Team: 08, Dieter Pisarewski, Maxim Rjabenko

Aufgabenaufteilung:

Maxim Rjabenko und Dieter Pisarewski:
Gemeinsame Entwurf und Implementierung

Quellenangaben:

http://www.erlang.org/doc/

http://dl.acm.org/citation.cfm?id=357200

Bearbeitungszeitraum:

16.11.2013 – 23.11.2013

Aktueller Stand:

Algorithmus ist implementiert und auf einem Rechner getestet.

Änderungen im Entwurf:

keine

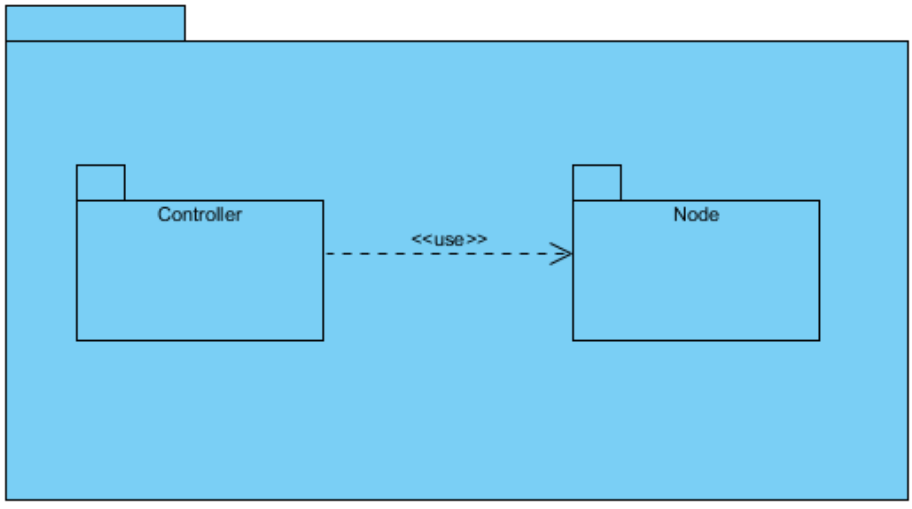
Entwurf:

Unser System besteht aus zwei Komponenten:

- Controller – Hat einen eigenen Prozess, der sich lokal unter einem Namen registriert, sich mit anderen Rechnern verbindet(durch ping), Konfiguration der Nodes aus Dateien lädt, Nodes startet und das Ergebnis der Berechnung abfragt.
- Node – Jeder Node hat einen eigenen Prozess und enthält Code für den Algorithmus.

Die beiden Komponenten sind erlang Module.

Komponentendiagramm



Realisierung:

Der Controller startet einen Node durch Funktion start() oder mehrere Nodes auf einem Rechner durch Funktion start_all(). Um mehrere Nodes zu starten, liegen Konfigurationsdateien für jeden Node im Verzeichnis nodes.

Als erstes registriert der Controller seinen Prozess lokal und liest Namen der Nodes aus den Konfigurationsdateien der Nodes(nur in start_all() Funktion). Danach sendet er einen ping an alle Rechner, die in der Datei hosts aufgelistet sind.

Als nächstes startet er einzelne Nodes in separaten Prozessen und wartet bis er von einem Node halt Nachricht bekommt, die das Ende der Berechnung signalisiert.

Dann sammelt der Controller Daten von allen Nodes und gibt den berechneten minimalen Spannbaum aus.

Der Node enthält den Code des Algorithmus. Er kann folgende Nachrichten empfangen und verarbeiten:

{initiate,Level,FragName,NodeState,Edge}
{test,Level,FragName,Edge}
{accept,Edge}
{reject,Edge}
{report,Weight,Edge}
{changeroot,Edge}
{connect,Level,Edge}

