APUVS, Blatt 10

Jan Fajerski and Kai Warncke and Magnus Müller

25. Januar 2011

Aufgabe 10.1

1 Beide Transaktionen können validiert werden, da U ein leeres Readset hat. i=55; j=66

2 T wird abgebrochen, da das Readset von T (enthält i) mit dem Writeset von U überlappt. T muss also davon ausgehen, einen inkonsistenten Wert gelesen zu haben. U wird validiert. i=55; j=66

› Beide Transaktionen werden validiert, da U zum Zeitpunkt der Validierung von T ein leeres Writeset besitzt. i=55; j=66

› T wird abgebrochen, da das Readset von T mit dem Writeset von U überlappt. T könnte also einen inkonsistenen Wert gelesen haben und wird aufgefordert sich abzubrechen. i=55; j=66

Aufgabe 10.2

Zum Start des Programms die Funktion `testcr(N)` aufrufen, wobei N die Anzahl der gewünschten Prozesse ist.

```
-module(changrob).
-export([initcr/1, cr/3]).

%%%%%%
% Initialize a chang-roberts process
% To initialize, usw spawn (changrob, initcr,[C]), where C is an et_collector. Then, send the ID of
the
% successor in the circle.
%
initcr(Collector) ->
    receive
        {cr_next_pid, Next} ->
            cr(Next, false, Collector)
    end.

%%%%%%
% cr main function
% Next - Id of the next chang-roberts process in the circle
%   NOTE: we ignore the case that Next could be crashed.
% Participant - Boolean flag wether this process took part in a vote
% Collector - et_collector eventtracer
%
cr(Next, Participant, Collector) ->
    receive
        % we are told to start an election
        {start_election, _} ->
            io:fwrite("~w:␣starting␣election\n", [self()]),
            Collector ! {c_state_change, {self(), start_election}},

            Next ! {{election, self()}, self()},
            cr(Next, true, Collector);

        % we receive an election message
        {{election, Pred}, Sender}->
            Collector ! {c_collect, {Sender, self(), io_lib:format("<election,␣~w>", [Pred])}},

            % le (X,Y) = {true if X < Y, equal if X == Y, otherwise false
            case (le(self(), Pred)) of
                true ->
                    % our id is < than Pred. Therefore, we propagate the value Pred with a
                    % new election message
                    io:fwrite("~w:␣received␣<election,␣~w>...passing␣it␣on\n",[self(), Pred]),

                    Next ! {{election, Pred}, self()},
                    cr(Next, true, Collector);
                false ->
                    % our id is > than pred
                    if
                        % if we already participated, we discard the message
                        Participant ->
                            Collector ! {c_label, {self(), discards_message}},
                            discardMsg;
                        % if we didn't participate, we vote for ourselve
                        not Participant ->
                            io:fwrite("~w:␣received␣<election,␣~w>...proposing␣myself\n",[self(), Pred
                            ]),
                            Next ! {{election, self()}, self()}
                    end,
                    cr(Next, Participant, Collector);
                equal ->
                    % we reached a decision. let's tell everybody about it
                    io:fwrite("~w:␣received␣<election,␣~w>...i␣was␣elected\n",[self(), Pred]),

