COMPUTATIONAL CLUSTER

Authors:

- Michał Padzik
- Michał Mierzyński
- Łukasz Napora
- Kamil Żak - Contact

Warsaw University of Technology
Software Engineering I

Table of Contents

Introducion

Description

1. Class Diagram is composed of several classes:

    a. Client – Using this class is initial step of computation. This class can solve problem that is defined in *mainProblem* variable. At the end, it present solution on screen, so it is last step too.
    b. Server – Independent module that serves to data transfer. It is a bridge connecting the modules at different points in the network. Provides a special log mechanism in order to prevent data loss.
    c. Task Manager – Main core of calculations distraction. Split problem into smaller ones and makes a solution from partial solutions.
    d. Computational Node – Part of cluster, gets a partial problem and turns it into partial solution.
    e. Problem – Abstract base class of computation unit. Its instance (concrete type at concrete moment of computation) is in circulation in working cluster.
    f. Partial Solution, Final Solution, Partial Problem, Main Problem – all of them inherit from Problem class and serve to present actual state of information flow. Each of them have some variable that define their type e.g. *string Name;* .

2. State Diagrams present a work of particular class from Class Diagram. Due to stability, architecture takes into account critical use-case's. All modules could be terminated due to lack of electricity, ignorance of users or unhandled exceptions (random errors), but all of this cases are not destroying for computation. Modules provides system of making a logs and restoring lost or crashed part of computation and resume solving the problem.

    a. The Client State Diagram shows the following state of the client. In the beginning, the client is initialized, then checks for wrong data. If data are correct execute next step, otherwise displays message error and stop working. The next step is finding a server. Again in case of error stop execution. The same thing is with sending data to founded server – if correct client waits for solution, if not – stop executing. Complete error messaging are taken into consideration (not shown on diagrams).

b. The Server State Diagram shows the following state of the server. In the beginning, the server is initialized, if initialization is correct it starts waiting for action. Having received some data, it checks what type of data it received, and passes to the conditions associated with the data. Server can receive the following types of data: new problem, partial solutions, partial data and final solution.
c. The Task Manager State Diagram shows the following state of the task manager. In the beginning, the task manager is initialized, if initialization is correct it starts waiting for data. Having received some data, it checks what type of data it received, and passes to the conditions associated with the data. Task manager can receive the following types of data: new problem and partial solutions.
d. The Computational Node State Diagram shows the following state of the Computational Node. In the beginning, the Computational Node is initialized and it presents itself to the server (in order to inform what problem it solves). Then waits for partial problem (state field – Unused) and if data has come, solve the problem (state field – Working). In case of failure it informs the server about failure and again wait for problem to compute. If solving was ended with Success, sends partial solution to the server (state field – Succeed), and wait for other data (state field – Unused).
e. The Problem Instance State Diagram shows the following state of the Problem Instance. In the beginning, the Problem Instance is initialized by Client. Then is sent (by Client too) through Server to Task Manager. In the next step is sent to Computational Nodes (through Server) as a partial problems (list of them). Further partials are sent to Task Manager (when composed into Final Solution) and sent to Client in order to display on screen.

3. Event Flow Diagram shows the following events of Computational Cluster.
   a. Choose problem – The client chooses a problem which should be solved.
   b. Problem available – The server approves the problem if it is available otherwise back to Choose problem and asks for choosing a new problem.
   c. Put data – The client enters data to system.
   d. Correct data – If the data are appropriate for the problem server starts converting it, otherwise asks client to send new data.
   e. Find problem – The server looks for suitable task manager and sends data to it.
   f. Divide for node – The task manager divides received data and sends partial data to the server.
   g. Choose node – The server chooses available node to solve partial problem.
   h. Node work – The node solves the partial problem it received form the server.
   i. Get partial solutions – The server receives solved partial problem and sends it to the task manager.
   j. Connect partials solutions – The task manager adds all partial problems and creates final result, which is sent to the server.
   k. Show result – The client receives the final result and prints it.

