COMPUTATIONAL CLUSTER

Software Engineering
Fall 2015

Viet Ba Mai
Jakub Kamiński
Patryk Abramczyk
Andrzej Frankowski

I. Description

The goal of our project was to design the architecture and communication protocol for the **Computational Cluster** and its components, which later on could be developed and implemented. Its pattern allows us to divide complex problems into smaller parts, parallelly executed, hence makes the whole problem easier to solve.
Computational Cluster consists of *Communication server*, *Computational node*, *Computational client* and *Task manager*, each is runned by a person (an actor).

In the following sections we will present the design documentation and diagrams showing system activities.

II. Structure

Communication server
This is the main part of the system. It is responsible for communication between components of Computational Cluster. It receives partial problems and sends them to the Computational node. Afterwards it retrieves partial solutions and sends them to the Task manager. At the end it receives a complete solution and sends it to the Computational client. Communication server additionally creates a copy of its states to the backup and informs other components of existence of that backup.

Computational node
It is responsible for solving received partial problems and sending partial solutions to the Task manager through the server.

Computational client
It is an application through which a problem instance can be sent to the sever. It gets the final solution.

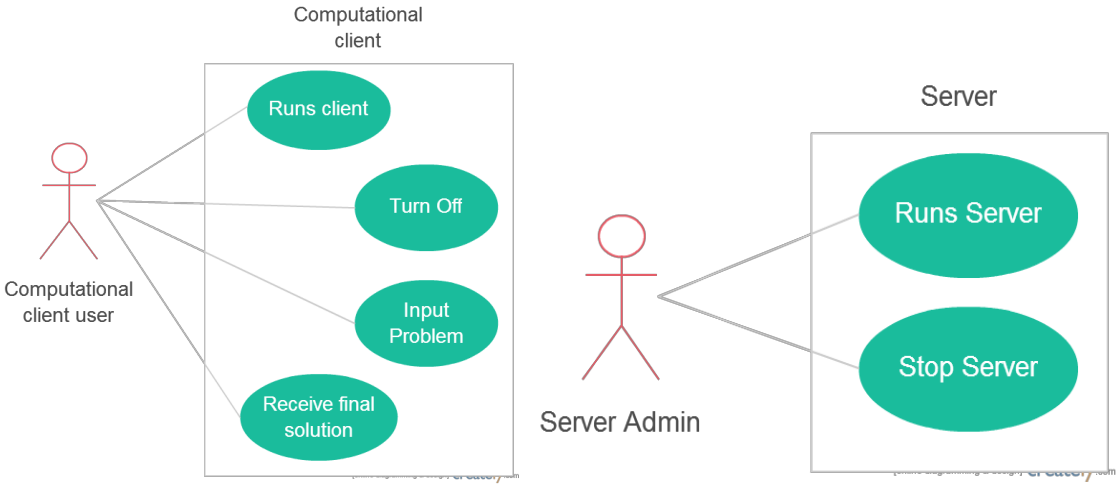
Task manager
It is responsible for dividing tasks into partial problems among Computational nodes. Afterwards, it receives partial solutions and chooses the best one which it then sends back to the Communication server.

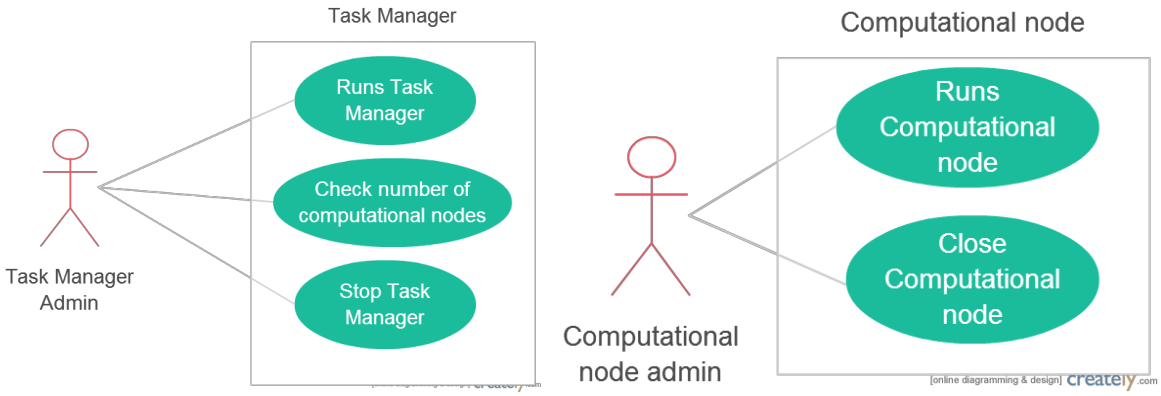
Task solver
It is a module managed by the Computational client to find partial solutions. It loads at runtime libraries for processing problems.