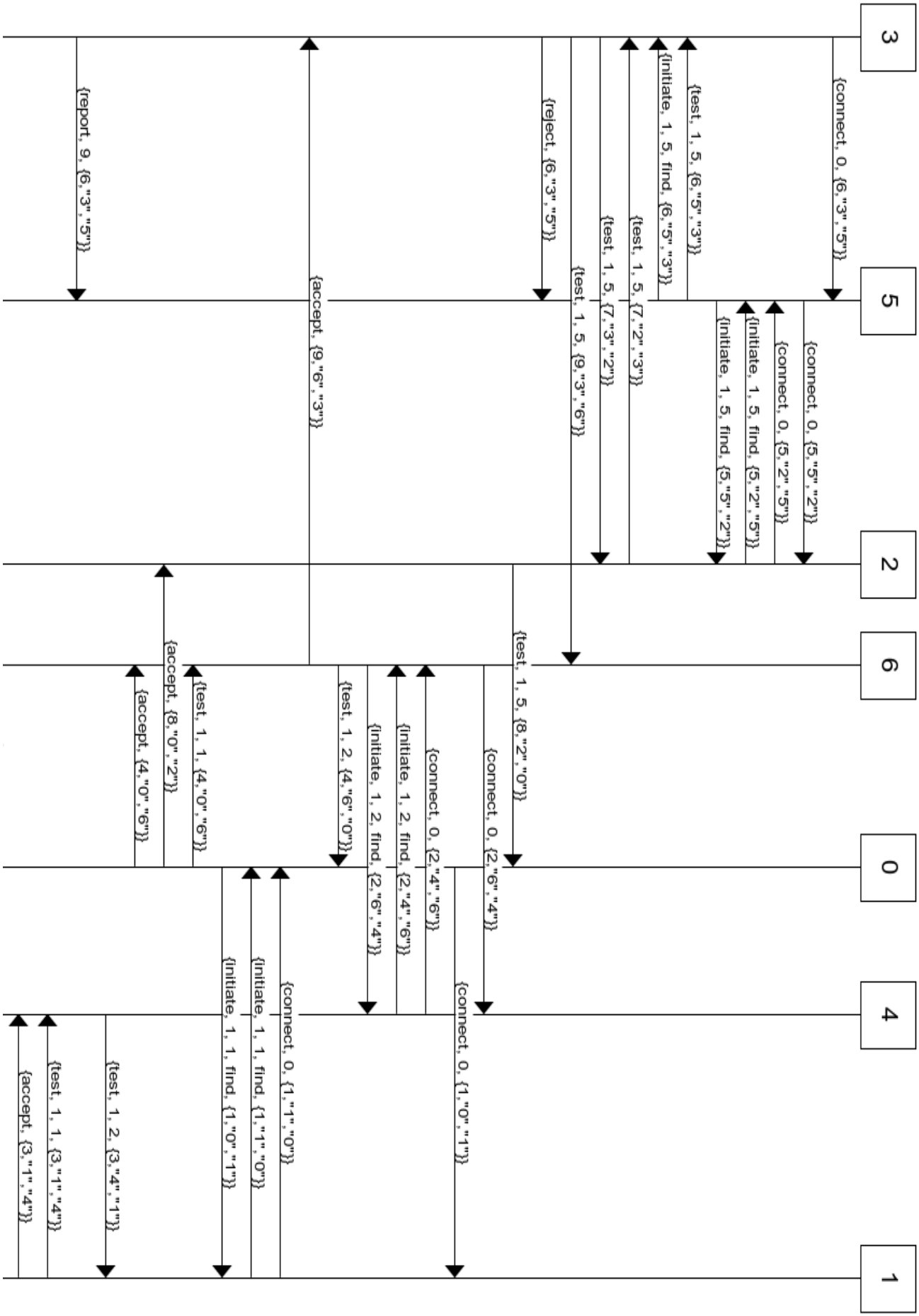
sowie {get_data, Pid}, die dem Controller Daten zurückgibt.

