Applying Ontologies and Agent Technologies to Generate Ambient Intelligence Applications

Artur Freitas, Daniela Schmidt, Alison Panisson
Rafael H. Bordini, Felipe Meneguzzi and Renata Vieira

Pontifical Catholic University of Rio Grande do Sul - PUCRS
Postgraduate Programme in Computer Science, School of Informatics (FACIN)
Porto Alegre - RS - Brazil
{artur.freitas,daniela.schmidt,alison.panisson}@acad.pucrs.br,
{rafael.bordini,felipe.meneguzzi,renata.vieira}@pucrs.br

Abstract. The specification of agent systems comprises different dimensions normally defined using distinct formalisms. Since this lack of a uniform representation makes harder to express how each level affects the others, we propose an ontology to integrate the formalisms that originally cover a single multi-agent system dimension. In doing this, we align semantic technologies and knowledge representation for agents, environments, and organisations providing agent-oriented designers with a unified approach for developing complex systems. In our approach, we represent the abstractions typical of each multi-agent system dimension as an ontology, and we exemplify both the use of such ontologies to model an eldercare application in the context of ambient intelligence and smart cities, as well as how the ontology concepts support coding in agent platforms. We discuss the implications of such integrated view for designing agents, and highlight its advantages for agent-based software development.

Keywords: ontology, multi-agent system, ambient intelligence

1 Introduction

The use of ontologies in the context of Multi-Agent Systems (MAS) is still an open issue, especially in relation to integrated frameworks that consider the co-specification of their different dimensions. The development of MAS in JaCaMo [1] comprises three distinct dimensions, namely: agent, organisation, and environment. However, these dimensions are not uniformly integrated into a single formalism: agents are programmed in Jason [2] using the AgentSpeak language; organisations are specified in Moise [3] in an XML-based document; and environments are coded in Java using the CArtAgO API [4]. This approach makes difficult to keep track of problems because errors in one level can affect the other levels, and it also becomes cumbersome to explore interconnections between the different layers and requires the programmer the knowledge about different paradigms. To address these issues, we propose a unified representation which covers these three agent programming dimensions and integrates the various formalisms. Thus, we developed ontologies to represent the agency, environment, and organisation levels of MAS, which are aligned with a platform integrating these three levels in agent programming (*i.e.*, JaCaMo [1]). Until now, JaCaMo platform does not address the ontological level of MAS. Hence, we discuss an integrated semantic model to represent these three dimensions based on ontologies that represent each MAS level.

In order to demonstrate the need for and advantages of an integrated view for knowledge representation when developing complex multi-agent systems, we show how our approach can be used to model a multi-agent assisted living application aiming at supporting home care for an elderly patient. In particular, we support the collaborative work of a team of people including family members and professional carers who work together to allow an elderly patient with various debilitating health conditions to live in his own home. We use multi-agent systems techniques and ambient intelligence to give such support, but beyond this, our vision is towards systems integration in smart cities through multi-agent systems techniques to allow full integration of data from city transport, health systems, social services, smart grids, and so forth. That is, we envision all such sources of information coming together to give as much support as possible to the patient's family and carers. This is a very important social scenario in Brazil where the predominant culture is for families to care for their elders themselves.

An integrated ontology model to represent these MAS dimensions enables semantic reasoning and can be used as a common vocabulary in agent-oriented programming. In our proposal these dimensions are sub-ontologies that may interconnect with each other, and reuse relevant concepts from each other MAS dimension. Each dimension details different aspects, and these interconnections when combined have to result in an integrated knowledge model with a clear correspondence to an integrated programming platform, such as JaCaMo [1]. Some proposals for the knowledge level, such as Moise [3], are already related to a programming framework, allowing to convert the ontology specification to a programming level [5]. This is desirable for all dimensions, but these levels have to be aligned for that to work as a common specification. Also, an MAS can be modelled, reused and extended in one dimension while maintaining the others, which allows the designer to work without going into specifics of the programming languages that define each dimension. In this context, an MAS design is more easily expressed and communicated, and the model can be more easily converted to a formal verification system. Thus, our work is a step towards a knowledge level integration of MAS dimensions and platforms.

This paper is structured as follows. Section 2 refers to previous ontologies related to MAS aspects: agents, organisations, and environments. Section 3 shows the need for the integration of such aspects at the knowledge level on the light of an example in the area of health care. Section 4 concludes this paper and points to our next steps.

