

# OntologyBeanGenerator 5.0: Extending Ontology Concepts with Methods and Exceptions

Daniela Briola

*Department of Computer Sciences,  
Systems and Communications,  
University of Milano Bicocca  
Milan, Italy  
daniela.briola@unimib.it*

Viviana Mascardi    Massimiliano Gioseffi

*Department of Informatics, Bioengineering,  
Robotics, and Systems Engineering,  
University of Genova  
Genoa, Italy  
viviana.mascardi@unige.it*

**Abstract**—When modeling and implementing complex systems based on agents and artifacts, achieving semantic interoperability is not only useful, but often necessary. A commonly adopted solution to manage complex and real MASs is adopting a Model Driven methodology, which uses an ontology as the formal representation of the domain, and then exploiting some existing tool to automatically generate code for agents in the MAS, to let them interact according to the model. While this approach is satisfactorily supported when the target MAS environment is Jason, less support is provided to Jade MASs, despite Jade’s large adoption for real MASs development. So, considering the great support given by the automatic code generation starting from a formal model, and the large community working on Jade MASs, in this work we present an extension of the OntologyBeanGenerator plugin for Protégé, used to generate a Java representation of an OWL ontology for Jade. We improved the OntologyBeanGenerator tool to support the modeling of exceptions, formalized at the ontology level, and of methods associated with ontology elements, to set the interface of concrete objects (artifacts) at design stage. This extension allows us to integrate in a Model Driven approach a support for the formal definition of artifacts and provide an automatic generation of Jade code/interfaces to interact with them respecting the model.

**Index Terms**—Multiagent Systems; Model Driven Design; Jade; Automatic Code Generation; Ontologies; OntologyBeanGenerator

## I. INTRODUCTION

Semantic interoperability, namely the ability to process digital information without losing the true intended meaning of the information itself, plays an increasingly important role in countless areas. Being explicit specifications of a shared conceptualization, ontologies [1] are one of the most widespread tools for representing the entities characterizing a domain, in order to make applications semantically interoperable.

The biomedical domain is the one where the oldest and largest ontologies were born<sup>1</sup>; closely related to that domain is the e-Health one that witnesses a recent boost of both scientific [2], [3], [4] and commercial initiatives. With regard

to e-government, at the European level the ISA program and its ongoing continuation ISA2<sup>2</sup> (2016-2020) demonstrate the effort in developing interoperable digital solutions for public administrations. The report [5] shows how standards have become an essential part of the revolution in the information industry, aiming at interoperability for hardware and software. Finally, in the last few years we are witnessing an increasing need for semantic interoperability also in the IoT [6], [7], [8] and cyber-physical systems domains [9].

Since the advent of the ontologies first, and the Semantic Web later, agents and ontologies have been considered as two sides of the same coin [10], [11], [12]. On the one hand, agents may exploit ontologies to model the environment, along with the involved entities therein and the services they offer, and to achieve a semantically rich message exchange [13], [14], [15], [16]. On the other hand, many stages of the ontology engineering process may benefit from the exploitation of agents, as discussed for example in [17] w.r.t. ontology alignment.

Given this close connection, all the most widespread platforms for MAS engineering offer some facilities to import and manage ontologies, be them used as a representation of the internal agent knowledge like in the JASDL extension [18] of Jason [19], or as a vocabulary shared among agents and as a formal description of agents’s actions, like in Jade [20].

Jade offers facilities to directly import and manage ontologies, provided that they are represented as JavaBeans<sup>3</sup>. The OntologyBeanGenerator 4.1<sup>4</sup> plugin of Protégé<sup>5</sup> automatically creates a Java representation of the ontology coherent with Jade requirements, modeling domain concepts, predicates, and agent actions. Two features not supported by OntologyBeanGenerator 4.1 are:

(1) the capability of associating interfaces (methods) with the ontology concepts representing entities in the domain, and

<sup>2</sup><http://ec.europa.eu/isa/>

<sup>3</sup><https://docs.oracle.com/javase/tutorial/javabeans/>

<sup>4</sup><https://protegewiki.stanford.edu/wiki/OntologyBeanGenerator>

<sup>5</sup><https://protege.stanford.edu/>

<sup>1</sup>See for example <http://www.openclinical.org/ontologies.html> and <http://www.obofoundry.org/>.