Architecture

Class diagrams

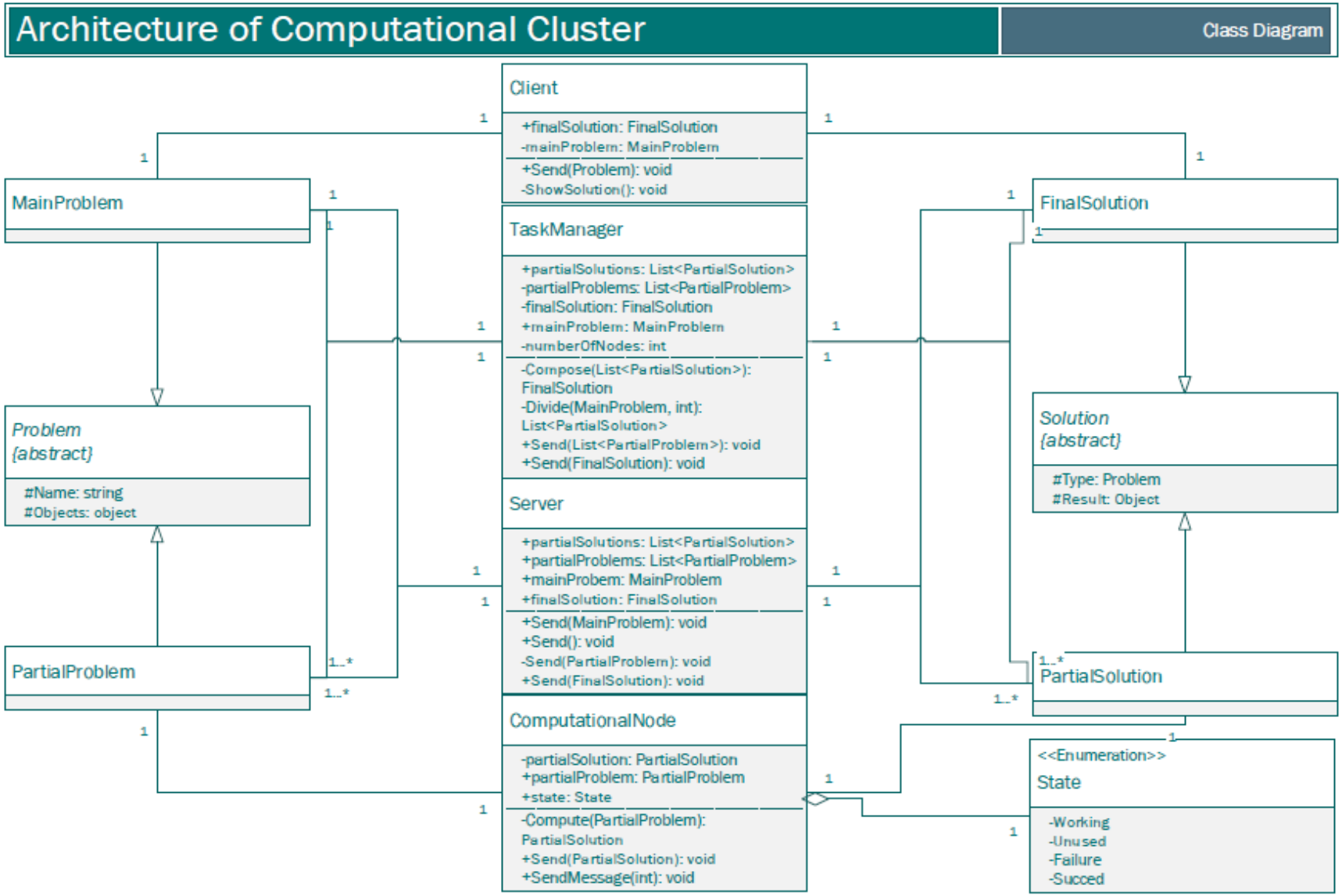
Architecture of Computational Cluster
Class Diagram
Client
+finalSolution: FinalSolution
-mainProblem: MainProblem
+Send(Problem): void
-ShowSolution(): void
MainProblem
TaskManager
+partialSolutions: List<PartialSolution>
-partialProblems: List<PartialProblem>
-finalSolution: FinalSolution
+mainProblem: MainProblem
-numberOfNodes: int
-Compose(List<PartialSolution>): FinalSolution
-Divide(MainProblem, int): List<PartialSolution>
+Send(List<PartialProblem>): void
+Send(FinalSolution): void
FinalSolution
Problem
{abstract}
#Name: string
#Objects: object
Server
+partialSolutions: List<PartialSolution>
+partialProblems: List<PartialProblem>
+mainProbem: MainProblem
+finalSolution: FinalSolution
+Send(MainProblem): void
+Send(): void
-Send(PartialProblem): void
+Send(FinalSolution): void
Solution
{abstract}
#Type: Problem
#Result: Object
PartialProblem
ComputationalNode
-partialSolution: PartialSolution
+partialProblem: PartialProblem
+state: State
-Compute(PartialProblem): PartialSolution
+Send(PartialSolution): void
+SendMessage(int): void
PartialSolution
<<Enumeration>>
State
-Working
-Unused
-Failure
-Succed