2 Ontologies for Multi-Agent Systems

Agents are reactive systems that can independently determine how to best achieve their goals and perform their tasks [2] while demonstrating properties such as autonomy, reactivity, proactiveness and social ability. Agents are situated in an environment, where they can perceive and modify it, and they should be able to exchange information, cooperate and coordinate activities. Jason [2] is an AgentSpeak language platform implementation that focuses on agent actions and mental concepts. It is an open source interpreter that offers features such as speech-act based agent communication, plans annotation, architecture customisation, distributed execution and extensibility through internal actions. On the environment side of agent systems, CArtAgO [4] is a platform to support the artifact notion in MAS. Artifacts are function-oriented computational devices which provide services that agents can exploit to support their individual and social activities [4]. Lastly, the specification of agents at the organisation level can be achieved using an organisation modelling language, such as Moise [3]. Moise explicitly decomposes the specification of an organisation into its structural, functional and normative dimensions. Each of these three agent-oriented platforms addresses a specific dimension of MAS programming, and JaCaMo [1] integrates these three dimensions. Thus, an MAS programmed in JaCaMo is given by an agent organisation defined in Moise [3], organising autonomous agents coded in Jason [2], working in shared distributed artifact-based environments developed in CArtAgO [4]. In this paper, we are interested in the use of ontologies in these multi-agent platforms.

Ontologies are knowledge representation structures composed of concepts, relationships, instances and axioms, which empower the execution of semantic reasoners that provide functionalities such as *consistency checking*, *concept satisfiability*, *classification*, and *realisation*. It is natural to think that there are advantages in using ontologies more expressively in agent development. One of the first approaches to consider the use of ontologies to enhance agent-oriented programming was AgentSpeak-DL [6], which extended AgentSpeak with description logic concepts. However, their focus was on using ontologies during agent reasoning, instead of modelling aspects of MAS in ontologies. In other words, there are many ways in which these areas can be connected and explored in the literature; in fact, JaCaMo dimensions have been considered at the knowledge level in previous work, as done for instance in [5]. However a multi-dimensional unified view has not been proposed. The work in [6] points advantages of integrating agents and ontologies: (*i*) more expressive queries in the belief base, since results can be inferred from the ontology and thus are not limited to explicit knowledge; (*ii*) refined belief update given that ontological consistency of a belief addition can be checked; (*iii*) the search for a plan to deal with an event is more flexible because it is not limited to unification, *i.e.*, it is possible to consider subsumption relationships between concepts; and (*iv*) agents can share knowledge using ontology languages such as OWL (Web Ontology Language). Next we focus on some examples of ontologies proposed for MAS, specially considering the distinct dimensions which divide JaCaMo. Although the advantages of using ontologies for agents are clear, few agent-oriented platforms are currently integrated with ontology techniques.

2.1 Ontologies in Agent Programming and Reasoning

Considering the agent dimension, AgentSpeak-DL [6] is an agent-oriented programming language based on Description Logic (DL). AgentSpeak-DL extends agents' belief base with DL in which the belief base includes: (*i*) one immutable TBox (terminological box, or conceptualisation) that characterises the domain concepts and properties; and (*ii*) one ABox (assertion box, or instantiation) with dynamic factual knowledge that changes according to the results of environment perception, plan execution and agent communication. In this approach, the agent belief base can be enriched with the definition of complex concepts that can go beyond factual knowledge [6].

JASDL [7] followed these ideas to transparently merge agent belief base and ontological reasoning. JASDL [7] is an AgentSpeak-DL implementation that extended Jason to provide agents with ontology manipulation capabilities using the OWL API [7]. This offers a practical approach to agents use ontologies and semantic reasoning in declarative paradigms. Agent programmers benefit from features such as plan trigger generalisation based on ontological knowledge and the use of such knowledge in belief base querying [7]. Some Jason modules were altered to implement JASDL, *e.g.*, the belief base was extended to partly resides within an ontology ABox, which, combined with a DL reasoner, facilitates the reuse of available knowledge in ontologies. This feature increases the inferences that an agent can make based on its beliefs and assures knowledge consistency. JASDL [7] also modified Jason plan library to enable enhanced plan searching; and agent architecture to augment it with message processing to obtain semantically-enriched inter-agent communication.

This section presented approaches for incorporating ontological reasoning in agents to enable them to relate with knowledge and not only with the observation of facts. However, to the best of our knowledge, approaches for represent the agent dimension in ontologies concerning the Jason platform do not exist yet. Next section shows a knowledge representation of the environment dimension of MAS using ontologies.

2.2 Ontologies for Environments and Artifacts

Environments play an essential role in MAS, and their semantic representation can improve the way agents reason about the objects with which they interact and the overall environment where they are situated. In [8] an environment ontology is proposed based on environment aspects of agent programming technologies that is integrated into a platform for developing cognitive multi-agent simulations. Thus, it can be used to specify environments and derive a project-level, complete, and executable definition of multi-agent environments [8]. An environment description is a specification of its properties and behaviour, which includes concepts such as: *objects* (*i.e.*, resources of the environment); *agents* (*i.e.*, their "physical" representation in the environment that is visible to other agents); *actions* that each type of agent can perform in the environment; *reactions* of the environment and objects when an agent's actions affect them; *perception types* available to each type of agent; and *observable properties*, that is, the information about the simulation to which observers (*e.g.*, the agents) have access.