(2) the capability of modeling the exceptions thrown by those methods as ontology concepts.

These features are extremely relevant when the MAS under development is as complex as a cyber-physical system, or in general foresees a large set of artifacts (referring to the A&A model [21]). In these cases, the ontology models physical devices that agents may control and act upon via some unified interfaces (their public methods), and malfunctioning of such devices (the exceptions they throw) must be explicitly represented in order not only to immediately react to them, but also to exchange information and reason about them.

Associating the pieces of information related with the standard and exceptional behavior of the entities in the model, with the ontology entities themselves, may improve semantic interoperability by adding compatibility between software systems via reuse of standardized interfaces.

Adopting an ontology as formal model to be used both at design time, and then as a starting point for the automatic code generation of agents and artifacts in the MAS, is the approach foreseen by Model Driven Engineering [22].

In this work we present *OntologyBeanGenerator 5.0*, an extension of *OntologyBeanGenerator 4.1*, designed and implemented to address its shortcomings, having in mind Jade MASs developed exploiting one (or more) ontology to model the domain (seen as the set of environment, agents and artifacts involved in the MAS). So, *OntologyBeanGenerator 5.0* could be used not only in those Jade MASs already exploiting an OWL ontology – in its Java version – but also in those MASs where a formal representation of artifacts, plus an “usage interface” defined at design time, is needed to achieve a more flexible solution.

The paper is organized as follows: Section II motivates the need for *OntologyBeanGenerator 5.0*, Section III describes its implementation, Section IV presents an example of use, Section V discusses the related works and concludes.

## II. MOTIVATION

When engineering a complex, safety-critical MAS, a software engineer might need to pursue the two soft goals below, besides the hard goals imposed by the stakeholder requirements and by privacy and security issues:

- reusability: (s)he should look for existing models of the domain, rather than starting from scratch;
- semantic interoperability: (s)he should look for models represented as ontologies, in order to exploit all the advantages that their adoption brings.

These soft goals may be integrated, for example, into an agile “Gaia-like” agent-oriented software engineering methodology [23] consisting of the following steps:

- 1) identify the Environment model, namely the appliances, devices and resources – more in general, the “artifacts” [21] – characterizing the domain;
- 2) look for an ontology suitable for modeling that Environment: only if none can be reused, create a new one from scratch;

- 3) identify the roles model, plus the services offered by each role;
- 4) identify the interaction model by describing use cases involving agents and artifacts;
- 5) describe the programming interfaces with which artifacts provide their services;
- 6) describe the exceptions such services may raise;
- 7) associate the programming interface and its exceptions with the ontology concepts representing artifacts and move from the MAS design to its implementation.

Depending on the agent-oriented development environment used for the final implementation of the MAS, the last step might take advantage of different supporting tools. In particular, if the target system is Jason, the description of how artifacts provide their services could be integrated into the environment ontology using the tool presented in [24] (see Section V). On the other hand, if the development environment is Jade, some tools and languages (for example [25]) exist which can support the modeling phase, such as the automatic generation of small parts of code but, as far as we know (see more details in Section V), a complete tool for a Model Driven generated MAS developed over Jade is not available to the community, even if Jade is known to be one of the most adopted platform for developing multiagent systems. One quite widespread solution is to use OWL to create a model of the agents, along with the services they offer, and of the known artifacts, and then to automatically generate the Jade skeletons of the agents actions and concepts to be used by Jade agents. Our *OntologyBeanGenerator 5.0* Protégé plugin makes a step forward helping designers in the modeling phase of the artifacts, letting them to model the programmable interfaces of the artifacts, that is, the methods to be used to manage the Java objects representing the artifacts, along with their Exceptions, if any, as foreseen in the 7th step of the “Gaia-like” methodology presented before. This feature can be useful when modeling an artifact managed by an agent which requests a specific interface to interact with and, more in general, can be seen as a new step towards a more solid and exhaustive platform for automatically generate code for Jade MASs. *OntologyBeanGenerator 5.0* (OBG5.0 in the sequel) seamlessly integrates with the previous *OntologyBeanGenerator 4.1* tool available to the community, which is one of the very few able to partially generate Jade-compliant code.

