|  |  |
| --- | --- |
| Projekt  **Systemy odporne na błędy**  Wydział Elektrotechniki Automatyki i Informatyki  Politechnika Świętokrzyska | |
| Studia: **Niestacjonarne II stopnia** | Kierunek: **Informatyka** |
| Grupa: **1IZ21** | Wykonanie: **Bartosz Rokita, Adam Markowski** |
| Temat projektu:  **Paxos głosowanie** | |

Spis treści

[1. Cel projektu 3](#_Toc93509144)

[2. Technologie Błąd! Nie zdefiniowano zakładki.](#_Toc93509145)

[3. Algorytmy Błąd! Nie zdefiniowano zakładki.](#_Toc93509146)

[4. Diagramy klas Błąd! Nie zdefiniowano zakładki.](#_Toc93509147)

[5. Diagramy przypadków użycia Błąd! Nie zdefiniowano zakładki.](#_Toc93509148)

[6. Działanie aplikacji 7](#_Toc93509149)

[7. Wnioski Błąd! Nie zdefiniowano zakładki.](#_Toc93509150)

# Cel projektu

Celem projektu było zaimplementowanie protokołu paxos na którym zostało oparte głosowanie na wartością liczbową i symulacja awarii części systemu.

# Paxos

Paxos to protokół rozwiązywania konsensusu w systemach rozproszonych. Pozwala na uzgodnienie(konsensus) jednego wyniku wśród wielu uczestników. Takie uzgodnienie staje się problemem, gdy uczestnicy lub komunikacja między nimi ulega awarii.

W tym protokole można wyróżnić kilka ról:

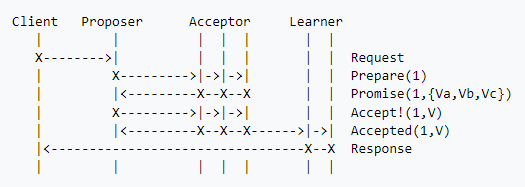
* Klient – wysyła żądanie do systemu.
* Akceptor – odbiera komunikaty od wnioskodawcy i zwraca wynik.
* Proposer – odbiera od klienta żądanie i koordynuje cały proces ustalania wyniku   
  z akceptantami.

Pełna komunikacja w protokole odbywa się w następujący sposób:

1. Proposer wysyła wiadomość „Prepare” zawierającą numer identyfikacyjny  
   do kworum akceptorów.
2. Akceptorzy odbierają wiadomość od Proposera. Po odebraniu wiadomości porównuje jej numer identyfikacyjny z zapisanym w pamięci. Jeśli nadesłany numer jest większy od zapisanego to wysyła do Proposera wiadomość „Promise” informującą o tym, że zignoruje przyszłe wiadomości, jeśli będą miały mniejszy numer od przesłanego. Dodatkowo, jeśli zaakceptował już jakąś poprzednią wiadomość to odsyła ją w raz  
   z wiadomością „Promise”.
3. Proposer oczekuje na wiadomości „Promise” z kworum akceptorów. Jeśli odebrał wiadomości od akceptorów to rozsyła do kworum wiadomość „Accept” z numerem identyfikacyjnym i wartością, która ustalana jest w następujący sposób:

* Jeśli część akceptorów już wcześniej zaakceptowała jakąś wiadomość  
  to wartość jest ustalana na podstawie przesłanych.
* Jeśli żaden z akceptorów nie zaakceptował wcześniej wiadomości  
  to Proposer zostaje przy swojej wartości.

1. Akceptor odbiera wiadomość „Accept” od Proposera jeśli obiecał mu to wcześniej  
   i wysyła do niego wiadomość „Accepted”.



