CooL-AgentSpeak Short Manual

MAS implementation

Your MAS configuration file (the one with .mas2j extension) should look like the following:

```
MAS test {

    infrastructure: Centralised
      environment: custom.env.OntologyEnvironment // this line is mandatory

    agents:
    finance1
        [
        jasdl_ontologies="ont1", // identifier of the only ontology known by
finance1
        jasdl_ont1_uri="/onts/Scenario3-JWNL/onto1.rdf" // path of the ontology
(note that the "ont1" identifier of the ontology, appears in the jasdl_ont1_uri
parameter name: this pattern is mandatory)
        ]
        agentArchClass jasdl.architecture.JASDLAgArch // this line is mandatory
        agentClass custom.asSemantics.CustomAgent     // this line is mandatory
        beliefBaseClass jasdl.bb.JASDLBeliefBase;     // this line is mandatory
    finance2
        [
        jasdl_ontologies="r,p", // identifiers of the ontologies (two in this case)
        jasdl_r_uri="/onts/room.owl", // path of the first ontology, with "r" in
the parameter name
        jasdl_p_uri="/onts/Ontology.owl" // path of the second ontology, with "p"
in the parameter name)
        ]
        agentArchClass jasdl.architecture.JASDLAgArch // this line is mandatory
        agentClass custom.asSemantics.CustomAgent     // this line is mandatory
        beliefBaseClass jasdl.bb.JASDLBeliefBase;     // this line is mandatory

      classpath:  "../lib/*.jar"; "../"; // path to both the jar files and to the
upper directory, where the configuration files file_properties.xml and oa.config
are saved

      aslSourcePath: "asl/Scenario3-JWNL"; // path of the directory where the
source files of the agents are saved

}
```

CooL-AgentSpeak Agents implementation

Your agents (agent.asl) should look like the following:

```
{include("../cool.asl")} // include the file cool.asl, provided in the .zip file

/*********************/
/** CooL parameters ***/
```

```
/**********************/

// you always need to specify all the CooL parameters

cooperationStrategy(
      trustedAgents([finance2]), // list of names of trusted agents
      retrievalPolicy(always),   // retrieval policy may be always or noLocal
      acquisitionPolicy(add),    // acquisition policy may be add or replace
      threshold(0.6), // threshold below which correspondences are discarded
      timeout(4000),  // milliseconds an agent is willing to wait for getting an
answer from a trusted agent
      alignment_method(jwnl) // alignment method may be jwnl (Alignment API) or
aroma
).

!init.
!start.

+!init : true <-
      !
registered_ontology("file:///home/mascardi/Documents/Software/Cool2.4/Cool2.0/onts/
Scenario3-JWNL/onto1.rdf", ont1). // you should register the known ontologies here


+!start : true <-
      .wait(1000);
      .print("Agent finance1 starting...");
      .wait(1000);
        custom.stdlib.get_current_ms(MS1);
      .print("MS1 = ", MS1);
      !nationalBank(NB1)[o(ont1)]; /* concept in ont1 with one match in ont2, no
plans locally available; plans available from trusted agents */
        !nationalBank(NB2)[o(ont1)];

        !partyPrincipal(PP1)[o(ont1)]; /* concept in ont1 with one match in ont2,
no plans locally available; plans available from trusted agents */
        !partyPrincipal(PP2)[o(ont1)];

        !shortTermLiabilities(STL1)[o(ont1)]; /* concept in ont1 with one match in
ont2, no plans locally available; plans available from trusted agents */
        !shortTermLiabilities(STL2)[o(ont1)];

        !primaryMarketOrderFee(IR1)[o(ont1)]; /* concept in ont1 with one match in
ont2, plans locally available; plans available from trusted agents */
        !primaryMarketOrderFee(IR2)[o(ont1)];

        !clientSecuritiesAccount(Acc1)[o(ont1)]; /* concept in ont1 with one match
in ont2, plans locally available; plans available from trusted agents */
        !clientSecuritiesAccount(Acc2)[o(ont1)];

        !liquidityRiskProfile(LRP1)[o(ont1)]; /*  concept in ont1 with one match in
ont2,  plans locally available; no plans available from trusted agents */
        !liquidityRiskProfile(LRP2)[o(ont1)];

        !referentialSystem(RS1)[o(ont1)]; /*  concept in ont1 with one match in
ont2,  plans locally available; no plans available from trusted agents */
        !referentialSystem(RS2)[o(ont1)];

        !conceptWithNoMatch1(C11)[o(ont1)]; /* concept in ont1 with no match in
```

```
ont2,  plans locally available */
        !conceptWithNoMatch1(C12)[o(ont1)];
        !conceptWithNoMatch2(C21)[o(ont1)]; /* concept in ont1 with no match in
ont2,  plans locally available */
        !conceptWithNoMatch2(C22)[o(ont1)];
        !conceptWithNoMatch3(C31)[o(ont1)]; /* concept in ont1 with no match in
ont2,  plans locally available */
        !conceptWithNoMatch3(C32)[o(ont1)];
        custom.stdlib.get_current_ms(MS2);
      .print("MS2 = ", MS2);
      ?mydiff(MS1, MS2, TotalTime);
        .print("FMA, Total execution time = ", TotalTime).

+!clientSecuritiesAccount(3234448)[o(ont1),source(self)] <-
      .print("Using finance1 plan for clientSecuritiesAccount").

+!primaryMarketOrderFee("My primary Market order fee")[o(ont1),source(self)] <-
      .print("Using finance1 plan for primaryMarketOrderFee").

+!liquidityRiskProfile(low)[o(ont1),source(self)] <-
      .print("Using finance1 plan for liquidityRiskProfile").

+!referentialSystem(refSys)[o(ont1),source(self)] <-
      .print("Using finance1 plan for referentialSystem").

+!conceptWithNoMatch1(only_local)[o(ont1),source(self)] <-
      .print("Using finance1 plan for conceptWithNoMatch1").

+!conceptWithNoMatch2(only_local)[o(ont1),source(self)] <-
      .print("Using finance1 plan for conceptWithNoMatch2").

+!conceptWithNoMatch3(only_local)[o(ont1),source(self)] <-
      .print("Using finance1 plan for conceptWithNoMatch3").
```

JASDL Agents implementation

You can simulate pure JASDL agents (that exploit neither Coo-BDI features given by plan exchange, nor ontology matching features) by setting the list of trusted agents to the empty list [] in the cooperation strategy. Note that the belief defining the cooperation strategy must be set anyway for each agent, otherwise a MAS failure will take place.