To make our approach and tool clearer, let us consider a concrete scenario where the final goal is to develop a controlling system (as we did for example in [26], [27]) for a smart home implemented as a Jade MAS. The first six steps in the above methodology can be addressed by a domain expert who will design and implement the ontology modeling the controlling system environment, along with its interactions with the agents:

- 1) (S)he realizes that the environment includes a set of Sensors and Smart elements which can be managed thanks to a wifi (or wired) network, plus a set of concepts such as Measurements and Locations;

- 2) (S)he discovers that the DogOnt ontology [28] <http://elite.polito.it/ontologies/dogont.owl> satisfies many of her/his needs and decides to adopt it, taking into consideration the possibility to refine or extend it, if necessary;
- 3) (S)he identifies two roles in the controlling system: ElementController and HouseManager, along with their services;
- 4) (S)he creates use cases, where agents playing the roles identified in step 3 interact and manage the artifacts to solve a set of problems;
- 5) (S)he lists a set of needed interfaces for Sensors and Smart elements;
- 6) (S)he lists a set of needed exceptions to model their known errors/problems.

After this first design phase, the software engineer (possibly, with the help of an expert in Jade MAS design) faces step 7 and exploits OBG5.0 to create an OWL ontology able to model the identified elements:

- 7.1. (s)he imports into Protégé the identified DogOnt ontology, and adds the needed new Concepts (with their properties);
- 7.2. (s)he adds a set of Exceptions;
- 7.3. (s)he models the AgentActions;
- 7.4. (s)he adds the desired methods (raising Exceptions when needed) to the identified elements in the ontology;
- 7.5. (s)he uses the OBG5.0 to export the ontology into Java interfaces and classes. At this point, (s)he is able to implement the MAS by adding the real behavior of the agents and implementing the AgentActions, and adding the body to the methods defined in the ontology for the elements managed in the MAS.

Exceptions in our work have two meanings: the first one is related directly to the methods we are adding to the Concepts, so we need them to let the designer, and then the programmer, define the methods with their errors as normally done in Object Oriented languages; the second is related to the formalization of an “Error/Exceptional State/Situation”, which in this case should be modeled in the ontology and, if needed, should be sharable between agents, for example, as the result of a service execution that an agent was not able to satisfy. Since we are targeting Jade, the only way an agent referring to a specific ontology could answer with an Exception (as for example for AgentAction result, as the content of a message with Failure performative) is to have the Exception as subclass of `Jade:Concept`. For this reason, to let the designer the freedom of using the exceptions as (s)he prefers, we decided to model them under `Concept`.

In Sections III and IV we will use this running example to show how to extend the DogOnt ontology and to implement a simple Jade MAS exploiting it.

### III. ONTOLOGYBEANGENERATOR 5.0

OntologyBeanGenerator is a tab widget for Protégé and is distributed under an open source license. The last version available is 4.1. OntologyBeanGenerator

adds a tab to the Protégé interface to directly export ontologies in Java, meeting the requirements for being used in Jade. An ontology containing Jade entities (`OWLSimpleJADEAbstractOntology`) is distributed with the plugin: the entities of the ontology  $O$  we want to make available for Jade must be reorganized under the pre-defined concepts offered by `OWLSimpleJADEAbstractOntology` (`Concept`, `AgentActions`, `AID` and `Predicate`), according to their semantics. Once this refactoring has been completed, the `OntologyBeanGenerator` tab can be used to generate the Java version of the “extended” ontology (namely, the `OWLSimpleJADEAbstractOntology` extended with  $O$  entities positioned in the right place).