fig. 1

State diagrams

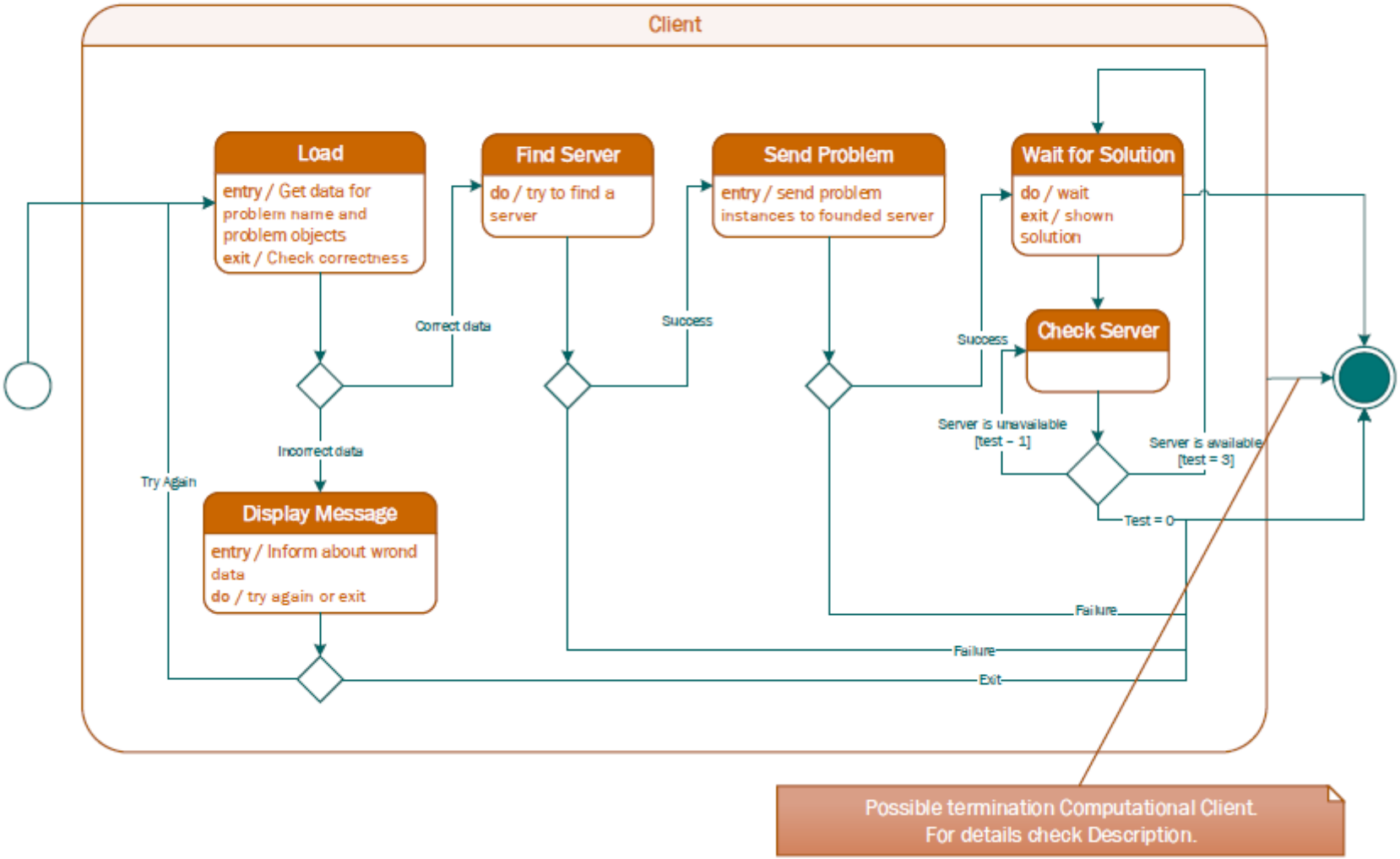
Client
Load
entry / Get data for problem name and problem objects
exit / Check correctness
Find Server
do / try to find a server
Send Problem
entry / send problem instances to founded server
Wait for Solution
do / wait
exit / shown solution
Check Server
Correct data
Success
Success
Incorrect data
Try Again
Server is unavailable [test - 1]
Server is available [test = 3]
Test = 0
Display Message
entry / Inform about wrond data
do / try again or exit
Failure
Failure
Exit
Possible termination Computational Client.
For details check Description.

