Computational Cluster

Katarzyna Węgiełek
Paweł Własiuk
Kamil Sienkiewicz
Marcin Wardziński

18 stycznia 2015

Spis treści

Rozdział 1

Wstęp

Tematem projektu jest stworzenie dokumentacji dla klastra obliczeniowego, który będzie umożliwiał rozwiązywanie problemów o dużej złożoności. System składa się z *serwerów komunikacyjnych*, *menadżerów zadań*, *klientów* i *węzłów obliczeniowych*.
Rolą *serwera* jest zapewnienie komunikacji pomiędzy wszystkimi komponentami. Elementy systemu wymieniają informacje wykorzystując protokół XML. W klastrze może być kilka serwerów. Jeden z nich jest serwerem głównym - zarządza przekazywaniem komunikatów i nadzoruje stan wszystkich elementów systemu. Pozostałe pełnią rolę pomocniczą - na nich znajdują się kopie zapasowe częściowych i ostatecznych rozwiązań problemów, zadań oczekujących na rozwiązanie i informacji o trwających obliczeniach. W przypadku awarii serwera głównego, jeden z serwerów pomocniczych przejmuje jego zadania. Wykonywaniem kopii danych zajmuje się serwer główny. On też informuje menadżerów zadań, klientów oraz węzły obliczeniowe o istnieniu serwerów pomocniczych, aby w przypadku awarii możliwa była ich zamiana.
Zadaniem *aplikacji klienckiej* jest pobranie danych wejściowych od użytkownika i wysłanie do serwera komunikacyjnego zlecenia rozwiązania podanego problemu. Gdy obliczenia zostaną zakończone, klient odbiera od serwera rozwiązanie zadania i prezentuje otrzymane wyniki użytkownikowi.
Menadżer zadań zajmuje się organizacją obliczeń i uzyskaniem ostatecznego wyniku z rozwiązań cząstkowych. Po otrzymaniu problemu od serwera komunikacyjnego, dzieli go na mniejsze podproblemy, przeznaczone do rozwiązania przez pojedyncze węzły obliczeniowe. Następnie wysyła je do serwera i oczekuje na wyniki częściowe. Na ich podstawie oblicza rozwiązanie końcowe i

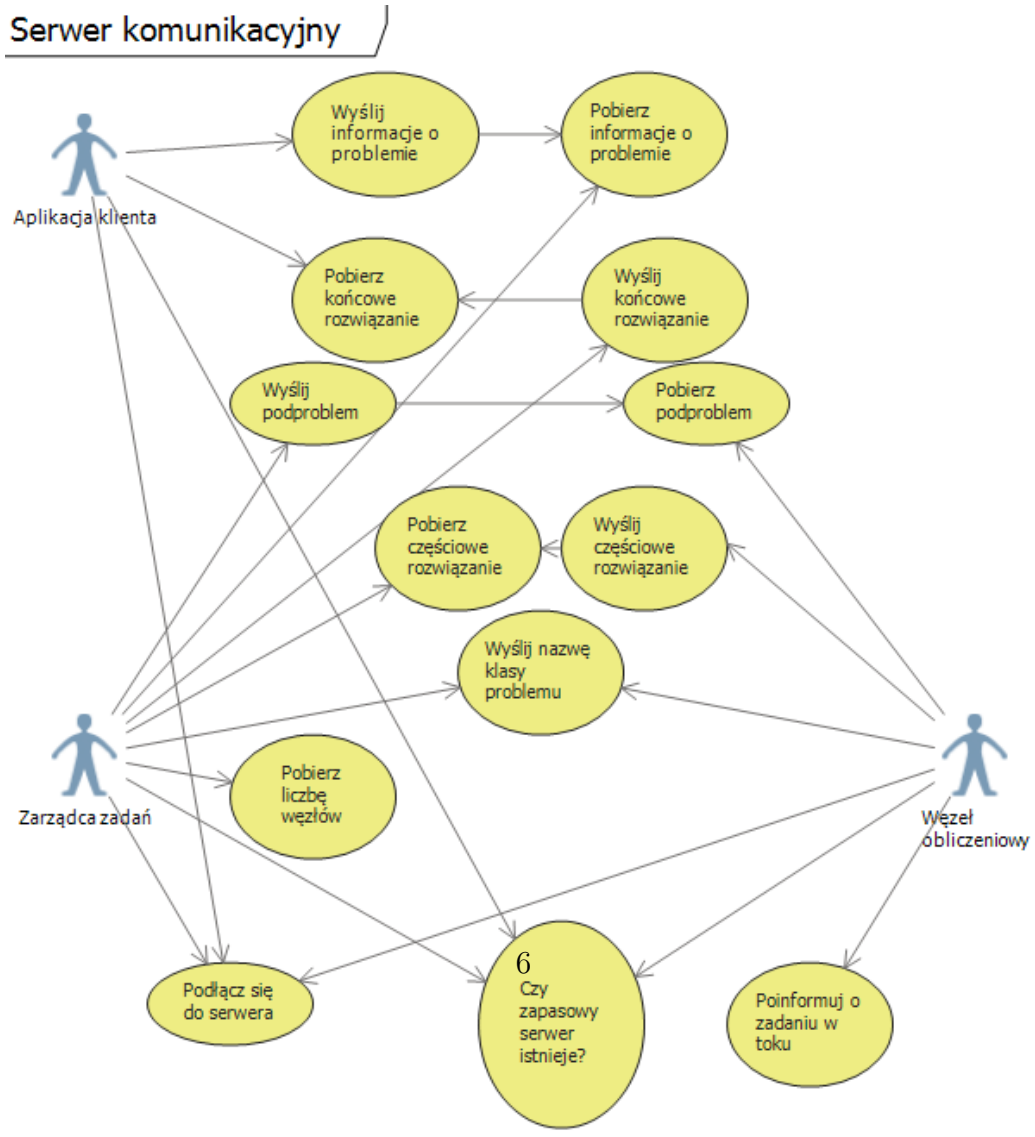
przesyła je na serwer. Każdy menadżer zadań zawiera jeden lub kilka modułów, odpowiedzialnych za rozwiązanie konkretnego typu problemu. Wysyła on do serwera nazwy typów problemów, jakie potrafi obsłużyć, dzięki czemu serwer wie, które zadania może do niego kierować.
Zadaniem *węzła obliczeniowego* jest rozwiązywanie podproblemów otrzymanych od serwera komunikacyjnego. Węzeł nie ma dostępu do informacji o całym zadaniu i nie zna wyników uzyskanych przez inne węzły. Otrzymuje jedynie problem częściowy, wyznaczony przez menadżera zadań, rozwiązuje go i wysyła do serwera wyniki swoich obliczeń. Węzeł składa się z jednego lub więcej modułów, zawierających algorytmy do rozwiązywania konkretnych typów podproblemów. Zanim otrzyma zadanie, wysyła do serwera informację dotyczącą rodzajów problemów, które potrafi rozwiązać. Podczas wykonywania algorytmu węzły na bieżąco informują serwer, że obliczenia trwają. W przypadku awarii węzła lub przerwania połączenia, do serwera przestaną dochodzić komunikaty i będzie on mógł skierować dany problem do innego węzła, aby uzyskać brakujący wynik częściowy.
Przykładowym zadaniem, które można rozwiązać korzystając z opisanego klastra obliczeniowego jest problem marszrutyzacji (Dynamic Vehicle Routing Problem).