The use of an environment ontology adds three important features to existing multi-agent approaches [8]: *(i)* ontologies provide a common vocabulary to enable environment specification by agent developers (since an ontology explicitly represents a consensual model for environment and agent essential properties, defining environments in ontologies can facilitate and improve the development of multi-agent simulations); *(ii)* an environment ontology is useful for agents acting in the environment because it provides a common vocabulary for communication within and about the environment (such explicit conceptualisation is essential to allow interoperability of heterogeneous systems); and *(iii)* environment ontologies can be defined in ontology editors with graphical user interfaces, making easier for those unfamiliar with programming to understand

and design such ontologies. In [8], the relationship between the environment and other MAS dimensions was already foreseen, since they mention the intention of looking at higher-level aspects of environments, *i.e.*, social environment aspects of agents, such as the specification of social norms and organisations in agent societies. Next section shows what has been proposed regarding the knowledge level of the social dimension.

2.3 Ontologies for Social Organisations

Agent organisations are required to provide the means for agents to query and reason about the structure of a society of agents [5]. Among the recent developments on Moise, there is already a semantic description of multi-agent organisations [5], using OWL to develop an ontology for organisational specifications of the Moise model (structural, functional, and normative levels). This approach may help agents in becoming aware, querying, and reasoning about their social and organisational context in a uniform way [5]. Also, this work makes possible to convert the ontology and the Moise specification, providing more flexibility for the development of agent organisations.

The semantic description of Moise [5] provides agent-side reasoning and querying features (*i.e.*, the agents are able to use this information). The benefits highlighted in [5] are increased modularisation, knowledge enriching with meta-data, reuse of specifications, and easier integration. With the semantic web effort aiming to represent the information in semantic formats, the MAS community can take advantage of new semantic technologies in MAS development tasks such as to integrate organisational models, to monitor organisations, and to analyse agent societies [5].

3 Unifying the Three Dimensions

For each of the three dimensions described above it would be interesting to establish semantic representations of their particular type of abstractions. For each of them the advantages of semantic web technologies have been advocated, and they usually recognise the importance of the other dimensions. However, a global view of MAS is still missing. We aim to work towards the integration of these various dimension at the semantic level, since they are already being integrated at the programming level (for example in JaCaMo [1]) and each dimension has had proposals for a semantic account.

Agent programmers benefit from an integration among these ontological levels with each programming dimension since the knowledge represented in one dimension can be reused in another, resulting in a greater interoperability of agent platforms. This would enable to convert MAS defined in ontologies to code in specific agent platforms, and vice-versa. Also, a system designed with a higher degree of modularity is easier to maintain, given that it separates different concerns yet enables relations between them. For example, the characteristics of one dimension (*e.g.*, environment) could be used to define properties on another (*e.g.*, organisational). In fact, it is often the case that the concepts of one level are related to another but current MAS platforms do not allow for such relations to be explicitly represented. The JaCaMo ontology is structured in a way that it imports three ontologies in order to represent: the agent dimension in Jason, the environment layer in CArtAgO, and the organization elements in Moise. This modularization separates different MAS concerns in independent components, while allowing to include and interrelate them, therefore offering advantages sush as increased manutenability, usability and extensibility for the MAS developers.

More specifically, the classes and properties in the JaCaMo ontology we are proposing can be visualized in Figure 1, which shows an overview of the concepts in the dimensions modelled in three sub-ontologies: agent, environment and social organisation. Figure 1, obtained in the Protégé ontology editor[1], shows the concepts in yellow circles at left and the properties in blue rectangles at right. From the agent dimension, in Jason [2], the most important concepts are the agents, their plans and actions. From the CArtAgO [4] environment perspective, the main concepts are the artifacts, their operations, observable properties, and signals. Artifacts can be either the target (outcome) of agent activities, or the tools used by agents as means to support their activities (consequently, artifacts reduce the complexity of agents tasks' execution). Finally, the Moise [3] organisation elements are, for example, groups, roles, missions, norms, and so on. A role definition states that agents playing that role are willing to accept the behavioural constraints related to it. The organisation functional dimension specifies how global collective goals should be achieved, *i.e.*, how they are decomposed in global plans, grouped in coherent sets (missions) to be individually distributed to agents. The normative dimension binds the structural dimension with the functional one to specify role's permissions, prohibitions and obligations for missions [3]. The connections among concepts are encoded by means of the object properties, which determine how instances are allowed to relate among each other. Next we show how to use this ontology to model, reason and generate code for a JaCaMo ambient intelligence application that provides its users health care functionalities.