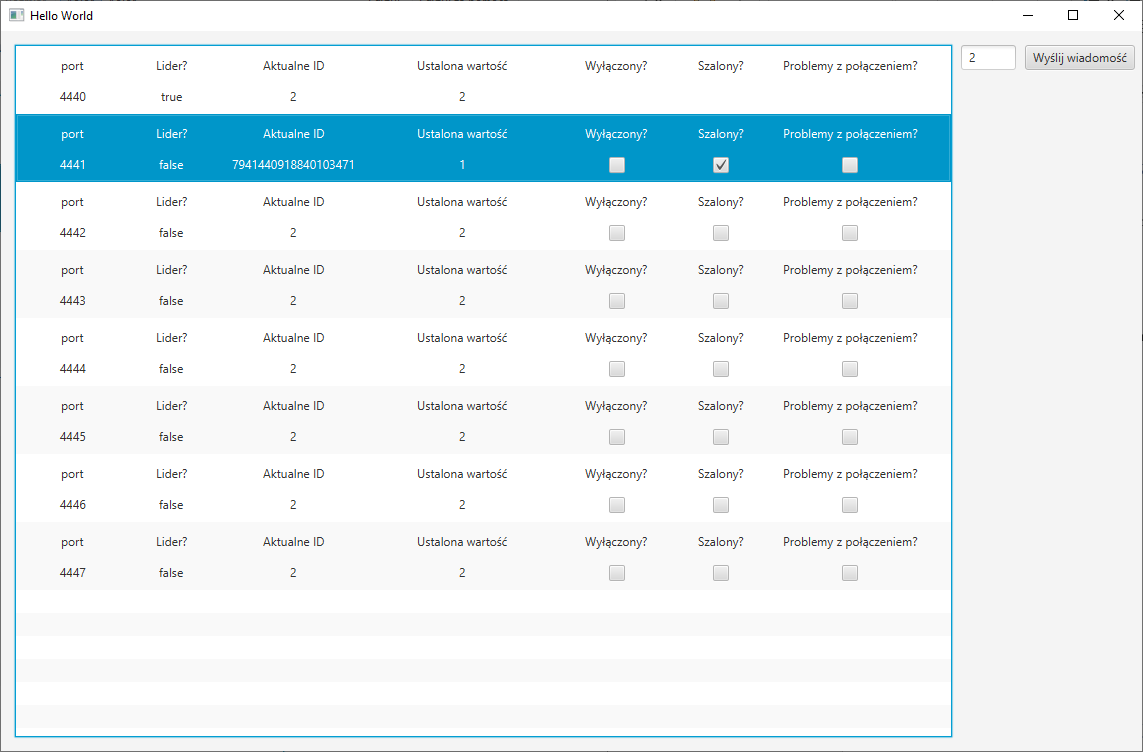
Rysunek 1 – graficzna prezentacja komunikacji protokołu paxos

# Symulacja awarii w protokole Paxos

W systemie zostały zaimplementowane 3 symulacje awarii w protokole Paxos. Wszystkie błędy zostały opisane poniżej.

## Szalony akceptor

Symuluje błędne działanie akceptora. Awaria polega na zwracaniu losowego numeru sekwencyjnego wiadomości i nieprzyjmowaniu wiadomości.



Rysunek 2 – losowy numer sekwencyjny

## Wyłączony akceptor

Symuluje błędne działanie akceptora. Awaria polega na braku połączenia akceptora do systemu. Nie dostaje żadnych wiadomości przez co nie bierze udziału w głosowaniu.

