periodica polytechnica

Electrical Engineering
51/3-4 (2007) 147–156
doi: 10.3311/pp.ee.2007-3-4.09
web: http://www.pp.bme.hu/ee

RESEARCH ARTICLE

An agent-based framework for supporting e-health: from OWL description to MAS

Gianfranco Pedone

Received 2007-10-03

Abstract

The increasing level of elder patients across the European countries obliges governments to face health services in a more flexible and uniform manner. The idea of this research project is to prospect the possibility of building an intelligent agent-based ICT (Information & Communication Technology) platform in order to grant home care services close to the patient's needs. In particular, this paper presents the unexplored possibility to add an intelligence automation to the implementation, creation and deployment of agents participating the e-Health MAS, starting from their Ontology Web Language description. This approach will convey the advantages inherent both the involvement of semantic representations and the agent paradigm.

Keywords

Ontologies · multi-agent systems · intelligent e-health · automatic agent creation · automatic agent deployment

Gianfranco Pedone

Computer and Automation Research Institute, Hungarian Academy of Sciences, Lágymányosi u 11 1111 Budapest, Hungary
e-mail: gianfranco.pedone@sztaki.hu

1 Introduction

Multi-Agent Systems (MAS) in health-care domains are showing a progressive and rapid increase, in order to manage complex tasks and adapt gracefully to unexpected events. On the other hand, the lack of well-established and industry-wide accepted MAS development methodologies represents a serious slack in the development process of a MAS. This inhibits the possibility to grant a sufficient degree of automation in the creation and population of a Multi-Agent System.

Multi-Agent Systems (MAS) can represent a real technological core competence for an organization in the actual panorama of artificial systems [22]. The enormous circulation of information within all application domains obliges actors involved into a specific context to find intelligence solutions [23] in order to support decision-making processes. This also holds, with a higher emphasis, for those ones in health services area.

The work realized and presented in this paper is part of a much wider project, which aims at building an intelligent agent-based platform for granting uniform and well-suited care services for elder people, across all European countries. In particular way, in this paper we will present an approach for automatically building and setting up a Multi-Agent System's[10] preliminary structure, starting from its ontological description.

The objective is to extract all necessary knowledge concerning the MAS described by the Ontology Web Language[26] (OWL) and delegate the composition and deployment of agents to an automation framework. As previously stated, the lack of well-established development methodologies[7] causes an important slow down in performance and it does not ease the engineering process of analysis, design and implementation of a MAS. Despite this, we will adopt some precise development premises in order to suggest an ontological representation roadmap of the system.

We consider of fundamental importance the introduction of some key concepts which will clarify their use within this work. This paper will be structured as follows:

- in the course of next section, we justify the reason for the use of agents and their most important characteristics, their

agglomeration into MAS and the need of ontologies for a semantic description of artificial resources;

- in section three, we propose all technological elements involved into the realization of the OWL-to-MAS framework;
- in the subsequent section we will expose in details the philosophy and the functioning of the mentioned framework, focusing on obtained outcomes;
- we will conclude the presentation of this paper by highlighting the obtained results and tracing future works.

2 Why the agent paradigm?

Artificial Intelligence (AI) deals with intelligent behaviours in artificial systems [27], which do not exist in nature, and it has found application in a natural way in the context of agents-based software. Behaviours involve abilities, such as reasoning, learning, communication and social perception[3] in complex environments. *The agents are the natural solution to all these expectations* [23]. *They refer to the intention of man to reproduce behavioural-model-driven artificial systems*. The enormous circulation of information obliges us to dominate a decisional complexity more and more onerous. From here, the necessity to employ technologies demonstrating proactive capabilities, such as learning behaviours and delegation skills. The use of intelligent technologies can give an essential and competitive approach to the achievement of goals[24]. In addition, they can permit us to evaluate their transportability and adaptability on the industrial context.

The possibility to express behavioural models based on ontological knowledge makes agents a resource ready for the semantic web[25]. In particular contexts, like health care, the information can play an even more critical role, dealing with humans surviving issues. The care of chronic and disabled patients can involve life-long treatments under the continuous supervision of experts and this could be, in part, based on artificial intelligent systems. All agents' definitions presuppose a base of intelligence, which, depending on the context, can be manifested emphasizing some characteristics in place of others: we can assert that intelligence is the behavioural premise of an agent. In other words, the undertaken actions of an agent are nothing else but the result of internal intelligence synthesis processes. For this reason, we propose to view an agent through the "agent-metric net" Fig. 1: just as a net is composed of rays, this vision places the characteristics of agents long radial directions. Therefore, as all rays are critical elements for the resistance of a net, the thoroughness of the agent-metric net guarantees the critical adaptability of the agent to the environment.