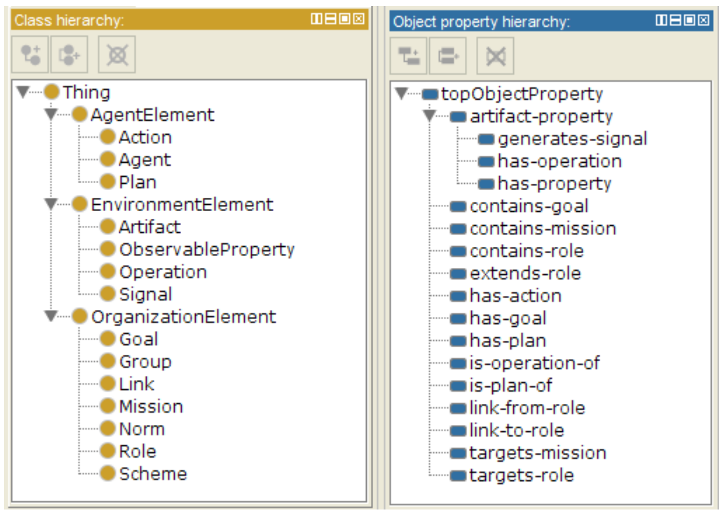


Fig. 1. Classes and object properties in our proposed JaCaMo ontology.

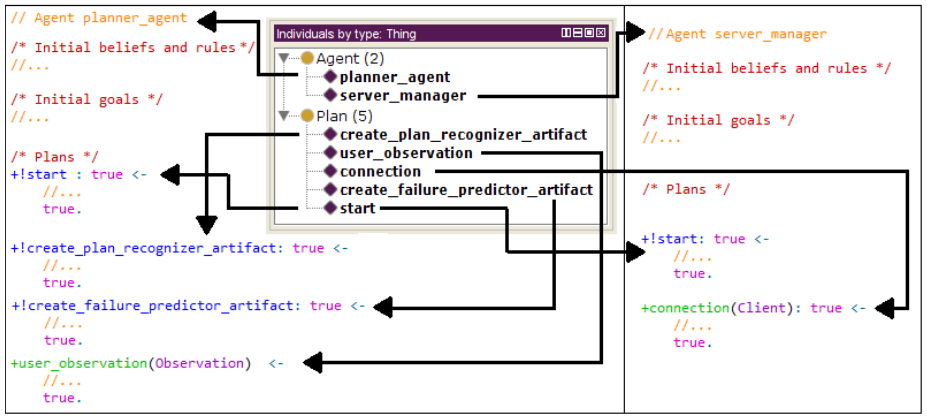
[1] Available open source at http://protege.stanford.edu/

3.1 Modelling a Health Care Application in the JaCaMo Ontology

The designed JaCaMo ontology was instantiated to model and generate a multi-agent assisted living application; as mentioned in Section 1, at the moment we focus on ambient intelligence and multi-agent systems to support team collaboration, but in the future we aim to take advantage of smart cities technology to give further support to family eldercare. We applied the proposed ontology to represent this scenario in order to generate the corresponding MAS code in Jason, CArtAgO and Moise. The resulting MAS application was designed to provide the functionalities of activity recognition and task negotiation among agents through the utilization of planning, agent and semantic technologies. More specifically, the assisted living multi-agent application is going to provide the following functionalities for its users:

1. Allocate tasks and commitments considering the context of patient care.
2. Detect if the responsible for the patient is following its appointments/commitments.
3. Detect problems which may prevent a responsible for the patient to attend its tasks.
4. Reallocate commitments among users if required (using a negotiation approach).
5. Generate reminders for users to monitor the patient schedule.

Figure 2 shows how the instances (depicted by purple diamonds) in the agent dimension of the ontology are converted to AgentSpeak code. Each individual of Agent becomes an .asl file, and each Plan instance is written in the corresponding agent file, which is retrieved according to the ontology properties (*e.g.*, **has-plan** and **is-plan-of**).