                    Next ! {{elected, self()}, self()},
                    cr(Next, false, Collector)
            end;

        % we receive an elected message
        {{elected, Leader}, Pred} ->
            %report the leader to somewhere
            case (self() == Leader) of
                false ->
                    % propagate the elected message
                    io:fwrite("~w:␣elected␣~w␣to␣be␣the␣leader\n", [self(),
                            Leader]),
                    Collector ! {c_collect, {Pred, self(), io_lib:format("<elected,␣~w>", [Leader])}},
                    Next ! {{elected, Leader}, self()},
                    cr(Next, false, Collector);
                true ->
                    % the elected message went around the circle. let's abort.
                    io:fwrite("~w:␣elected␣~w␣to␣be␣the␣leader\n", [self(),
                            Leader]),
                    Collector ! {c_collect, {Pred, self(), io_lib:format("<elected,␣~w>", [Leader])}},
                    Collector ! {c_state_change, {self(), leader}},
                    cr(Next, false, Collector)
            end
    end.

%%%%%%
% Helper function
% compare two values. return true if X < Y, false if X > Y, equal otherwise.
le (X,X) -> equal;
le (X,Y) -> X < Y.
```

```
-module(collector).
-export([collector/0, start_collector/0, convert_process_id/1]).


start_collector () ->
    spawn (?MODULE, collector, []).

collector () ->
    {ok, C} = et_collector:start_link ([]),
    collector (C).

collector (C) ->
    DONE = receive
        {c_collect, {Sender, Receiver, Message}} when is_atom(Message) ->
            et_collector:report_event(C, 1, Sender, Receiver, Message, []);
        {c_collect, {Sender, Receiver, Message}} when is_list(Message) ->
            et_collector:report_event(C, 1, Sender, Receiver, list_to_atom(lists:flatten(Message)), []
            );
        {c_collect, _} ->
            exit(bad_arg);
        {c_state_change, {Sender, State}} ->
            et_collector:report_event(C,1,Sender,Sender,state_change,[State]);
        {c_label, {Sender, Label}} ->
            et_collector:report_event(C,1,Sender,Sender,label,[Label]);
        {c_name_process, {Sender, Name}}  ->
            et_collector:report_event(C,1,Sender,Sender,name_process,[Name]);
        {c_print} ->
            io:format ("~s", [string_representation(C)]);
        {c_print_to_file, Filename} ->
            file:write_file(Filename, string_representation(C));
        {c_clear_cache} ->
            et_collector:clear_table(C);
        {c_stop} ->
            finish_collector
    end,
    case DONE of
        finish_collector -> ok;
        _ -> collector (C)
    end.

%%%%
% String represenation
% return a msc string representation
%
string_representation (C) ->
    Processes = iterate (C,
        fun({event,_,_,_,Sender,Receiver,_,_}, Acc) ->
                TempAcc = sets:add_element(Sender, Acc), % try to add the Sender
                sets:add_element(Receiver, TempAcc)  % try to add the Receiver
        end,
        sets:new ()),
    [LP|LT] = sets:to_list (Processes),

    "msc␣{\n␣hscale=2;\n"
    ++
    io_lib:format("\"~s\"", [convert_process_id (LP)])
    ++
    [io_lib:format(",␣\"~s\"",[convert_process_id(LPP)]) || LPP <- LT]