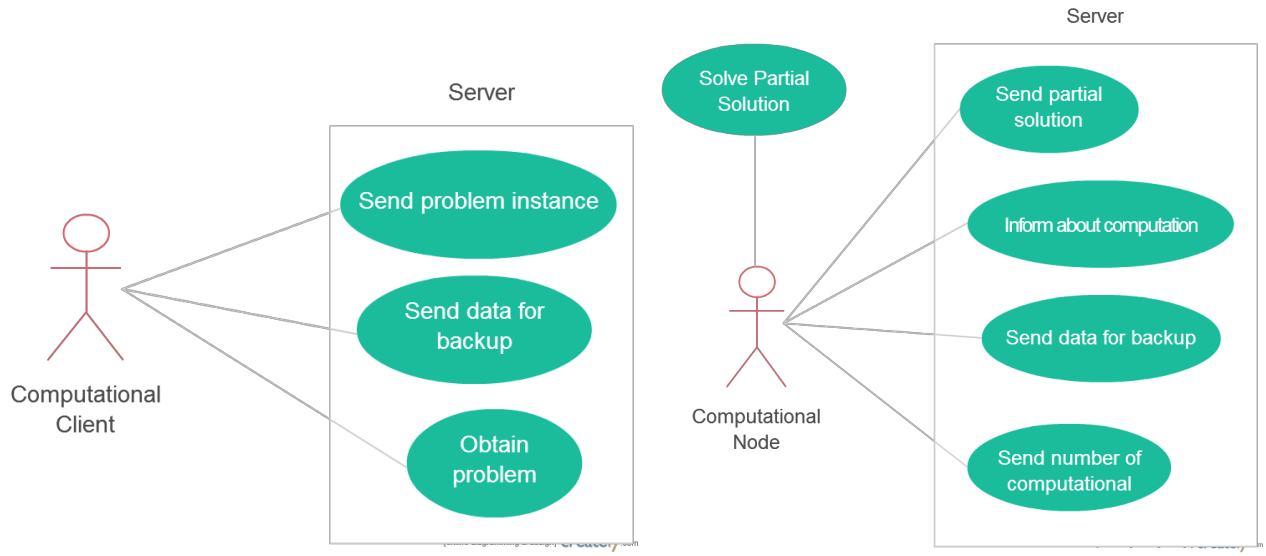
III. Diagrams

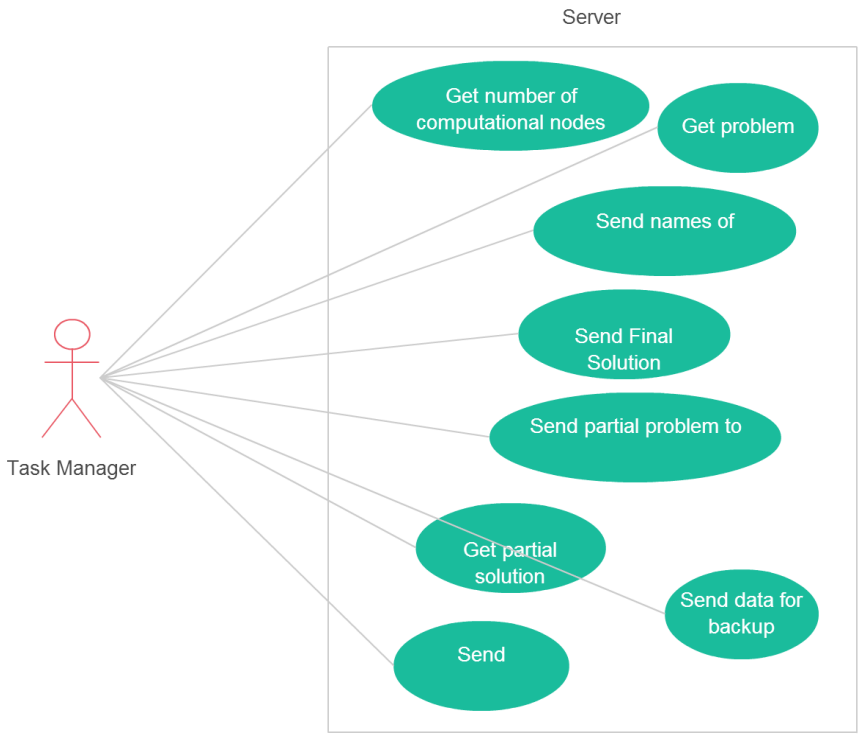
Remark: All included diagrams can be found in folder /diagrams as full resolution images.