fig. 2

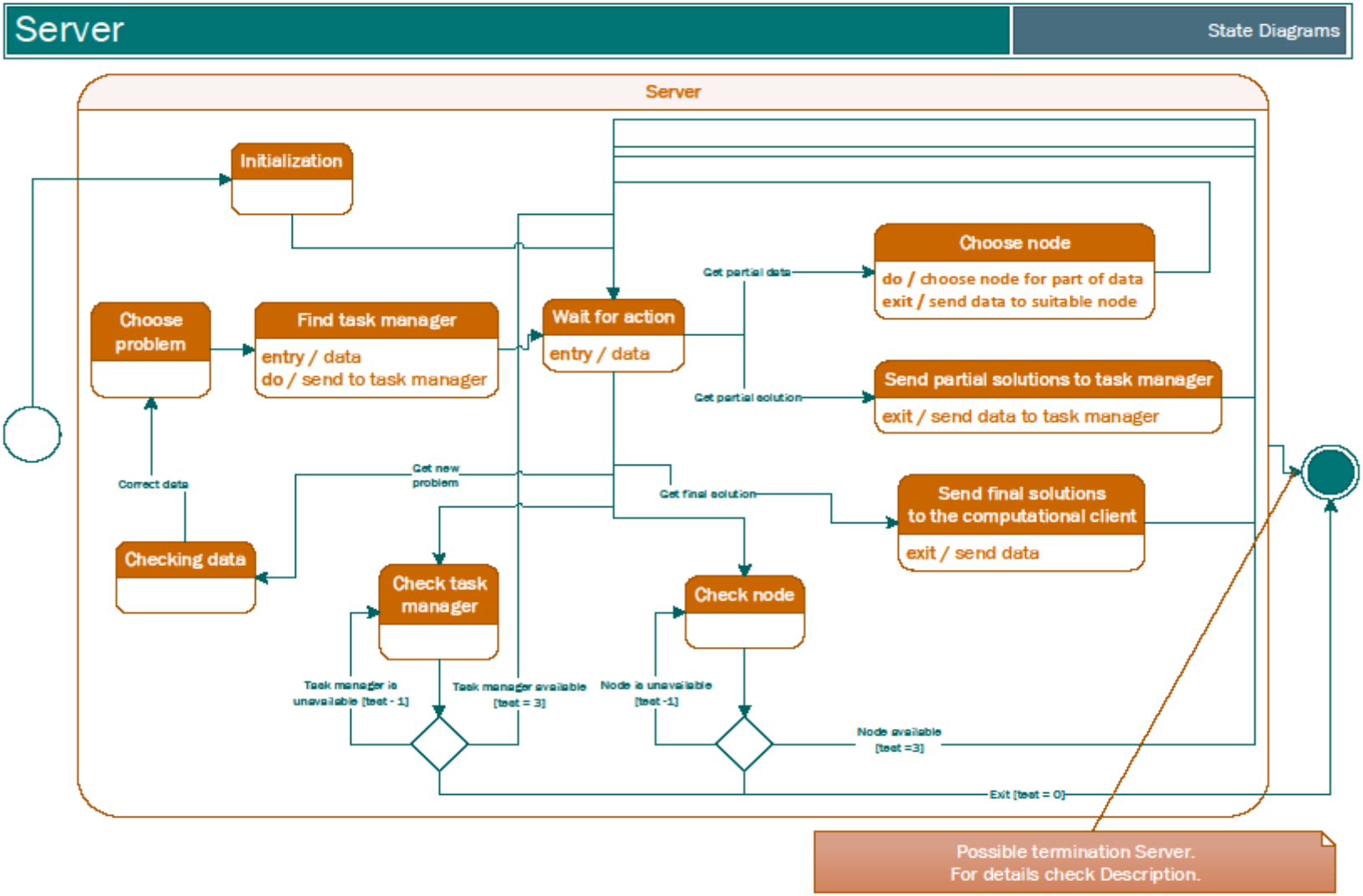
Server
State Diagrams
Server
Initialization
Choose problem
Find task manager
entry / data
do / send to task manager
Wait for action
entry / data
Get partial data
Choose node
do / choose node for part of data
exit / send data to suitable node
Get partial solution
Send partial solutions to task manager
exit / send data to task manager
Correct data
Get new problem
Get final solution
Send final solutions to the computational client
exit / send data
Checking data
Check task manager
Check node
Task manager is unavailable [test - 1]
Task manager available [test = 3]
Node is unavailable [test -1]
Node available [test =3]
Exit [test = 0]
Possible termination Server.
For details check Description.