    ++
    ";|||;\n"
    ++
    iterate (C, fun(Event, Acc) -> collector_string_representation (Event, Acc) end, "")
    ++
    "}\n"
    .

%%%%
% String representation of the collector's content

%% show state change
collector_string_representation ({event, _Priority, _Time1, _Time2, Sender, Sender,
        state_change,[State]}, Acc) ->
    Acc ++ io_lib:format("\"~s\"␣rbox␣\"~s\"␣[label=\"~s\"];\n",
        [convert_process_id (Sender),convert_process_id(Sender), State]);

%% print label
collector_string_representation ({event, _Priority, _Time1, _Time2, Sender, Sender,
        label,[Label]}, Acc) ->
    Acc ++ io_lib:format("\"~s\"␣note␣\"~s\"␣[label=\"~s\"];\n",
        [convert_process_id (Sender),convert_process_id(Sender), Label]);

%% name process
collector_string_representation ({event, _Priority, _Time1, _Time2, Sender, Sender,
        name_process,[Name]}, Acc) ->
    Acc ++ io_lib:format("\"~s\"␣box␣\"~s\"␣[label=\"~s\"];\n",
        [convert_process_id (Sender),convert_process_id(Sender), Name]);

%% call -> arrows
collector_string_representation ({event, _Priority, _Time1, _Time2, Sender, Receiver,
        Message, _More}, Acc) -> Acc ++
    io_lib:format("\"~s\"␣=>␣\"~s\"␣[label=\"~w\"];\n", [convert_process_id(Sender),
    convert_process_id(Receiver), Message])
    .

%%%%
% iterate over a collector
iterate (Collector, Fun, Acc) ->
    et_collector:iterate (Collector, first, infinity, Fun, Acc).


%%%%
% the msc program doesn't like < and >
convert_process_id (Pid) ->
    lists:filter(fun(E) -> (E =/= $<) and (E =/= $>)
        end,
        io_lib:write(Pid)).
```

```
-module(test).
-export([testcr/1]).

%%%%%%
% testcr (N)
%  test the changrob module with N processes
```

```
%

testcr([String]) when is_list(String) ->
    testcr(list_to_integer(String));

testcr(N) when is_integer (N) ->
    % create a collector to build msc trace
    C = collector:start_collector(),
    % initialize the chang-roberts processes
    [H|T] = Pids = [spawn(changrob, initcr, [C]) || _ <- lists:seq(1, N)],
    % tell them about their successor
    SendTupels = lists:zip(Pids, lists:append(T, [H])),
    lists:map(fun ({Pred, Next}) -> Pred ! {cr_next_pid, Next} end, SendTupels),

    % start an election at each process
    [start_single_election (lists:nth(I, Pids), C) || I <- lists:seq(1,N)],

    % start two concurrent elections
    start_concurrent_elections([s(1,Pids), s(2,Pids)], C),
    start_concurrent_elections([s(1,Pids), s(3, Pids), s(5, Pids)], C),
    start_concurrent_elections([s(2,Pids), s(3, Pids), s(5, Pids)], C),
    start_concurrent_elections([s(3,Pids), s(4, Pids), s(5, Pids)], C),
    ok.

%%%%% shorter wrapper for lists:nth (..)
s(N,L) -> lists:nth(N,L).

start_single_election(Pid, C) ->
    Pid ! {start_election, self()},
    timer:sleep (1000),
    C ! {c_print_to_file, io_lib:format("msc/single_election_at_~s.msc",[collector:convert_process_id(Pid)])},
    C ! {c_clear_cache}.

start_concurrent_elections(Pids, C) ->
    [ P ! {start_election, self()} || P <- Pids],
    timer:sleep(1000),
    C ! {c_print_to_file,
        io_lib:format("msc/concurrent_with_~s.msc",
            [lists:map(fun(P) -> collector:convert_process_id(P) end, Pids)])
        },
    C ! {c_clear_cache}.
```