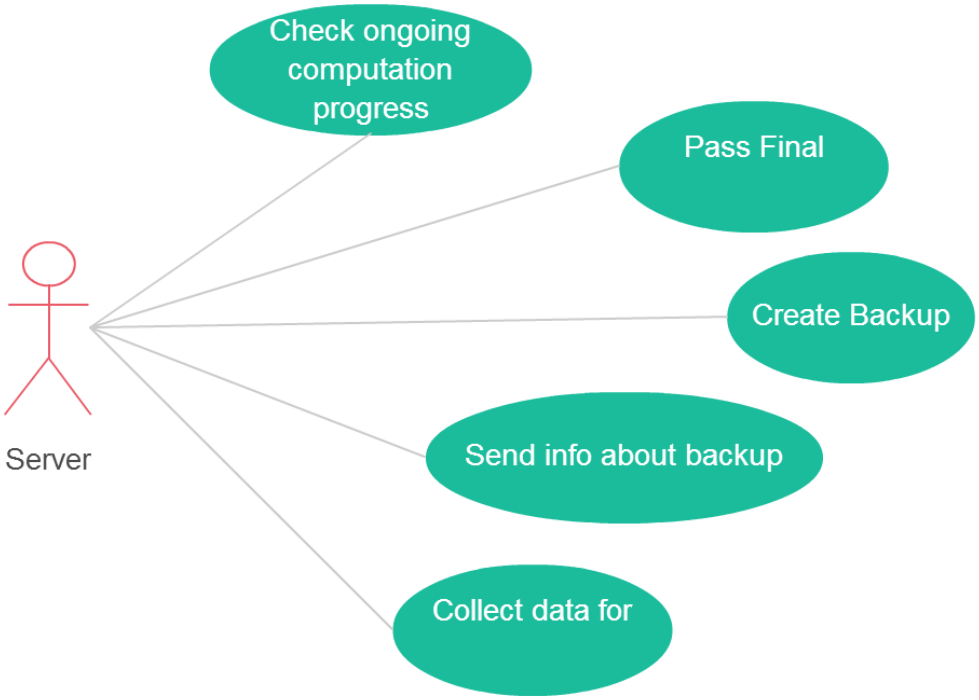
1. Use - case diagrams



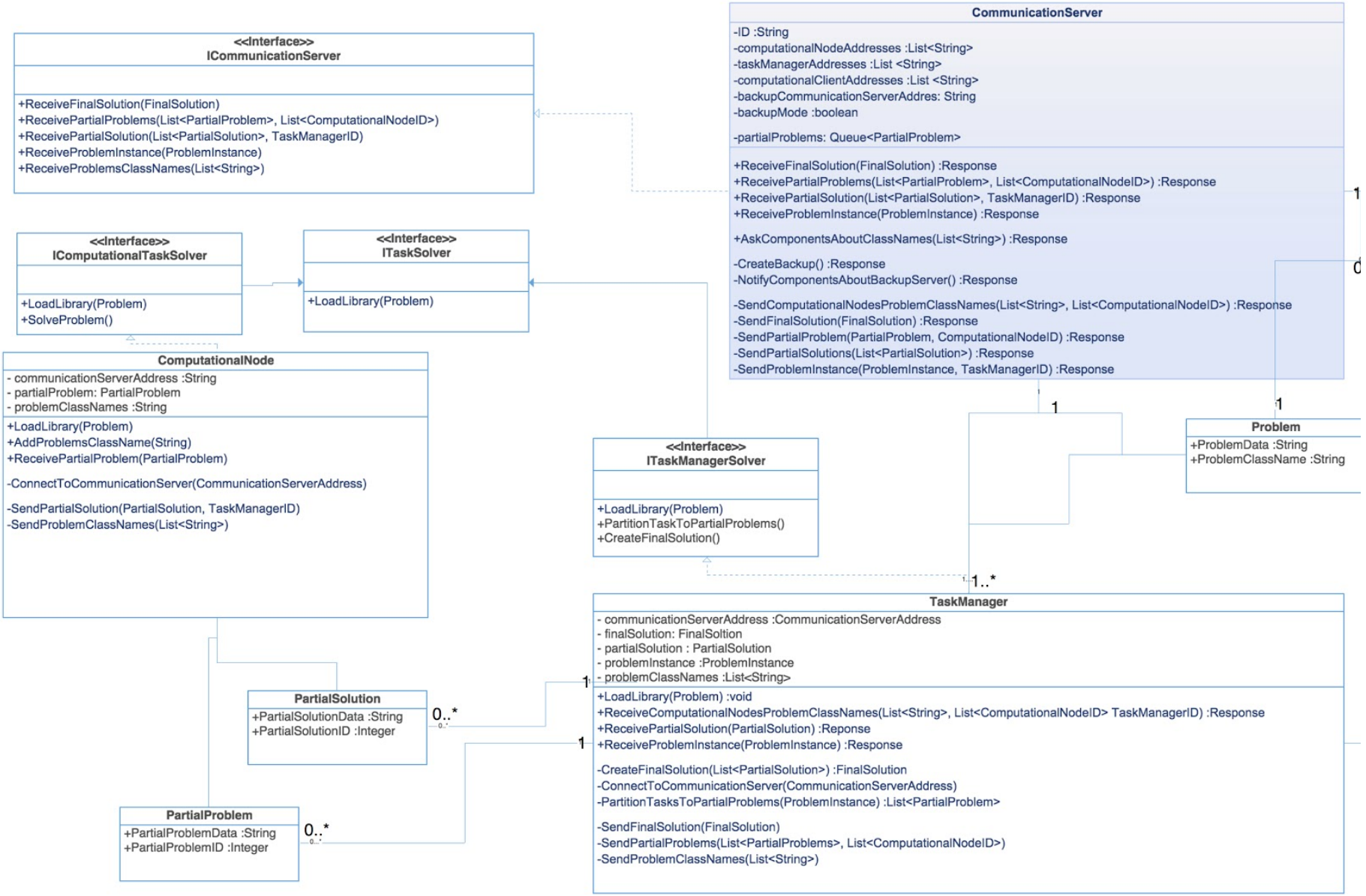


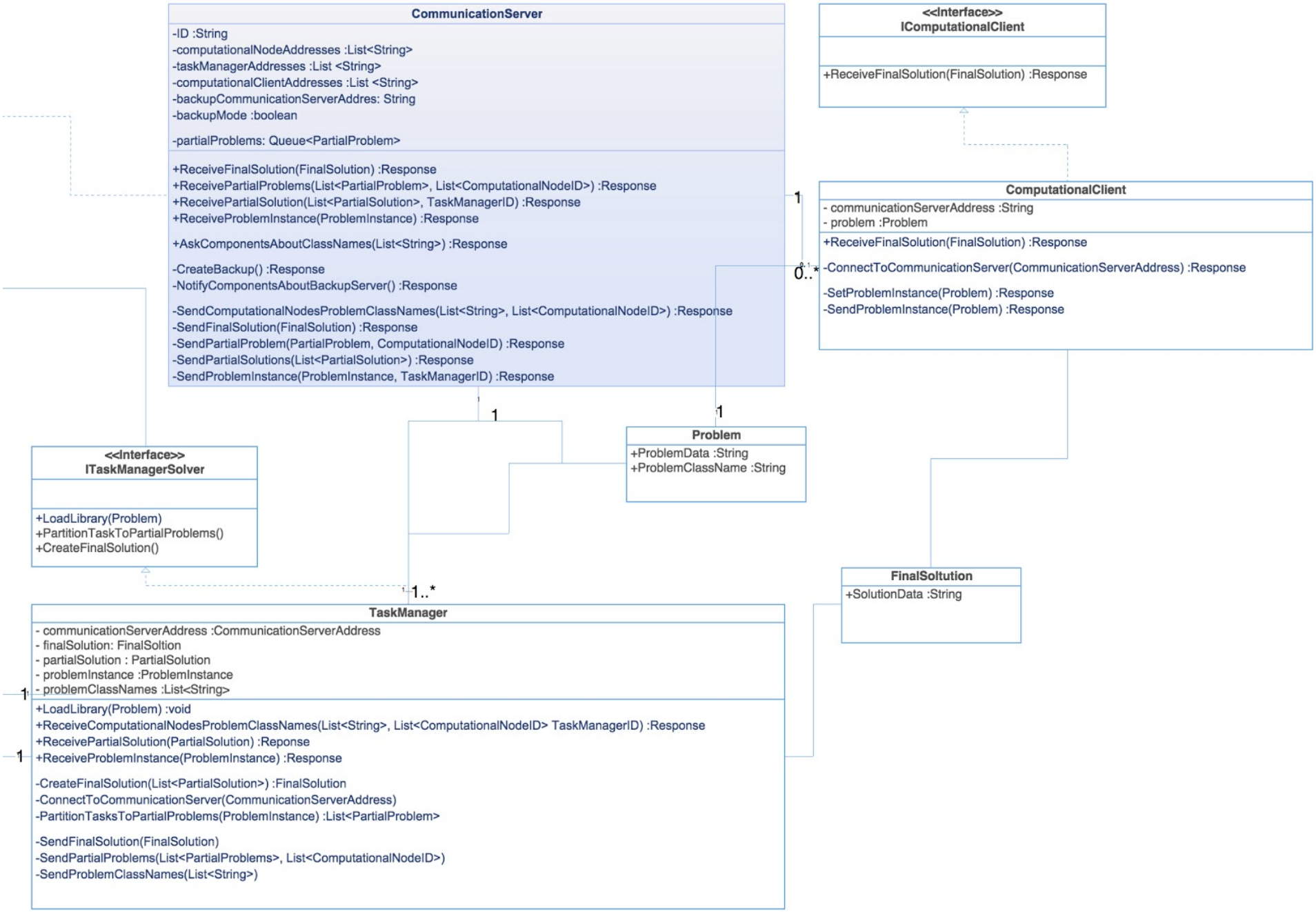



Server
Get number of computational nodes
Get problem
Send names of
Send Final Solution
Send partial problem to
Get partial solution
Send data for backup
Send
Task Manager


Check ongoing computation progress
Pass Final
Create Backup
Send info about backup
Collect data for
Server

