```
infrastructure infr // refers to infr.ginf

// Responder
process A {
    send {"arrived", comp.a} @ (true);
}

process D {
    loop {
        receive (proc.x == "dissolve" && proc.y == comp.partner) {x, y} [
            comp.rank := 2,
            comp.exPartner := comp.partner,
            comp.partner := -1
        ];
    }
}

process R {
    loop {
        receive (proc.x == "propose") {x, y, l, n};
        spawn(Rprime)
    }
}

process Rprime {
    if (comp.lock == 0 && rank(proc.y) < comp.rank){
        send{"accept", comp.idr, proc.n} @ (receiver.idi == proc.l) [comp.lock := 1];
        receive {
            case (comp.partner != -1 && proc.a == "ack" && proc.b == comp.idr && proc.c == proc.l){a, b, c}[
                comp.exPartner := comp.partner, comp.partner := proc.l
            ]:{
                send{"dissolve", comp.idr} @ (receiver.idi == comp.exPartner) [
                    comp.rank := rank(proc.y),
                    comp.lock := 0
                ] print("M $comp.partner$ with W $comp.idr$ !");
            }

            case (comp.partner == -1 && proc.a == "ack" && proc.b == comp.idr && proc.c == proc.l){a, b, c}[
                comp.lock := 0, comp.partner := proc.l, comp.rank := rank(proc.y)
            ] print("M $comp.partner$ with W $comp.idr$ !"):{
            }

            case (proc.a == "ack" && proc.b != comp.idr && proc.c == proc.l){a, b, c}[comp.lock := 0]:{
            }
        }
    } else if (comp.lock == 0) {
        set;
    }
}

// Initiator
process I {
    send{"propose", comp.a, comp.idi, comp.ref} @ (receiver.a == attributeForRank(comp.ref)) [comp.timer := 0];

    loop {
        if (comp.lock == 0 && !comp.success && comp.timer == comp.timeout){
            send{"propose", comp.a, comp.idi, next(comp.ref)} @ (receiver.a == attributeForRank(next
(comp.ref))) [
                comp.timer := 0,
                comp.ref := next(comp.ref)
            ];
        } else if (comp.lock == 0 && comp.dissolve) {
            send{"propose", comp.a, comp.idi, 0} @ (receiver.a == attributeForRank(0)) [
                comp.timer := 0,
                comp.ref := 0,
                comp.dissolve := false,
                comp.success := false,
                comp.rank := noRank(),
                comp.partner := -1
            ];
        } else if (comp.lock == 0 && comp.arrival && comp.bof <= comp.rank - 1 && comp.partner != -1) {
            send{"dissolve", comp.idi} @ (receiver.idr == comp.partner) [
                comp.success := false,
                comp.ref := comp.bof,
                comp.bof := noRank(),
                comp.rank := noRank(),
                comp.exPartner := comp.partner,
                comp.partner := -1
            ];

            send{"propose", comp.a, comp.idi, comp.ref} @ (receiver.a == attributeForRank(comp.ref))
[comp.timer := 0];
        }
    }
}

process T {
    loop {
        waitfor(comp.timer < comp.timeout)[comp.timer := comp.timer + 1];
        waitfor(100);
    }
}

process N {
    loop {
        receive(proc.x == "accept" && (proc.z != comp.ref || comp.success)) {x, y, z};
        spawn(Nprime)
    }
}

process Nprime {
    send{"ack", -1, comp.idi} @ (receiver.idr == proc.y);
}

process M {
    loop {
        receive {
            case (!comp.success && proc.x == "accept" && proc.z == comp.ref){x, y, z}[
                comp.lock := 1,
                comp.success := true,
                comp.rank := comp.ref,
                comp.exPartner := comp.partner
            ]:{
                send{"ack", proc.y, comp.idi} @ (true) [
                    comp.lock := 0,
                    comp.partner := proc.y
                ];
            }

            case (proc.x == "dissolve" && proc.y == comp.partner){x, y}[comp.dissolve := true]:{
            }

            case (proc.x == "arrived" && comp.bof >= rank(proc.y)){x,y}[
                comp.arrival := true,
                comp.bof := rank(proc.y)
            ]:{
            }
        }
    }
}

// Functions

function int next(int rank){
    return (rank + 1) % noRank()
}

function int noRank(){
    return 2
}

function int rank(int y){
    return y
}

function int attributeForRank(int rank){
    return rank
}

// Components

component {
    a := 0, // attribute value, 0 or 1
    partner := -1,
    exPartner := -1,
    idi := 0,
    ref := 0,
    success := false,
    arrival := false,
    dissolve := false,
    rank := 2,
    bof := 2,
    lock := 0,
    timer := 0,
    timeout := 20
} : I | T | M | N

environment E {
    a := 0, // attribute value, 0 or 1
    partner := -1,
    exPartner := -1,
    idr := 0,
    rank := 2,
    lock := 0
}
component E : A | R | D
```