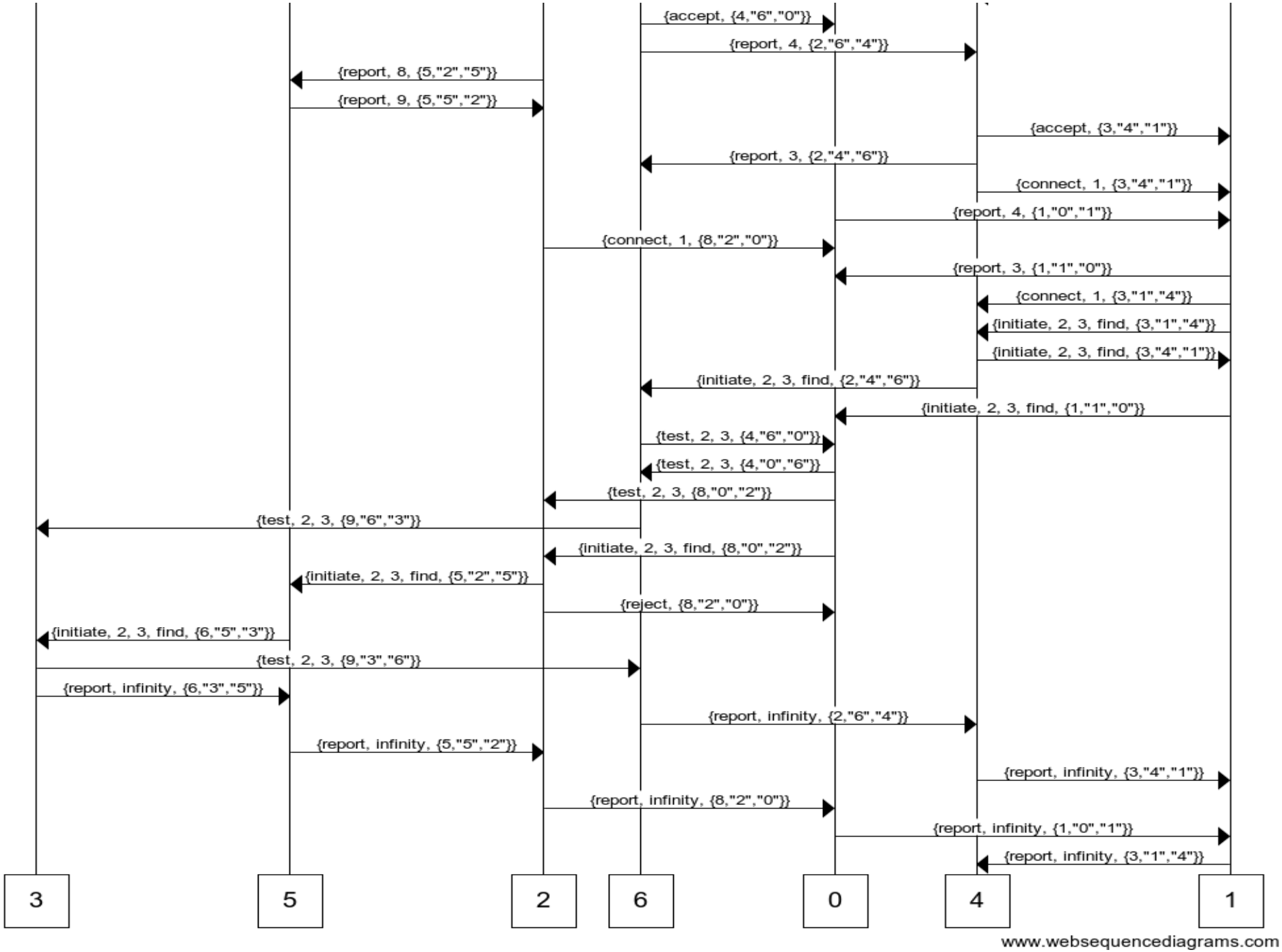
Wenn der Node eine unbekannte Nachricht empfängt, gibt er eine Warnung aus.

Für die Speicherung und die Übergabe der Werte zwischen den Funktionen wird eine Record benutzt, die folgende Daten enthält:

```
-record(data, {
  name          = <String>,
  level         = <Number>,
  frag_name     = <String>,
  node_state    = <Atom>,
  edges         = <List>,
  best_edge     = <Tuple>,
  best_weight   = <Number>,
  found_count   = <Number>,
  test_edge     = <Tuple>,
  in_branch     = <Tuple>,
  edge_states   = <List>,
  log_file      = <String>
}).
```

Kommunikationsbeispiel für den Graph aus der Aufgabenstellung:




{accept, {4,"6","0"}}
{report, 4, {2,"6","4"}}
{report, 8, {5,"2","5"}}
{report, 9, {5,"5","2"}}
{accept, {3,"4","1"}}
{report, 3, {2,"4","6"}}
{connect, 1, {3,"4","1"}}
{report, 4, {1,"0","1"}}
{connect, 1, {8,"2","0"}}
{report, 3, {1,"1","0"}}
{connect, 1, {3,"1","4"}}
{initiate, 2, 3, find, {3,"1","4"}}
{initiate, 2, 3, find, {3,"4","1"}}
{initiate, 2, 3, find, {2,"4","6"}}
{initiate, 2, 3, find, {1,"1","0"}}
{test, 2, 3, {4,"6","0"}}
{test, 2, 3, {4,"0","6"}}
{test, 2, 3, {8,"0","2"}}
{test, 2, 3, {9,"6","3"}}
{initiate, 2, 3, find, {8,"0","2"}}
{initiate, 2, 3, find, {5,"2","5"}}
{reject, {8,"2","0"}}
{initiate, 2, 3, find, {6,"5","3"}}
{test, 2, 3, {9,"3","6"}}
{report, infinity, {6,"3","5"}}
{report, infinity, {2,"6","4"}}
{report, infinity, {5,"5","2"}}
{report, infinity, {3,"4","1"}}
{report, infinity, {8,"2","0"}}
{report, infinity, {1,"0","1"}}
{report, infinity, {3,"1","4"}}
3
5
2
6
0
4
1
www.websequencediagrams.com