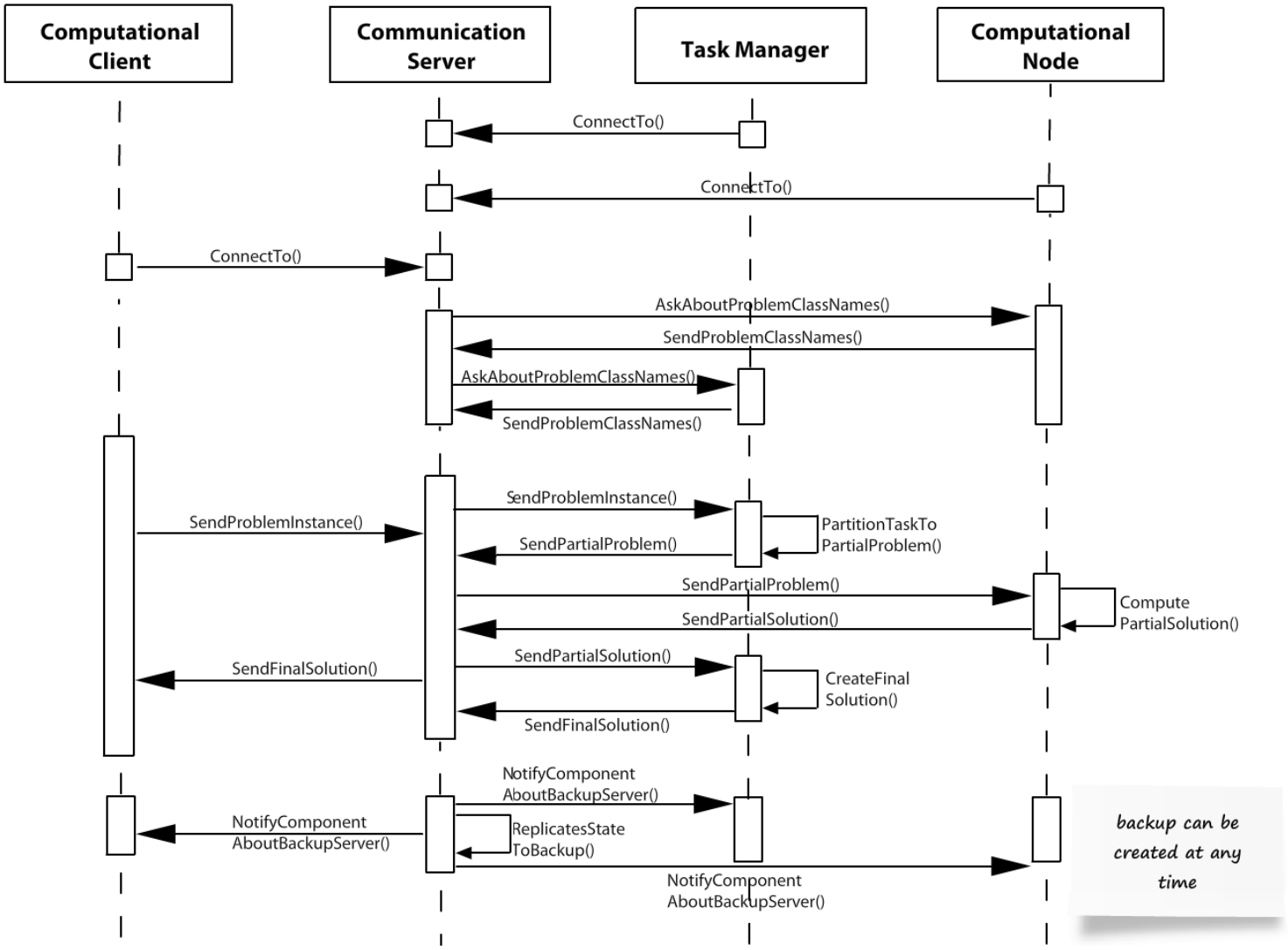
2. Class diagram




CommunicationServer
-ID :String
-computationalNodeAddresses :List<String>
-taskManagerAddresses :List <String>
-computationalClientAddresses :List <String>
-backupCommunicationServerAddres: String
-backupMode :boolean
-partialProblems: Queue<PartialProblem>
+ReceiveFinalSolution(FinalSolution) :Response
+ReceivePartialProblems(List<PartialProblem>, List<ComputationalNodeID>) :Response
+ReceivePartialSolution(List<PartialSolution>, TaskManagerID) :Response
+ReceiveProblemInstance(ProblemInstance) :Response
+AskComponentsAboutClassNames(List<String>) :Response
-CreateBackup() :Response
-NotifyComponentsAboutBackupServer() :Response
-SendComputationalNodesProblemClassNames(List<String>, List<ComputationalNodeID>) :Response
-SendFinalSolution(FinalSolution) :Response
-SendPartialProblem(PartialProblem, ComputationalNodeID) :Response
-SendPartialSolutions(List<PartialSolution>) :Response
-SendProblemInstance(ProblemInstance, TaskManagerID) :Response
<<Interface>>
IComputationalClient
+ReceiveFinalSolution(FinalSolution) :Response
ComputationalClient
- communicationServerAddress :String
- problem :Problem
+ReceiveFinalSolution(FinalSolution) :Response
-ConnectToCommunicationServer(CommunicationServerAddress) :Response
-SetProblemInstance(Problem) :Response
-SendProblemInstance(Problem) :Response
Problem
+ProblemData :String
+ProblemClassName :String
<<Interface>>
ITaskManagerSolver
+LoadLibrary(Problem)
+PartitionTaskToPartialProblems()
+CreateFinalSolution()
FinalSoltution
+SolutionData :String
TaskManager
- communicationServerAddress :CommunicationServerAddress
- finalSolution: FinalSoltion
- partialSolution : PartialSolution
- problemInstance :ProblemInstance
- problemClassNames :List<String>
+LoadLibrary(Problem) :void
+ReceiveComputationalNodesProblemClassNames(List<String>, List<ComputationalNodeID> TaskManagerID) :Response
+ReceivePartialSolution(PartialSolution) :Reponse
+ReceiveProblemInstance(ProblemInstance) :Response
-CreateFinalSolution(List<PartialSolution>) :FinalSolution
-ConnectToCommunicationServer(CommunicationServerAddress)
-PartitionTasksToPartialProblems(ProblemInstance) :List<PartialProblem>
-SendFinalSolution(FinalSolution)
-SendPartialProblems(List<PartialProblems>, List<ComputationalNodeID>)
-SendProblemClassNames(List<String>)
1
0..*
1
1
1
1..*
1
1

PartialSolution and *tialProblem* classes are only implemented in *ComputationalNode* and TaskManager components. In *CommunicationServer* property *Queue<PartialProblems>* is not a Queue of *PartialProblems*, but of XML data instead. It was written in this form for clarity. Queue of XML of *PartialProblems* is stored in case there are more *PartialProblems* than *ComputationalNodes*.
ITaskSolver is a generic interface that provides method for loading library for a specific problem. There are subclasses of this interface that provide more methods specialized for different jobs, e.g. *ITaskManagerSolver* which inherits from it and provides an interface for *TaskManager* with methods like CreateFinalSolution().