2.1 Multi-Agent Systems

A paradigm itself is not enough to give form to an architectural structure: we need the necessary technology enabling such a paradigm. Multi Agent Systems are the necessary infrastructures for agents to act: a physical environment in which agents

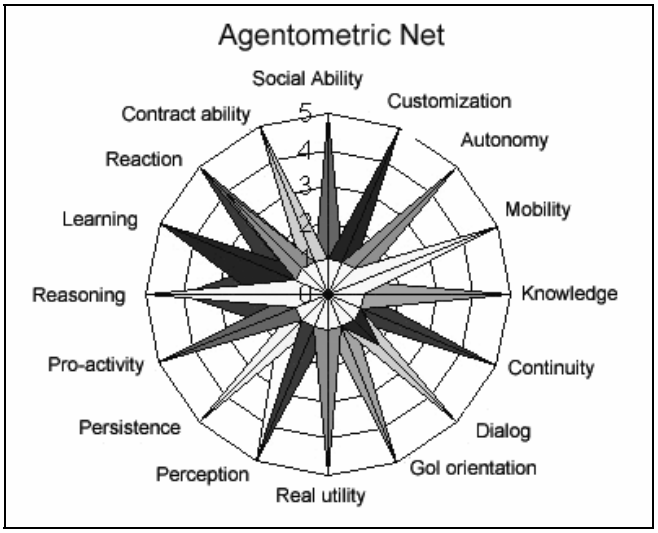

Fig. 1. : Representation of Agents Capabilities

are able to operate in order to achieve their goals and complete their life cycles.

Abstraction leads to have concepts much stronger than mathematical logics: considering that software is nothing else than a machine-level representation of human needs, abstracting just means to bring the "cold" computer closer to human necessities. Cooperation, on the other hand, is the solution for closed systems: every single program has to interface and communicate in order to lighten computational difficulties and reduce the amount of resources needed to achieve goals. In fact, agents combine several characteristics in order to interact with other entities available in the environment and find solutions to their goals.

2.2 Ontologies

The goal of the ontological representation of knowledge is to make explicit the semantics of a particular domain of interest, for the purposes of sharing the knowledge among humans and computer artefacts [1,2]. Sowa [4] subdivides knowledge representation into categories:

- Logic, provides the formal structure and rules of inference;
- Ontology defines the kinds of things that exist in the application domain;
- Computation supports the applications that distinguish knowledge representation from pure philosophy.

There is a strong relationship between some specific ontology and the logical rules and computational artefacts that use that ontology, in that when they communicate among themselves, they have some level of assurance that the same terms have the same meanings to all. However, this use requires that the logical rules and the computational artefacts have explicit linkages with the ontology; often in the form of hard-coding the ontological terms into the rules and/or the application code itself. In an agent-based system, common ontologies specify the ontological

commitments of a set of participating agents [3]. An ontological commitment is an agreement to use a vocabulary in a way that is consistent with ontology. An agent or human committed to an ontology understands (some subset of) the ontology and agrees to use it in a manner consistent with its semantics.

Agents and humans committed to the same ontology can share knowledge among themselves with some certainty that they share an underlying understanding of what is being said. Commitment to common, shared ontologies facilitates openness in an agent-based system.

In this paper, furthermore, we will evaluate the possibility to settle the whole MAS implementation on its ontological description, elevating the mutual acceptance of the concept of MAS at the level of human designer.

3 Technological requirements

3.1 Protégé

Protégé[1] is a free, open-source platform that provides a suite of tools to construct domain models and knowledge-based applications with ontologies. It implements a rich set of knowledge-modelling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats.

Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, it can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

The Protégé platform supports two main ways of modelling ontologies:

- the Protégé-Frames editor, that enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity[2] protocol (OKBC);
- the Protégé-OWL editor, that enables users to build ontologies for the Semantic Web, in particular in the W3C[3]'s Web Ontology Language (OWL).

3.2 Protégé OWL Plug-in

The OWL Plug-in is a semantic web extension of the Protégé ontology development platform [6]. The OWL Plug-in can be used to edit ontologies in the Web Ontology Language (OWL), to access description logic reasoners, and to acquire instances for semantic mark-up. In many of these features, the OWL Plug-in has created and facilitated new practices for building semantic web contents.

3.3 Java Agent DEvelopment Framework

JADE[4] is a middleware for the development and run-time execution of peer-to-peer applications which are based on agents paradigm and which can seamless work and interoperate both in wired and wireless environment. Two main aspects of the conceptual model are the following: distributed system topology with peer-to-peer networking and software component architecture with agent paradigm. JADE is a software framework completely implemented in JAVA language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA[5] specifications and through a set of tools that supports the debugging and deployment phases. The agent platform can be changed at run-time by moving agents from one machine to another one. JADE is a free software and it is distributed and copyrighted by TILAB[6] under the terms of the LGPL (*Lesser General Public License*).

3.4 Ontology Beangenerator for JADE

The Beangenerator helps creating java files representing an ontology that can be used for within the JADE environment (version 2.5). The Beangenerator is implemented as a plug-in for Protégé (version 1.6), by which we are enabled in importing and exporting RDF and RDFS. This tool permits us to generate FIPA/JADE compliant ontologies from RDF(S), XML and Protégé projects.

