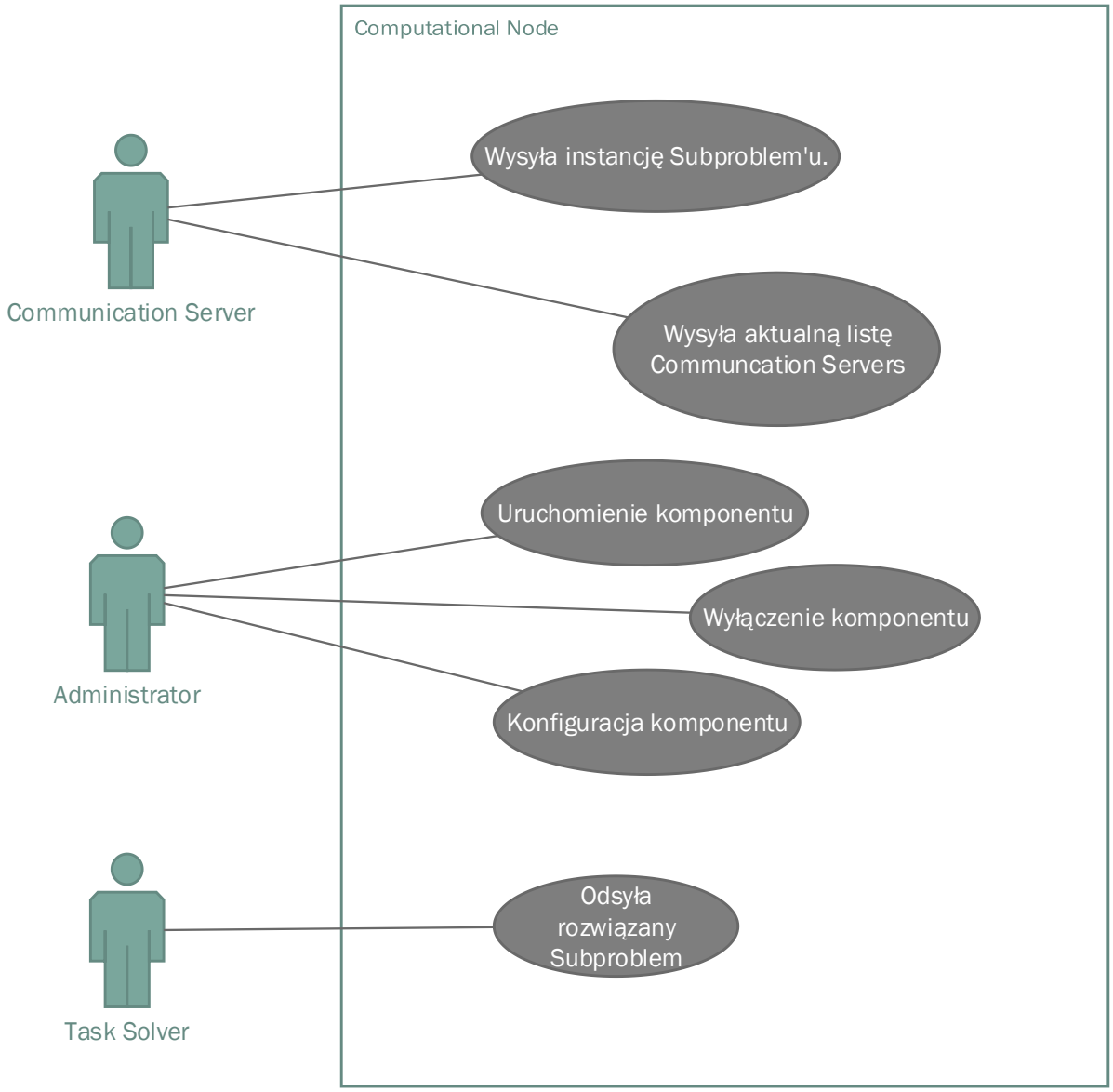
Communication Server



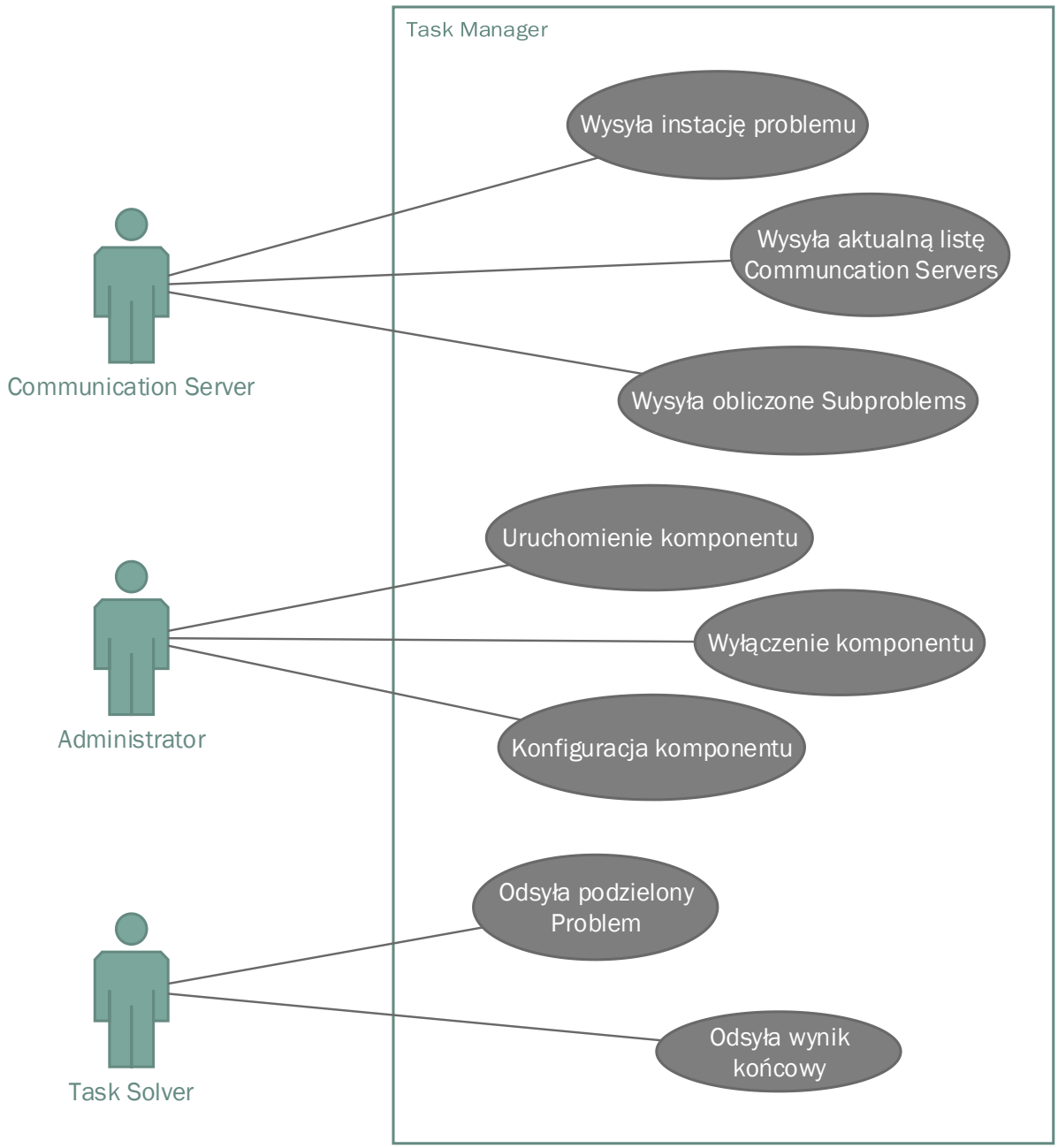
Computational Client



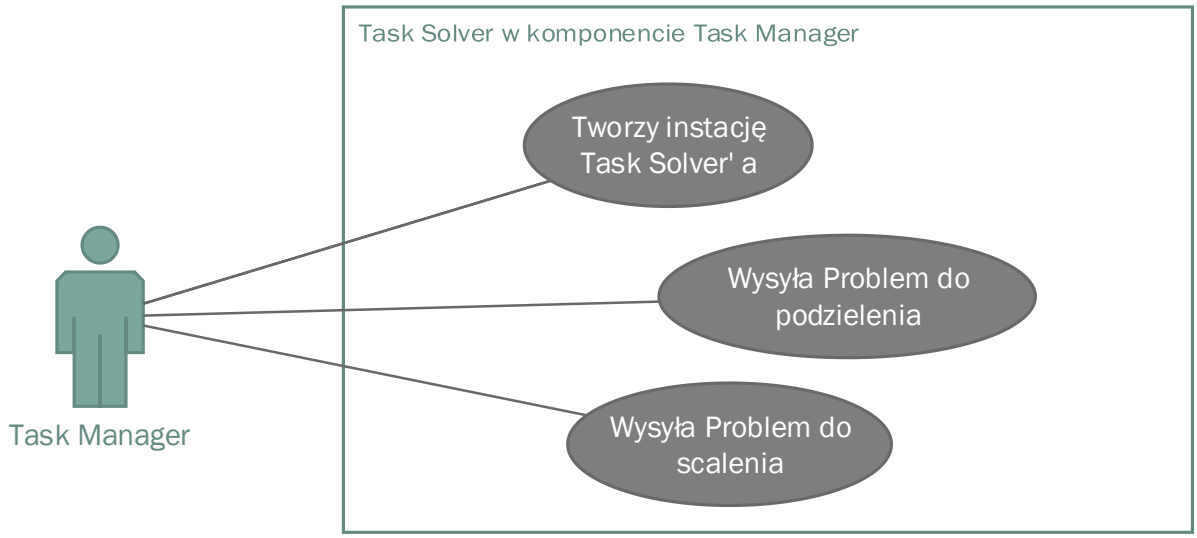
Computational Node



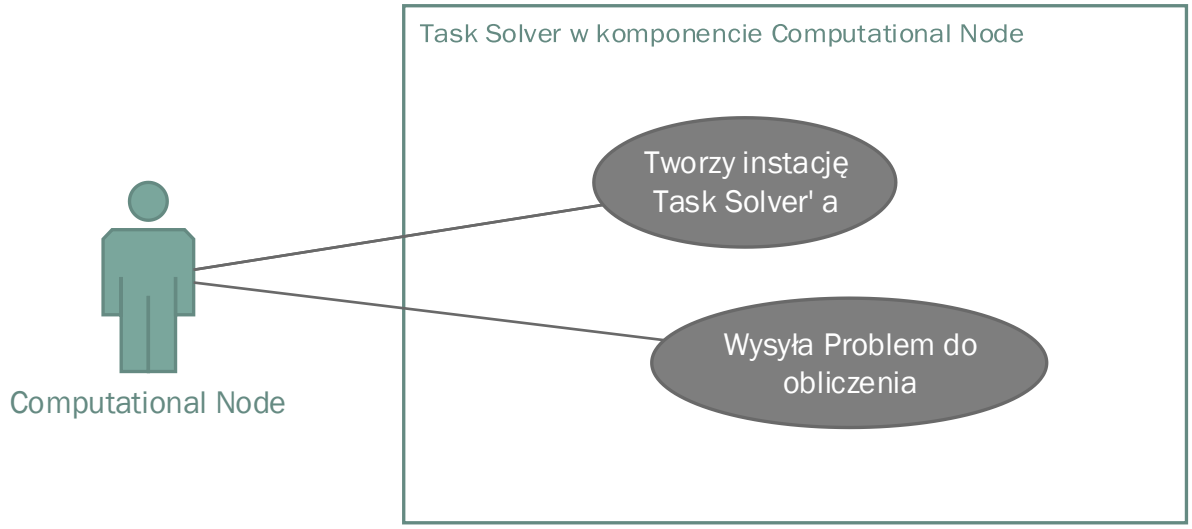
Task Manager



Task Solver dla komponentu Task Manager



Task Solver dla komponentu Computational Node



Każdy z komponentów systemu jest uruchamiany i konfigurowalny przez użytkowników (zwykłych lub administratorów).

Interfejs Systemu

Każdy z komponentów systemu ma oddzielny interfejs:

- Node, TM, CS i BCS uruchamiane i konfigurowane są przez administratora systemu.
- Client uruchamiany i obsługiwany jest przez użytkownika wysyłającego zadanie do rozwiązania.
- TSt uruchamiany jest automatycznie na podstawie żądania otrzymanego przez TM.
- TSn uruchamiany jest automatycznie na podstawie żądania otrzymanego przez Node.

Konfiguracja komponentów