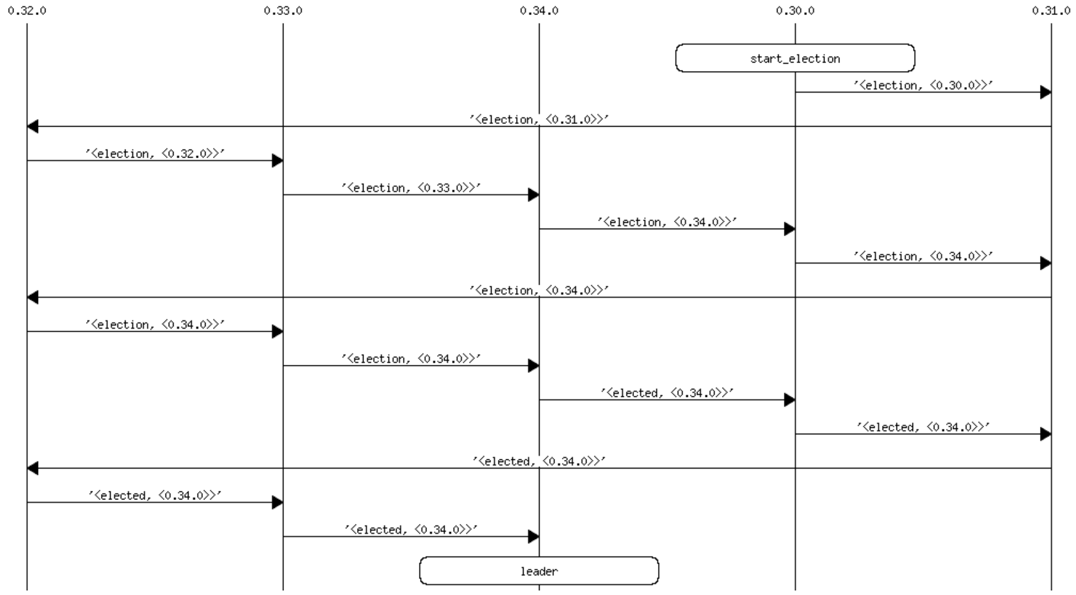

Abbildung 1: Wahlstart bei 0.30.0; Anzahl der Nachrichten: 14

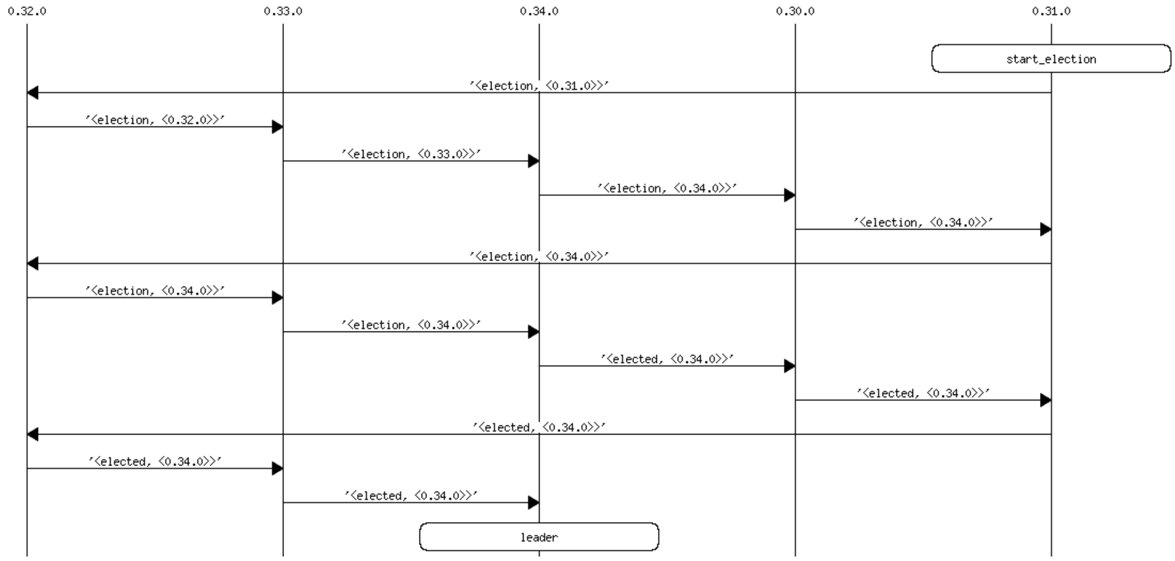

Abbildung 2: Wahlstart bei 0.31.0; Anzahl der Nachrichten: 13

