Defining Ontologies in a Multi Agent Scenario Using the JADE Framework

Dominik Fabrici

dominik.fabrici@tuke.sk; Technical University of Košice, Slovakia

Abstract: This paper describes the way in which JADE, a multi agent development framework, deals with the problem of message content consistency in inter agent communication by means of providing content language and ontology support features for syntactic and semantic validation of content expressions. Included is the describtion of a usecase example ontology created to support and validate the information exchange inside of a group of maze navigating reactive agents used for demonstration purposes.

1 Introduction

The communication between any number of agents in a multi agent system comprises the exchange of small amounts of information in the form of a message, which had been preformated to comply with the conventions of an agent communication language (ACL) which was agreed upon by said agents beforehand, sooner than any actual data is sent. This data, embeded into an ACL message, constitutes it's content and expressed in a suitable content language forms a content expression.This representation, usually a string or a byte sequence, cannot be considered convenient for internal purposes of an agent, where data is usually stored as data objects.

Each information exchange taking place requires the part taking agents to convert their internal representations of the information exchanged into an equivalent content expression which gets wrapped inside an ACL message before being sent to the receiver, who performs the same operation in reverse with the additional overhead of semantic checks to see whether the received message’s content can be interpreted, that is whether it is meaningful. This procedure gains in importance when designing open applications, where not all agents' messages are expected to carry semantically correct and consistent content.

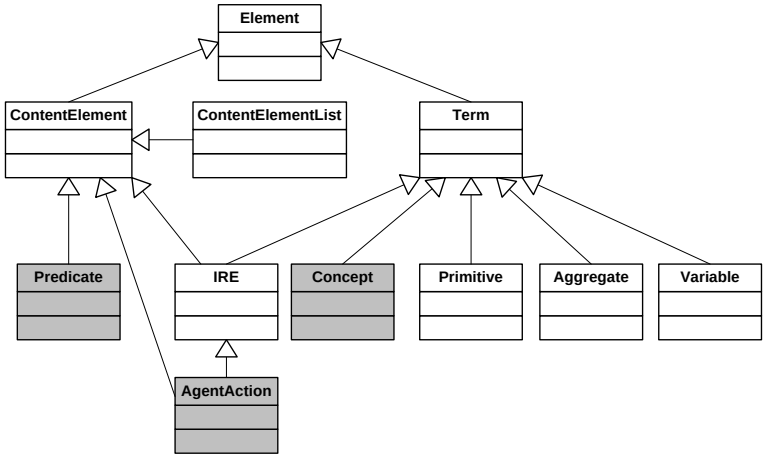
A meaningful content conforms to a set of predefined rules of an ontology. With a properly defined ontology a programmer can leverage the power of automatic message content validation provided by a FIPA compliant agent framework like JADE for example.

In JADE as an example, the conversion and validation operations are performed by a dedicated content manger objct. The content manager object's class provides all the necessary methods for content conversion, but in reality delegates all the work done to special ontology and content codec object instances. While the codec performs syntactic translations with direct support for two content languages (one defined in a FIPA specification – a human readable string-encoded SL content language [1], the other a non-readable byte-code encoded LEAP language defined solely for JADE agent interaction), the ontology validates all information from a semantic perspective, which requires all elements of a content expression to be known and classified. With this information, an ontology basically serves as a dictionary or vocabulary for inter agent communication.

FIPA's agent communication language specification dictates all messages to have a semantic conforming to its performative (type of action taken), where the content reference model discerns between predicates (status describing expression holding a truth value) and terms (expressions describing entities from the world where they exist and are the subject of agent discussions).

Terms can be further categorised into six types [2]:

- concepts (structured entities consisting of data slots)
- agent actions
- primitives (primitive data-type entities)
- aggregates (groups of other entities)
- identifying reference expressions (expressions using a predicate as the identification criterion, typicaly used in queries)
- variables (elements unknown beforehand)



Picture 1
The content reference model

An ontology defines the structure of a domain's elements by means of schemas. Specifficaly, schemas need to be defined for three types of elements, those being predicates, concepts and agent actions, while a class needs to be implemented for each of these elements defined in the ontology. Because an ontology as a collection of schemas typically doesn't evolve during the life cycle of an agent, it's usually defined as a singleton object that can be shared among all agents existing on a platform run by a single java virtual machine. User defined ontologies extend the basic ontology, which defines schemas for all primitive typs, the aggregate type and some framework speciffic generic types. Each schema included in an ontology is associated to a corresponding class that is expected to be coherent with the schema by implementing a proper interface, by having the correct inheritance relations and by having the correct member fileds with accessor methods. Each slot in a schema is defined by name and data type, which is significant for later validation purposes. Slots in a schema are declared either optional or mandatory, signifying the requirements for this piece of data to be filled for a message to be declared valid. Slots can also have a cardnality larger than one, meaning that more than one element is expected to be inserted into them. Schemas themselves allow to be hierarchisized in a “specializes/extends” relationship among concepts, which allows for the creation of so called super schemas.