Wszystkie komponenty wymagają wprowadzenia konfiguracji początkowej zawierającej dane potrzebne do połączenia z CS (lub dowolnym BCS, patrz “sytuacje wyjątkowe”). W przypadku bezawaryjnego działania systemu nie powinna zajść potrzeba wprowadzania tych konfiguracji ponownie (nawet po jego wyłączeniu), ponieważ komponenty nawiązując połączenia aktualizują posiadane dane dot. komunikacji.

Ponadto każdy Node i TM specyfikują jakie typy problemów są w stanie rozwiązać (jakie TSt i TSn potrafią zainicjalizować). Oprócz tego należy skonfigurować częstotliwość wysyłania LifeStatusReport przez te komponenty oraz zachowanie w przypadku rozłączenia/awarii (ile razy ponawiać połączenie, przy której próbie wyświetlić administratorowi komunikat o zaistniałej sytuacji).

W konfiguracji CS i BCS należy określić timeout, po którym jeśli nie otrzymamy od komponentu komunikatu LifeStatusReport uznajemy ten komponent za odłączony (służy m. in. wykrywaniu awarii). Można również wyspecyfikować parametry określające metody koordynowania rozwiązywania problemów (wybieranie ilości Node’ów, metody przyporządkowywania Node’ów do podproblemów).