OBG5.0, available from [www.disi.unige.it/person/MascardiV/Download/OBG5.0.zip](http://www.disi.unige.it/person/MascardiV/Download/OBG5.0.zip), has been built as a Protégé plugin extending the `OntologyBeanGenerator` 4.1 (OBG4.1). The `OntologyBeanGenerator` developer Chris van Aart authorized this extension, which aims at: 1) correcting some errors in OBG4.1; 2) adding a way to specify methods in the ontology; 3) adding a way to model, and then directly exploit, exceptions. The main improvements of OBG5.0 w.r.t OBG4.1 are:

- Addition of a new tab (called **Java Method Mapper**) to manage methods creation and exportation, to allow the MAS/ontology designer to add methods to the Java version of the ontology directly. This is necessary if we want to add methods that cannot be mapped into setters/getters of a specific field (which were already supported by OBG4.1). Methods are saved in an XML file which is loaded by the tab and overwritten when exporting the ontology, and are added to the interfaces and to their default implementations;
- Exception management: methods are allowed to raise Exceptions. To do this, a specific ontology has been created;
- Update of the ontology type, now is `BeanOntology`;
- Automatic import of the Default Implementations of the interfaces in the Java Ontology;
- Conversion of the `boolean` and `int` Protégé types into the wrapper `Boolean` and `Integer` Java classes, to support the exportation of multiple properties into lists (currently, OBG4.1 exports them as a List of Object, but this does not work with the methods for adding or removing elements from the list);
- Automatic disambiguation of Java types: we now use the fully qualified names when exporting types from `java.lang` or `java.util` to avoid ambiguities with the Jade classes;
- Creation, after the compilation step, of the Java jar.

**The Java Method Mapper tab widget.** Figure 1 shows the OBG5.0 **Java Method Mapper**: when selecting a class from the left area, the user can see the methods (if any) associated with that class on the right area. Setters and getters do not appear, since they are managed by the OBG4.1 “core” code. If the method name is preceded by “this::”, it means that it is

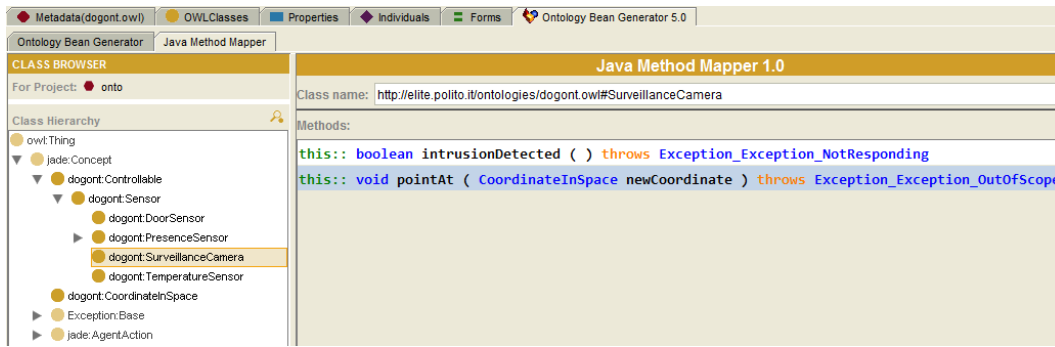


Fig. 1: The **Java Method Mapper** tab with the methods list.

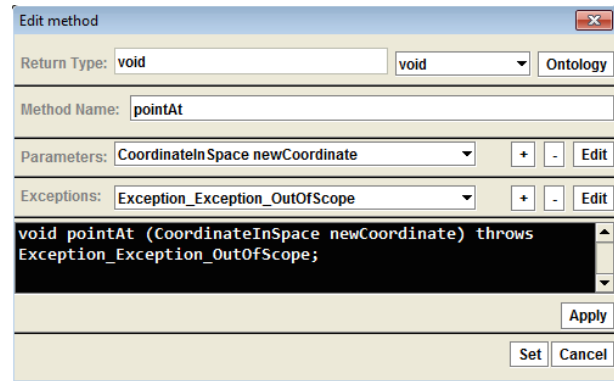
defined in the current class: if the method name is preceded by another string, that string is the name of a parent class (the method is inherited from the parent class). A user can remove or modify methods of the selected class only: with a double click on an inherited method, it is possible to override it.

The “Add” and “Remove” buttons add a new method to the selected class and remove the selected method respectively. In the lower part of the tab, the button “Method File” loads, for the entire ontology (not only for the selected class), all the methods previously created for the ontology. The two “Save” buttons save the current changes or create a new XML file, which is used when exporting the ontology using the other tab of the plugin.