Rozdział 2

Diagramy Przypadków Użycia

2.1 Komunikacja

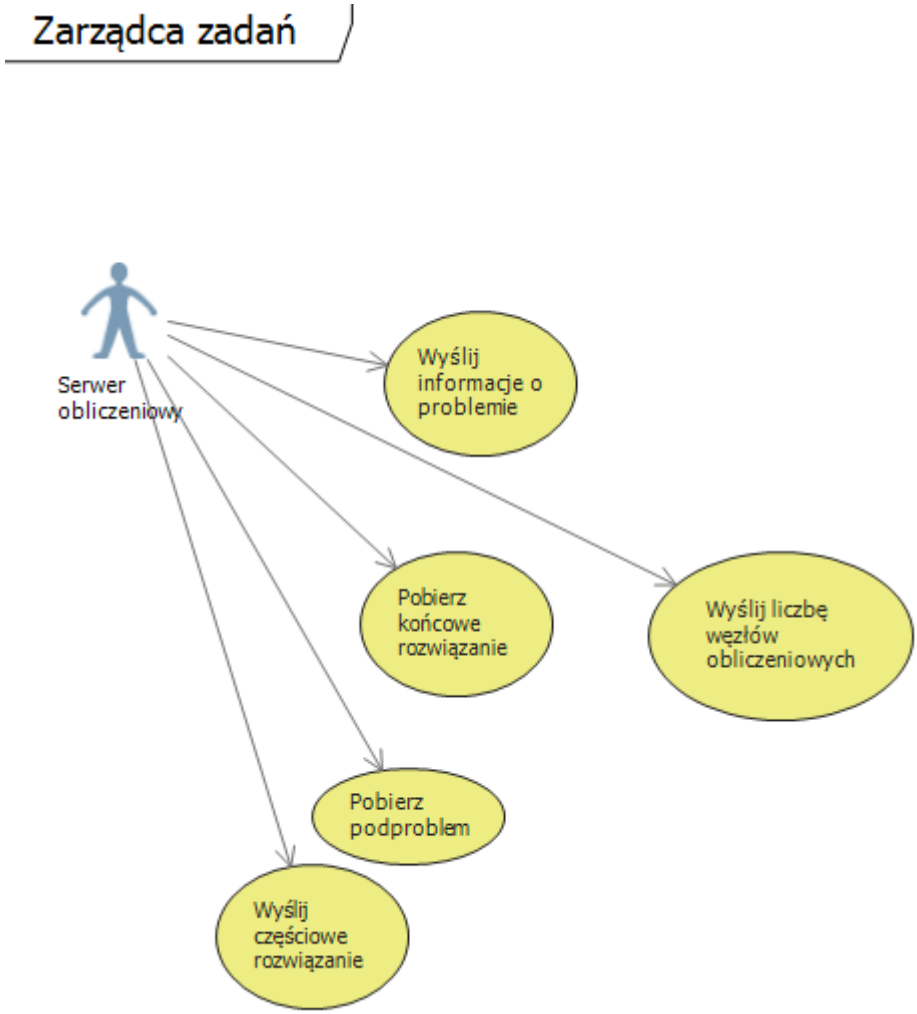
2.1.1 Komponenty z serwerem



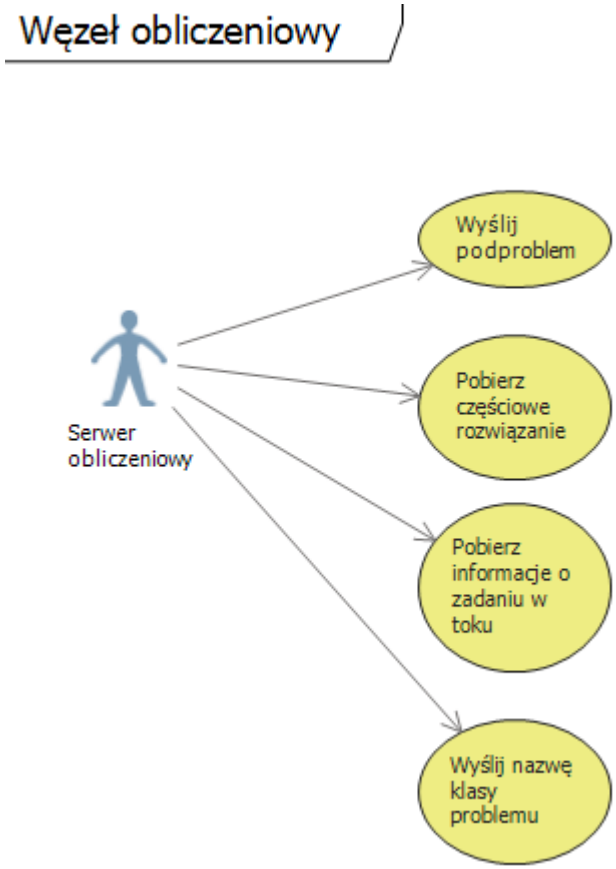
Cała komunikacja musi odbywać się przez serwer, dlatego na początku swojego działania każdy komponent musi się z nim połączyć. Serwer pośredniczy w przesyłaniu informacji o problemie między klientem a Menadżerem Zadań, końcowego rozwiązania od Menadżera Zadań do Klienta, jak również rozwiązań pośrednich między Węzłami Obliczeniowymi, a Menadżerem Zadań. Jego użycie jest również wymagane do przesyłania częściowych podproblemów między Menadżerem Zadań i Węzłami obliczeniowymi.

Do dodatkowych funkcji serwera, nie związanych bezpośrednio z komunikacją należy podanie liczby dostępnych Węzłów obliczeniowych Menadżerowi zadań oraz informowanie o istnieniu zapasowego serwera. Serwer może zażądać informacji o klasie problemu od Menadżera Zadań i Węzłów Obliczeniowych. Węzły obliczeniowe mają obowiązek informować serwer o tym, czy obliczenia nadal się odbywają.