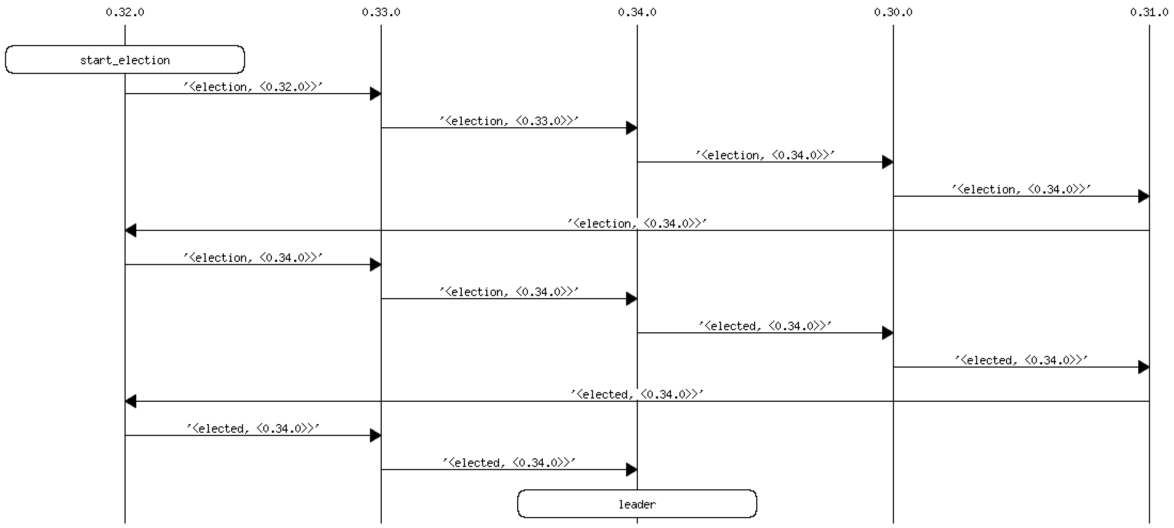

Abbildung 3: Wahlstart bei 0.32.0; Anzahl der Nachrichten: 12

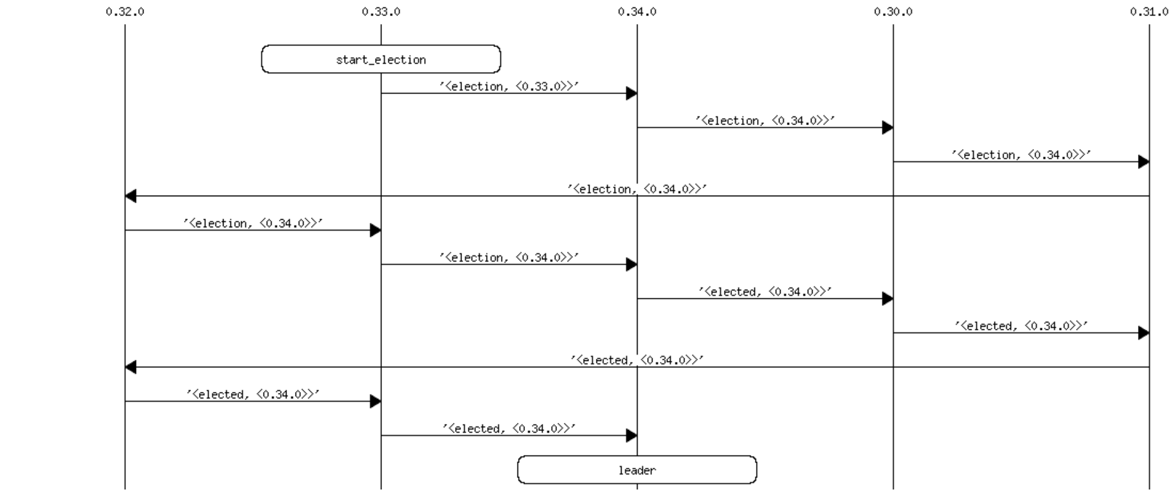

Abbildung 4: Wahlstart bei 0.33.0; Anzahl der Nachrichten: 11