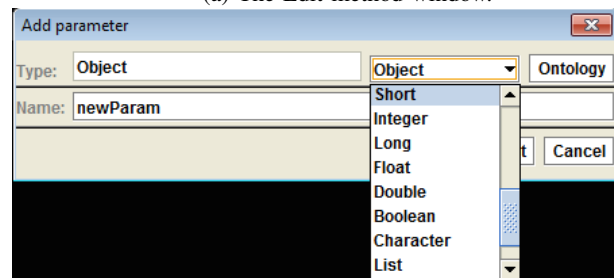
**The add/edit method interface.** When we add a new method to a class or we edit an already existing method, the Edit Window opens (shown in Figure 2a). The first part of the dialog window is composed by these elements:

- Return Type of the method: allows to select the return type from a combobox. The user can chose between primitive Java types, Wrapper types, and the collections List, Set and Map from `java.lang`. With the button “Ontology” the user can instead select one class from the ontology (a windows with the ontology structure is opened, and the user can navigate it and select a class).
- Method name: the user can insert a valid string as a name (an alphanumeric string starting with `_` or a character, not containing a Java keyword).
- Parameters: The button “+” opens the window Add/Edit Parameters (shown in Figure 2b). The button “-” deletes the selected parameter.
- Exceptions: with the button “+”, a window showing the ontology part regarding the Exceptions is opened, so that the user can select one (see below). The button “-” removes the selected Exception.

In the lowest part of the window, a text area shows the resulting Java code for the method. This area can be manually inspected and its content can be changed: with the “Apply” button, a syntactic checker is performed thanks to ANTLR<sup>6</sup>, to verify that the method is well written, then a semantic check



(a) The Edit method window.



(b) The Add parameters window.

Fig. 2: The Edit method dialog and the Add new parameter dialog.

is performed, to verify that the types involved in the method exist either in Java or in the ontology, and if all is correct, the previous fields are then updated.

With the “Set” button, a check over the method and parameters names is performed. If we are adding a new method, the check verifies that the method does not already exist. Overloading is permitted, and checked when adding/modifying a method in a class. Overriding is permitted as well: if the user double clicks on an inherited method from the Java Method Mapper tab, a new dialog windows will open: from this, the user can redefine the exceptions or the return type, changing them into a more specific type. If the user confirms the changes, and they are correct, the method is added (if it was the first time it was modified) or updated in the tab.

<sup>6</sup><http://www.antlr.org/>

**Exception management.** To let methods use Exceptions, the user is requested to modify a template ontology modeling the Exceptions (s)he wants to use: in this new ontology, under the already existing Base class (that is a subclass of `Jade:Concept`), the user must create all the Exceptions (s)he will use. Then, this Exception ontology must be imported in the main ontology, and must be given the prefix Exception (that can be set up in the Metadata tab in Protégé). During the export phase, the tool will automatically modify the class `Exception:Base` so that it extends `java.lang.Exception`: in this way, all its subclasses will be automatically Exceptions. The `Exception:Base` class is already a subclass of `Jade:Concept`, and can be consequently used in the Jade MAS as the user prefers, for example, as content in a message.

It is not allowed to add methods to the Exception classes.

**The BeanOntology class.** Starting from Jade 3.6.1, a class called `jade.content.onto.BeanOntology` is adopted by Jade to represent an ontology: this class lets the programmer to directly create ontology schemas starting from Java classes (if they respect the Bean convention), without having to manually specify all the elements in the ontology. For example, if we want to import a new Java implementation of the C ontology class (called `CImpl.java`), we can simply write the code `ontology.add(CImpl.class)` avoiding to manually create each element present in the class.

Another functionality provided by OBG5.0 is directly importing a package of classes (using the code `ontology.add("packageName")`), so that to avoid registering every class in the package. That also means that one line of code is enough to include a new set of concrete classes, so the user may develop its classes (implementing the exported interfaces) and, adding one line of code, (s)he can directly use them in Jade: anyway, if the user is not using methods (or (s)he is adopting other classes composition), (s)he can directly import the automatically created jar with no need to modify the code.

#### IV. OBG5.0 EXAMPLE

Let us consider the SmartHome Controlling system domain presented in Section II. We started from the already available DogOnt ontology [28], which gives a device/network independent description of houses with smart entities, including both “controllable” and architectural elements, and we kept only a very small subset of the existing elements (those coherent with our example, that is a very simple example foreseen to show the capabilities of our tool and its functioning, with no claim to be completely realistic).