Schemat rozwiązania pojedynczego problemu

1. Użytkownik specyfikuje problem oraz dane wejściowe.
2. Client wysyła w.w. dane do CS, który rejestruje instancję problemu.
3. CS rozsyła do dostępnych węzłów żądanie zainicjowania TSn.
4. Node próbuje zainicjalizować TSn do danego problemu i informuje CS o sukcesie lub porażce.
5. CS zapisuje Node’y, które odpowiedziały informacją o sukcesie (brak odpowiedzi liczony jest jako porażka) i rejestruje je jako powiązane z danym problemem.
6. CS wysyła zadanie powołania TSt do dostępnego TM i powiązuje go w danych dotyczących komunikacji z zadanym problemem.
7. CS wysyła zadanie rozwiązania problemu wybranemu TM wraz z ilością przydzielonych do niego Node’ów (patrz punkt 5).
8. TSt wewnątrz TM dzieli problem na podproblemy, a następnie TM odsyła je do CS.
9. CS rejestruje podproblemy i koordynuje przydział podproblemów do wolnych Node’ów.
10. TSn wewnątrz Node’a rozwiązuje podproblem a następnie Node odsyła rozwiązanie podproblemu do CS.
11. CS po otrzymaniu wszystkich rozwiązań odsyła je do TM.
12. TM wewnątrz TSt łączy rozwiązania podproblemów w rozwiązanie problemu i odsyła je do CS.
13. CS rejestruje rozwiązanie problemu, usuwa dane dot. podproblemów i wysyła żądanie zakończenia działania TSn do powiązanych z problemem Node’ów.
14. Client wysyła do CS RequestSolution. Jeśli CS posiada już rozwiązanie zadania odsyła Clientowi rozwiązanie.
15. Client wyświetla rozwiązanie użytkownikowi.

Diagram aktywności rozwiązywania pojedynczego problemu.

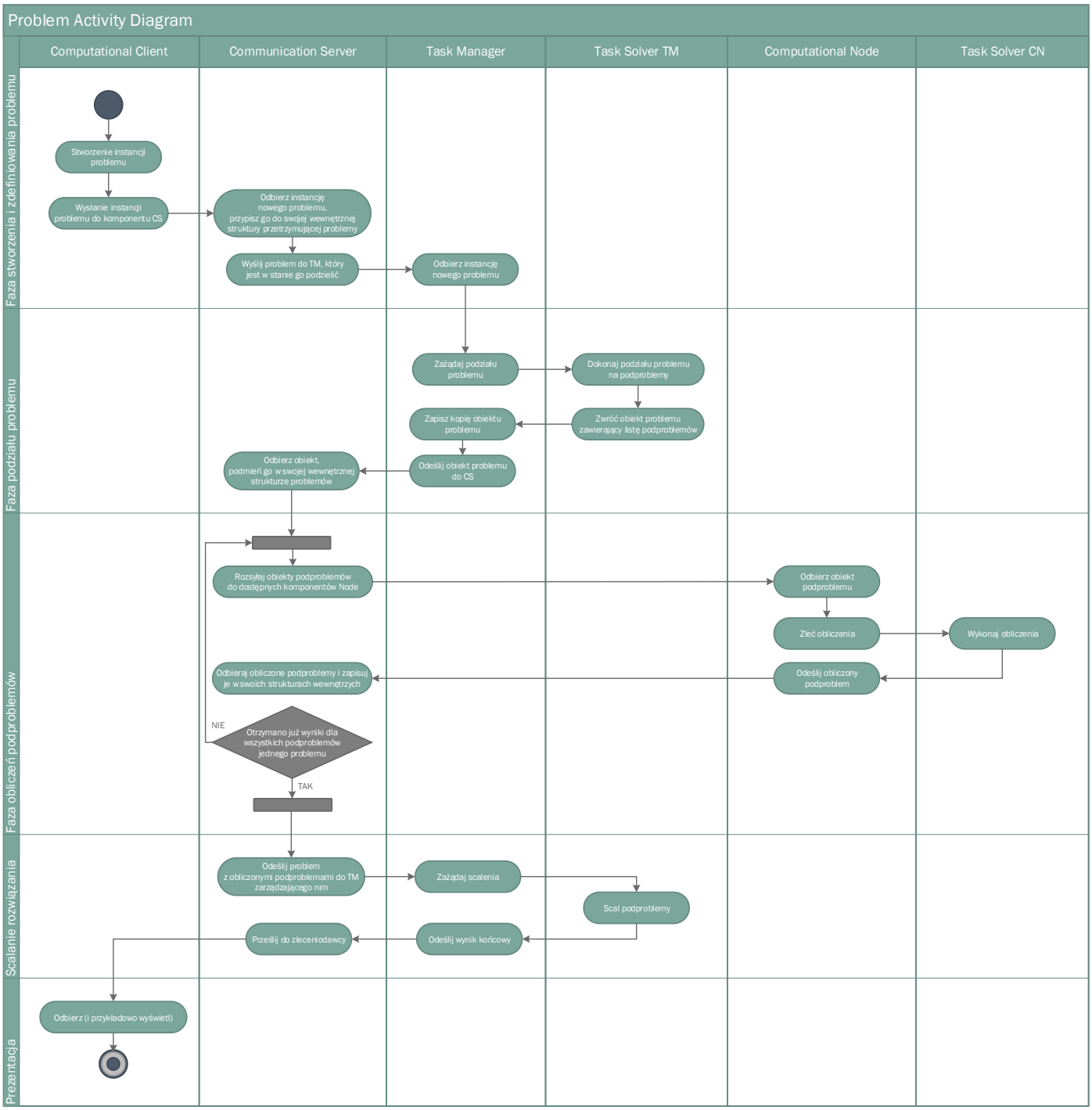


Diagram prezentuje ideę przepływu obiektów w systemie od momentu narodzenia instancji problemu, przez jego rozwiązywanie aż po moment ostatecznej prezentacji wyniku obliczeń.

Założone zostało, że wszystkie komponenty systemu w momencie utworzenia instancji problemu są już włączone i gotowe do pracy. W trakcie obliczeń nie występują żadne awarie ani problemy. Ponadto istnieją komponenty Task Manager'ów i Computational Node'ów, które są w stanie obliczyć zadany przez użytkownika problem.

Działanie warstwy transportowej

Komunikacja CS-Client

1. Synchronizacja adresów CS.
    a. Client wysyła do CS komunikat RequestTimestamp z częstotliwością określoną w konfiguracji.
    b. CS odsyła komunikat ResponseSimpleTimestampzawerającytimestamp określający datę ostatniej aktualizacji listy serwerów.
    c. Jeśli otrzymany timestamp różni się od timestampu obecnie posiadanej listy wysyłany jest RequestNodeClient.
    d. W odpowiedzi na RequestNodeClient CS wysyła RequestResponse zawierający jedynie listę serwerów.
2. Dołączenie do systemu odbywa się tak samo jak synchronizacja adresów, z jedną różnicą: po otrzymaniu RequestTimestamp CS dodatkowo rejestruje nadawcę.

Komunikacja CS-Node/CS-TM

1. Synchronizacja oraz dołączenia odbywają się tak samo jak w przypadku Clienta.
2. W trakcie działania komponenty wysyłają do CS LifeStatusReport ze skonfigurowaną wcześniej częstotliwością, CS odpowiada na to komunikatem ACK (pozwala to wykryć rozłączenia).

Komunikacja CS-BCS

1. Synchronizacja zarejestrowanych danych.
    a. BCS wysyła do CS RequestTimestamp.
    b. CS w odpowiedzi odsyła ResponseComplexTimestamp, w którego skład wchodzą timestampy następujących zbiorów danych: dane dot. Node'ów, dane dot. TM, dane dot. Clientów, dane dot. serwerów, dane dot. podproblemów.
    c. Jeśli którykolwiek timestamp wskazuje na nieaktualność przechowywanych w BCS danych to BCS odsyła do CS RequestServerTables specyfikując, które dane są do zaktualizowania.
    d. CS odsyła w odpowiedzi na RequestServerTablesRequestResponse zawierający żądane dane.