fig. 3

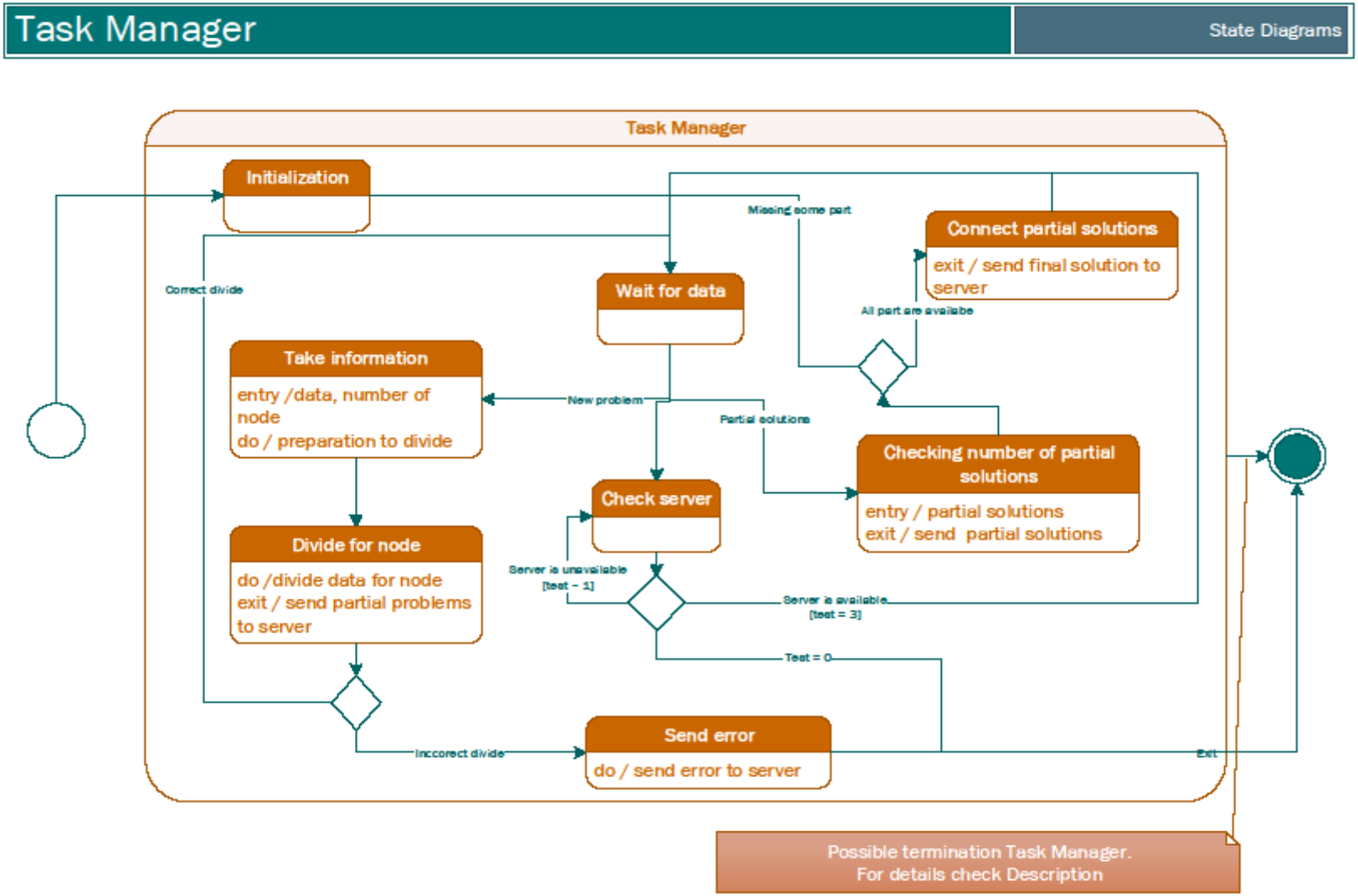
Task Manager
State Diagrams
Task Manager
Initialization
Missing some part
Connect partial solutions
exit / send final solution to server
Correct divide
Wait for data
All part are availabe
Take information
entry /data, number of node
do / preparation to divide
New problem
Partial solutions
Checking number of partial solutions
entry / partial solutions
exit / send partial solutions
Check server
Divide for node
do /divide data for node
exit / send partial problems to server
Server is unavailable
[test - 1]
Server is available
[test = 3]
Test = 0
Inccorect divide
Send error
do / send error to server
Exit
Possible termination Task Manager.
For details check Description