Figure 3 shows the DogOnt (part of) extended ontology, while the methods added to `SurveillanceCamera` are reported in Figure 1.

**The new concepts and the artifacts’s methods.** `SurveillanceCamera` comes from the DogOnt existing ontology, but the software engineer in charge for the ontology modeling added to it a property `pointingAT` of type

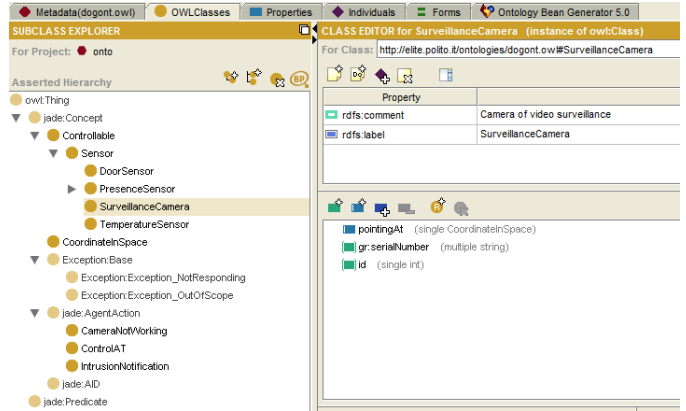


Fig. 3: (Part of) DogOnto ontology, extended with exceptions and new concepts.

`CoordinateInSpace` (a new concept for specifying a point in the 2D area of the camera). The camera identifies the movement of an intruder and, thanks to the boolean method `intrusionDetected`, can notify the caller – its own controlling agent in this case – of this situation. This method raises an exception of type `Exception_NotResponding` if the camera is not working properly or is unable to perceive a clear image (the image is completely filled with a unique color) because, for example, the intruder covered the camera with a piece of paper. The camera may change its orientation (the `pointingAT` value) thanks to the new method `pointAt`, which actually rotates the camera and raises an exception if the requested point cannot be framed.

**Use case and interactions.** We consider a use case related to home security. We assume to have an agent associated with each sensor in the house: in this very simple example, we have two `SurveillanceCameras`, each one associated with a `CameraAgent` (with the role `ElementController`), and a `Manager` agent (with the role of `HouseManager`) which receives either notifications of intrusion from the `CameraAgents` (`IntrusionNotification AgentAction`) or notifications of problems (for example, a camera not responding): when a `CameraAgent` sends the `ManagerAgent` the `CameraNotWorking AgentAction`, it will also specify the last coordinate `LastC` the camera was pointing at: we could otherwise include the Exception itself in the message and let the receiving agent decide what to do, but in this example due to space limitation, we decided to not exchange Exceptions between agents, and we adopt a simpler behavior where the `CameraAgent` already knows how to react when an Exception is raised by its associated device. In that situation, the `Manager` will select another `Camera` and will ask the associated `CameraAgent` to point to that coordinate `LastC`: consequently, the camera which moved because of this request will start patrolling that point and, if an intrusion is detected, it will inform the `Manager`. In this way, if a camera is not working or has been deactivated, the `Manager` is immediately notified and is able to coordinate

the other cameras to check the situation.

Figure 4 provides a general overview of how agents interact by means of a UML sequence diagram: agents exchange messages containing AgentActions defined in the ontology, and CameraAgents interact with their respective SurveillanceCamera artifacts using methods in the ontology.

**Jade implementation.** We implemented this simple MAS in Jade. Figure 5 shows a screenshot from the Jade Sniffer Agent. At the beginning of the surveillance activity, Camera\_1 and Camera\_2 are patrolling two disjoint areas. Then, Camera\_1 (a CameraAgent) notifies the Manager that the camera is no longer working properly: the Manager requests Camera\_2 (another CameraAgent) to point where Camera\_1 was pointing, and at that point Camera\_2 perceives an intruder and notifies the Manager, which will for example contact the police. Later on we report the content of the FIPA-ACL request sent by the Manager to Camera\_2, so that the reader can see how the ontological concept (ControlAT AgentAction) is used.