Individuals by type: Thing
Agent (2)
planner_agent
server_manager
Plan (5)
create_plan_recognizer_artifact
user_observation
connection
create_failure_predictor_artifact
start

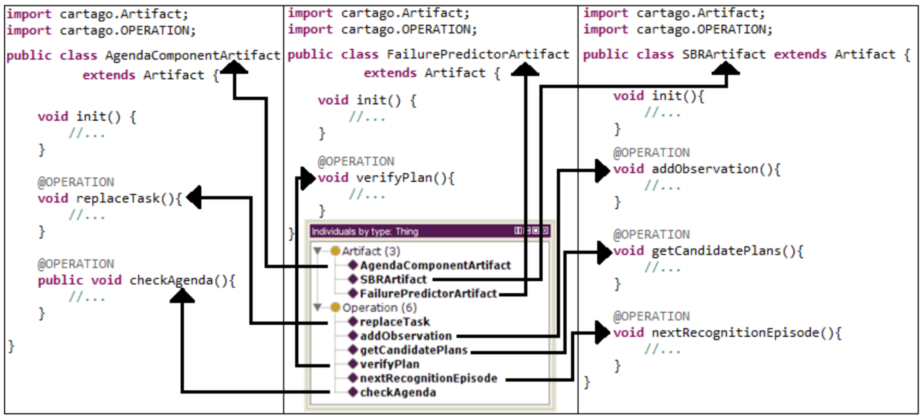
```
// Agent planner_agent

/* Initial beliefs and rules */
//...

/* Initial goals */
//...

/* Plans */
+!start : true <-
    //...
    true.

+!create_plan_recognizer_artifact: true <-
    //...
    true.
+!create_failure_predictor_artifact: true <-
    //...
    true.
+user_observation(Observation)  <-
    //...
    true.
```

```
//Agent server_manager

/* Initial beliefs and rules */
//...

/* Initial goals */
//...

/* Plans */

+!start: true <-
    //...
    true.

+connection(Client): true <-
    //...
    true.
```

Fig. 2. Agent dimension instantiation in our ontology and generation of AgentSpeak code.

The **planner_agent**, shown in Figure 2, is responsible for the heavy reasoning tasks such as plan recognition, failure prediction and plan negotiation and reallocation (in case of failure). The **planner_agent** extracts the relevant information (*e.g.*, specific sensor readings) from the received observation, and, with the extracted contextual information, it tries to recognize the plan that the user is executing using the Plan Recognizer component (the **SBRArtifact** explained later in this paper). After recognizing a plan, the **planner_agent** tries to predict if the plan is going to fail using the Failure Predictor component (implemented in the **FailurePredictorArtifact**, to be explained later). If the agent can not determine which of the plans the user is executing (there is more than one candidate plan), the agent stops the recognition and waits for the next observation. After a new observation is received, the recognition process is resumed. The **server_manager** agent is responsible for setting the system initial infrastructure and for establishing and managing the connections between an interface agent and a planner agent. The interface agents are responsible for collecting sensor readings from the system related devices, such as the user smartphone, and send them to their respective planner agents as observations. However, the interface agent code lie outside the scope of the MAS project exemplified here. The connection of an interface agent is signalized through the event "+connection(Client)" in the **server_manager** agent, in which Client corresponds to the name (identification) of the connected agent.

Figure 3 shows how the instances in the environment dimension of the ontology are converted to CArtAgO code. Each instance of Artifact generates an .java file, and each Operation individual generates a Java method in the corresponding artifact file (according to the **has-operation** property). The integration of agents with artifacts, such as the Plan Recognizer and the Failure Predictor, is based on a set of CArtAgO and Jason agent language elements such as events and beliefs.



```
import cartago.Artifact;
import cartago.OPERATION;

public class AgendaComponentArtifact
          extends Artifact {

    void init() {
        //...
    }

    @OPERATION
    void replaceTask(){
        //...
    }

    @OPERATION
    public void checkAgenda(){
        //...
    }

}
```

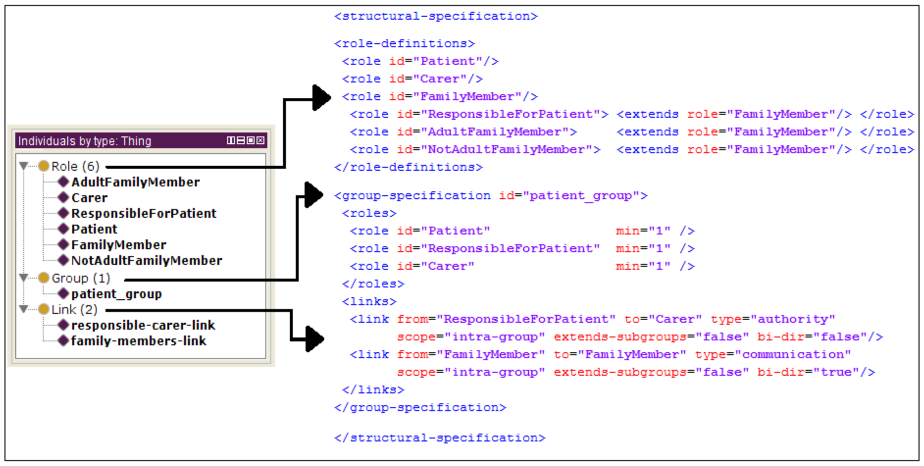
```
import cartago.Artifact;
import cartago.OPERATION;

public class FailurePredictorArtifact
           extends Artifact {

    void init() {
        //...
    }

    @OPERATION
    void verifyPlan(){
        //...
    }
}
```

```
import cartago.Artifact;
import cartago.OPERATION;

public class SBRArtifact extends Artifact {

    void init(){
        //...
    }

    @OPERATION
    void addObservation(){
        //...
    }

    @OPERATION
    void getCandidatePlans(){
        //...
    }

    @OPERATION
    void nextRecognitionEpisode(){
        //...
    }
}
```