4 OWL to MAS Automating Framework

The intention to develop a *whole* Multi-Agent System preliminary structure starting from its ontological description is new in literature. Well-established procedures can be found in agents' communication and interactions through ontologies, as described by FIPA and implemented by JADE. Here our intention is to capitalize the benefits coming from the semantics of an ontology exploitation.

We have identified six main modules in the composition of the framework, each of them oriented to a precise step in the automation process. The design refinement has led to logically divide the framework into 3 main phases: *extraction* (OWL description and analysis), *coding* (ontology translation and MAS code composition) *and deployment* (agents' creation and loading). One of the most important components of the framework is represented by the OWL-based MAS description: all successive modules depend on this one, which contains all relevant agent-oriented concepts: behaviours, interaction, communication, service, actions and so on.

We would like to emphasize once more that the intention of the framework is to automate the composition and population of MAS in a sufficiently detailed manner, but we do not pretend to create a final MAS complete in its business and functioning logic.

[1] http://protege.stanford.edu/

[2] http://www.ai.sri.com/~okbc/

[3] http://www.w3.org/

[4] http://jade.tilab.com/

[5] FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies - http://www.fipa.org/

[6] http://www.telecomitalia.it/cgi-in/tiportale/TIPortale/ep/home.do?LANG=IT{\&}tabId=2

4.1 MAS OWL Description Module

This module contains the OWL-based ontology description representing the Multi-Agent System. The most considerable effort requested by this framework component was to conciliate an appropriate conceptualization of the e-Health MAS together with the absence of an Agent-Oriented Software Engineering (AOSE) methodology. According with [7], none of the actually existing AOSE seems to completely suite this purpose. Numerous methodologies and metaphors for developing agent-based systems have been proposed in the literature [7, 8, 28]:

- *the ant algorithms metaphor* [11, 12] has shown to be useful in efficiently solving complex distributed problems;
- *physical metaphors* [13, 14], focusing on the spontaneous reshaping of a system's structure, may have useful applications in pervasive and mobile computing;
- *societal metaphors* have been effectively applied in robotics applications[15][16] and in the understanding and control of highly decentralized systems [17, 18].

The GAIA methodology approach[09] focuses on the development of large size systems, that have to guarantee predictable and reliable behaviours. For these kinds of systems, we are persuaded by the fact that the most appropriate metaphor is that one reflecting the *human organization* [19–21].

Evaluating strengths and weaknesses of a methodology is a fundamental phase toward its adoption as a standard and it is of crucial importance for the success of the whole system realization. After the analysis of the project requirements our decision has focused on GAIA methodology, in order to model the MAS ontology description. The choice has been justified by the approach of the methodology in conceptualizing the environmental elements and their characteristics Fig. 2, which seemed to naturally match with the concept of an agent within the JADE platform.

Every agent living in a MAS container plays a well-defined organizational role, following the specifications of the Health care system. Each agent provides (guarantees) one or more services fruition. A service has been modelled as a pair of object properties in Protégé: *behaviour and interaction*. The first represents the *dynamic intelligence* of the agent, whilst the second produces the environmental knowledge-base of each MAS entity. Due to its connecting functionality between the MAS design and its physical implementation, this module had to conciliate concepts coming from the chosen methodology (role, environment, permission, obligations, etc.), as well as those ones necessary to the JADE MAS platform (agent identification, directory facilitator, mobility, message, etc.). Finally, all interactions (i.e. communications) are based upon the ontological description of concepts, such as *behaviour* and *message*.

4.1.1 Scenario description

The conceptualization of our MAS is defined around the following elements: *Environmental Role* (agent in the sense of JADE)*, Behaviour, Communication, Health Structure, Interaction Protocol, Interaction Role, Language, Message, Ontology, Service.*

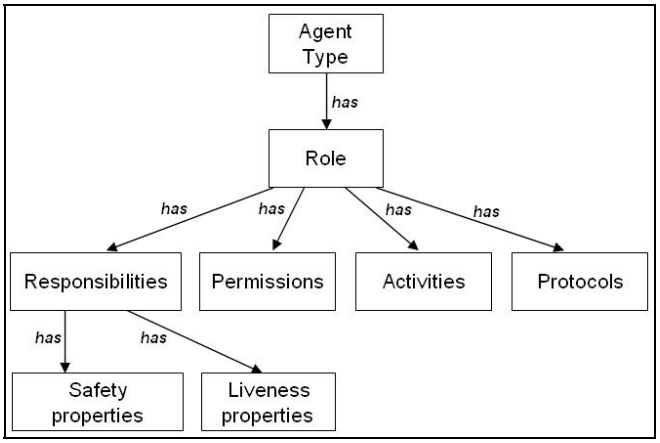

Fig. 2. An abstract vision of an agent in GAIA

We simplified the MAS implementation by three agents role definition: a *Gynaecologist Agent,* a *Home Assistant Agent* and a *Nurse Agent.* Each of them will provide a specific service, which will always be translated into a behavioural model and an interaction pattern. A simplified example of the gynaecologist agent model (in square brackets we intend the Protégé object property range class) is the following:

a*GynaecologistAgent*, offersGinaecologistService [MonitoringService] and actsInside[HealthStructure];

a *MonitoringService*, hasMonitoringBehaviour [CyclicBehaviour] and asInteractionBy[Communication];

a *CyclicBehaviour*, hasActionBy[Activity];

a Communication, hasInferenceOn[Ontology], hasInteractionInitiator[Initiator], hasInteractionResponder[Responder], hasLanguage[Language], hasMessage[Message] and hasProtocol[Protocol].