The validation of message content against the schemas of the ontology used can also be disabled in situations, where its small performance impact is considered unnecessary, which is especially true when developing closed applications where content expressions are expected to be consistent.

As already mentioned, a user defind ontology in JADE usually extends tha basic, platform speciffic, ontology but is not constrained to that, as it can extend a number of predefined ontologies, not strictly related to that particular domain.

Abstract decriptors are another, more general, way of representing content expressions. They consist of a type name defining the element's type and of a number of named slots, which hold the attributes of the mentioned element. Each type of element from the content reference model has its own predefined abstract descriptor class (Predicate, AbsConcept, AbsAgentAction) and all primitives (atomic elements) are instances of the AbsPrimitive class.

The developer is free to use both types of descriptors, user defined classes or abstract descriptors, there are situations though which require the use of abstract discriptors, like situations when dealing with queries which typically consist of an abstract IRE and an abstract variable.

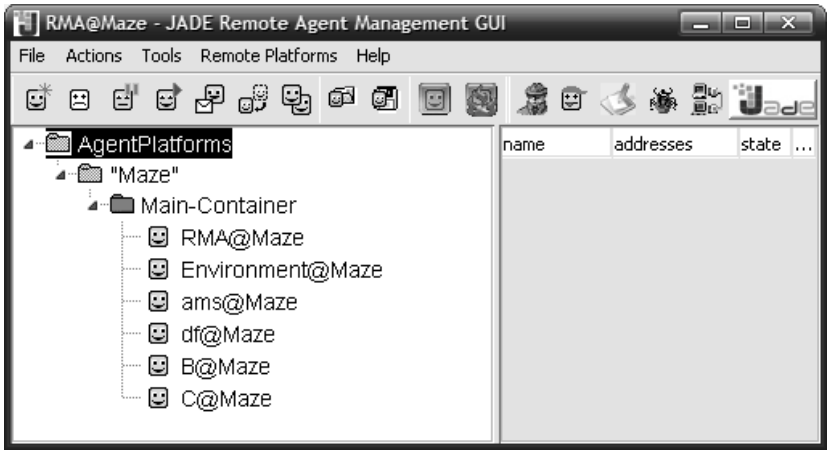
JADE's content language and ontologies support also allows the user to add additional constraints to predicates, concepts and agent actions. These constraints are reffered to as facets and adding a facet to an alement requires only defining a new class that implements the Facet interface.

2 The Usecase

The following example illustrates the use of a simple, custom defined ontology that enables a flowless exchange of messages between agents navigating an unknown maze and an environment agent who solely shares information about the environment which it posesses. This is a typical representation of a shared information space role model [3] which consists of one centralized information resource and of a number of subsriber roles.

The Environment agent stores a model of a simple maze, his resposibility is to update the location of agents navigating the maze and to provide them with information about their immediate surroundings.

Tha Navigator agents are pre-programed with the task of finding an exit and escaping the maze. They are practicaly blind in respect to their surroundings, have no maze representation available and completely rely on information provided to them by the Environment agent. Each Navigator agent is equiped with a simple reactive wall following behaviour (right-hand rule) which uses as its inputs information requested from and provided by the Environment-agent. To take advantage of agent collaboration and to extend the usability of the mentioned behaviour beyond simply connected mazes, each Navigator-agent leaves behind a signature, a so called pheromone trail that informs other agents and the trail originator as well, about already scouted territory - this information is also managed by the Environment-agent.