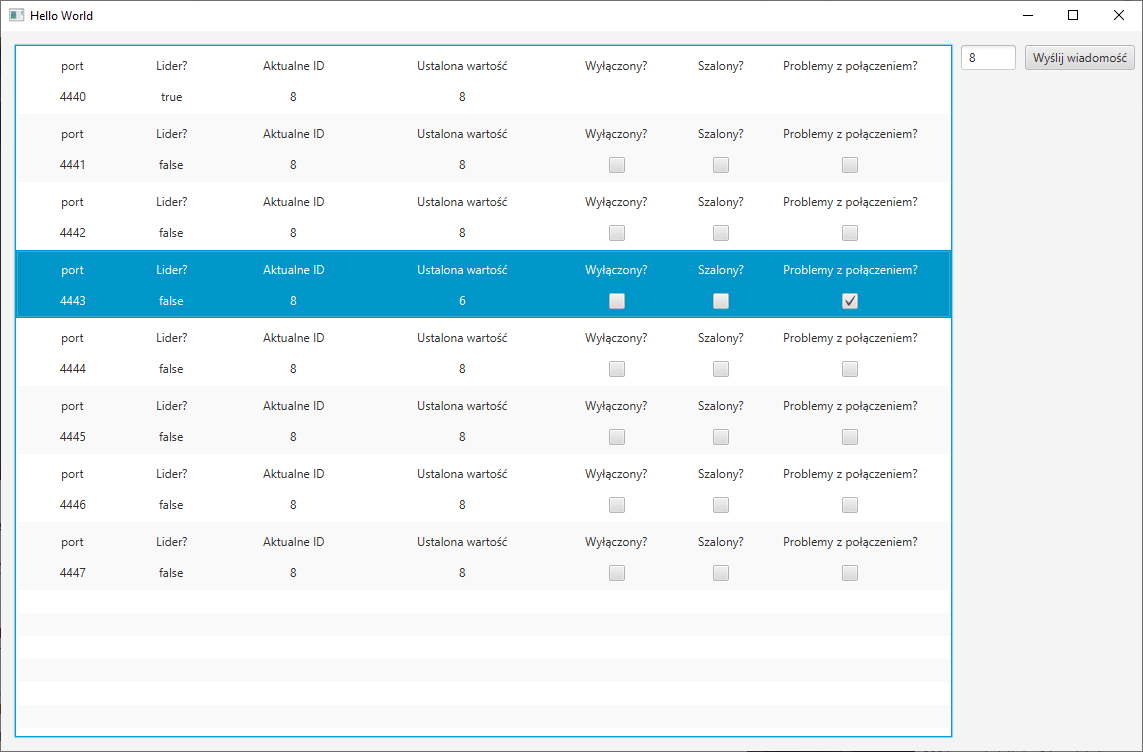
Obraz zawierający stół

Opis wygenerowany automatycznie

Rysunek 3 – Wyłączony akceptor

## Problem z połączeniem akceptora

Symuluje błędne działanie akceptora. Awaria polega na utracie wiadomości. Gdy symulacja jest włączona akceptor ma szanse 50% na utratę przysłanej wiadomości.



Rysunek 4 – Problem z połączeniem akceptor

# Działanie aplikacji

W tym rozdziale zostaną przedstawione fragmenty kodu odpowiedzialne za działanie systemu.

## Serwer

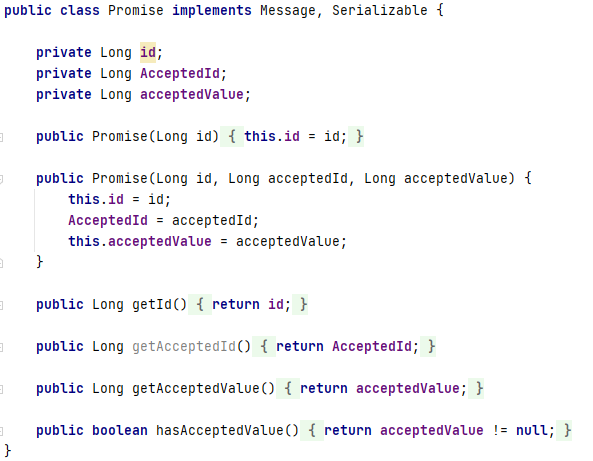
Instancje klasa *Server* odwzorowują fizyczne serwery połączone w sieci. Każdy z serwerów posiada swój port oraz mechanizmy do wysyłania i odbierania pakietów. Wewnątrz obiektu zapisywany jest stan awarii, aktualny numer sekwencyjny oraz ustalona wartość. *Server* steruje wysyłaniem wiadomości i decyduje co zrobić z wiadomościami przychodzącymi.



Rysunek 5 – Część klasy *Server*

## Odbieranie i wysyłanie wiadomości

Każda wiadomość zaimplementowana jest jako osobna klasa rozszerzona o interfejs *Message*. Wysyłanie wiadomości odbywa się poprzez klasę *Connection*, która posiada w sobie obsługę UDP.