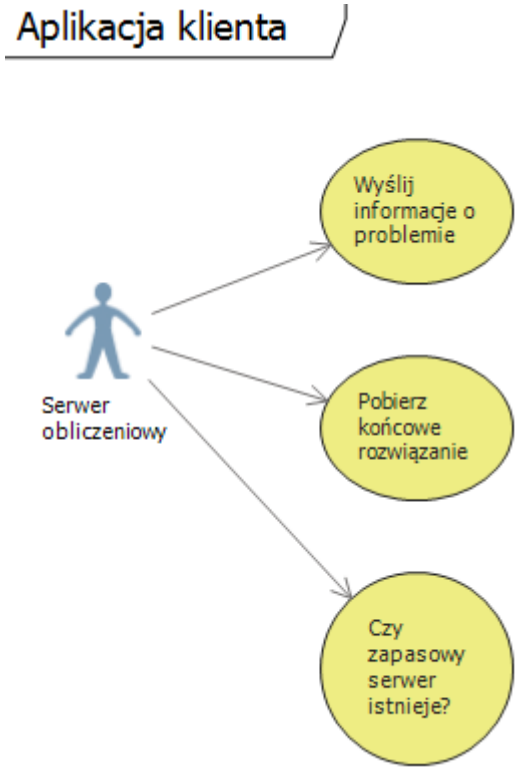
2.1.2 Serwer z Menadżerem Zadań



2.1.3 Serwer z Węzłem obliczeniowym



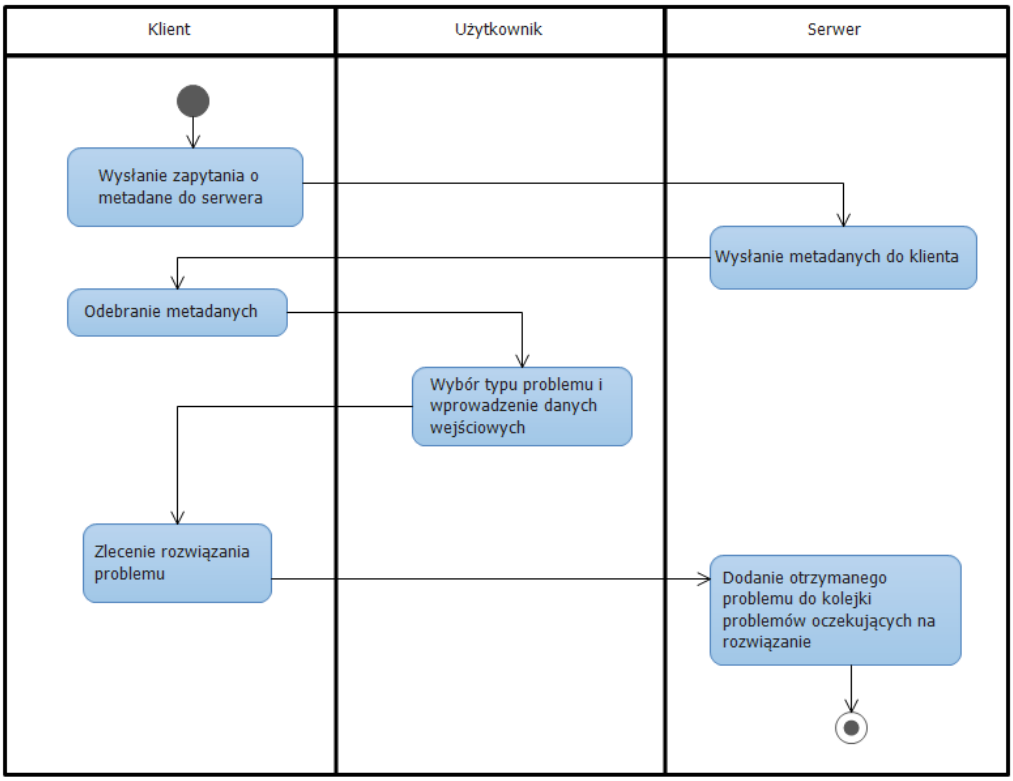
2.1.4 Serwer z Klientem



Rozdział 3

Diagramy Aktywności

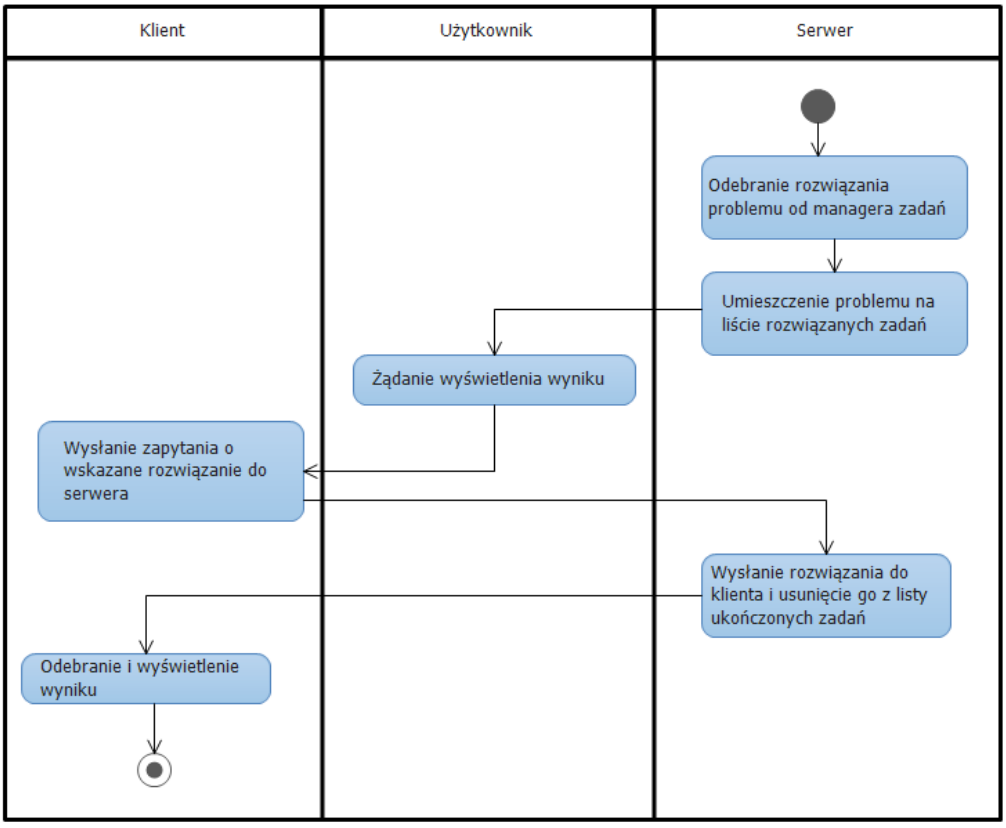
3.1 Zlecenie rozwiązania problemu



Aplikacja kliencka nawiązuje połączenie z serwerem głównym, używając adresu IP serwera zapisanego w pliku konfiguracyjnym. Wysyła zapytanie o metadane - adresy IP serwerów backup'owych oraz nazwy klas problemów,

które mogą być rozwiązane przez ten klaster obliczeniowy (takie, że istnieje przynajmniej jeden *menadżer zadań* potrafiący obsłużyć dany typ problemu). Serwer przesyła metadane do klienta. Następnie użytkownik wybiera spośród dostępnych nazw klas problemów typ zadania jakie ma zostać rozwiązane i wprowadza do programu wszystkie potrzebne dane wejściowe. Aplikacja kliencka wysyła do serwera zlecenie rozwiązania problemu i podane przez użytkownika dane w formacie XML. Serwer odbiera zlecenie przysłane przez klienta i umieszcza problem w kolejce problemów danego typu oczekujących na rozwiązanie. Zadanie znajduje się w kolejce, dopóki któryś z *menadżerów zadań*, potrafiących rozwiązać problem tej klasy, nie zakończy obliczeń i nie będzie mógł się nim zająć.