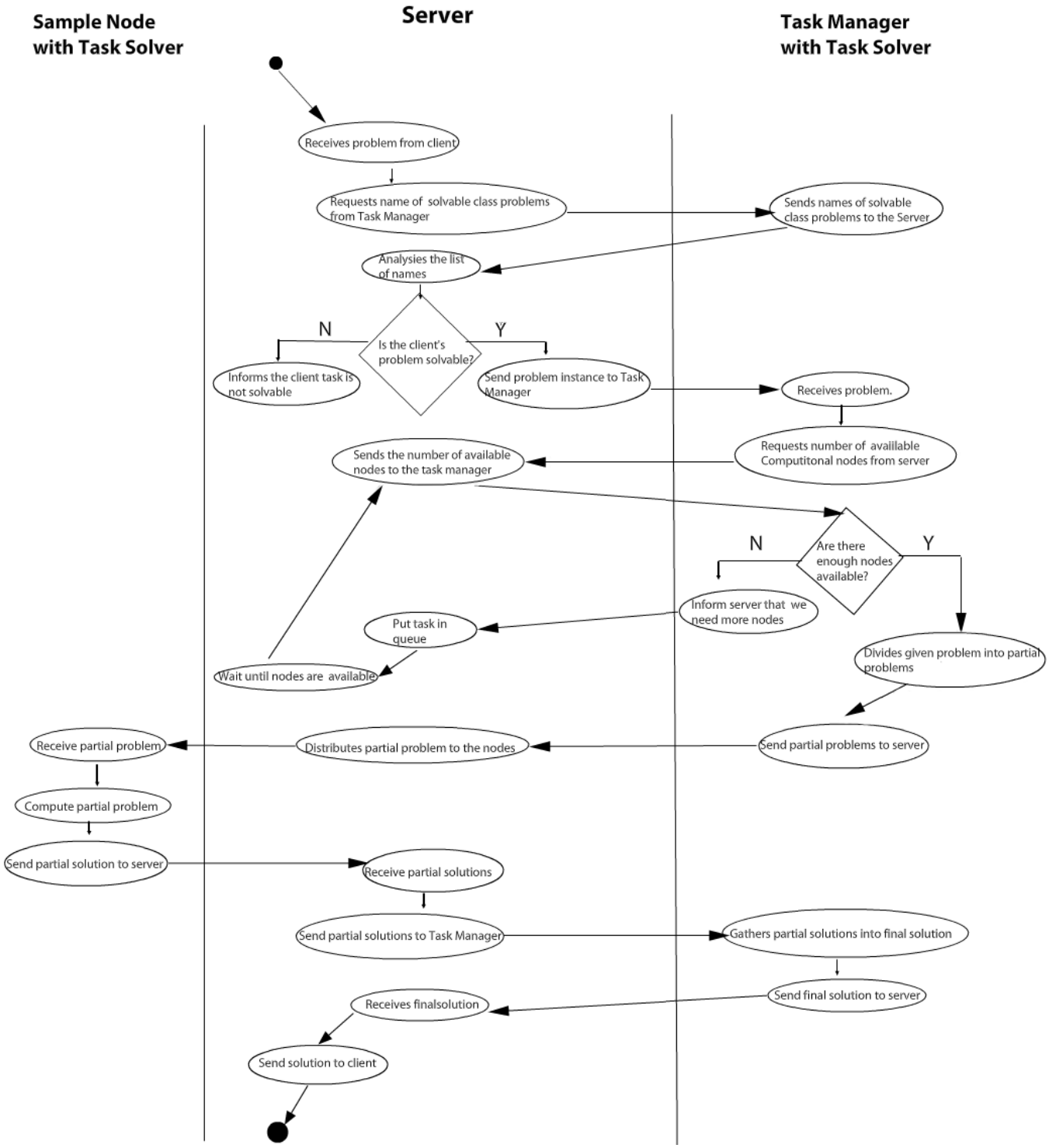
3. Sequence diagram



Workflow description:

1. Task manager, Computational node and Computational client connect to the server.
2. Communication server asks and then receives problem class names from the Computational node and Task manager.
3. The instance of a problem is sent from the Computational client to the Task manager through the server. The problem is then divided into partial problems by the Task manager.
4. Communication server receives partial problems and sends them to the Computational node, which computes partial solution and sends them back to the server.
5. Task manager receives partial solutions from which it creates a final solution. This final solution is sent to the client through the server.
6. At any time, Communication server creates server backups and informs other components about the existence of the backup.

4. Activity diagram

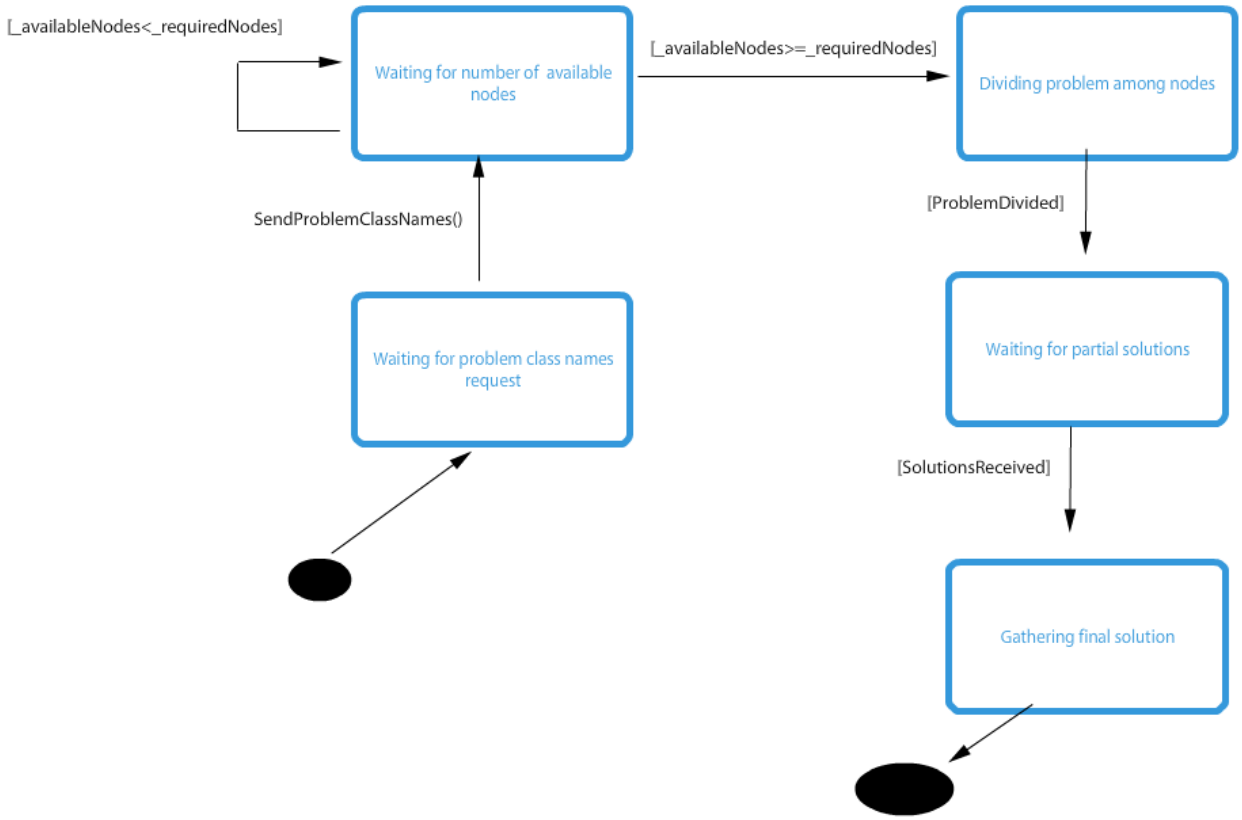


Activity diagram showing the exchange of information between Communication Server, Task Manager and Computional Node, once the client asked the cluster to solve the problem.