fig. 4

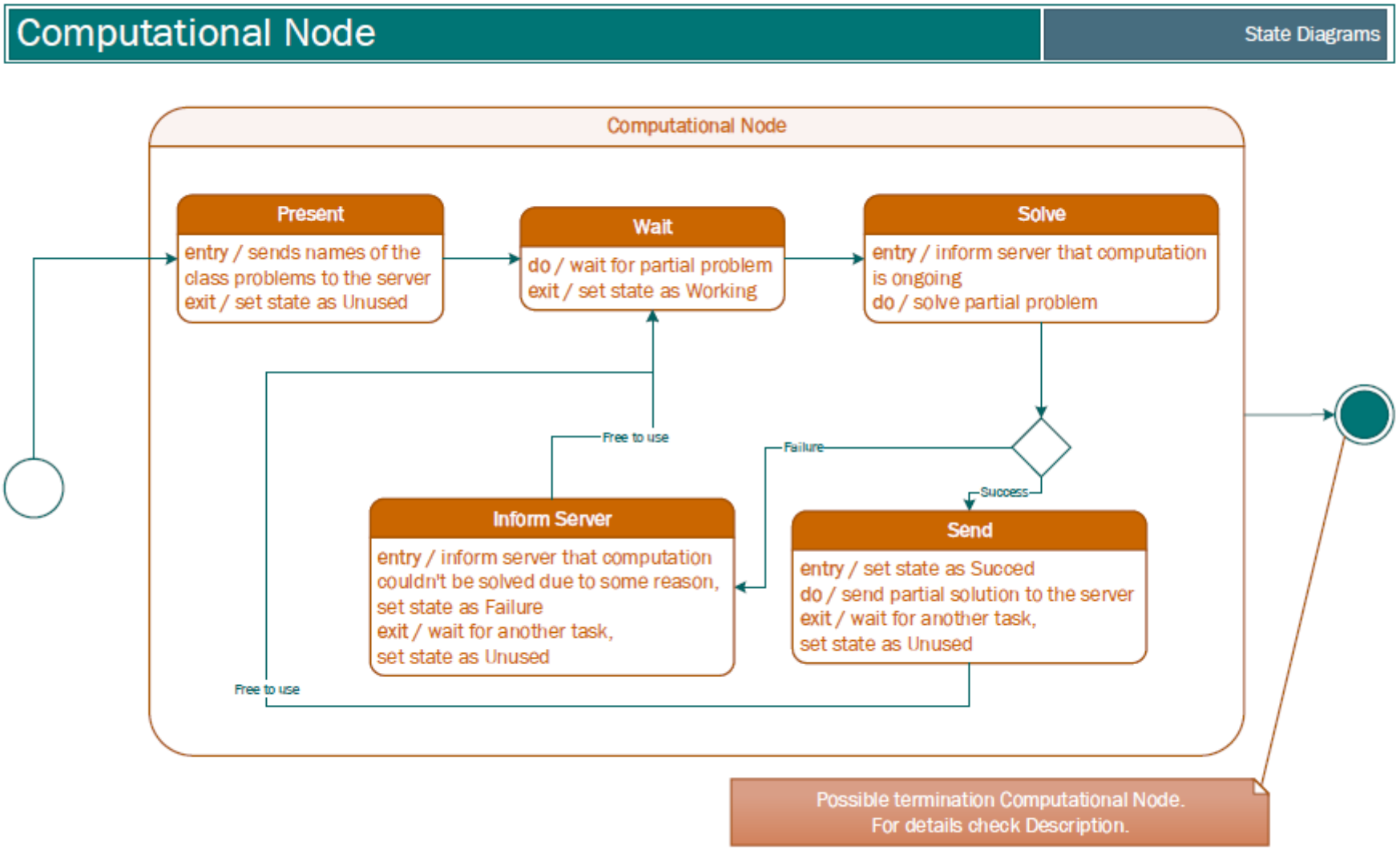
Computational Node
State Diagrams
Computational Node
Present
entry / sends names of the class problems to the server
exit / set state as Unused
Wait
do / wait for partial problem
exit / set state as Working
Solve
entry / inform server that computation is ongoing
do / solve partial problem
Free to use
Failure
Success
Inform Server
entry / inform server that computation couldn't be solved due to some reason, set state as Failure
exit / wait for another task, set state as Unused
Send
entry / set state as Succed
do / send partial solution to the server
exit / wait for another task, set state as Unused
Free to use
Possible termination Computational Node.
For details check Description.