Sytuacje wyjątkowe

Awaria primary CS

BCS otrzymuje komunikat, który powinien był trafić do CS

1. BCS inicjuje standardową próbę synchronizacji z CS.
    a. W razie sukcesu BCS przesyła komunikat do CS.
    b. W razie porażki BCS stwierdza awarię CS (co wywołuje zachowanie opisane dalej).

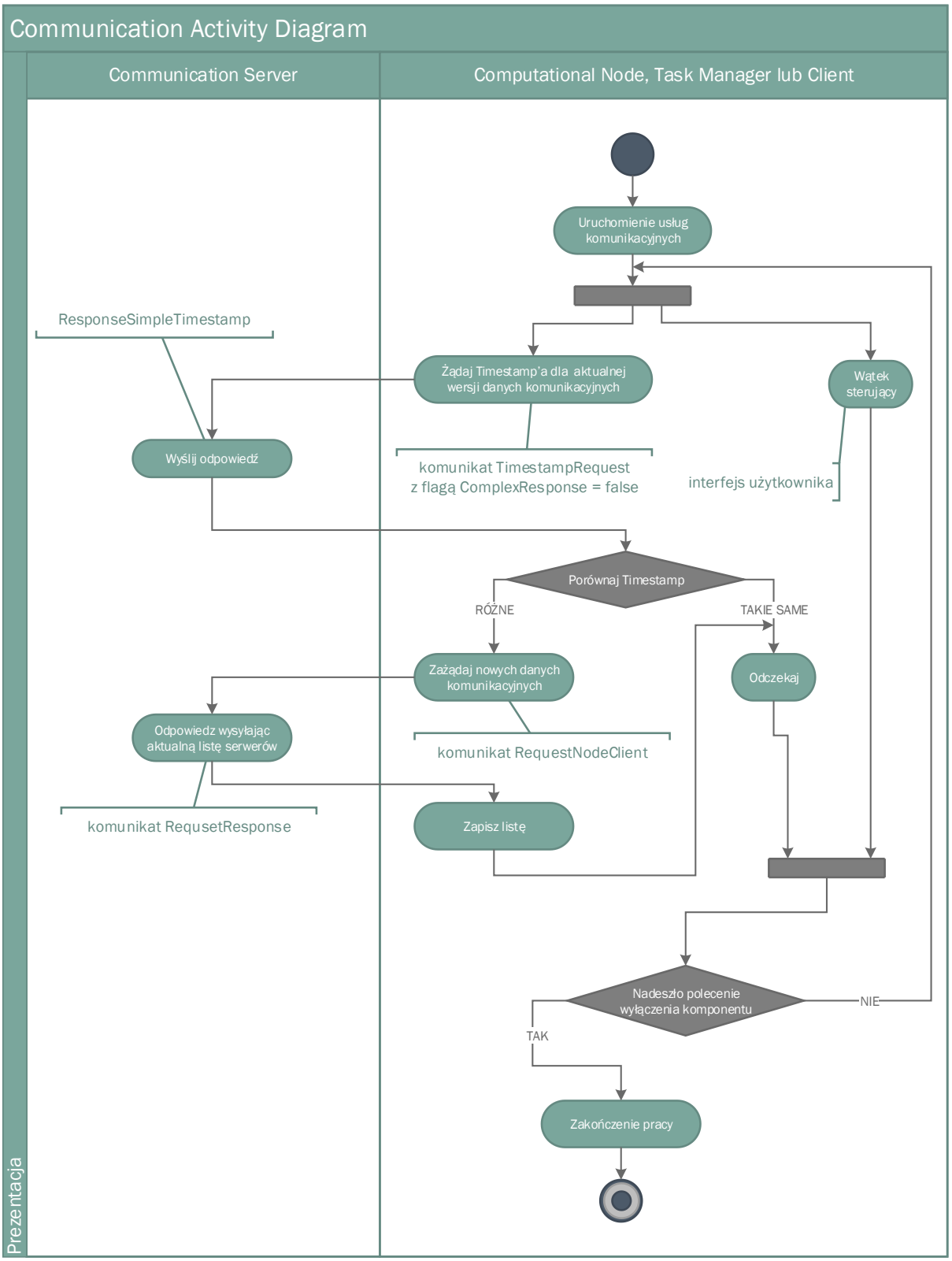
Awarie pozostałych komponentów - warstwa obliczeniowa działa tak samo jak zawsze (z możliwymi opóźnieniami). Zadaniem warstwy transportowej jest przywrócenie lub utrzymanie poprawnej komunikacji między komponentami. Przewidujemy następujące scenariusze:

2. Awaria CS - wykrywana poprzez brak odpowiedzi na TimestampRequest lub LifeStatusReport. Wyróżniamy dwie sytuacje:
    a. Komponent, który nie otrzymał odpowiedzi to BCS.
        i. Komponent ponawia próbę iterując po liście serwerów aż do otrzymania odpowiedzi (przypadek a. poniżej) lub dotarcia do samego siebie (przypadek b. poniżej).
            1. Przesyła komunikat, na który wcześniej nie otrzymał odpowiedzi do serwera, od którego otrzymał odpowiedź (może spowodować zachowanie opisane w poprzednim paragrafie)
            2. Uznaje siebie za nowy CS, aktualizuje listę serwerów tak by był na niej pierwszy.
    b. Komponent, który nie otrzymał odpowiedzi nie jest BCS'em
        i. Komponent ponawia próbę iterując po liście serwerów aż do otrzymania odpowiedzi (może spowodować sytuację opisaną w poprzednim paragrafie)
        ii. Komponent nie otrzymał odpowiedzi od żadnego serwera - mamy do czynienia z awarią komponentu, a nie awarią CS.
3. Awaria Node'a
    a. CS wykrywszy awarię Node'a oflagowuje go jako rozłączony (oczekując na jego powrót).
    b. Node, który się odłączy próbuje połączyć się ponownie w sposób ustawiony w jego konfiguracji.
4. Awaria Clienta
    a. CS wykrywszy awarię (lub wyłączenie) Clienta czeka ustawiony w konfiguracji "czas przetrzymywania rozwiązania" po którym usuwa wszelkie dane dotyczące problemów wysłanych przez tego Clienta czekając wcześniej na ich rozwiązanie.
5. Awaria BCS
    a. CS wykrywając awarię BCS usuwa go z listy serwerów.
    b. BCS, który wykrył swoje rozłączenie próbuje połączyć się ponownie w sposób ustawiony w jego konfiguracji.
6. Awaria TM
    a. TM, który wykrył swoje rozłączenie próbuje połączyć się ponownie w sposób ustawiony w jego konfiguracji.
    b. CS po wykryciu rozłączenia/awarii TM oflagowuje go jako rozłączony.
    c. Jeśli wszystkie podproblemy zostały rozwiązane a TM wciąż jest rozłączony CS przeprowadza ponownie całą procedurę inicjalizacji obliczeń i po otrzymaniu listy podproblemów nie nadpisuje poprzedniej. Następnie kontynuuje działanie jakby nic się nie stało.