Rysunek 6 – Klasa wiadomości Promise

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 7 – Klasa *Connection*

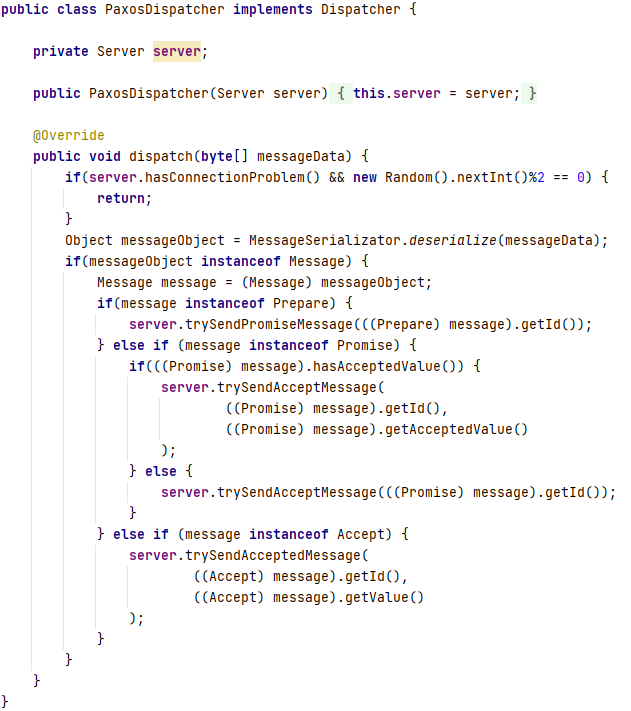
Każdy pakiet, który trafia do nadawcy odbierany jest przez *PacketReceiver* w celu wrzucenia go na kolejkę, która obsługiwana jest przez klasę *PacketDispatcher*. To w niej następuje deserializacja wiadomości poprzez dispatchery i przekazanie jej do klasy *Server*.



Rysunek 8 – Klasa *PacketReceiver*

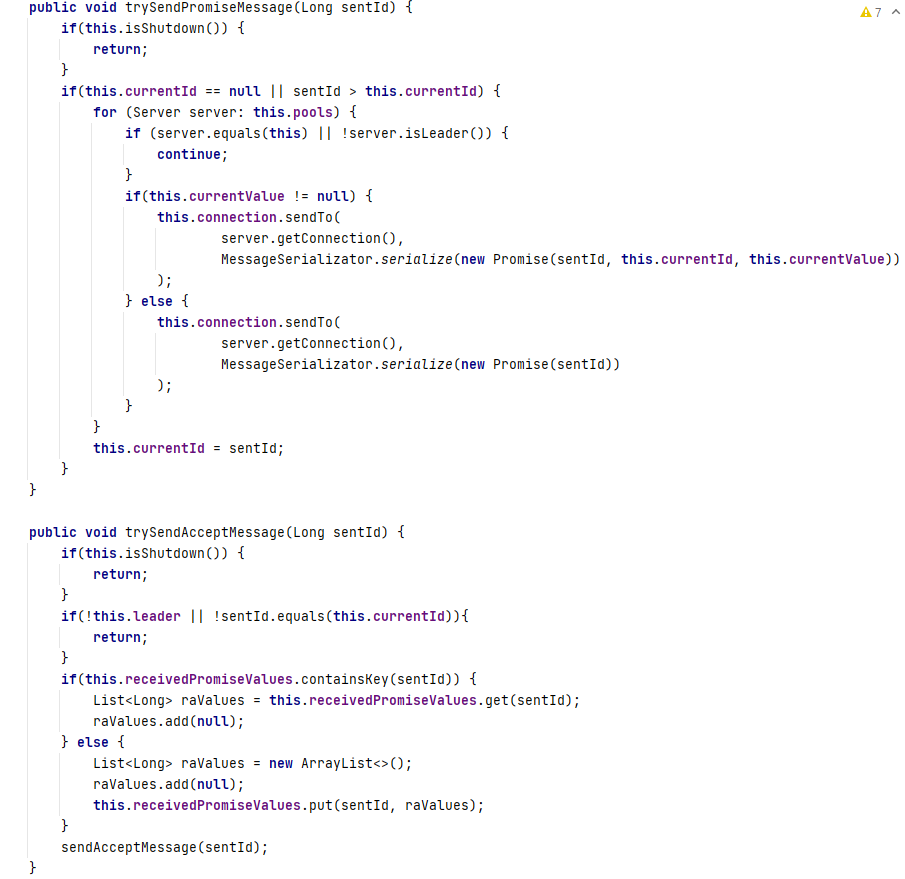


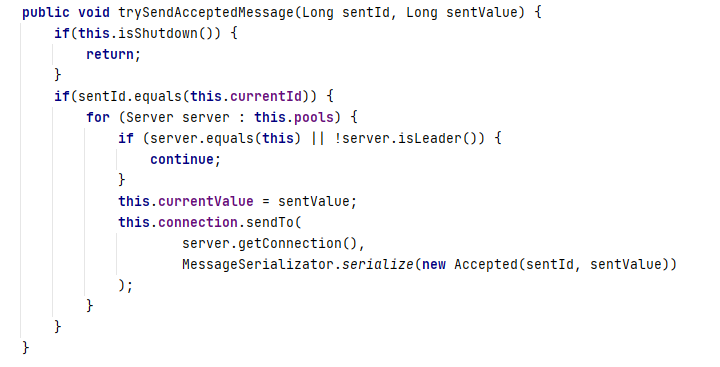
Rysunek 9 – Klasa *PacketDispatcher*



Rysunek 10 – Klasa *PaxosDispatcher*

Logika każdego z etapów jest rozbita na osobną metodę w klasie *Server*.