IV. System states description

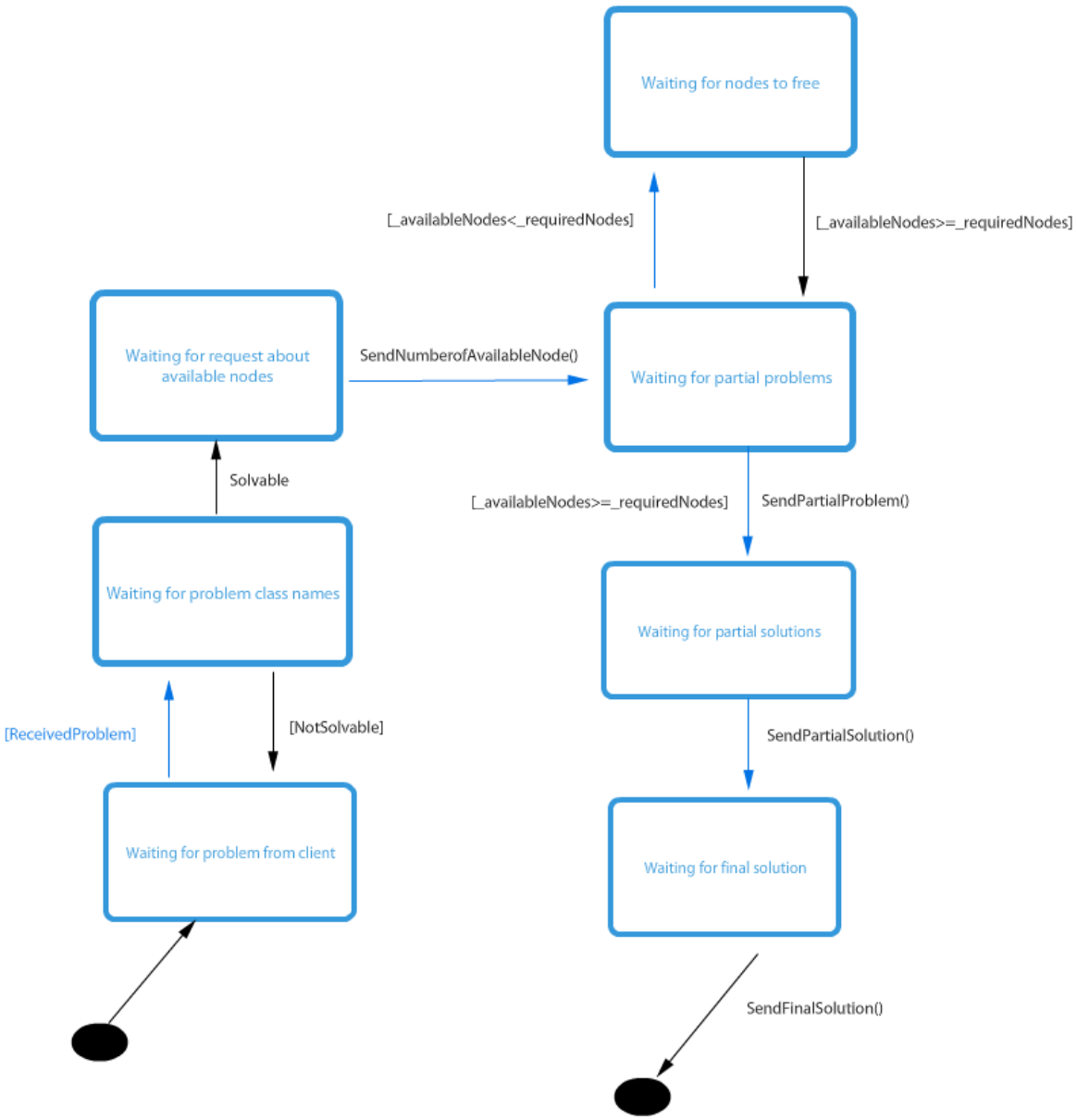
a) Task Manager

States of Task Manager are associated with waiting for information from Communication server as well as solutions from Computional Node.



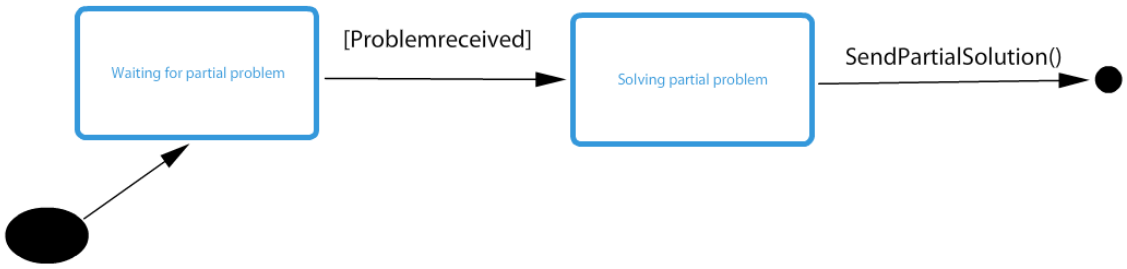
b) Communication Server

Server works as an intermediate between Node, Client and Task Manager. As a consequence its states are mostly associated with waiting for information to be forwarded.