The goal of this agent is to continuously check the status of its patients and to trigger the intervention of the nurse agent in case of assistance plan invocation. We assume that a standard message of alarm will be sent by the doctor to the nurse, in order to inform on the number of the action room. The *NurseAgent*, on the other hand, has to provide the alarm and monitoring services, customizing of course its behavioural model, its communication patterns and its interaction protocols and responder(s). Similar considerations can be asserted for the third agent, the *HomeAssistantAgent*, whose aim is to provide an intervention service in case of alarm and to register a new patient to the e-Health platform from an external care structure.

4.1.2 Module results

In the following, we report the output of the first module of the automating framework: an OWL description (simplified in

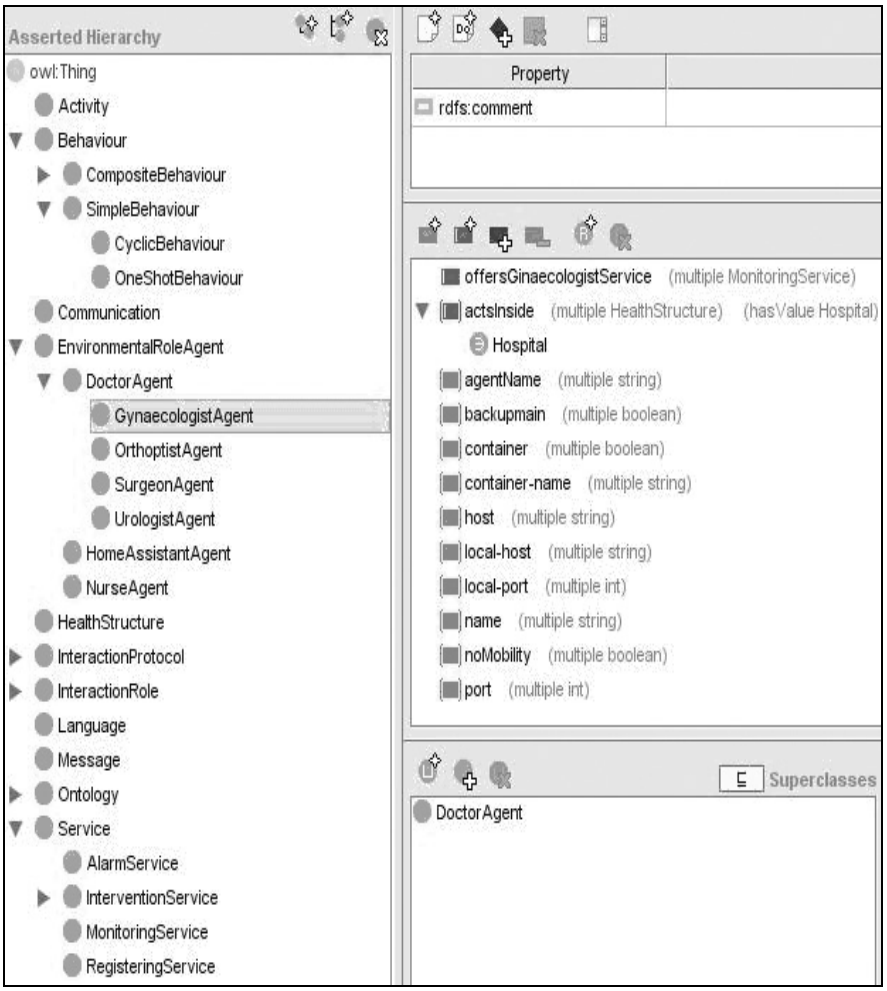

Fig. 3. OWL Classes of the MAS Description in Protégé

(Fig. 3) in Protégé of the MAS organizational relationships and a snippet of the OWL-to-MAS description file.

The ontological representation of the MAS is composed of several important diagrams. First of all we have realized the necessary OWL classes and properties (Fig. 3 and Fig. 4), listing all relevant concepts for the model. After that, we could proceed with the instantiation of the individuals, which will physically populate our Multi-Agent System: the final result of the automated modelling framework.