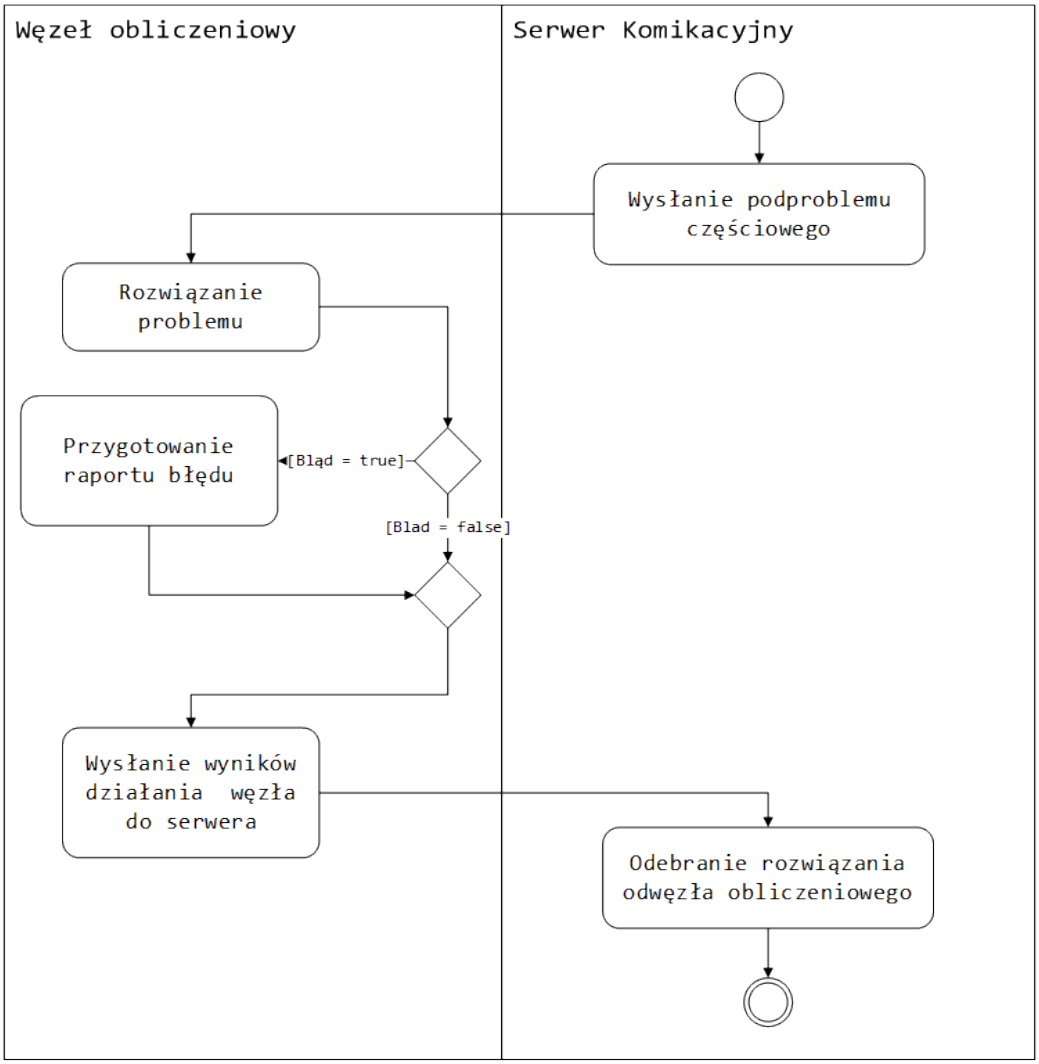
3.2 Odczytanie wyniku



Po otrzymaniu ostatecznego rozwiązania od *menadżera zadań*, serwer umieszcza problem na liście ukończonych, ale jeszcze nieodczytanych rozwiązań.

Następnie użytkownik wybiera z listy problem, którego rozwiązanie chce zobaczyć. Aplikacja kliencka wysyła do serwera żądanie pobrania wskazanego wyniku. Serwer wysyła rozwiązanie odpowiedniego problemu do klienta i usuwa je z listy ukończonych zadań. Klient odbiera wyniki i wyświetla je użytkownikowi.

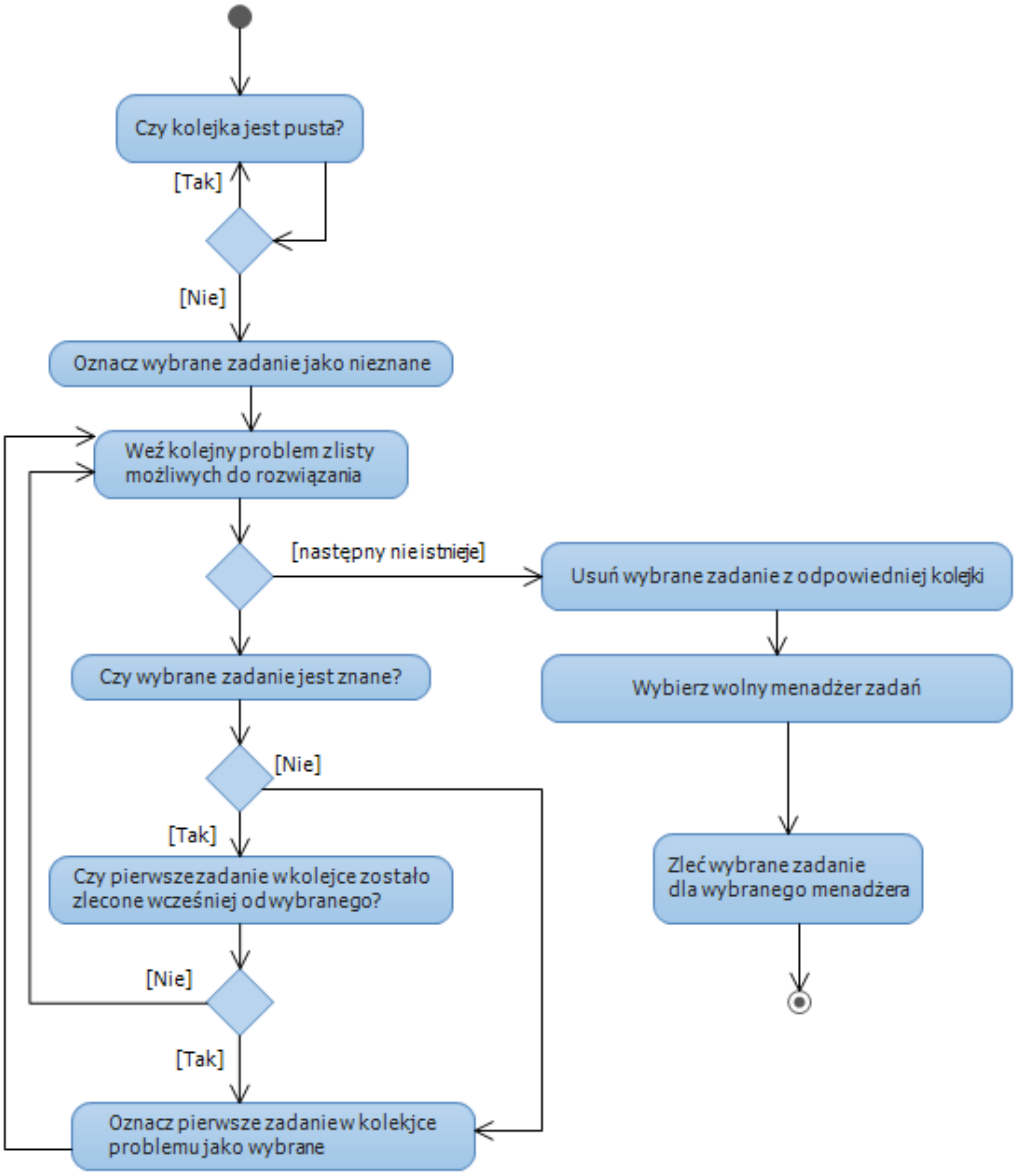
3.3 Węzeł obliczeniowy



Węzeł obliczeniowy przy uruchomieniu zgłasza swoją obecność serwerowi komunikacyjnemu. Informacje temat serwera znajdują się w pliku konfiguracyjnym węzła. Węzeł wysyła do serwera informacje na temat typów proble-