Fig. 3. Environment dimension instantiation in our ontology and generation of CArtAgO code.

The **AgendaComponentArtifact** is used to store and manage commitments, thus it uses the operations **replaceTask** and **checkAgenda** to achieve this. The other artifact, named **FailurePredictorArtifact**, was designed to identify whether a plan is likely going to fail. To accomplish this, the **verifyPlan** operation tries to predict if the plan (correspondent to planID) is going to fail given the contextual information (context). The context parameter corresponds to the same list of predicates used to generate the observation in the Plan Recognizer example. The third and last artifact, called **SBRArtifact**, was developed to recognize plans by applying a symbolic approach which derives candidate plans according to a sequence of observations. To achieve this, the **addObservation** operation converts a set of binary predicates (received as a list of java objects) into an observation object that is used as input for the Symbolic Plan Recognition technique. The **getCandidatePlans** operation returns a list of candidate plans (determined based on the previous added observations). The returned list is included in the agent belief base as a predicate containing a list of plan names (identifications). The **nextRecognitionEpisode** operation stops the current recognition episode. When the recognition is stopped, all the information relating to the episode is erased (*e.g.*, observations counter).

Figures 4 and 5 show how the instances in the organization dimension of the ontology are converted to Moise code. The mapping is straightforward since individuals of concepts such as Role, Group, Mission and Norm are directly mapped to the XML Moise code. Figure 4 demonstrates the Moise structural specification for the assisted living application MAS scenario, which defines the following roles: patient, carer, family member, responsible for patient, adult family member and not adult family member. In this context, a group must be composed of at least one patient, one carer and one agent playing the role of responsible for the patient. Moreover, the links establish communication constraints, *e.g.*, agents playing the role of responsible for the patient have authority over carers (*i.e.*, they are allowed to send goals to be achieved by carer agents).

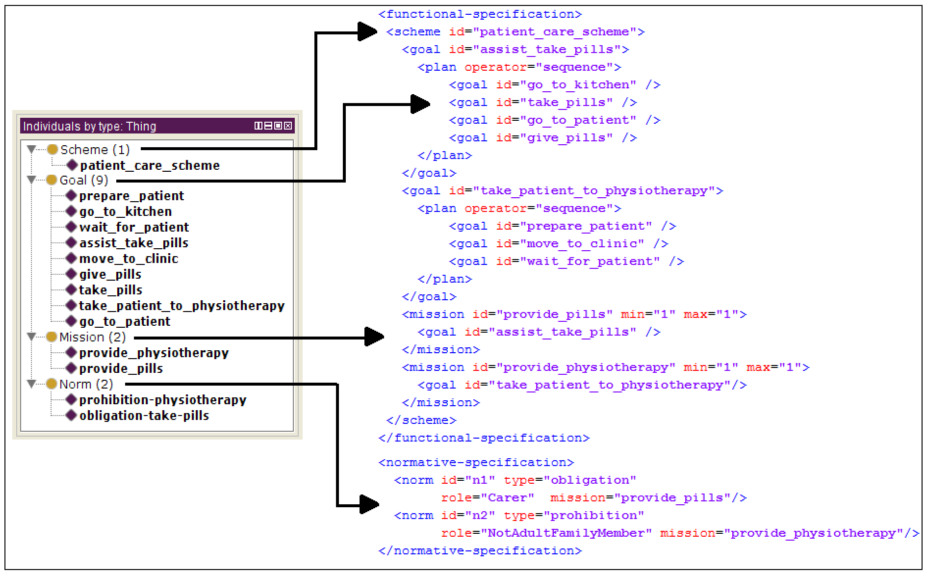


```
Individuals by type: Thing
▼ Role (6)
    AdultFamilyMember
    Carer
    ResponsibleForPatient
    Patient
    FamilyMember
    NotAdultFamilyMember
▼ Group (1)
    patient_group
▼ Link (2)
    responsible-carer-link
    family-members-link
```

```
<structural-specification>

<role-definitions>
 <role id="Patient"/>
 <role id="Carer"/>
 <role id="FamilyMember"/>
  <role id="ResponsibleForPatient"> <extends role="FamilyMember"/> </role>
  <role id="AdultFamilyMember">     <extends role="FamilyMember"/> </role>
  <role id="NotAdultFamilyMember">  <extends role="FamilyMember"/> </role>
</role-definitions>

<group-specification id="patient_group">
 <roles>
  <role id="Patient"               min="1" />
  <role id="ResponsibleForPatient"  min="1" />
  <role id="Carer"                  min="1" />
 </roles>
 <links>
  <link from="ResponsibleForPatient" to="Carer" type="authority"
        scope="intra-group" extends-subgroups="false" bi-dir="false"/>
  <link from="FamilyMember" to="FamilyMember" type="communication"
        scope="intra-group" extends-subgroups="false" bi-dir="true"/>
 </links>
</group-specification>

</structural-specification>
```