The following code represents a snippet of the file describing the Multi-Agent Sytem:

```
<rdf:RDF>
. . . .
<Activity rdf:ID="gmbActionTwo">
  <activitySignature
   rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   gmbActionTwo(Object paramTwo)
  </activitySignature>
</Activity>
<Activity rdf:ID="gmbActionOne">
  <activitySignature rdf:datatype="http://www.w3.org
    /2001/XMLSchema#string">gmbActionOne(Object paramOne)
  </activitySignature>
</Activity>
<MonitoringService
  rdf:ID="K4CareGynaecologistAgentHospitalMonitoringService">
<hasInteractionBy
  rdf:resource="#GynaecologicalMonitoringCommunication"/>
<hasMonitoringBehaviour>
  <CyclicBehaviour
    rdf:ID="GynaecologicalMonitoringBehaviour">
    <hasActionBy rdf:resource="#gmbActionTwo"/>
    <hasActionBy rdf:resource="#gmbActionOne"/>
    <behaviourName rdf:datatype=
      "http://www.w3.org/2001/XMLSchema#string">
      gynaecologicalMonitoringBehaviour</behaviourName>
      <hasActionBy>
        <Activity rdf:ID="gmbActionThree">
          <activitySignature rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string">
            gmbActionThree(Object paramThree)
          </activitySignature>
        </Activity>
      </hasActionBy>
  </CyclicBehaviour>
</hasMonitoringBehaviour>
<serviceName
  rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  hospital_gynaecological_monitoring
</serviceName>
<dfServiceName
  rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  HOSPITAL_GYNAECOLOGICAL_MONITORING
</dfServiceName>
```

```
</MonitoringService>
<HealthStructure rdf:ID="HealthCenter"/>
<GynaecologistAgent rdf:ID="K4CareGynaecologistAgent">
  <agentName
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    k4cGynaecologist
  </agentName>
  <offersGinaecologistService
    rdf:resource=
     "#K4CareGynaecologistAgentHospitalMonitoringService"/>
  <container-name
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    K4CareGynaecologistContainer
  </container-name>
  <container
    rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
    true</container>
</GynaecologistAgent>
<HealthStructure rdf:ID="ConsultingRoom"/>
</rdf:RDF>
```

Fig. 4. OWL Object Properties List

4.2 MAS Description Analysis Module

The objective of this module is to analyse, recognize and extract all the FIPA/JADE compliant agent-oriented information contained into the OWL file produced in the previous step. The designated class to achieve this goal is the *OWLJenaExtractor*. We have already mentioned the fact that, for this purpose, we will be helped by the Protégé OWL plug-in API set, in order to query and traverse the contents of the MAS.

Among all the available APIs we have to highlight the importance of those ones granting the instantiation of the OWL model and its interrogations.

The Protégé OWL API makes a clear distinction between named classes and anonymous classes. Named classes are used to create individuals (i.e. classes instantiations), while anonymous classes are used to specify logical characteristics (restrictions) of named classes. Note that the class *OWLModel* from the APIs contains many more classes than those returned by the: *owlModel.getUserDefinedOWLNamedClasses()* method. We used this method to filter out the system resources such as the owl:Class metaclass, owl:Thing and rdfs:subClassOf. If you want to access other resources by their names, you can easily use methods such as:

OWLModel.getOWLNamedClass()

or *OWLModel.getOWLObjectProperty()*.

To summarize, the class OWLModel provides access to all resources in the model, and then you can call other getter methods on the objects that are delivered by the OWLModel. Other queries, like:

OWLModel.getRDFResourcesWithPropertyValue() get all resources that have a certain property value.

Starting from the results of the previous module, in the MAS there will be the three agents mentioned above, and, by the appropriate classes and methods exposed before, we will pass their information to the composing module. The agent-oriented information extracted from the OWL file will be stored as a knowledge-base inside an *AgentInsightDTO* object.The structure of this object complies the need to store all data used by the module that will compile and deploy the agents. We can summarize the properties of the class *AgentInsightDTO*, as follows:

- **agentName** (String) [the name of the agent within the container];
- **container** (Boolean) [specifies that this instance of JADE is a container and that it must join with a main-container];
- **backupmain** (Boolean) [specifies that this instance of JADE is a backup main container and that it must join with a main-container];
- **host** (String) [specifies the host name where the main container to register with is running; its value is defaulted to *localhost*;
- **port** (String) [this option allows to specify the port number where the main container to register with is running];
- **local-host** (Integer) [specifies the host name where this container is going to run];
- **local-port** (Integer) [this option allows to specify the port number where this container can be contacted];
- **name** (String) [this option specifies the symbolic name to be used as the platform name; this option will be considered only in the case of a main container; the default is to generate a unique name from the values of the main container's host name and port number];

– **container-name** (String) [this option specifies the symbolic name to be used as the name of this container];

– **nomobility** (Boolean) [disable the mobility and cloning support in the launched container. In this way the container will not accept requests for agent migration or agent cloning, option that might be useful to enhance the level of security for the host where this container is running. The platform can include both containers where mobility is enabled and containers where it is disabled. In this case an agent that tries to move from/to the containers where mobility is disabled will die because of a Runtime Exception];

– **service** (ServiceClass) [OO representation of the service carried out by the agent];

Within the service class we will find the following further structured information:

– **behavioralModel** (BehaviorClass);

– **communication** (CommunicationClass);
  – **interacionProtocol** (InteractionClass);
  – **interactionInitiator** (InitiatorClass);
  – **interactionRespoder** (ResponderClass);

– **healthStructure** (HealthStructureClass) [OO representation of the care structure in which the agent acts].