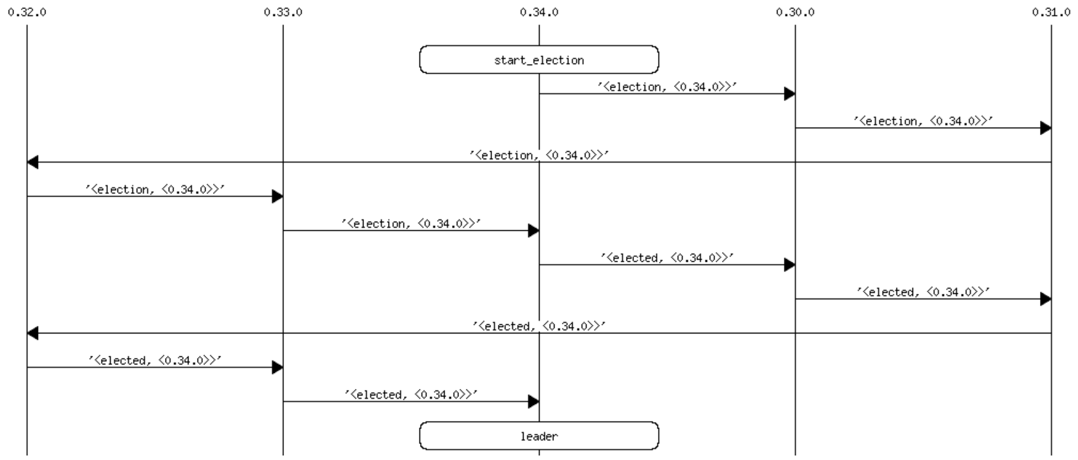

Abbildung 5: Wahlstart bei 0.34.0; Anzahl der Nachrichten: 10

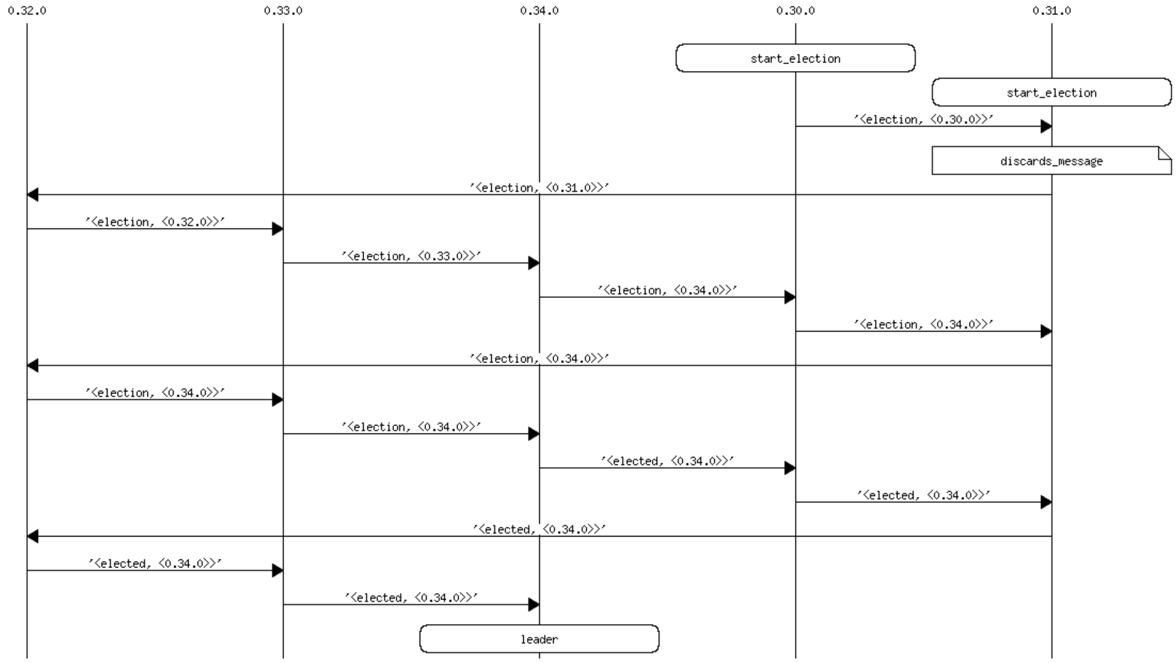

Abbildung 6: Wahlstart bei 0.30.0 und 