Lydia Ould Ouali[1], Charles Rich[2] and Nicolas Sabouret[1]

[1] LIMSI-CNRS, UPR 3251, Orsay, France
Université Paris-Sud, Orsay, France
{ouldouali, nicolas.sabouret}@limsi.fr
[2] Worcester Polytechnic Institute
Worcester, Massachusetts, USA
rich@wpi.edu

Abstract.

1 Algorithms

1.1 StatePreference

```
1: function isAcceptable(value, relation)
2:     if (relation = dominant) then
3:         return (score(value) > bestScoreOfPreference × 0.7)
4:     if (relation = peer) then
5:         return (score(value) > 0)
6:     if (relation = sub) then
7:         return (score(value) > 0 or isInOAS(value))
```

Fig. 1: Function to compute the acceptability of a value of preference

```
1: function ComputeAcceptableProposals(Value, relation)        ▷ Value is a
   criterion type (eg:cuisine,cost) or option(eg: restaurant)
2:     SelfAcceptable(Value)                    ▷ List of acceptable values. see Fig. 7
3:     OtherAcceptable(Value)    ▷ List of values which are acceptable for the user.
4:     proposals                                              ▷ ListofProposals
5:     if (relation = dominant) then
6:         for all (V : SelfAcceptable) do
7:             if V was not proposed in the negotiation then add(V) to proposals.
8:     if relation = peer or submissive then
9:         for all (V : OtherAcceptable) do
10:            if V was not rejected or accepted in the negotiation and isAcceptable(V)
   then add(V) to proposals.
11:    return proposals.
```

Fig. 2: Function to compute proposals values

1.2 Propose

```
1: function ComputeProposals(relation)
2:     Proposals = ComputeAcceptableProposals(currentDiscussedCriterion, relation)
3:     if (Proposals is empty) then
4:         Proposals = ComputeAcceptableProposals(OpenNewCriterion, relation)
   ▷ Open new criterion which has not been negotiated yet
5:     if (Proposals is empty) then Proposal = ComputeAcceptableProposals(Options, relation)
   ▷ compute acceptables options if all criteria have been already discussed
6:     return proposals.
```

Fig. 3: Function to compute proposals in a negotiation.

```
1: function reactToUser(less, more)
2:     if (more = MostPrefferedValue() and (∗, more) ∉ OAS) then
3:         return (∗, more)
4:     if (more = LeastPrefferedValue() and (more, ∗) ∉ OAS) then
5:         return (more, ∗)
6:     The conditions (2, 4) are also applied for less
7:     (less′, more′) = computePreference(less, more)   ▷ calculates a preference from a criterion
8:     if (less′, more′) ∈ OAS then
9:         (less1, more1) = reactToCriterion(more)
10:        if (less1, more1) = null then
11:            (less2, more2) = reactToCriterion(less)
12:            if (less2, more2) = null then
13:                return (less′, more′)
14:            elsereturn (less2, more2)
15:    return (less′, more′)
```

Fig. 4: Function to react to a preference (less,more)

```
1: function ReactToRejectableProposal((Proposal))
2:     if (Proposal isCriterionProposal) then
3:         return reactToCriterion(Proposal)
4:     if (Proposal isOptionProposal) then
5:         return reactToCriterion(getLeastScoredValue(Proposal))
6:                 ▷ getLeastScoredValue(Proposal) returns the criterion value of the proposal with the least score
```

Fig. 5: Function to compute a preference if the proposal is not acceptable.

```
1: function CounterPropose((Proposal))
2:     if (Proposal isCriterionProposal and isAcceptable(Proposal)) then
3:         return ComputeAcceptableProposal()   ▷ compute an acceptable option that contains the acceptable criterion
4:     if isNotAccepted(Proposal) then
5:         return computeProposal()   ▷ Compute a counter proposal
```

Fig. 6: Function to compute a counter proposal.

```
1: function canPropose(())
2:     if isNotSub() and isCriterionProposal(LastAcceptedProposal) then
3:         return true
4:     if isSub() and openNewCriterion = null then
5:         return true
```

Fig. 7: Function to compute a preference if the proposal is not acceptable.