4.3 Ontology Translation Module

The aim of this module is to produce conversational ontologies, used by agents during their interactions. The Protégé Beangenerator creates the necessary representation of exchanged concepts in JAVA classes, defining a JADE compliant structure of classes that will be imported inside our project as added packages. We call attention to the fact, that even if the Beangenerator is able to create agent-oriented concepts, such as agent and activity, it is absolute inappropriate for our purpose: the Benagenerator composes agents code structure singularly, without any perception of the fundamental interconnecting concepts that are necessary to the MAS modelling, such as roles, services, communications, interactions and, above all, there is no methodology at the basis of such process. For this reason, we confine the tool at the creation of the ontologies that will be queried by an ACL-based (Agent Communication Language) message exchange. A snippet of extracted ontology can be viewed in the following code, concerning the concept *PatientRecord*:

```
// ......
public class PatientRecordOntology extends Ontology {

  // The name identifying this ontology
  public static final String ONTOLOGY_NAME =
    "Patient-record-ontology";
  // VOCABULARY
  public static final String ENVIRONMENTAL_ROLE =
    "Environmental_role";
  public static final String PATIENT = ‘‘Patient’’;
  public static final String NAME = "Name";
  public static final String SURNAME = "Surname";
  public static final String ADDRESS = "Address";
  public static final String BIRTHDATE = "Birthdate";
  public static final String CITY = "City";
  public static final String STATE = "State";
  public static final String ASSISTENCE_NUMBER =
    "Assistance_number";
  public static final String ASSURANCE_NUMBER =
    "Assurance_number";
  // The singleton instance of this ontology
  private static Ontology theInstance =
    new PatientRecordOntology();
  // This is the method to access the singleton
  patient record ontology object
  public static Ontology getInstance() {
    return theInstance;
  }
  // Private constructor
  private PatientRecordOntology() {
  // The patient record ontology extends the basic ontology
  super(ONTOLOGY_NAME, BasicOntology.getInstance())
    try {
      add(new ConceptSchema(ENVIRONMENTAL_ROLE),
        Environmental_role.class);
      add(new ConceptSchema(PATIENT), Patient.class);
      // Structure of the schema for the Item concept
      ConceptSchema
        cs = (ConceptSchema) getSchema(ENVIRONMENAL_ROLE);
        cs = (ConceptSchema) getSchema(PATIENT);
        cs.add(NAME, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
        cs.add(SURNAME, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
        cs.add(ADDRESS, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
        cs.add(BIRTHDATE, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
        cs.add(CITY, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
        cs.add(STATE, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
        cs.add(ASSISTENCE_NUMBER, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
        cs.add(ASSURANCE_NUMEBR, (PrimitiveSchema)
          getSchema(BasicOntology.STRING));
    } catch (OntologyException oe) {
      oe.printStackTrace();
}}}
```

4.4 MAS Code Composition and Orchestration Module

Once collected all information relating to the agents of the MAS, their behaviours, activities, communications, basic messages and ontologies, we can delegate this module to compose the agents final code structure. The elected main class for this purpose is the *MASCodeComposer*.

The agent code can be divided into two categories of elements: *fixed and invariable*, regarding all keywords involved into the JAVA programming and JADE agent-oriented coding,

and *automation-framework-derived*, necessary to the embedding of those expressions which comply the correct code completion of an agent. In other words, this module owns the parsing knowledge needed to decide whether to insert a fixed expression from the "composing vocabulary" or whether to get data from the *AgentInsightDTO* object.

Here an example of automatically composed agent class code follows (note that bolded expressions between **[]** indicate the *automation-framework-derived* ones).

```
// ......
public class [gynaecologistAgentInsideDTO.getAgentClassName()]
extends Agent {
  protected void setup() {
    addBehaviour(new
    [gynaecologistAgentInsideDTO.getMonitoringService()](this)
    {
    private boolean finished = false;
    public void action() {
      try{
          // ......
          // .......
      ACLMessage inform = new ACLMessage(ACLMessage.INFORM);
      inform.setProtocol(FIPANames.
      InteractionProtocols.FIPA_INFORM);
      inform.setOntology([gynaecologistAgentInsideDTO.
      getOntologyName()]);
      inform.setContent([gynaecologistAgent
      InsideDTO.getMessageContent()]);
      inform.addReceiver(new
      AID([gynaecologistAgentInsideDTO.
      getInteractionResponderLocalName()],
      AID.ISLOCALNAME));
      addBehaviour(new AchieveREInitiator(this,
      [gynaecologistAgentInsideDTO.getProtocol().getName()])
      {
      protected void handleInform(ACLMessage reply) {
        System.out.println("Protocol finished.
        Rational Effect achieved." +"Received
        the following message: "+reply);
        }
      });
      // .......
```

4.5 Agents Creation Module

The simple goal of this module is to compile under the right directives the .java files previously created, arranged and saved by the module explained in the previous paragraph.