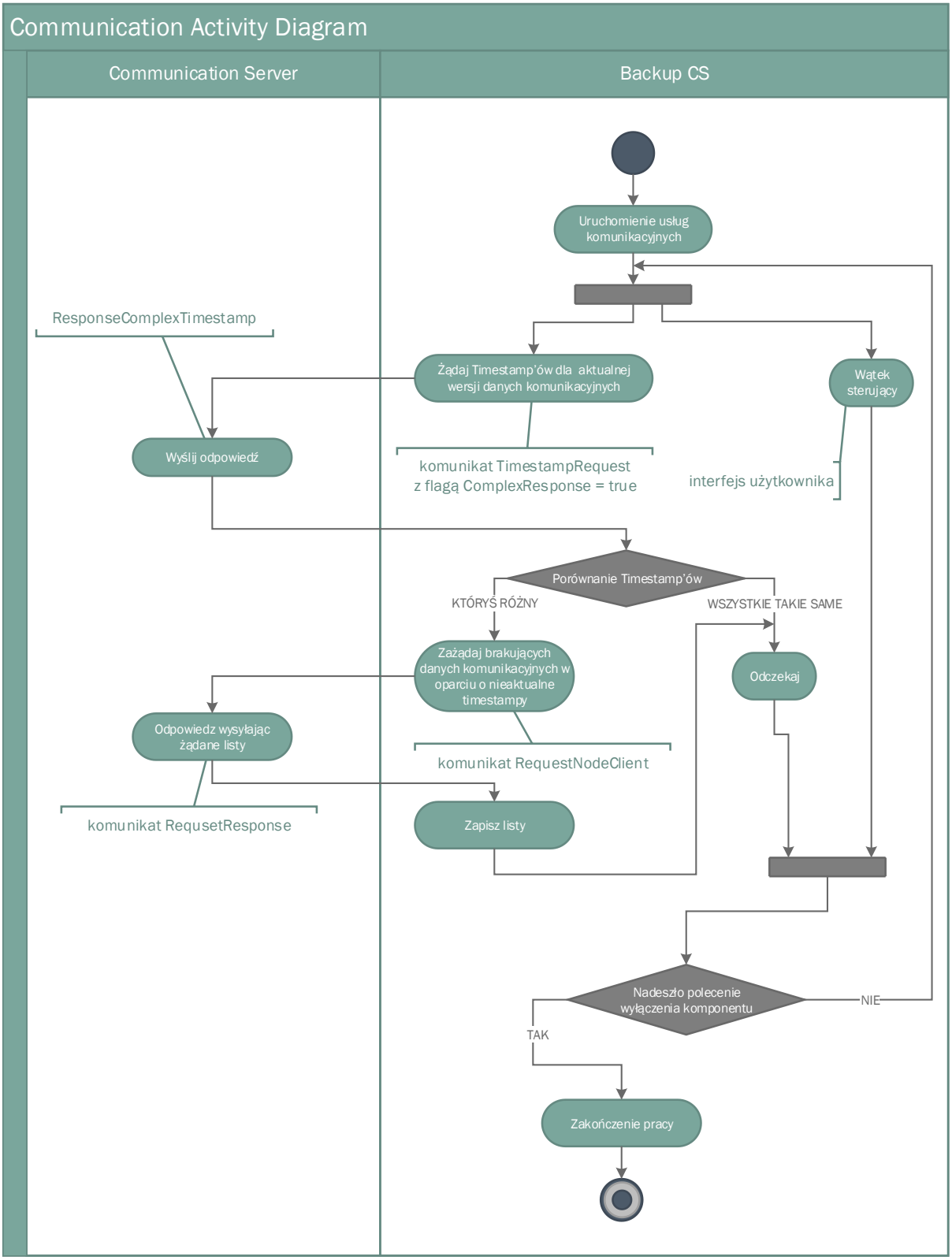
Sytuacje nieprzewidziane - pełny reboot systemu.

Diagramy aktywności warstwy transportowej dla poszczególnych komponentów systemu

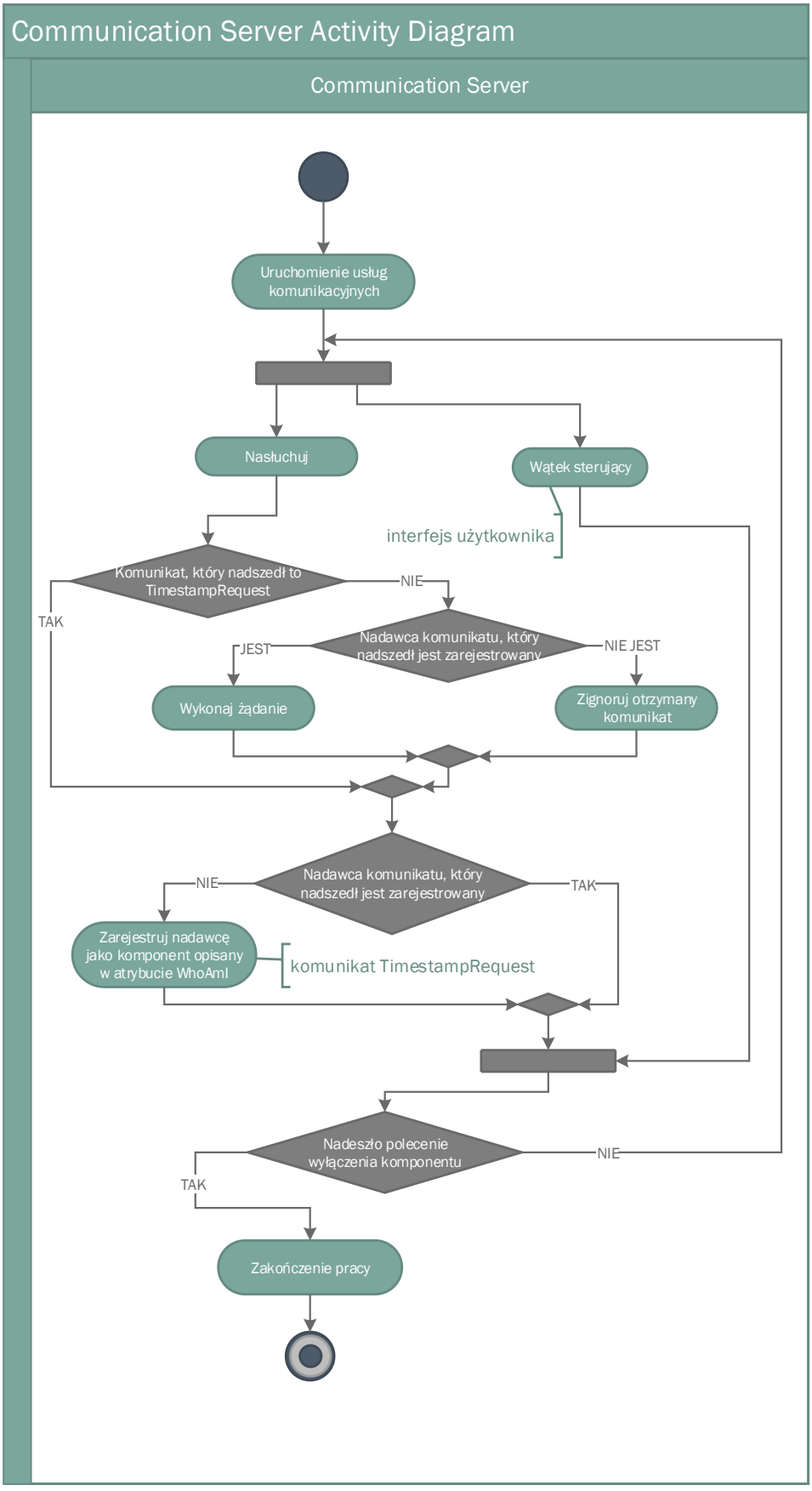
Computational Node, Task Manager, Client.



