# Multiagent Systems

## 7b. Jason: Contract Net Protocol

B. Nebel, C. Becker-Asano, S. Wölfl

Albert-Ludwigs-Universität Freiburg

June 20, 2014



# Multiagent Systems

June 20, 2014 — 7b. Jason: Contract Net Protocol



## Recommended reading

Bordini, Hübner, Wooldridge, "Programming multi-agent systems in AgentSpeak using Jason", Chapter 6.3, Wiley, 2007



## Communicating agents in AgentSpeak

We have learned how to use the following performatives:

- Information exchange:
  - tell
  - untell
- Goal delegation:
  - achieve
  - unachieve
- Information seeking:
  - askOne
  - askAll
- Exchanging means (plans):
  - tellHow
  - untellHow
  - askHow

How can we implement the Contract Net Protocol?

# 7b.1 Contract Net Protocol





# Contract Net Protocol

A UML diagram of the Contract Net Protocol:

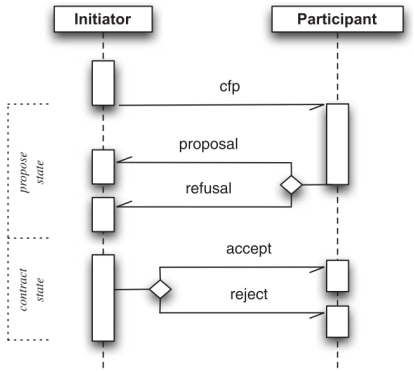


Figure 6.4 Contract net protocol.

(JasonBook page 131)





# CNP in Jason

The Jason implementation can be found in `<Jason_path>/examples/contract-net-protocol`.
It has the following MAS file `ContractNetProtocol.mas2j` (shortened a bit):

```
MAS cnp {
    agents:
        c;     // the CNP initiator
        p #3; // the participants (3)
        pr;    // a participant that always refuses
        pn;   // a participant that does not answer
}
```





# The simplest participant

The agent `pn` is the simplest participant:

```
// the name of the agent playing initiator in the CNP
plays(initiator,c).

// send a message to the initiator introducing myself
// as a participant
+plays(initiator,In)
   :  .my_name(Me)
   <- .send(In,tell,introduction(participant,Me)).
```

Although it registers with initiator, it never send `proposals` or `refusals`.

## The "always reject" participant

The agent pr:

```
plays(initiator,c).
+plays(initiator,In)
   :  .my_name(Me)
   <- .send(In,tell,introduction(participant,Me)).

// plan to answer a CFP
+cfp(CNPId,_Service)[source(A)]
   :   plays(initiator,A)
   <- .send(A,tell,refuse(CNPId)).
```

It registers with initiator and always rejects/refuses proposals.





## CFP procedure

A Call For Proposals (CFP) by agent c is processed by participants as follows:

1. Agent c uses `tell` perfomative to send a message `cfp` that has two arguments:
   - A "unique identification" so that participants can partake in more than one proposal at a time
   - A "task description" telling the agents what needs to be achieved
2. When received by participants, a belief (such as `cfp(1,fix(computer))[source(c)]` is added to belief base and a (similar) event to the event queue
3. Participants with a plan to react to this event will `refuse` to participate in the call or `propose` an offer
4. Agents not responding at all, need to be ignored by agent c ⇒ timeout necessary!





## Answering a CFP

The agents p answer a `cfp` by proposing an offer:

```
@c1 +cfp(CNPId,Task)[source(A)]
   : plays(initiator,A) & price(Task,Offer)
   <- // remember my proposal
      +proposal(CNPId,Task,Offer);
      .send(A,tell,propose(CNPId,Offer)).
```

and they react, when agent c sends an accept message:

```
@r1 +accept_proposal(CNPId)
   : proposal(CNPId,Task,Offer)
   <- .print("My proposal '", Offer,"' won CNP ",
             CNPId, " for ", Task,"!").
      // do the task and report to initiator
```





## Starting a CFP

The agent c uses the following plan to start a `cfp`:

```
+!startCNP(Id,Object)
  <- .wait(2000); // wait participants introduction
     +cnp_state(Id,propose); // remember state of CNP
     .findall(Name,introduction(participant,Name),LP);
     .print("Sending CFP to ",LP);
     .send(LP,tell,cfp(Id,Object));
     .concat("+!contract(",Id,")",Event);
     // the deadline of the CNP is now + 4 seconds,
     // so the event +!contract(Id) is generated
     // at that time
     .at("now +4 seconds", Event).
```

The `.at(...)` is important as a timeout to handle the non-responding agent(s)!