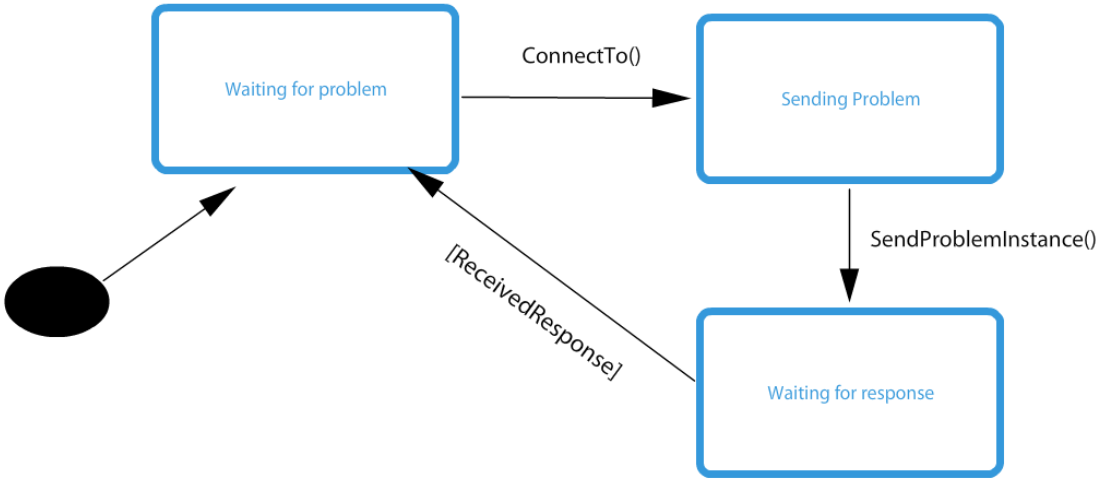
c) Computational Node

First it waits for partial problem to be sent by the server and then computes it.



d) Computational Client

This diagram represents a computer with user inputted problem. It connects to the server, sends the user's problem and waits for response. Either solution or information about unsolvability



V. Communication protocol

get-problem-types is a message sent from CommunicationServer to TaskManagers and ComputationalNodes. Requests names of problems that those components are able to process.

Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ClusterMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="get-problem-types"/>
      </xs:sequence>
      <xs:attribute type="xs:long" name="md5"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<ClusterMessage md5="ae4b7c59bcfa98772aed75ae33bdae0c">
  <get-solvable-problems />
</ClusterMessage>
```

get-problem-types response from TaskManagers or ComputationalNodes back to CommunicationServer. List problem types (*problem class names*) that those components are able to process.

Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ClusterMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="response">
```

```
          <xs:complexType>
            <xs:sequence>
              <xs:element name="solvable-problems">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="command"/>
                    <xs:element name="list">
                      <xs:complexType mixed="true">
                        <xs:sequence>
                          <xs:element type="xs:string" name="problem" maxOccurs="unbounded"
minOccurs="0"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:long" name="md5"/>
      <xs:attribute type="xs:byte" name="id"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<ClusterMessage md5="ae4b7c59bcfa98772aed75ae33bdae0c" id="12">
  <response>
    <problem-types>
      <command>get-problem-types</command>
      <list>
        <problem>TSP</problem>
```

```
      </list>
    </problem-types>
  </response>
</ClusterMessage>
```

problem-instance sends a problem instance data (e.g. to TaskManager)

Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ClusterMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="problem-instance">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="name"/>
              <xs:element type="xs:string" name="data">
                <xs:annotation>
                  <xs:documentation>List of solvable problems</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:string" name="md5"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<ClusterMessage md5="ae4b7c59bcfa98772aed75ae33bdae0c">
  <problem-instance>
    <name>TSP</name>
    <data>
      <!-- Data of the problem instance -->
    </data>
  </problem-instance>
</ClusterMessage>
```

partial-solutions sends a partial problem data (e.g. to TaskManager)

Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ClusterMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="partial-solutions">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="list"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:string" name="md5"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<ClusterMessage md5="ae4b7c59bcfa98772aed75ae33bdae0c">
  <partial-solutions>
     <list />
  </partial-solutions>
</ClusterMessage>
```

final-solution sends a final solution (e.g. from TaskManager to CommunicationServer)

Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ClusterMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="response">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="final-solution">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="data"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:long" name="md5"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<ClusterMessage md5="ae4b7c59bcfa98772aed75ae33bdae0c">
  <response>
    <final-solution>
     <data />
    </final-solution>
  </response>
</ClusterMessage>
```

data-corrupted specifies, that received data (e.g. *final-solution*) is corrupted (md5 does not match the data checksum)

Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ClusterMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="data-corrupted"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<ClusterMessage>
  <data-corrupted />
</ClusterMessage>
```

invalid-problem-type specifies, that received problem type is not accepted by the receiver (e.g. ComputationalNode that has loaded only SAT solver module and is asked to compute different type of problem).

Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ClusterMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="invalid-problem-type">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="message"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:byte" name="length"/>
      <xs:attribute type="xs:long" name="checksum"/>
      <xs:attribute type="xs:byte" name="id"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<ClusterMessage length="109" checksum="9234212312236578" id="2">
<invalid-problem-type>
    <message>Problem type is not accepted</message>
</invalid-problem-type>
</ClusterMessage>
```

VI. Errors

1. Client

- Connection with server lost while sending the problem.
- Connection with server lost while obtaining the solution.
- Connection with server lost while waiting for solution.
- Final solution: data corrupted

Communication server enqueues data until the client is reconnected and responds back. In case when data was being sent while the connection was lost, it becomes invalid.

2. Server

- Connection lost with client.
- Connection lost with node.
- Connection lost with task manager.
- Data corrupted

In case of connection failure with the server all components try to resend data.

3. Computational Node

- Node stops responding.
- Connection with server lost while computing.
- Connection with server lost while receiving partial problem.
- Connection with server lost while sending partial problem.
- Data corrupted

If a computational node fails to connect, the server determines whether the remaining nodes are able to solve problems which were already sent.

4. Task Manager

- Connection with server lost while sending class names.
- Connection with server lost while sending partial problem.
- Connection with server lost while receiving partial solution.
- Connection with server lost while receiving number of nodes.
- Connection with server lost while receiving main problem.
- Data corrupted

Communication server will enqueue problems and all partial solutions awaiting for the Task manager to reconnect to the server.

Each ‘data corrupted’ error will cause to send ClusterMessage to the sender to resend the data.
Each error of type ‘connection lost’ will put a component into a state of trying to re-establish connection.

VII. Comments

- Problem can be solved by one or more nodes.
- Most of errors are caused by losing connection with the server.
- Due to architecture of our cluster, server is the weakest element.
- Components communicate through the server, they do not see each other directly.