Fig. 4. Organization dimension instances in our ontology and generation of Moise code (part 1).

Figure 5 demonstrates the functional and normative specification of the Moise for the assisted living application. In this context, the carer is obligated through a norm to perform the mission of provide pills to the patient. Also, not adult family members are prohibited of executing the mission of proving physiotherapy to the patient. This specification also defines how the missions are decomposed in goals. For example, the mission to provide the patient physiotherapy requires to achieve the goals of prepare the patient, move to clinic and wait for patient.



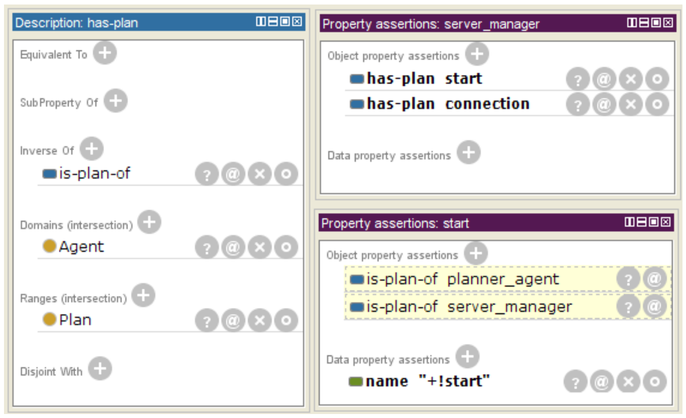
Individuals by type: Thing

- Scheme (1)
  - patient_care_scheme
- Goal (9)
  - prepare_patient
  - go_to_kitchen
  - wait_for_patient
  - assist_take_pills
  - move_to_clinic
  - give_pills
  - take_pills
  - take_patient_to_physiotherapy
  - go_to_patient
- Mission (2)
  - provide_physiotherapy
  - provide_pills
- Norm (2)
  - prohibition-physiotherapy
  - obligation-take-pills

```
<functional-specification>
 <scheme id="patient_care_scheme">
   <goal id="assist_take_pills">
     <plan operator="sequence">
         <goal id="go_to_kitchen" />
         <goal id="take_pills" />
         <goal id="go_to_patient" />
         <goal id="give_pills" />
     </plan>
   </goal>
   <goal id="take_patient_to_physiotherapy">
     <plan operator="sequence">
         <goal id="prepare_patient" />
         <goal id="move_to_clinic" />
         <goal id="wait_for_patient" />
     </plan>
   </goal>
   <mission id="provide_pills" min="1" max="1">
     <goal id="assist_take_pills" />
   </mission>
   <mission id="provide_physiotherapy" min="1" max="1">
     <goal id="take_patient_to_physiotherapy"/>
   </mission>
 </scheme>
</functional-specification>

<normative-specification>
  <norm id="n1" type="obligation"
        role="Carer"  mission="provide_pills"/>
  <norm id="n2" type="prohibition"
        role="NotAdultFamilyMember" mission="provide_physiotherapy"/>
</normative-specification>
```

Fig. 5. Organization dimension instances in our ontology and generation of Moise code (part 2).

The representation of MAS in ontologies provides not only the code generation functionality, but also reasoning features. Figure 6 exemplifies (through the Protégé interface) asserted object properties(*e.g.*, that the server_manager Agent has-plan connection) and inferred properties (*e.g.*, that the start Plan is-plan-of planner_agent).



Description: has-plan

Equivalent To

SubProperty Of

Inverse Of
- is-plan-of

Domains (intersection)
- Agent

Ranges (intersection)
- Plan

Disjoint With

Property assertions: server_manager

Object property assertions
- has-plan start
- has-plan connection

Data property assertions

Property assertions: start

Object property assertions
- is-plan-of planner_agent
- is-plan-of server_manager

Data property assertions
- name "+!start"

Fig. 6. Object properties asserted and inferred through the execution of semantic reasoning.