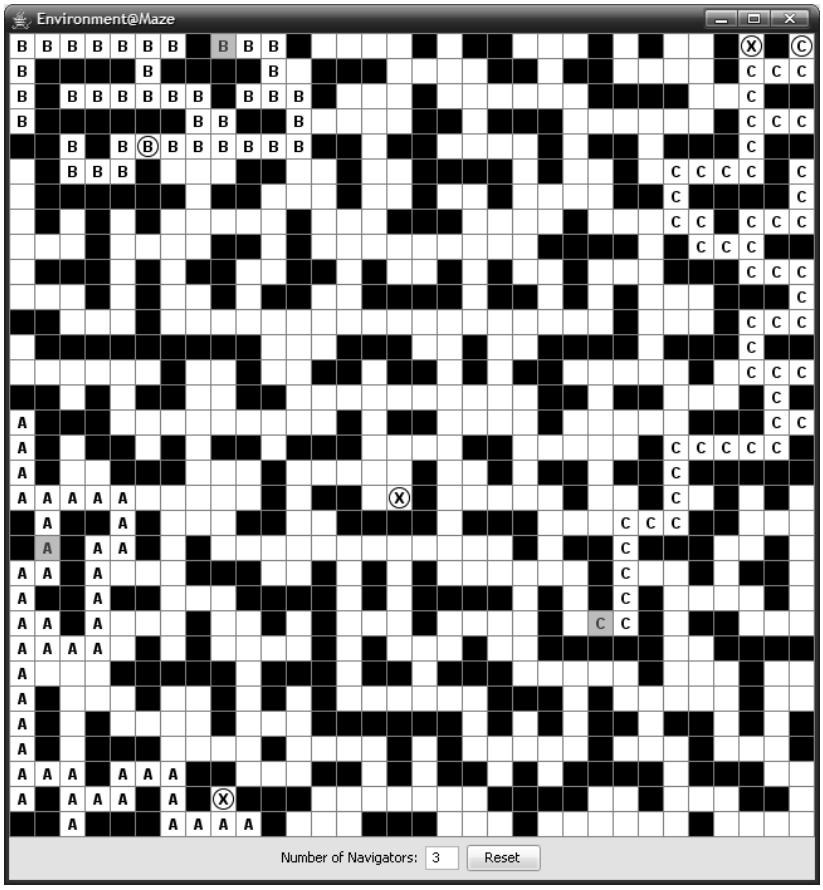


Picture 2
Remote Monitoring Agent's GUI

Other agents can use this information in a situation when they cross the trail left by a successful agent, in which case they abandon their own progress and follow

the mentioned trail. Picture number 3 depicts the graphical representation of the information stored by the Environment-agent after 61 consecutive steps by each navigator (marked with a circled letter).

Picture 4 shows both remaining agent's GUIs which show all information these agents possess after querying the Environment-agent for their location. Their heading direction is the result of the last move made and directly influences the order in which the next move direction is being decided about.

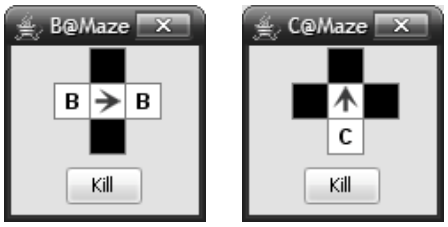


Picture 3

Environment agent's GUI

In case an agent finds one of the available exits (marked with a circled 'X'), it sends a message addressed to all registered subscriber agents (other navigators)

informing them about its success before deregistering and deleting itself from the container/agent platform.



Picture 4
Navigator agents' GUIs

Agent 'A' already left the maze successfully and is going to be followed by navigator agent 'C' after 4 more steps, considering the rulebase described previously. The starting positions chosen randomly are marked by a darkened trail for each agent respectively.

3 Ontology Definition

Considering the usecase scenario described, we can now with certainty describe all communication acts which are expected to take place in the domain of this multi agent environment. Appart from the agent platform inherent messages exchange which is framed by the FIPA defined agent management ontology [4], we want to define a custom ontology to support all application defined object tranlation and validation tasks.