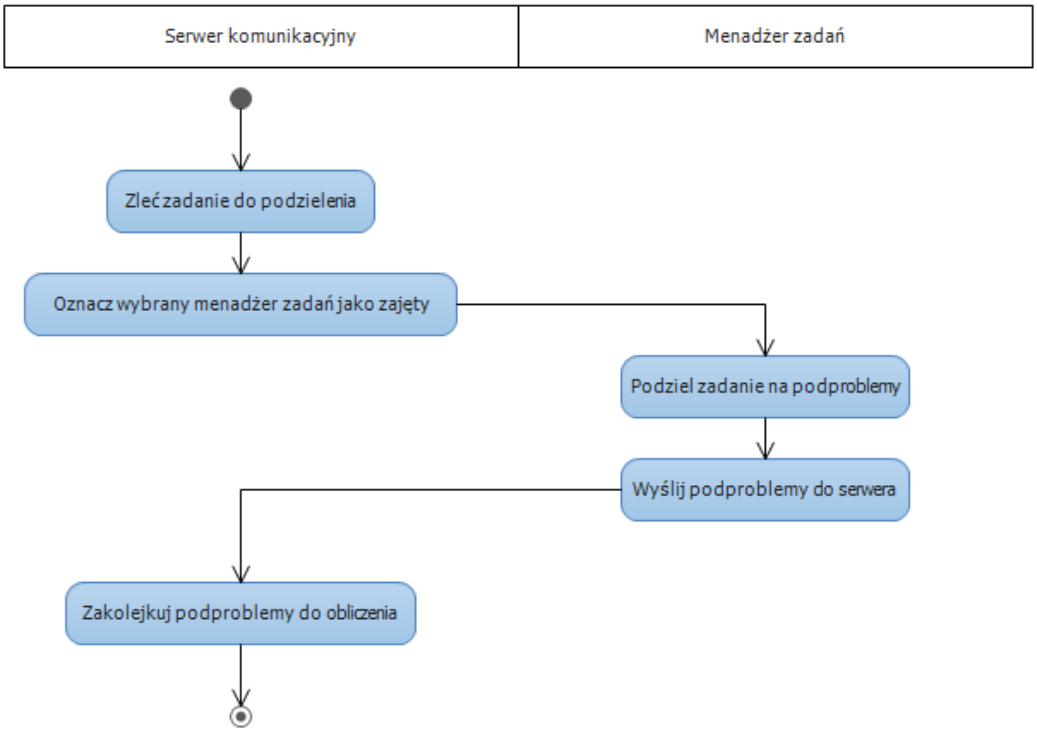
mów które jest w stanie rozwiązać, dzięki temu serwer komunikacyjny może uwzględniać go przy przesyłaniu do **Menadżera Zadań** informacji na temat ilości węzłów potrafiących rozwiązać dany typ problemu. Węzeł obliczeniowy nie wykonujący w danym momencie obliczeń otrzymuje od serwera skolejkowany podproblem, podzielony na części przez **Menadżer Zadań**. Zadaniem węzła obliczeniowego jest rozwiązanie otrzymanego zadania oraz przesłanie rozwiązania do serwera. Jeżeli podczas wykonywania obliczeń wystąpi błąd, węzeł ma za zadanie przygotować raport o błędach a następnie wysłać go do serwera.

3.4 Wybieranie zadania do podzielenia



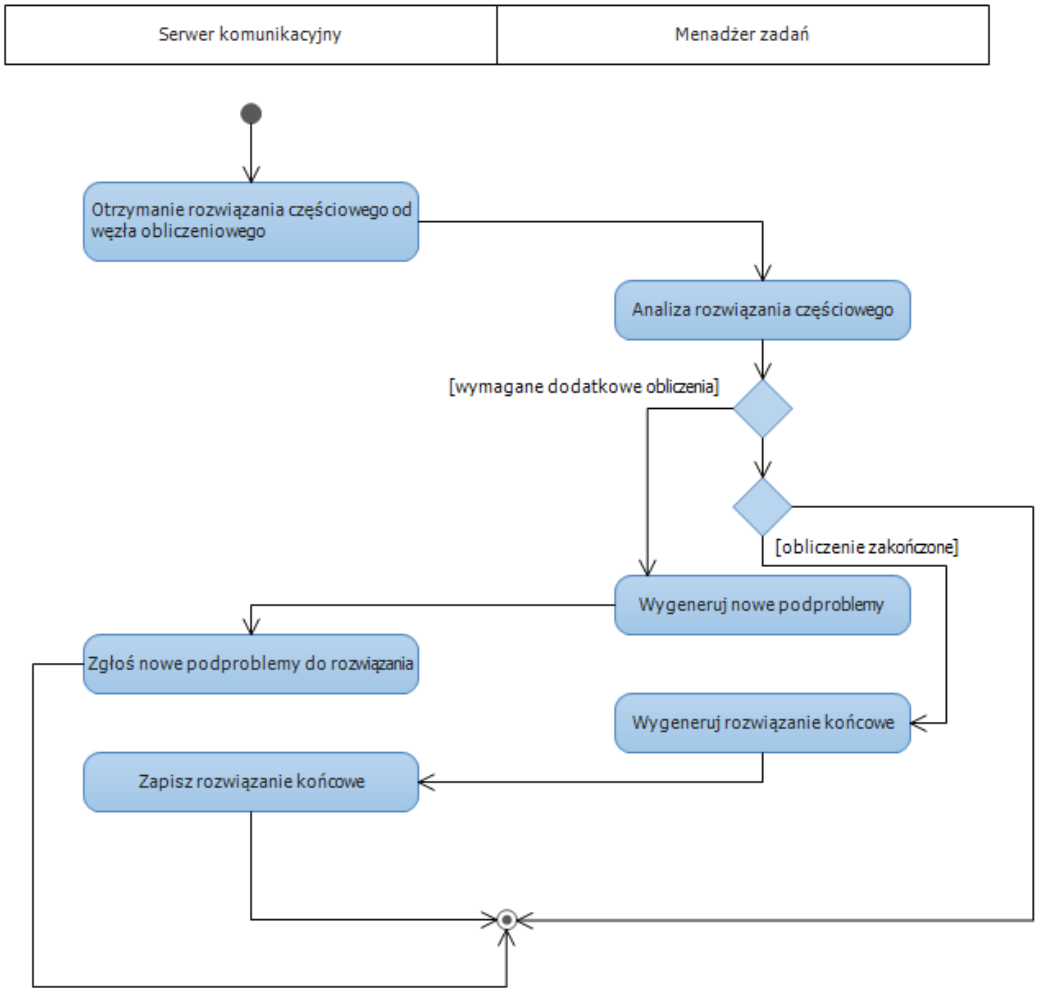
W celu przeprowadzania procesu dzielenia problemów na podproblemy w sposób optymalny z zachowaniem kolejności potrzebne było opracowanie specjalnej struktury danych do ich przechowywania. (Gdzie opisać tą strukturę? Tutaj to się nie nadaje. Najlepiej byłoby to umieścić oddzielnie.)