fig. 5

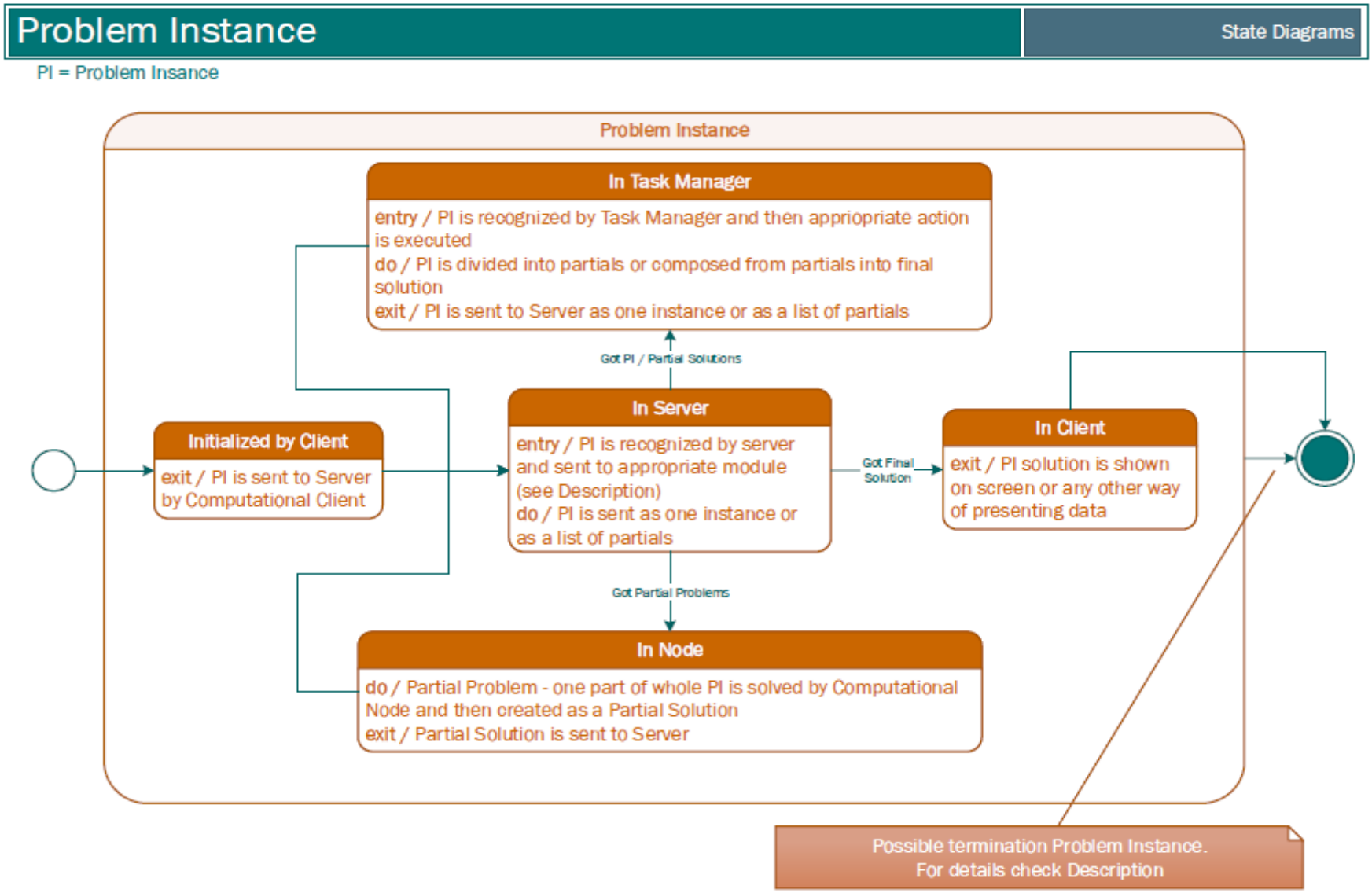
Problem Instance
State Diagrams
PI = Problem Insance
Problem Instance
In Task Manager
entry / PI is recognized by Task Manager and then appriopriate action is executed
do / PI is divided into partials or composed from partials into final solution
exit / PI is sent to Server as one instance or as a list of partials
Got PI / Partial Solutions
Initialized by Client
exit / PI is sent to Server by Computational Client
In Server
entry / PI is recognized by server and sent to appropriate module (see Description)
do / PI is sent as one instance or as a list of partials
Got Final Solution
In Client
exit / PI solution is shown on screen or any other way of presenting data
Got Partial Problems
In Node
do / Partial Problem - one part of whole PI is solved by Computational Node and then created as a Partial Solution
exit / Partial Solution is sent to Server
Possible termination Problem Instance.
For details check Description