Communication Server z dowolnym Backup Communication Server'em.



Communication Server



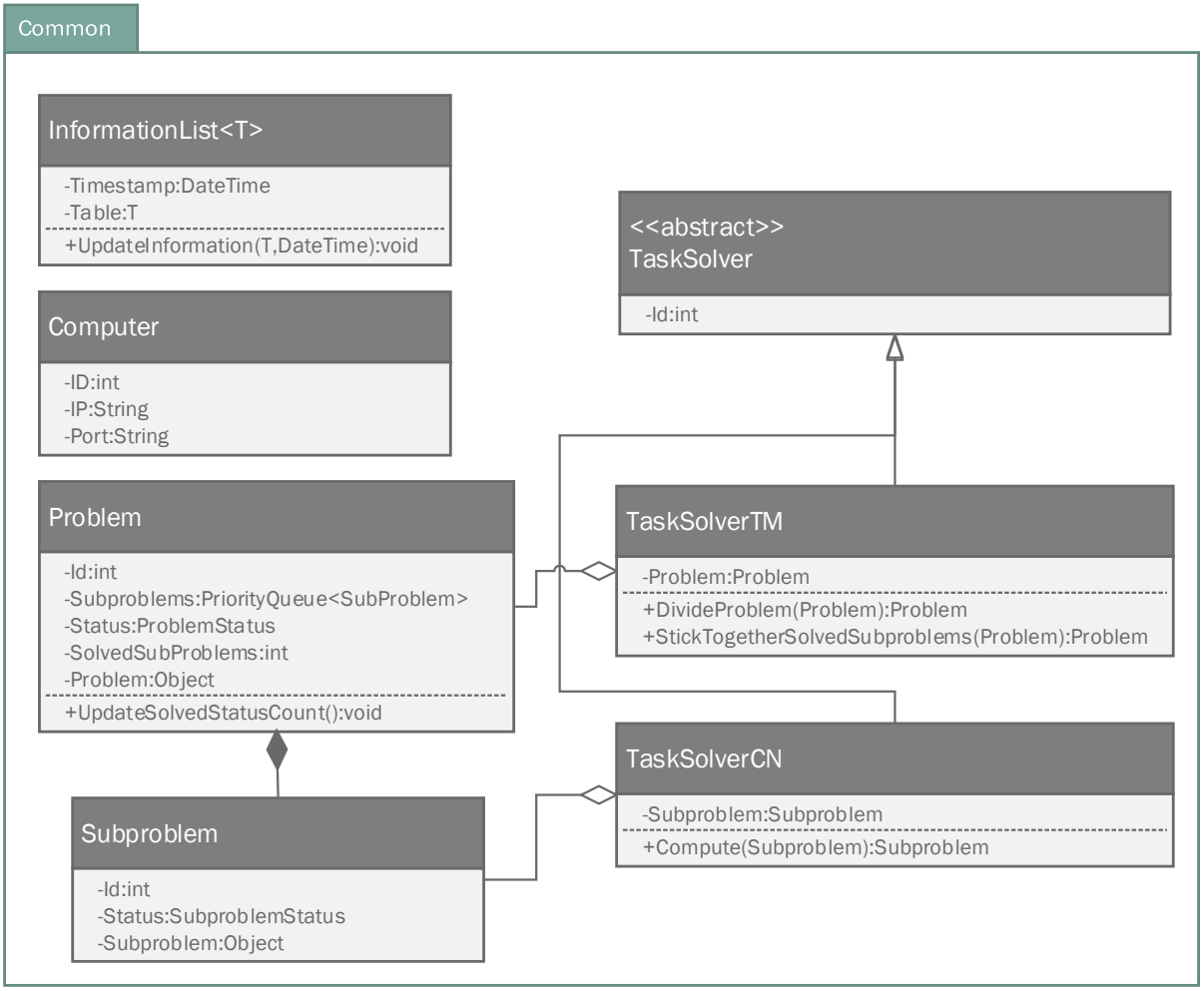
Stany elementów systemu

Stany, w jakich mogą znajdować się wybrane komponenty i elementy systemu zostały opisane w Załączniku *"Stany elementów systemu"*.

Struktury danych

Package Common

Pakiet zawierający klasy będące wspólną częścią użytkową dla wszystkich komponentów systemu.



InformationList<T>

Generyczna klasa pozwalająca na tworzenie tabel wybranego typu T. Wykorzystywana jest ona do zapisywania informacji o przydziałach zadań, lokalizacjach itp. Zawiera zawsze informację o ostatniej zmianie danych w tablicy Table - Timestamp. Metoda UpdateInformation ustawia ponownie Timestamp i Table, po otrzymaniu nowych danych.

Computer

Klasa przetrzymująca dane o lokalizacji sieciowej komponentu. Wszystkie komponenty systemu po niej dziedziczą. Przyjęcie takiego rozwiązania (lista pól) sugeruje, że system przygotowany będzie do stabilnej pracy wewnątrz jednej sieci lokalnej.

TaskSolver

Klasa bazowa dla dwóch dziedziczących po niej klasach służących do rozwiązywania problemu na różnym etapie ich przetwarzania - TaskSolverTM (dla komponentu Task Manager) i TaskSolverCN (dla komponentu ComputationalNode).

TaskSolverCN

Metoda Compute przyjmuje Subproblem do rozwiązania i zwraca ten sam obiekt.

TaskSovlerTM

Metoda DivideProblem przyjmuje Problem do podzielenia na Subproblemy i ze względu na strukturę klasy Problem zwraca ten sam obiekt, ale już z wypełnioną listą Subproblemów (czyli wszystkie podproblemy, na które został podzielony zadany Problem). Metoda StickTogetherSolvedSubproblems przyjmuje objekt Problem i zwraca już rozwiązany Problem, który ma status Finished (ponieważ wszystkie Subproblemy z listy SubProblemów tego obiektu były już rozwiązane i wykonano sklejenie wszystkich rozwiązań w całość).

Problem

Obiekty tej klasy mają najdłuższy cykl życia. Tworzone są w komponencie Computational Client i przesyłane dalej. Po rozwiązaniu zadanego problemu obiekt odsyłany zostaje z powrotem do komponentu Computational Client, który go stworzył (zawiera np. kompletne rozwiązanie zadanego problemu lub komunikat o błędzie obliczeń). To właśnie ta klasa jest kontenerem dla częściowych problemów i ich rozwiązań. Klasa umożliwia serializację. Klasa przetrzymuje kolejkę priorytetową zawierającą wszytskieSubproblemy, na które został podzielony Problem lub też jest ona pusta, gdy Problem nie został jeszcze podzielony. Status informuje na jakim etapie rozwiązywania jest Problem. Metoda UpdateSolvedStatusCount powoduję zmianę wartości SolvedSubproblems oraz powoduje zamianę obiektu podproblemu, który został zgłoszony jako obliczony (z nieobliczonego na ten właśnie otrzymany, podmiana dokonywana jest w kolejce priorytetowej Subproblemów).