3.5 Podział zadania na podproblemy



Wybrane zadanie do przetworzenia poprzez **serwer komunikacyjny** trafia do wolnego **menadżera zadań** potrafiącego rozwiązać dany problem. Po podzieleniu go na podproblemy, **menadżer zadań** odsyła definicje podproblemów z powrotem do **serwera komunikacyjnego**, który umieszcza je na innej kolejce do późniejszego obliczenia.

3.6 Zgłaszanie wyniku częściowego do menadżera zadań

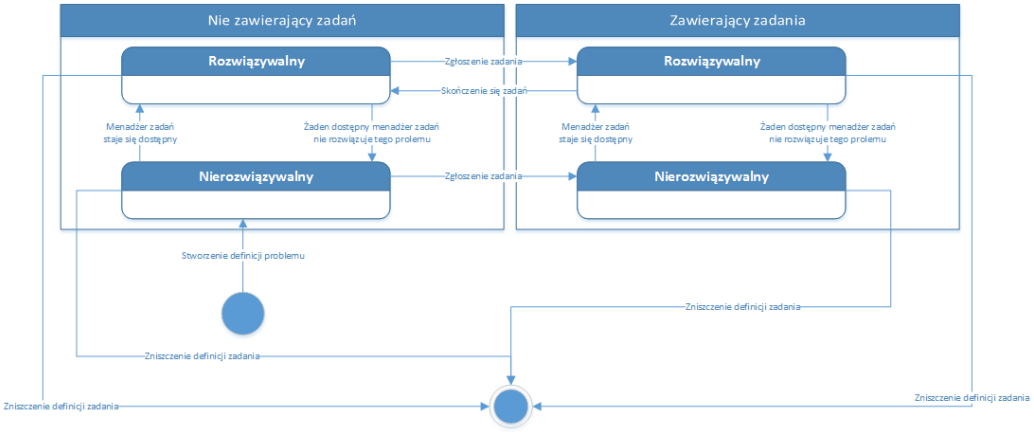


Zaraz po otrzymaniu wyniku częściowego od **węzła obliczeniowego serwer komunikacyjny** odsyła go do **menadżera zadań** który zajmuje się daną instancją problemu. **Menadżer zadań** po analizie rozwiązania częściowego może wygenerować nowe podproblemy, które zostaną zgłoszone analogicznie jak w przypadku początkowego podziału zadania. Może również oznaczyć problem jako rozwiązany i odesłać pełne rozwiązanie do późniejszego pobrania przez użytkownika.