fig. 6

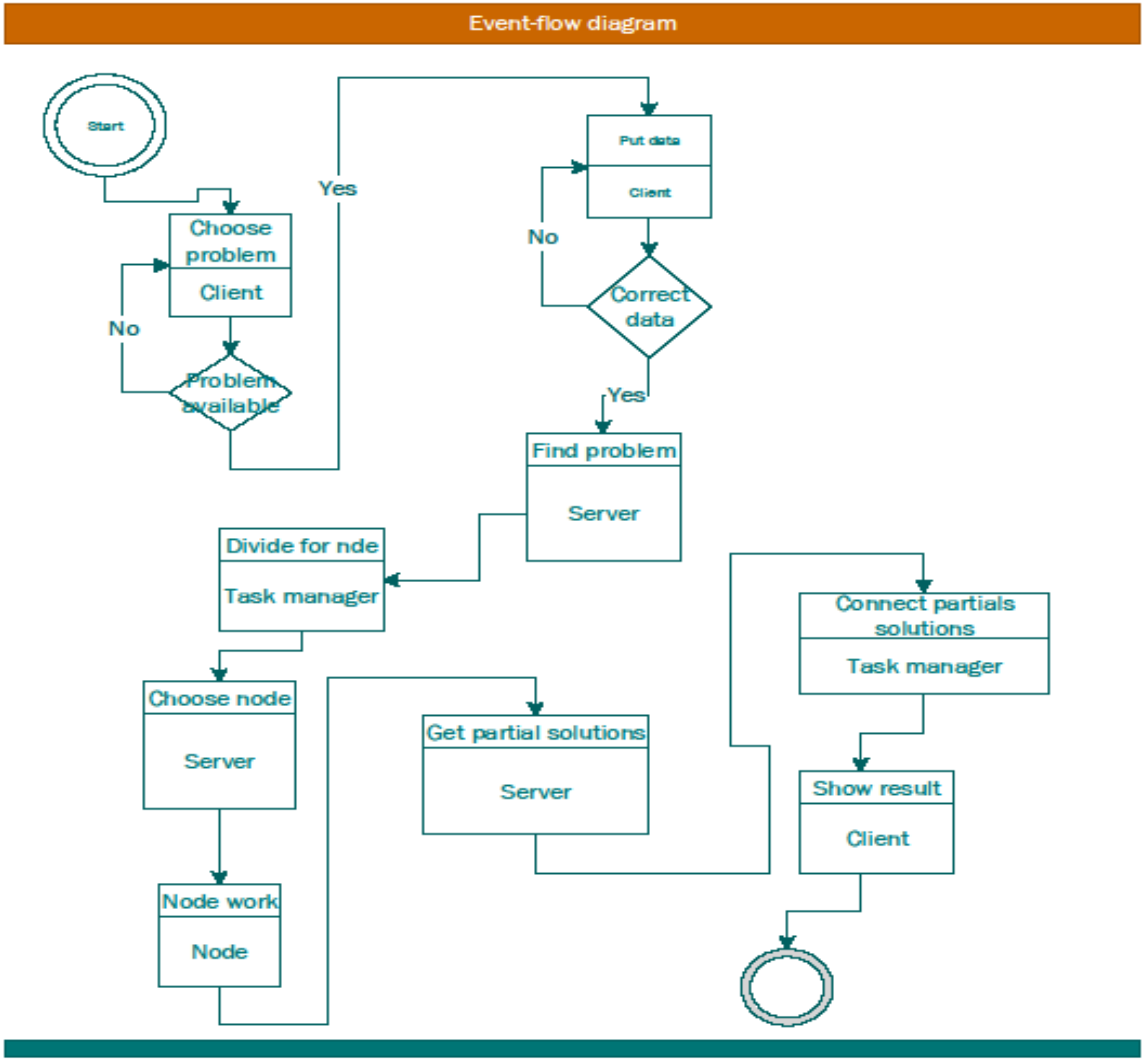
Event-flow diagram
Start
Choose problem
Client
No
Problem available
Yes
Put data
Client
No
Correct data
Yes
Find problem
Server
Divide for nde
Task manager
Choose node
Server
Node work
Node
Get partial solutions
Server
Connect partials solutions
Task manager
Show result
Client

fig. 7

Activity diagrams

Sequence diagrams

Communication protocol desing

Input data format specification

Special system states description

Example class problem