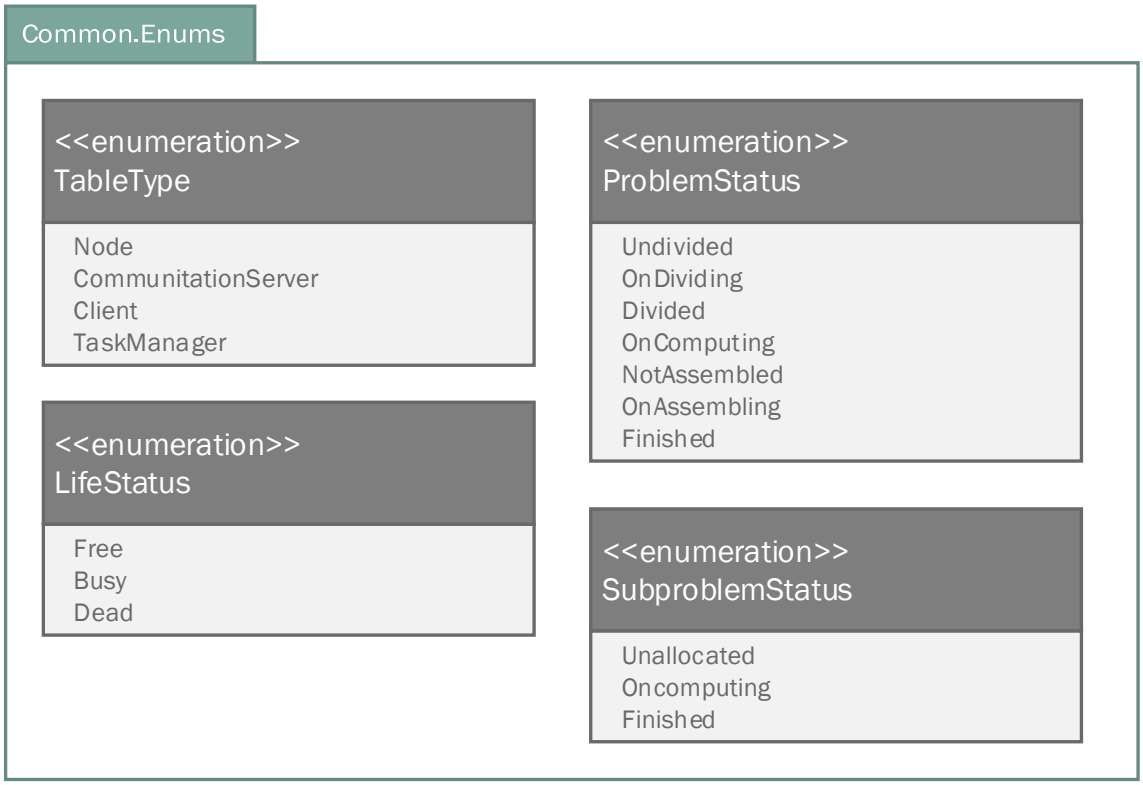
Subproblemy przetrzymywane są w kolejce priorytetowej, ponieważ zakładamy, że niektóre z nich mogą być ważniejsze i muszą zostać obliczone wcześniej niż inne.

Subproblem

Instancje tej klasy przetrzymywane są w obiektach klasy Problem. O tym, na jakim etapie znajduje się dane rozwiązanie częściowe możemy dowiedzieć się z pola Status. Dzięki takiemu rozwiązaniu wewnątrz jednego obiektu tej klasy przetrzymywać będziemy mogli obliczenia na różnym etapie zaawansowania.

Package Common.Enums

Pakiet ten zawiera wspólne dla pozostałych klas enumeratory umożliwiające wszystkim komponentom systemu używanie wspólnego języka mówiącego o stanie przesyłanych między nimi obiektów.



TableType

Określa typ tabeli. Wyliczenie to jest używane jako określenie typu argumentu funkcji ResponseSimpleTimeStamp w klasie CommunicationServer.

LifeStatus

Typ wyliczeniowy używany do zdefiniowania życia poszczególnych komponentów systemu.

ProblemStatus

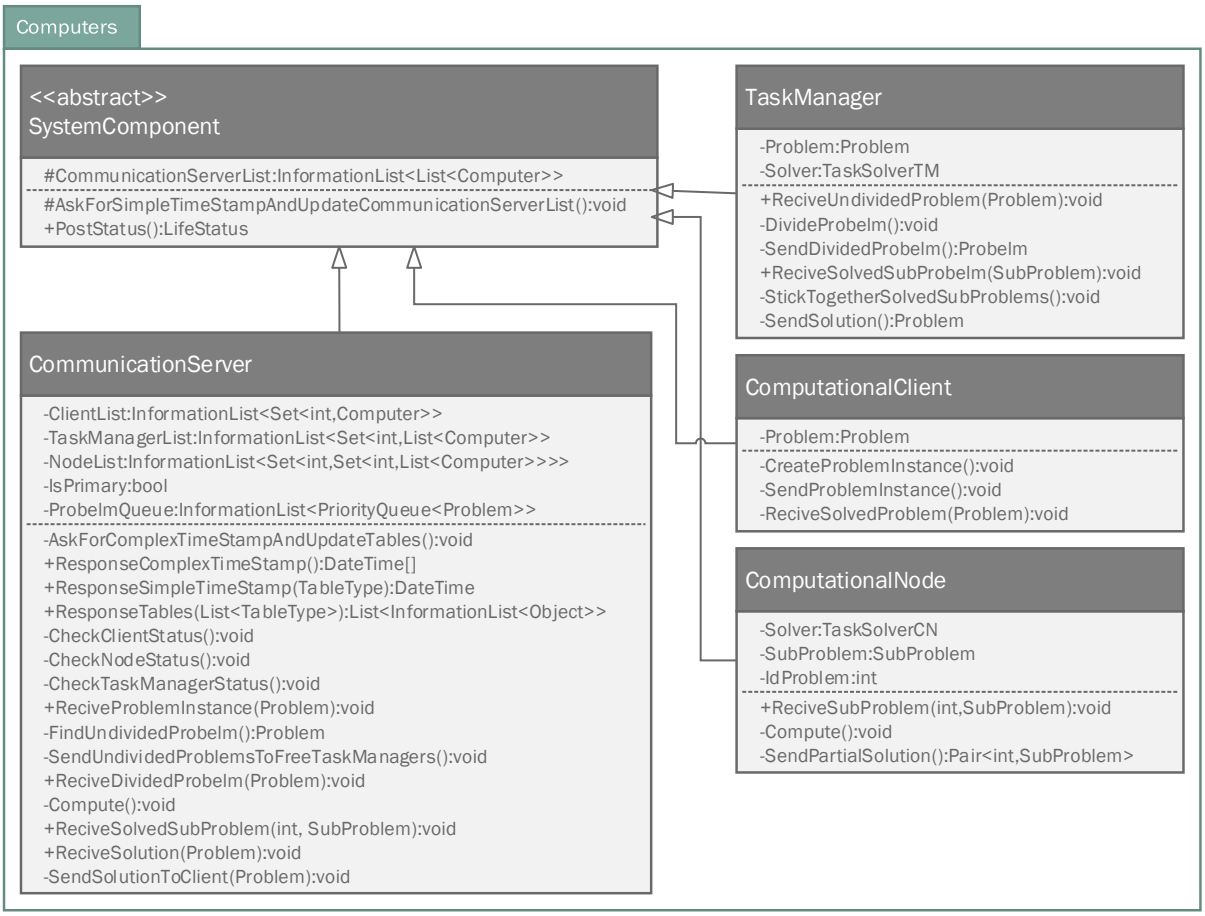
Używany jest w celu określenia stanu w jakim obecnie znajduje się problem.