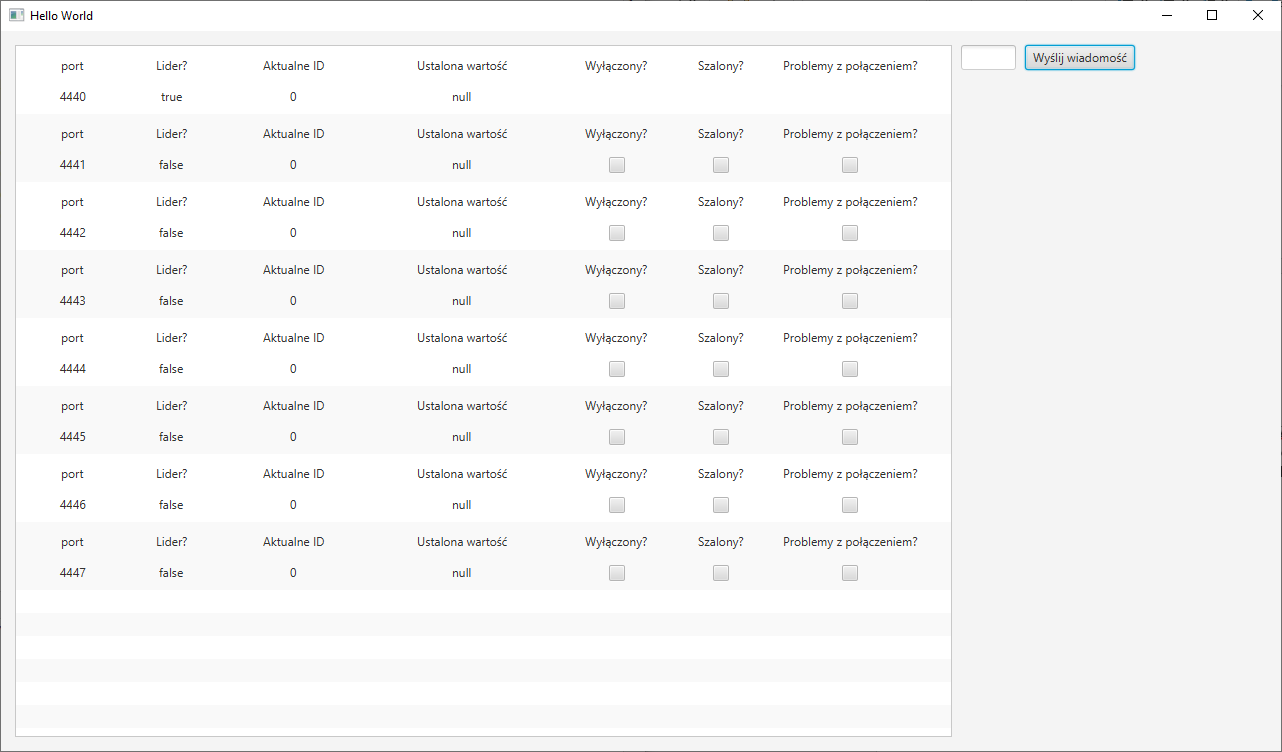




Rysunek 11 – Metody do wysyłania wiadomości Promise, Accept i Accepted

## Interfejs

Interfejs został stworzony przy użyciu *SceneBuilder*. W okienku aplikacji znajduje się lista serwerów z informacjami o ich aktualnym stanie oraz checkboxy, które informują o tym czy dana symulacja awarii została aktywowana. Obok listy znajduje się pole z przyciskiem, którym wysyłana jest wartość do Proposera.



Rysunek 12 – Interfejs

Obsługę pola tekstowego i przycisku „Wyślij wiadomość” zapewnia klasa *Controller*. Znajduje się w niej również konfiguracja i tworzenie serwerów.

**

Rysunek 12 – Klasa *Controller*