Management of Multi-Agent Systems by Means of Automated Reasoning

Ondřej Kazík

Faculty of Mathematics and Physics, Charles University in Prague

January 12, 2011
Multi-Agent Systems

- Agent
  - Situatedness
  - Autonomy
- Agent societies
  - Distributed systems
  - Collaboration
  - Delegacy of subproblems
- Development of large and open MAS
Roles

- Insipired by sociological concept
- Roles in other fields, e.g.
  - OOP
  - Design patterns
  - Computer-to-human interface
  - Security
- Organizational view of MAS
- Separation of concerns
- Reusability

- Assignment by agents
- Expected behavior of agent
  - Capabilities (rights)
  - Obligations (duties)
- Interface between agent and environment
- Definition of interactions
Case Study: Computational Intelligence

- **Hybrid computational models**
  - Combination of AI methods (ANN, EA, FLC)
  - Benefits over individuals methods
  - No underlying theory, importance of heuristics
  - Complexity of system, unpredictable data exchange between components

- **Computational MAS**
  - Agents encapsulation computational algorithms
  - Distributed execution
  - Interchangeability of computational methods
  - Autonomous behavior
Case Study: Computational Intelligence

- Computational MAS example
  - Data classification problem
  - Training data provided by data source agent
  - Classification algorithm (e.g. RBF net) employed
  - Different learning algorithms (inverse pseudofunction, gradient method, VQ, EA)
  - Supplementary agents

- AGR model
  - Modular model
  - Flexibility of role-based approach
Organizational Structure

- **Computational group**
  - Task manager represents the problem
  - It employs computational agent
  - CA asks for training data

- **Simple learning group**
  - Computational agent requests for training a learning algorithm

- **Evolutionary alg. group**
  - Evolutionary methods require supplementary agents
  - Representation of chromosome
  - Fitness function
  - Settings of EA algorithm
• Cheeseboard diagram
• System starts with computational group
• After learning request, RBF net accepts Learned CA and Evoluted CA roles and establish corresponding groups
• At the end, these groups can be terminated and the resulting model is tested
Description Logic Model of Roles

- Agents, Groups – individuals in assertional box (A-Box)
- Roles – classes of the agents
  - Initiators (capabilities)
  - Responders (obligations)
- Abstract classes (concepts) in terminological box (T-Box):
  - Agent
  - Group
  - Initiator
  - Responder
- Properties (roles):
  - hasInitiator, inverse isInitiatorOf
  - hasResponder
  - hasAgent
  - sendsTo
Integrity constraints

- **OWL** adopts the Open World Assumption (OWA)
  - a statement cannot be inferred to be false on the basis of a failure to prove it
- Data validation needs CWA
- IC validator extending Pellet reasoner
- Constraint:
  
  $$\text{Product} \sqsubseteq_C \exists \text{isManufacturedBy}.\text{Manufacturer}$$

- This is translated into SPARQL query:
  
  ```sparql
  ASK WHERE {
    ?x rdf:type Product.
    OPTIONAL {
      ?x isManufacturedBy ?y.
      ?y rdf:type Manufacturer.
    }
    FILTER(!BOUND(?y))
  }
  ```
Schema axioms (OWA)

- Responder class and individual:
  \[ \text{ControlMsgResponder} \sqsubseteq \text{Responder} \]
  \[ \text{ControlMsgResponder}(\text{ControlMsg}) \]

- Initiator class definition:
  \[ \text{DataMsgInit} \sqsubseteq \text{RequestInit} \]

- Role definition:
  \[ \text{CompAgent} \sqsubseteq \text{Agent} \]
  \[ \exists \text{hasResponder}.\text{ControlMsg} \]

- Group structure definition:
  \[ \text{CompGroup} \sqsubseteq \text{Group} \]
Integrity Constraints (CWA)

- Request initiator:

\[ \text{RequestInit} \sqsubseteq C \sqsubseteq \text{sendsTo}.1 \]

- Allowed roles for initiator of communication:

\[ \text{DataMsgInit} \sqsubseteq C \ \forall \text{sendsTo} \exists \text{hasResponder}.\text{DataMsgResp} \]
\[ \sqcap \ \forall \text{isInitiatorOf}.\text{CompAgent} \]

- Allowed members in a group:

\[ \text{CompGroup} \sqsubseteq C \ \forall \text{hasAgent}.\]  
\[ (\text{CompAgent} \sqcap \text{TaskManager} \sqcap \text{DataSource}) \]
A-Box - Actual State

- Role assignment:
  
  \[\text{CompAgent}(RBF)\]

- Group definition:
  
  \[\text{CompGroup}(\text{Group}_1)\]

- Group membership:
  
  \[\text{hasAgent}(\text{Group}_1, RBF)\]

- Establishing of connection:
  
  \[\text{DataMsgInit}(\text{Initiator}_1)\]
  
  \[\text{hasInitiator}(RBF, \text{Initiator}_1)\]
  
  \[\text{sendsTo}(\text{Initiator}_1, \text{ARFFReader})\]
Ontology Agent

- **Goals:**
  - Representation of current state
  - Correctness verification
  - Matchmaking
- **Communication ontology**
  - Actions
  - Queries
  - Result informations
Implementation: Creating Schema Inference Model

```
// create an empty ontology model using Pellet spec
o_model=ModelFactory.createOntologyModel(
    PelletReasonerFactory.THE_SPEC);
// read the schema file
o_model.read("AGR_Schema.owl");
```
protected boolean ontoIC(){
    // create the IC validator and
    // associate it with the ontology
    JenaICValidator validator = new JenaICValidator(o_model);
    // load the constraints into the IC validator
    validator.getConstraints().read("AGR_ICs.owl");
    // get the constraint violations
    Iterator i=validator.getViolations();
    if(i.hasNext()) {
        return false;
    }else{
        return true;
    }
}
void ontoEnterGroup(String grp_name, String ag_name) {
    Individual grp_indiv = o_model.getIndividual(grp_name);
    Property hasAgent_role = o_model.getProperty(onto_ns + "hasAgent");
    Individual ag_indiv = o_model.getIndividual(ag_name);
    grp_indiv.addProperty(hasAgent_role, ag_indiv);
}
Implementation: Querying with SPARQL

```java
Query query = QueryFactory.create(queryStr);
// create a query execution engine with a Pellet model
QueryExecution qe =
    QueryExecutionFactory.create(query, o_model);
// run the query
ResultSet results = qe.execSelect();
// query results
List result_ags = new LinkedList();
while(results.hasNext()){
    QuerySolution solution = results.next();
    String agURI = solution.getResource("?Agent").getURI();
    result_ags.add(agent_addrs.get(agURI));
}
```