SubproblemStatus

Enumerator wykorzystywany do określenia w jakim stanie znajduje się częściowe rozwiązanie problemu.

Package Computers

Pakiet ten zawiera komponenty odpowiedzialne za komunikację w całym systemie i zarządzanie instancjami Problemu będącymi na różnych etapach rozwiązywania.



SystemComponent

Klasa abstrakcyjna będąca klasą bazową dla wszystkich komponentów komunikujących się w systemie (CommunicationServer, ComputionalClient, ComputionalNode, TaskManager). Zawiera hierarchiczną listę wszystkich CommunicationServerów czyli tak na prawdę wszystkich serwerów (CS oraz BSC). Informuje o swoim statusie oraz potrafi wysłać zapytanie o najnowszą listę wszystkich serwerów w systemie.

CommunicationServer

Najważniejsza klasa do komunikacja w obrębie systemu. Odpowiedzialna jest za udzielanie wszystkich informacji dotyczących statusów komponentów oraz problemów. Zawiera listę wszystkich Clientów, TaskManagerów, Nodów oraz CommunicationServerów (czyli serwerów CS i BCS). Wszystkie wymienione listy są obiektami klasy InformationList zwierające zbiory informujące jakieIdProblemu ma przydzielony dany komponent (nie dotyczy CommunicationServerów). CommunicationServer posiada również kolejkę priorytetową Problemów, która umożliwia szybkie update'owanie instancji Problemów. Wyżej wymienione listy posiadają swoje Timestampy informujące o czasie ostatniej aktualizacji listy. Obiekty tej klasy posiadają metody sprawdzające statusy wszystkich komponentów z wyżej wymienionych list, odpowiadają również na prośby wysłania aktualnych InformationList'ów, potrafią przesyłać i odbierać instancje SubProblemów oraz Problemów, update'ując na bieżąco wszystkie listy. Odpowiadają również za odebranie instancji

Problemu od Clienta i odesłanie mu rozwiązania lub informacji o błędach związanych z rozwiązywaniem Problemu.

ComputionalClient

Klasa odpowiadająca Clientowi. Obiekty tej klasy posiadają instancje Problemu do rozwiązania, łączą się z CommunicationServerem wysyłając mu instancję swojego Problemu czekającego na rozwiązanie. Obiekty ComputionalClient otrzymują rozwiązanie lub informacje o błędach od CommunicationServera.

ComputionalNode

Obiekty tej klasy odpowiedzialne są za przesyłanie instancji Subproblemu do TaskSolverów (przeznaczonych do wyliczania pojedynczych instancji odpowiadających typowi Problemu zgodnemu z typem Problemu otrzymanego Subproblemu). Komponent te otrzymuje od swojego TaskSolver'a rozwiązanySubproblem lub też informację o błędach i przesyłają te dane dalej do CommunicationServera. Każdy ComputionalNode zawiera identyfikator Problemu oraz obiekt klasy SubProblem, którym aktualnie się zajmuje.

TaskManager

Obiekty tej klasy zarządzają przydzielonej im instancji Problemu. Potrafią przekazać niepodzielony Problem do TaskSolvera (przeznaczonego do dzielenia problemów tego samego typu) oraz otrzymać podzielony już obiekt lub też informację o błędach. Na bieżąco otrzymuje rozwiązane Subproblemy i zapisuje je. Gdy już pojawią się wszystkie Subproblemy obiekt Solver ma za zadanie skleić ostateczne rozwiązanie. Po otrzymaniu kompletnego wyniku TaskManager wysyła rozwiązanie Problemu, który został mu przydzielony lub informacje o błędach do CommunicationServera.

Opracowane algorytmy

Algorytm koordynacji podproblemów w komunikacji z komponentami Computational Node

1. Wszystkie podproblemy na początku oflagowane są jako nierozwiązane i nieprzydzielone.
2. Wszystkie Node'y na początku oflagowane są jako bezczynne.
3. Wybieramy pierwszego bezczynnego Node'a przydzielonego do tego problemu.
    a. W przypadku nie znalezienia - czekaj.
4. Wybieramy pierwszy nieprzydzielony i nierozwiązany podproblem.
    a. W wypadku nieznalezienia znajdź pierwszy przydzielony i nierozwiązany problem.
    b. W przypadku nie odnalezienia niezakończonego podproblemu, odeślij rozwiązania do TM oraz zarejestruj Node'y jako bezczynne i wyślij im żądanie zakończenia obliczeń związanych z tym problemem.
5. Wysyłamy wybranemu Node'owi wybrany podproblem, oflaguj podproblem jako przypisany a Node'a jako zajęty.
6. Powróć do punktu 3.
7. W trakcie całego procesu:
    a. Każdy podproblem, który został przydzielony ma timeout. Rozróżniamy trzy sytuacje:
        i. Otrzymujemy rozwiązanie - Podproblem zostaje oflagowany jako rozwiązany (nawet jeśli jest oznaczony jako nieprzydzielony), timeoutu już nie liczymy.
            1. Jeśli został rozwiązany już wcześniej to poprzedniego rozwiązania nie nadpisujemy.
        ii. W trakcie timeoutu otrzymaliśmy LifeStatusReport dotyczący tego podproblemu - resetujemy timeout.
        iii. W trakcie timeoutu nie otrzymaliśmy żadnego LifeStatusReport dotyczącego tego podproblemu - przenosimy podproblem na koniec listy i flagujemy go jako nieprzydzielony.

Załączniki

Następujące przykładowe pliki komunikacyjne xml oraz opisujące je XML schema'y:

- ACK.xml oraz .xsd
- LifeStatusReport.xml oraz .xsd
- PostProblem.xml oraz .xsd
- PostProblemInternal.xml oraz .xsd
- PostProblemResponse.xml oraz .xsd
- RequestNodeClient.xml oraz .xsd
- RequestResponse.xml oraz .xsd
- RequestServerTables.xml oraz .xsd
- RequestSolution.xml oraz .xsd
- ResponseComplexTimestamp.xml oraz .xsd
- ResponseSimpleTimestamp.xml oraz .xsd
- SendSubproblems.xml oraz .xsd
- SolutionResponse.xml oraz .xsd
- TerminateCalculations.xml oraz .xsd
- TimestampRequest.xml oraz .xsd

Dodatkowe pliki PDF:

- Stany elementów systemu.pdf