0.31.0; Anzahl der Nachrichten: 14

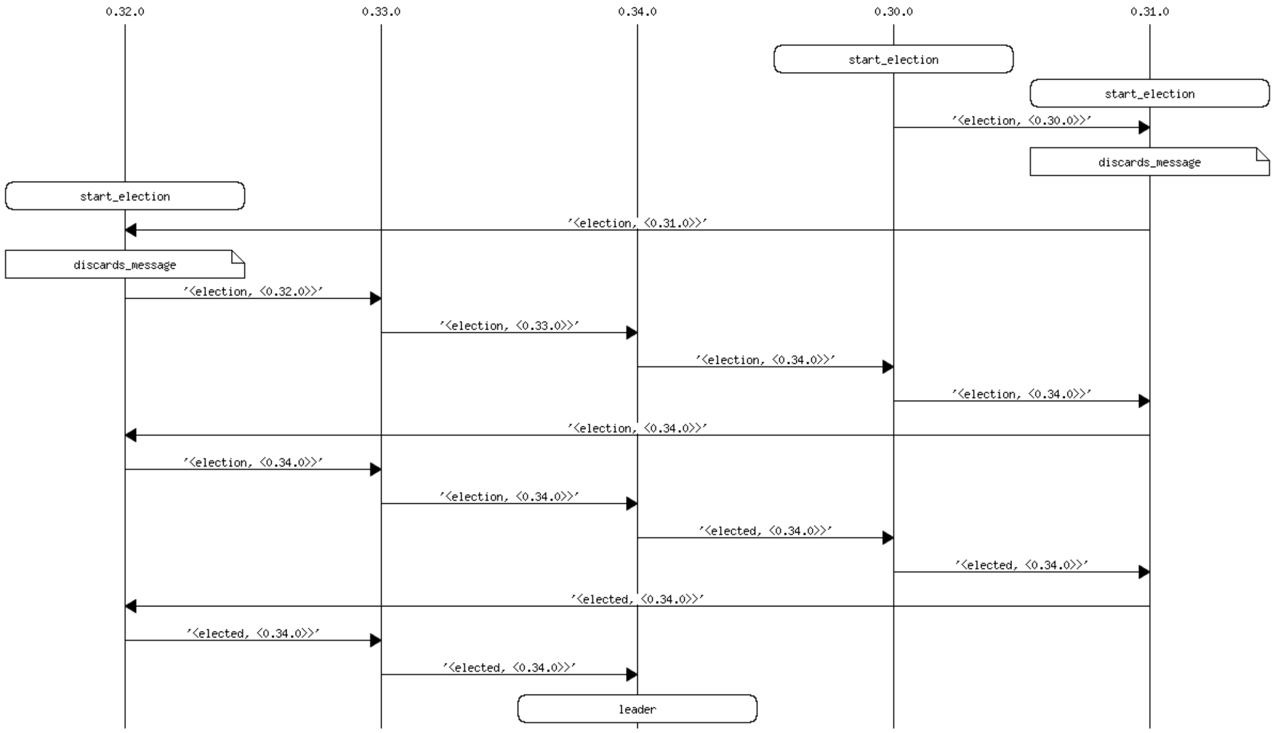

Abbildung 7: Wahlstart bei 0.30.0, 0.31.0 und 0.32.0; Anzahl der Nachrichten: 14

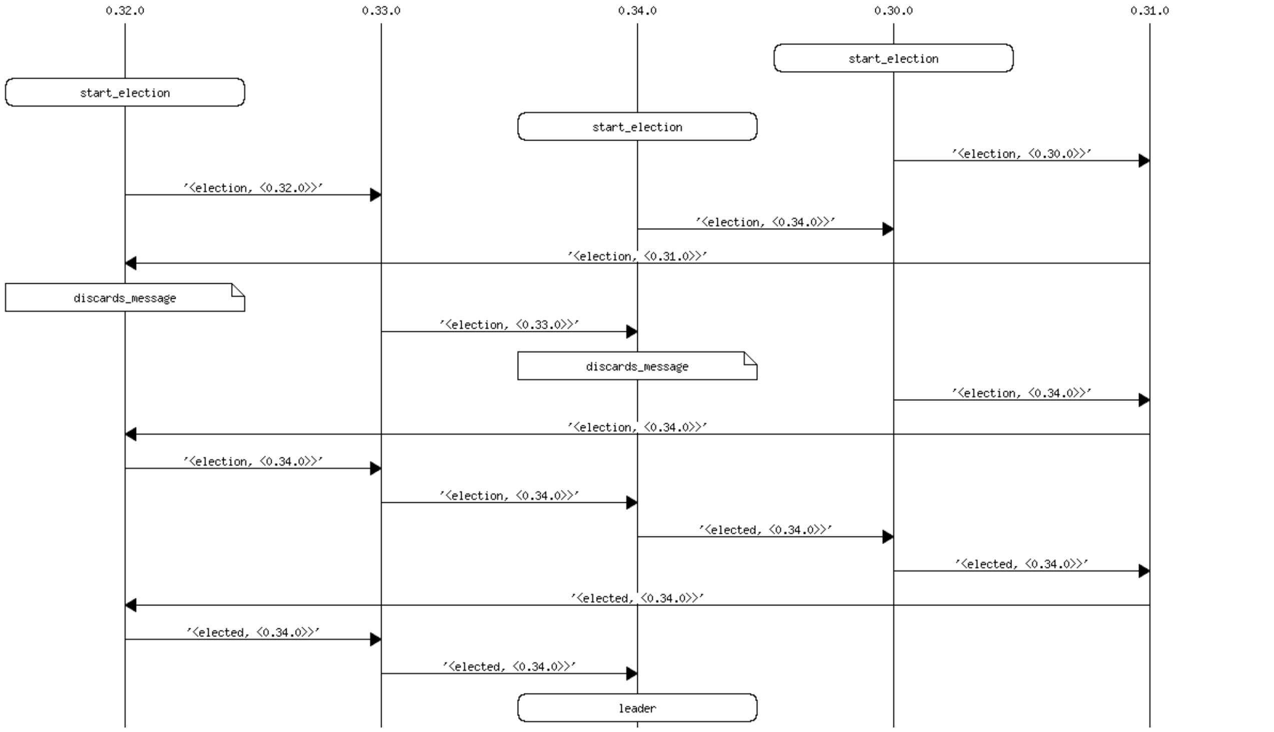