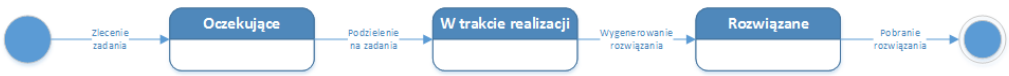
Rozdział 4

Diagramy stanów

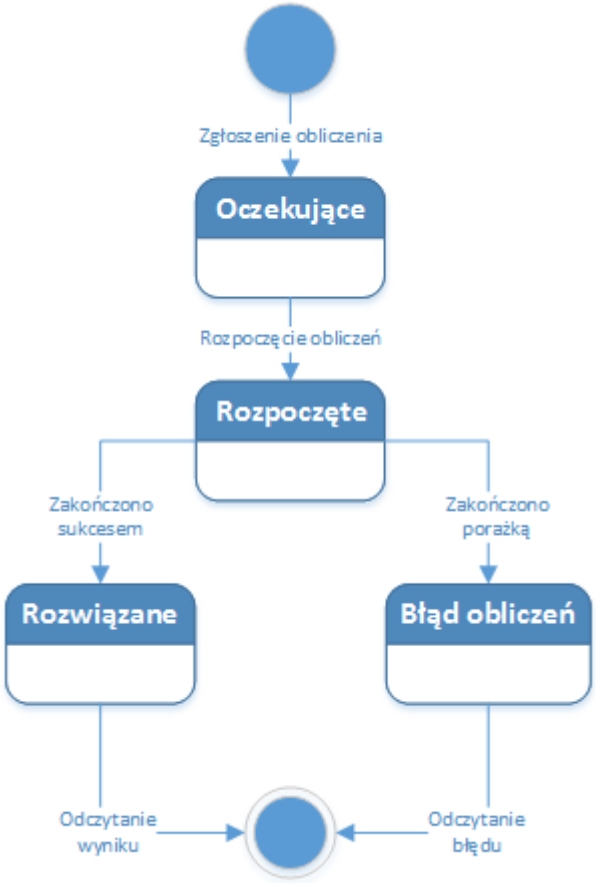
4.1 Rodzaj zadania



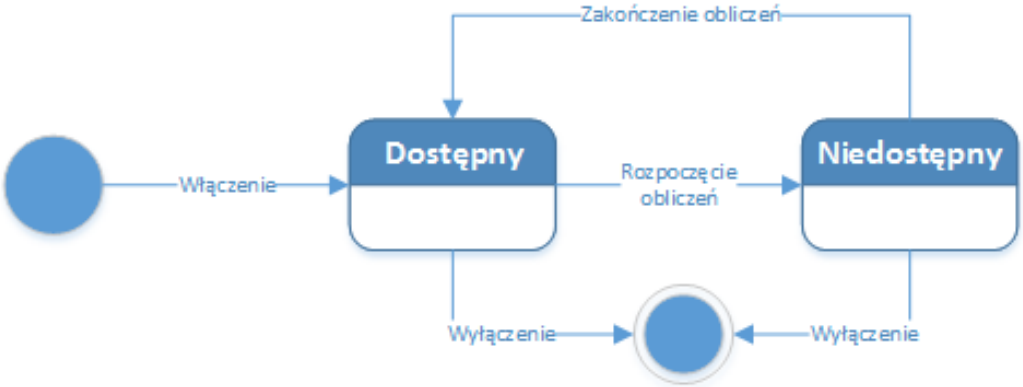
4.2 Zadanie



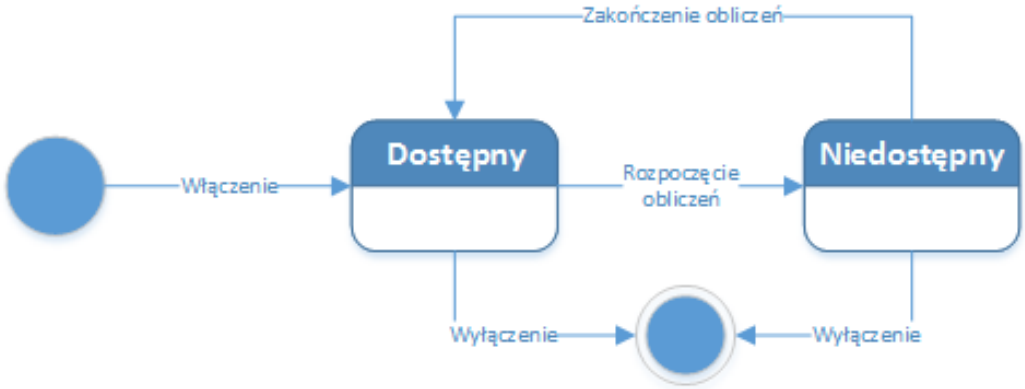
4.3 Obliczenie



4.4 Menadżer zadań



4.5 Węzeł obliczeniowy



Rozdział 5

Komunikacja

5.1 Lista rozwiązywalnych problemów

Proces rozwiązywania zadania przez *klaster obliczeniowy* rozpoczyna się na poziomie aplikacji klienckiej. Przed wysłaniem problemu do rozwiązanie, aplikacja kliencka musi pobrać z serwera informację na temat klas problemów rozwiązywalnych w danej chwili przez klaster obliczeniowy (wynika ona z obecnie dostępnych *Menadżerów zadań* i *węzłów obliczeniowych*). Aplikacja wysyła do *serwera* zapytanie postaci:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema" messageGuid="70c61dd5-afcb-452f-9ad4
    -621a0ef78e6c">
3   <MessageType>ProblemsListRequest</MessageType>
4 </Message>
```

examples.xml

gdzie wartość węzła `MessageType` (definiującego typ wiadomości) ustawiona jest na `ProblemsListRequest`.

Serwer w odpowiedzi na to żądanie odsyła wiadomość, w której zawarte są informacje o wszystkich typach problemów rozwiązywalnych przez klaster. Informacje zostają przetworzone i przedstawione użytkownikowi.

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

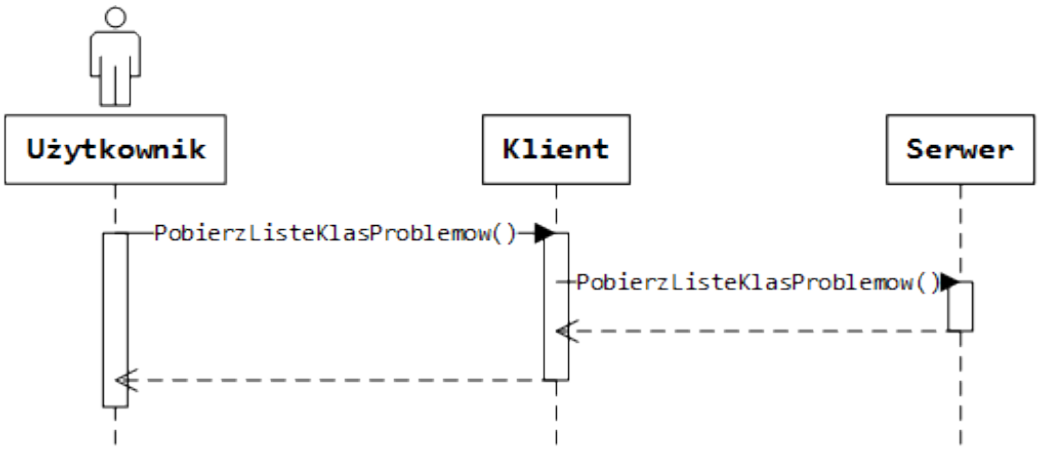
```xml
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema" messageGuid="70c61dd5-afcb-452f-9ad4
    -621a0ef78e6c">
  <MessageType>ProblemsList</MessageType>
  <ProblemClassList>
    <ProblemClass problemClassName="name1" problemClassId="ee28fec2-6361-4
        a58-aaf1-b9ff0f509743" />
    <ProblemClass problemClassName="name2" problemClassId="f9ed0a8f-a9f1
        -494a-aea6-68ffc533934e" />
  </ProblemClassList>
</Message>
```

examples.xml

Przedstawiony powyżej fragment kodu to przykładowa odpowiedź serwera. Dla każdego typu problemu zostają przekazane dwie informacje:

- `problemClassName` - nazwa problemu,
- `problemClassId` - guid problemu, który w jednoznaczny sposób identyfikuje typ problemu.

Ten etap komunikacji przedstawi poniższy diagram:



5.2 Rozwiązywania zadania

Po wykonaniu czynności opisanych w poprzednim rozdziale, użytkownik może zlecić zadanie *klastrowi obliczeniowemu* Aplikacja kliencka wysyła do serwera wiadomość z informacją o typie rozwiązywanego problemu, jak też danych

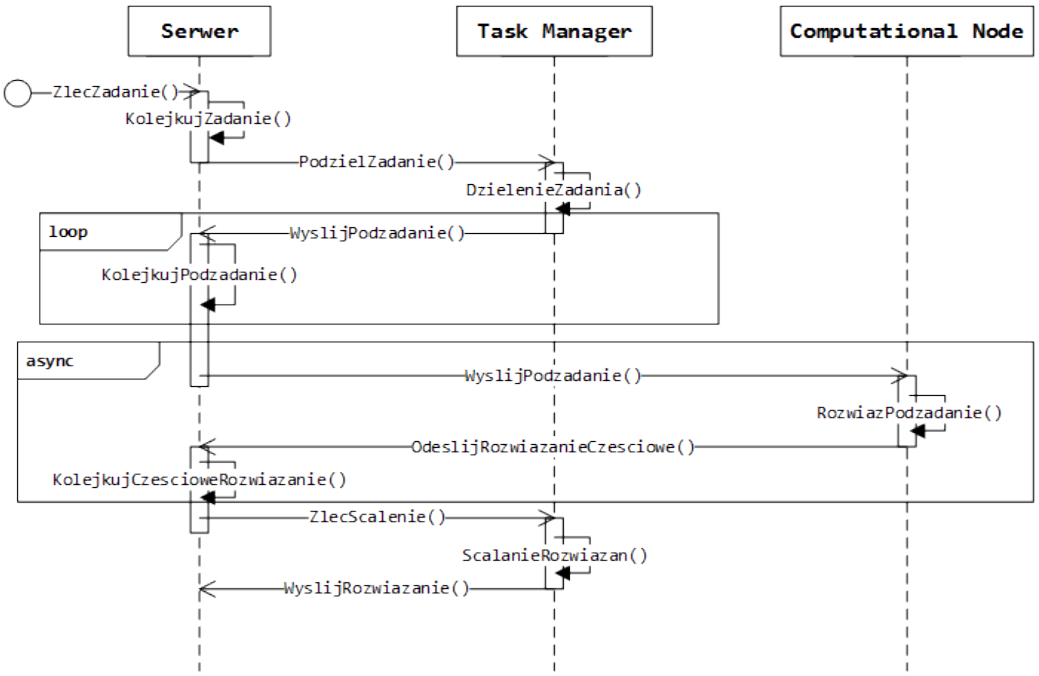
wejściowych. Przykładem takiej wiadomości jest poniższy przykład:

```
<?xml version="1.0" encoding="UTF-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema" messageGuid="70c61dd5-afcb-452f-9ad4
    -621a0ef78e6c">
  <MessageType>TaskOrder</MessageType>
  <problemClassId>4fbcbba1-014e-4643-b60f-f7888a95bb54</problemClassId>
  <Data>dane w postaci XML</Data>
  <TaskName>CustomTaskName</TaskName>
</Message>
```

examples.xml

Wiadomość zawiera 4 parametry konieczne do stworzenia, rozwiązania i zidentyfikowania zadania przez użytkownika:

- `MessageType` - parametr ustawiony na `TaskOrder` informuje serwer o typie wiadomości,
- `problemClassId` - identyfikator problemu, umożliwiający rozwiązanie go przed odpowiednie części systemu,
- `Data` - parametry typu `String` zawierający dane wejściowe dla danego typu problemu. Dane te przedstawione są w formacie XML specyficznym dla danego typu problemu.
- `TaskName` - nazwa zadania zdefiniowana przez klienta, umożliwi w przyszłości wyszukanie rozwiązania.



Klaster obliczeniowy wykonuje ciąg czynności koniecznych do otrzymania rozwiązania, tzn:

(a) przesłanie polecenia podziału zadania od *serwera* do *menadżera zadań*,

(b) podział zadania i odesłanie stworzonych podproblemów prowadzących do otrzymania rozwiązania,

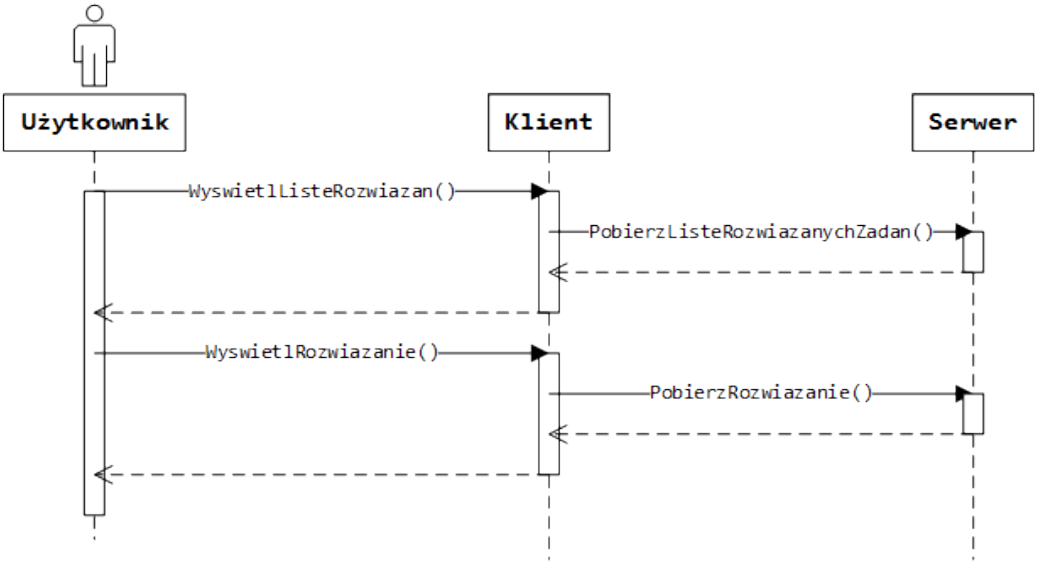
(c) przesłanie każdego z podzadań do *węzłów obliczeniowych*

(d) przeprowadzenie obliczeń i odesłanie częściowych rozwiązań do *serwera*

(e) przesłanie częsciowych rozwiązań do *menadżera zadań* w celu stworzenia rozwiązania

(f) odesłanie pełnego rozwiązania zadania do *serwera*

5.3 Odczytanie rozwiązania



Proces odczytania rozwiązania jest podobny do początkowego etapu obliczania. Tym razem *aplikacja kliencka* musi poprosić *serwer* o listę rozwiązanych zadań, wyświetlić je i umożliwić użytkownikowi wybór typu konkretnego zadania. Razem z lista rozwiązanych problemów serwer przesyła:

- `name` - nazwa zadania (wcześniej zdefiniowana przez użytkownika),
- `id` - identyfikator zadania, wykorzystywany jest przy zapytaniu o rozwiązanie konkretnego zadania,
- `problemClassName` - typ klasy problemu

Następnie zostaje przesłana prośba o pobranie rozwiązania dla wybranego zadania. Rozwiązanie zostaje przesłanie w postaci `stringu` w formacie XML specyficznym dla danego typu problemu. Poniżej przedstawione zostały dwa inne dwa przykłady komunikatów

- lista rozwiązanych zadań przesyłana do klienta

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns
      :xsd="http://www.w3.org/2001/XMLSchema" messageGuid="70c61dd5-
      afcb-452f-9ad4-621a0ef78e6c">
3   <MessageType>TaskSolvedList</MessageType>
4   <TasksList>
```

```
    <Task name="CustomName1" id="8844d38c-ac3a-4774-9e81-957a708c6f0b
        " problemClassName="className1" />
    <Task name="CistomName2" id="d104742a-76bb-4942-92bb-9f92149f4863
        " problemClassName="className2" />
  </TasksList>
</Message>
```

examples.xml

- rozwiązanie przesłane do klienta

```
<?xml version="1.0" encoding="UTF-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns
    :xsd="http://www.w3.org/2001/XMLSchema" messageGuid="70c61dd5-
    afcb-452f-9ad4-621a0ef78e6c">
  <MessageType>TaskSolved</MessageType>
  <problemClassId>4fbcbba1-014e-4643-b60f-f7888a95bb54</
      problemClassId>
  <problemId>d104742a-76bb-4942-92bb-9f92149f4863</problemId>
  <Data>rozwiazaniew postacie XML</Data>
  <TaskName>CustomTaskName</TaskName>
</Message>
```

examples.xml

5.4 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="guid">
    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
          ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA-F
          ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="messageTypes">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ProblemsListRequest"/>
      <xs:enumeration value="ProblemsList"/>
      <xs:enumeration value="TaskOrder"/>
```

```
14      <xs:enumeration value="TaskSolvedListRequest"/>
15      <xs:enumeration value="TaskSolvedList"/>
16      <xs:enumeration value="TaskSolved"/>
17      <xs:enumeration value="TaskSolvedRequest"/>
18      <xs:enumeration value="SubtaskOrder"/>
19      <xs:enumeration value="SubtaskSolved"/>
20      <xs:enumeration value="NodeError"/>
21    </xs:restriction>
22  </xs:simpleType>
23
24  <xs:element name="Message">
25    <xs:complexType>
26      <xs:sequence >
27        <xs:element type="messageTypes" name="MessageType"/>
28        <xs:element type="guid" name="problemClassId" minOccurs="0"/>
29        <xs:element type="guid" name="problemId" minOccurs="0"/>
30        <xs:element type="xs:string" name="Data" minOccurs="0"/>
31        <xs:element name="ProblemClassList" minOccurs="0">
32          <xs:complexType>
33            <xs:sequence>
34              <xs:element name="ProblemClass" maxOccurs="unbounded"
                    minOccurs="0">
35                <xs:complexType>
36                  <xs:simpleContent>
37                    <xs:extension base="xs:string">
38                      <xs:attribute type="xs:string" name="problemClassName"
                            use="required"/>
39                      <xs:attribute type="guid" name="problemClassId" use="
                            required"/>
40                    </xs:extension>
41                  </xs:simpleContent>
42                </xs:complexType>
43              </xs:element>
44            </xs:sequence>
45          </xs:complexType>
46        </xs:element>
47        <xs:element type="xs:string" name="TaskName" minOccurs="0"/>
48        <xs:element name="TasksList" minOccurs="0">
49          <xs:complexType>
50            <xs:sequence>
51              <xs:element name="Task" maxOccurs="unbounded" minOccurs="0">
52                <xs:complexType>
53                  <xs:simpleContent>
```

```
54                    <xs:extension base="xs:string">
55                      <xs:attribute type="xs:string" name="name" use="
                            required"/>
56                      <xs:attribute type="guid" name="id" use="required"/>
57                      <xs:attribute type="xs:string" name="problemClassName"
                             use="required"/>
58                    </xs:extension>
59                  </xs:simpleContent>
60                </xs:complexType>
61              </xs:element>
62            </xs:sequence>
63          </xs:complexType>
64        </xs:element>
65      </xs:sequence >
66      <xs:attribute type="guid" name="messageGuid" use="required"/>
67    </xs:complexType>
68  </xs:element>
69 </xs:schema>
```

messageSchema.xsd