During this process we must pay attention to the file system folders structure, in order to grant the right dependencies among all files containing the agents' code.

4.6 Agents Loading Module

Within JADE there is an in-process implemented interface that allows external Java applications to use JADE as a kind of library and to launch the JADE Runtime from within the application itself.

A singleton instance of the JADE Runtime can be obtained via the static method *jade.core.Runtime.instance()*, it provides two methods to create a JADE main-container or a JADE remote container (i.e. a container that joins an existing main-container, giving form in this way to a distributed agent platform); both methods require passing as a parameter a *jade.core.Profile* object that keeps the configuration options (e.g. the hostname and port number of the main container) required to start the JADE runtime.

Both these two methods of the Runtime return a wrapper object, belonging to the package *jade.wrapper*, that wraps the higher-level functionality of the agent containers, such as installing and uninstalling MTPs (Message Transport Protocols, FIPA), killing the container (where just the container is killed while the external application remains alive) and, of course, creating new agents. The *createNewAgent()* method of this container wrapper returns as well a wrapper object, which wraps some functionalities of the agent, but still tends to preserve the autonomy of agents.

In particular, the application can control the life-cycle of the Agent but it cannot obtain a direct reference to the Agent object and, as a direct consequence, it cannot perform method calls on that object. Notice that, having created the agent, it still needs to be started via the method *start()*. The following code lists a very simple way to launch an agent from within an external class:

```
// Create a peripheral container within the JVM
Profile p = new ProfileImpl(false);
p.putProperty(Profile.MAIN, "true");
AgentContainer another = rt.createAgentContainer(p);
// Launch the Gynaecologist agent
// and pass it 1 argument: a String
Object[] arguments = new Object[1];
arguments[0] = "In action. Good Morning!";
AgentController gynaecologistController =
another.createNewAgent("DR.Laszlo",
"k4care.mas.owlagents.K4CareGynaecologistAgent", arguments);
gynaecologistController.start();
```

An alternative to the previous code consists in the creation of batch file to compose, save and launch as an external application from within the JAVA class *MASAgentDeployer*:

```
java -classpath "{\$}CLASSPATH{\$}" jade.Boot -container-name
GynaecologistAgentContainer
DR.Laszlo:k4care.mas.owlagents.K4CareGynaecologistAgent
```

The Fig. 5 gives an abstracted representation of the framework functioning, highlighting the deployment phase of the agents entering the MAS.

4.7 Final considerations on obtained outcomes

Following all previous MAS development steps (representing the automation framework) we were able to automate the arrangement of a whole e-Health supporting MAS. Agents acting and cooperating inside JADE containers have the right capabilities (modelled by their behaviours) to provide the required care services. Some manual developing intervention will be mandatory in order to clarify the business logic of a single agent action (which is out of the scope of this paper).

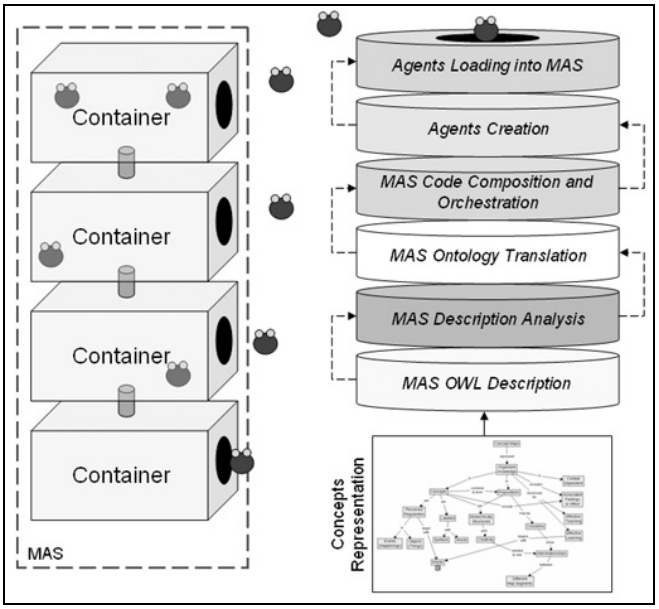

Fig. 5. Framework Functioning Abstraction

5 Conclusions and Future Works

Information technology's traditional approaches to modern applicative contexts are no longer sufficient, due to the enormous amount of available information and system's highly changing requirements. Software technologies must express a degree of sufficient intelligence in order to ease the human decisional process. The context of electronic care services manifests the same necessities. In this sense, Multi-Agent Systems can give a real competitive advantage, thanks to their capabilities, such as autonomy, mobility and social-capacity, all concepts based on a behavioural model. The development of MAS is not a simple task at all: a considerable help can derive from the use of the ontologies during the implementation phase. Ontologies can represent a fundamental semantic description of artificial systems. On the basis of this approach, we tried to give intelligence to an automation framework that builds and populates a MAS starting from its ontological representation, indeed.