We assume that all communication is initiated by the Navigato-agents where these agents request actions to be performed (locate the navigator in maze, move the navigator) by the Environment-agent in a discourse following the FIPA-request [5] protocol, with a predicate usefull for querying other navigator status. Taking into account all of this, we can define our vocabulary as follows:

```
// Concepts
public static final String NAVIGATOR = "NAVIGATOR";
public static final String NAVIGATOR_NAME = "navigator-name";
//
public static final String MAZE_CELL = "MAZE-CELL";
public static final String MAZE_CELL_OBSTACLE = "obstacle";
public static final String MAZE_CELL_SCENT = "scent";
//
public static final String MAZE_LOCATION = "MAZE-LOCATION";
public static final String MAZE_LOCATION_NORTH = "north";
```

```
public static final String MAZE_LOCATION_SOUTH = "south";
public static final String MAZE_LOCATION_EAST = "east";
public static final String MAZE_LOCATION_WEST = "west";
public static final String MAZE_LOCATION_HERE = "here";
// Predicates
public static final String EXITED = "EXITED";
public static final String EXITED_NAVIGATOR = "navigator";

// Actions
public static final String LOCATE = "LOCATE";
public static final String LOCATE_NAVIGATOR = "navigator";
//
public static final String MOVE = "MOVE";
public static final String MOVE_DIRECTION = "direction";
```

The ontology isn't complete without each concept, predicate and agent action having a schema defined and assigned to it, an exapmle of which would look something like this for our case:

```
add(new ConceptSchema(NAVIGATOR), Navigator.class);
add(new ConceptSchema(MAZE_CELL), MazeCell.class);
add(new ConceptSchema(MAZE_LOCATION), MazeLocation.class);
add(new PredicateSchema(EXITED), Exited.class);
add(new AgentActionSchema(LOCATE), Locate.class);
add(new AgentActionSchema(MOVE), Move.class);
ConceptSchema cs;
cs = (ConceptSchema)getSchema(NAVIGATOR);
cs.add(NAVIGATOR_NAME,
  (PrimitiveSchema)getSchema(BasicOntology.STRING),
  ObjectSchema.MANDATORY);
cs = (ConceptSchema)getSchema(MAZE_CELL);
cs.add(MAZE_CELL_OBSTACLE,
  (PrimitiveSchema)getSchema(BasicOntology.STRING),
  ObjectSchema.MANDATORY);
cs.add(MAZE_CELL_SCENT,
  (PrimitiveSchema)getSchema(BasicOntology.INTEGER),
  ObjectSchema.OPTIONAL);
cs = (ConceptSchema)getSchema(MAZE_LOCATION);
cs.add(MAZE_LOCATION_NORTH,
  (ConceptSchema)getSchema(MAZE_CELL), ObjectSchema.MANDATORY);
cs.add(MAZE_LOCATION_SOUTH,
  (ConceptSchema)getSchema(MAZE_CELL), ObjectSchema.MANDATORY);
cs.add(MAZE_LOCATION_EAST,
  (ConceptSchema)getSchema(MAZE_CELL), ObjectSchema.MANDATORY);
cs.add(MAZE_LOCATION_WEST,
```

```
  (ConceptSchema)getSchema(MAZE_CELL), ObjectSchema.MANDATORY);
cs.add(MAZE_LOCATION_HERE,
  (ConceptSchema)getSchema(MAZE_CELL), ObjectSchema.MANDATORY);
//
PredicateSchema ps = (PredicateSchema)getSchema(EXITED);
ps.add(EXITED_NAVIGATOR, (ConceptSchema)getSchema(NAVIGATOR));
//
AgentActionSchema as;
as = (AgentActionSchema)getSchema(LOCATE);
as.add(LOCATE_NAVIGATOR,
  (ConceptSchema)getSchema(NAVIGATOR), ObjectSchema.MANDATORY);
//
as = (AgentActionSchema)getSchema(MOVE);
as.add(MOVE_DIRECTION,
  (PrimitiveSchema)getSchema(BasicOntology.INTEGER),
  ObjectSchema.MANDATORY);
```

Conclusions

By defining a proper ontology we can take full advantage of JADE's content manager but its ontologies support offers much more functionality when working with abstract descriptors to create queries, with facets to constrain concepts, predicates and agent actions, or with custom defined introspectors for translation of abstract descriptors which are tasks beyond the scope of this usecase.

References

[1] FIPA SL Content Language Specification, Foundation for Intelligent Physical Agents, 2000, http://www.fipa.org/specs/fipa00008/

[2] Application-Defined Content Languages and Ontologies, JADE documentation, available from http://jade.tilab.com/

[3] E. Kendall et al., "Patterns of Intelligent and Mobile Agents", Proceedings of Autonomous Agents '98, ACM Press, 1998

[4] FIPA Agent Management Specification, Foundation for Intelligent Physical Agents, 2004, http://www.fipa.org/specs/fipa00023/

[5] FIPA Interaction protocols, Foundation for Intelligent Physical Agents, 2002, http://www.fipa.org/specs/fipa00026/