Abbildung 8: Wahlstart bei 0.30.0, 0.32.0 und 0.34.0; Anzahl der Nachrichten: 14

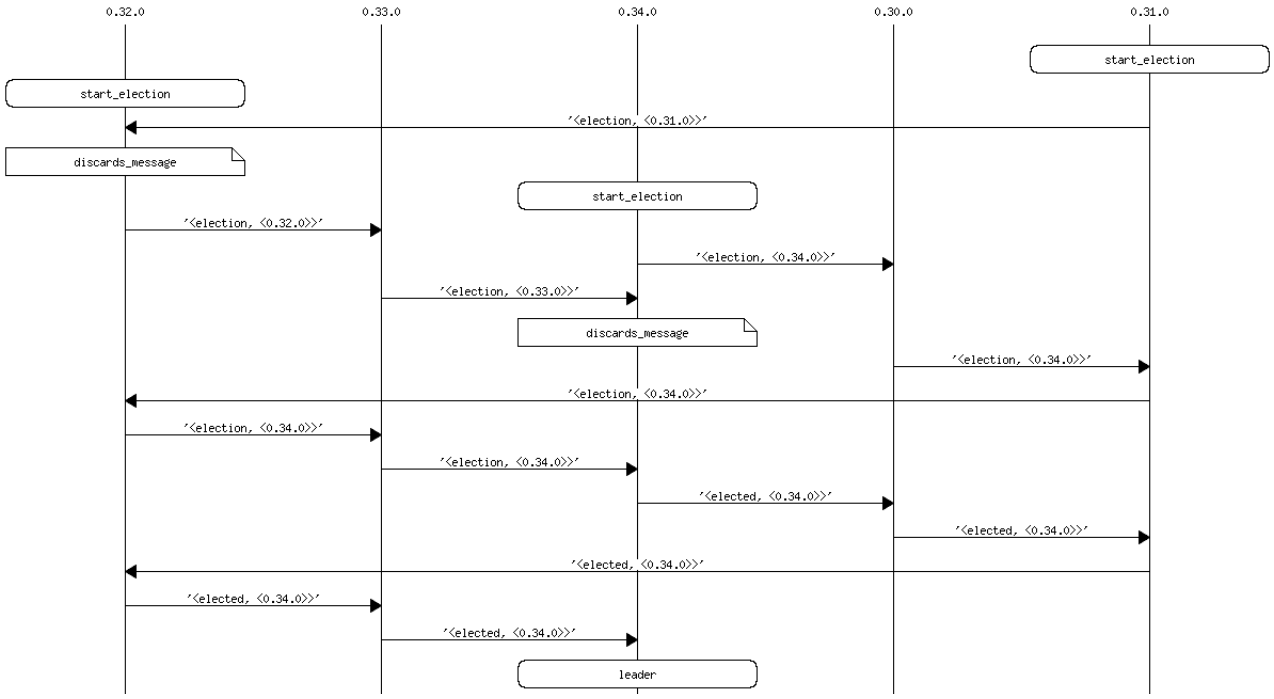

Abbildung 9: Wahlstart bei 0.31.0, 0.32.0 und 0.34.0; Anzahl der Nachrichten: 13