The outcome actually obtained is an ontology-based framework for MAS development automation which simplifies the creation of the whole architectural structure (without omitting important concepts of the agent paradigm, such as behaviours, communications, protocols, ontologies, messages). We have given a semantic weight not only to the agent communication level, but even to the MAS implementation itself. Future plans regard the intention to empower the automating framework, by trying to describe in a more formal manner the definition of agent's behaviours, in relation to the business logic of the service they provide. We are interested in a mapping procedure which could express, with a certain degree of intelligent automation, the logics deriving from a service (business) in terms of agent behavioural model.

References

1 **Cranefield S, Purvis M, Nowostawski M, Hwang P**, *Ontologies for interaction protocols*. Whitestein Series in Software Agent Technology, Ontologies for Agents: Theory and Experiences.
,

2 **Nodine MH, Fowler J**, *on The Impact Of Ontological Commitment*. Whitestein Series in Software Agent Technology, Ontologies for Agents: Theory and Experiences.

3 **Karunatillake NC, Jennings NR, Rahwan I, Norman TJ**, *Arguing and Negotiating in the Presence of Social Influences*, CEEMAS 2005, Budapest, Hungary – September 2005, Proceedings.

4 **Sowa JF**, *Knowledge Representation : Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co., 2000. Pacific Grove, CA.

5 **Gruber TR**, *Translation approach to portable ontology specifications*, Knowledge Acquisition **5** (1993), no. 2.

6 **Knublauch H, Fergerson RW, Noy NF, Musen MA**, *The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications - Stanford Medical Informatics*, Stanford School of Medicine.

7 **Dam KH, Winikoff M**, *Comparing Agent Oriented Methodologies*, Proceedings of the 5th Int'l Bi-Conference Workshop on Agent Oriented Information Systems (AOIS), 2003.

8 **Wooldridge M, Jennings NR, Kinny D**, *A methodology for agent-oriented analysis and design*, May 1999. ACM.

9 ———, *The Gaia methodology for agent-oriented analysis and design*, Autonomous Agents and Multi-Agent Systems **3** (2000), no. 3.

10 **Zambonelli F, Jennings NR, Wooldridge M**, *Developing Multiagent Systems: The Gaia Methodology*, Autonomous Agents and Multi-Agent Systems **3** (2000).

11 **Bonabeau E, Dorigo M, Theraulaz G**, *Swarm Intelligence. From Natural to Artificial Systems*, Oxford University Press, Oxford, U.K., 1999.

12 **Babaoglu O, Meling H, Montresor A**, *A framework for the development of agent-based peer-to-peer systems*, 2002.

13 **Abelson H, Allen D, Coore D, Hanson C, Homsy G, Knight T, Napal R, Rauch E, Sussmann G, Weiss R**, *Amorphous computing*, Commun. ACM **43** (5 May 2000), 43–50.

14 **Mamei M, Zambonelli F, Leonardi L**, *Distributed motion coordination in co-fields*, Apr. 2003, pp. 63–70. IEEE Computer Society Press,Los Alamitos, Calif.

15 **Moses Y., Tennenholtz M.**, *Artificial social systems*, Comput. Artif. Intel. **14** (1995), 533–562.

16 **Collinot A, Benhamou P**, *Agent-oriented design of a soccer robot team*, 1996. Los Alamitos, Calif.

17 **Hattori F, Ohguro T, Yokoo M, Matsubara S, Yoshida S**, *Socialware: Multiagent systems for supporting network communities*, Communi. ACM **42** (1999), no. 3, 55–61. (Mar.)

18 **Ripeani M, Iamnitchi A, Foster I**, *Mapping the gnutella network*, 2002, pp. 50–57.

19 **Handy C**, *Understanding Organizations*, PENGUIN BOOKS publishing, 1976.

20 **Demazeau Y, Costa R**, *Populations and organizations in open multi-agent systems*, 1996.

21 **Zambonelli F, Jennings NR, Omicini A, Wooldridge M**, *Agent-oriented software engineering for internet applications*, Springer-Verlag, Berlin, Germany, 2001.

22 **Jayalath N, Karunananda AS**, *Meeting competitive advantages through Mobile Agent Technology*,.

23 **Wooldridge MJ, Jennings NR**, *Intelligent agents: Theory and practice*, The Knowledge Engineering Review **10** (1995), no. 2, 115 152.

24 **M.B. Blake**, *B2B Electronic Commerce: Where Do Agents Fit In?*, July 28 2002.

25 **Petrie C, Bussler C**, *Service Agents and Virtual Enterprises: A Survey*, IEEE Internet Computing (August 2003), 1-12.
26 available at `http://www.w3.org/TR/owl-features/`.
27 **Rodney A. Brooks**, *MIT Artificial Intelligence Laboratory: Intelligence without representation*, Vol. 47, January 1991.
28 **Onn Shehory**, *Evaluation of Modelling Techniques for Agent-Based Systems*, 2001, pp. 624–631.