Semantic reasoning allows to infer knowledge which is implicit derived from axioms explicit asserted in the ontology. Besides inferences, modelling an MAS in ontologies offers the possibilities of enhancing how the developer design and visualize their JaCaMo projects. With this example, we intended to show the importance of the interrelation of these distinct levels, how complementary previous approaches are, and how crucial a unified view at the knowledge-level is for MAS development, which is being proposed by means of ontologies. Thus, this paper explores ways of integrating MAS with semantic technologies in the context of ambient intelligence applications.

4 Final Remarks

Integrating agent platforms with ontologies is expected to bring agents with the ability to operate in a Semantic Web context. We see efforts to integrate different programming dimensions in MAS development frameworks, however, our work is, to the best of our knowledge, the first to address the knowledge level integration of these dimensions. We claim that ontologies have an important role in the whole system development, rather than exclusively in the programming phase. Thus, we are investigating how to enable current agent-oriented development platforms to transparently merge with such semantic technologies. This is in line with CArtAgO development directions [4] in considering ontologies to represent the artifacts, and with Moise recent developments which proposes a semantic description of multi-agent organisations [5]. These approaches may help agents in becoming aware, querying, and reasoning about agent systems in an uniform way [5]. These are however all separate initiatives, whereas in the development of MAS these ontologies should be interconnected within the various specification levels. This allows for an unified view of systems engineering, and should co-exist with integrated agent platforms, such as JaCaMo [1]. As result, developers may obtain a new paradigm for developing complex software systems with a semantic infrastructure applying the software and knowledge engineering principles.

The development of services based on collaborative agents requires a comprehensive view of a complex problem, which includes knowledge about the environment, the relations the agents establish among each other, and the common plans that they have to deal with in a collaborative way. Unified MAS platforms such as JaCaMo [1] are being developed with the purpose of helping developers to build such complex solutions, however, such unification must happen during the system design and at the knowledge level. Thus, we investigated the integration of agent programming platforms and ontologies, by applying ontologies to streamline MAS development in JaCaMo. We exemplify our approach with an application in the smart cities context, which applies ambient intelligence to support eldercare.

The development of applications that integrate semantic and agent technologies is still an open challenge. To address this issue, considering the current development of agent technologies towards a semantic layer, we pointed out that ontology languages offering semantic querying and reasoning should be suitably integrated into agent development frameworks, for example in regards to transparency. We discussed previous work that first proposed merging MAS with semantic technologies, which is the case of AgentSpeak-DL [6], JASDL [7], and Semantic Moise [5]. JaCaMo [1] integrates different levels of MAS paradigms, *e.g.*, agents in Jason [2], artifacts in CArtAgO [4], and organisations in Moise [3]. This work discussed the integration of these levels through ontologies. The inclusion of ontology technologies in MAS is expected to bring together the power of knowledge-rich approaches and complex distributed systems. In terms of MAS design, such an integrated approach allows the design of a global conceptual view, and semantic tools make it possible to verify model consistency, perform inferences using semantic reasoners, query instantiated models, develop and visualize MAS specifications in ontologies, all of which can contribute to a more principled way to develop multi-agent systems.

Acknowledgements

Part of the results presented in this paper were obtained through research on a project titled "Semantic and Multi-Agent Technologies for Group Interaction", sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian federal law No. 8.248/91.

References

1. Boissier, O., Bordini, R.H., Hübner, J., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6) (2013) 747–761
2. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. John Wiley & Sons (2007)
3. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Autonomous Agents and Multi-Agent Systems **20**(3) (2010) 369–400
4. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: an infrastructure for engineering computational environments in MAS. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: 3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS). (2006) 102–119
5. Zarafin, A.M.: Semantic description of multi-agent organizations. Master's thesis, Automatic Control and Computers Faculty, Computer Science and Engineering Department – University "Politehnica" of Bucharest (2012)
6. Moreira, A.F., Vieira, R., Bordini, R.H., Hübner, J.F.: Agent-oriented programming with underlying ontological reasoning. In: Proceedings of the 3rd international workshop on Declarative Agent Languages and Technologies. DALT'05, Berlin, Heidelberg, Springer-Verlag (2006) 155–170
7. Klapiscak, T., Bordini, R.H.: JASDL: a practical programming approach combining agent and semantic web technologies. In: The 6th international workshop on Declarative Agent Languages and Technologies. Volume 5397., Springer (2008) 91–110
8. Okuyama, F.Y., Vieira, R., Bordini, R.H., da Rocha Costa, A.C.: An ontology for defining environments within multi-agent simulations. In: Workshop on Ontologies and Metamodeling in Software and Data Engineering. (2006)