ACL Message Content:

```
((action
  (agent-identifier
    :name Manager@10.0.1.5:1099/JADE)
  (DefaultControlAT
    :newCoordinate
      (DefaultCoordinateInSpace
        :x 3
        :y 4))))
```

To give the reader an idea of how the Java code output by OBG5.0 looks like, we show below a fragment of the code produced by the plugin. In particular, we report part of the code for adding concepts and properties to the ontology, and part of the code in the interface SurveillanceCamera to manage the new methods.

```
public class SensorsOntoOntology extends jade.content.onto.
BeanOntology {
  private SensorsOntoOntology(){
    super(ONTOLOGY_NAME, BasicOntology.getInstance());
    try {
      // Concepts, AgentActions and Exceptions (part of code)
      add(AIExportDougSensor.exceptions.Exception_Base.class);
      add(AIExportDougSensor.ControlAT.class);
      //new! done to automatically add all the default classes
      add("AIExportDougSensor.impl");
    } catch (BeanOntologyException e) {e.printStackTrace();}
  }
}

public interface SurveillanceCamera extends Sensor {
  public boolean intrusionDetected ( )
  throws Exception_Exception_NotResponding ;
  public void pointAt (CoordinateInSpace newCoordinate)
  throws Exception_Exception_OutOfScope ;
}
```

## V. RELATED WORKS AND CONCLUSIONS

The research field in the intersection of multiagent systems and semantic web technologies is an extremely lively one. As recently observed in [24], it has also close connections with model-driven engineering, since the model which drives the MAS engineering may be profitably represented as an ontology. In our research activity, we have often used OntologyBeanGenerator, for example in [29], [30], [31], [32], [33],

[34], [35], [36], where the MASs were centered around the domain ontology, and in many cases they were driven by it.

Starting from [37], a lot of effort has been devoted to the integration of ontologies into the AgentSpeak language [38], and more practically into Jason [18], [39], [40]. One of the last works in this series [24] presents a new modeling approach where MASs are designed by instantiating an ontology developed by the authors, and a tool uses these instances to generate code for such systems.

When moving from Jason to Jade, the number of available tools and scientific papers dramatically decreases. Since Jade ontologies must be represented in Java, it is common practice to model them in OWL using Protégé, and then to use some existing tool that automatically generates the Java representation of the ontology. The process of exporting an OWL ontology into a Java representation was faced in similar ways by [41]-[42] and [43]; anyway [43], which gave birth to the OntologyBeanGenerator tool, was more successful, probably due to its simplicity and its direct integration into Protégé, and is still today widely used.

Whereas we can already add properties while modeling entities in OWL aimed at being used in Jade, we have no way to model methods nor Exceptions.

Our understanding of the issues raised by MAS engineering along with our knowledge of the OntologyBeanGenerator tool and of its limitations, plus a comparison with platforms like CARtAgO [40] and JaCaMo [44], suggested us the extensions described in this paper. The analysis of the literature confirmed that, currently, no comprehensive Model Driven Tool with Jade as a target exists.

Our implementation of OBJ5.0, freely available to the community, improves the previous version of the tool by fixing a set of known problems; it also addresses some issues discussed for example in [45], where the authors argue that an exception ontology, along with a domain one, may increase the MAS reliability and enhance its fault tolerance capability. By adding Exceptions both to methods and as Concepts in the ontology, we believe OBJ5.0 may represent a very first step towards the design of a Model Driven platform for developing secure and reliable Jade MASs.

An ideally similar approach is the one proposed by the W3C with OWL-S (<https://www.w3.org/Submission/OWL-S/>), which is an ontology modeling Web Services: this ontology is born mainly to be used for the Semantic Web, to support the semantic annotation and invocation of web services and also of entities in IoT environments. In this ontology, each Service has a Profile, describing what the service offers, a Model, to explain how to use it, and a Grounding, to specifically describe how to interact with it. In some way, the grounding may be similar to our methods, but the difference is in the general approach. We created a way to add methods to Concepts in an ontology that represents agents, services and artifacts to be used in a Jade MAS, so, we are already acting in a specific model (the one requested by Jade), foreseen for the usage inside the MAS. OWL-S is foreseen to model how an agent could interact with an external Web Service, so its scope

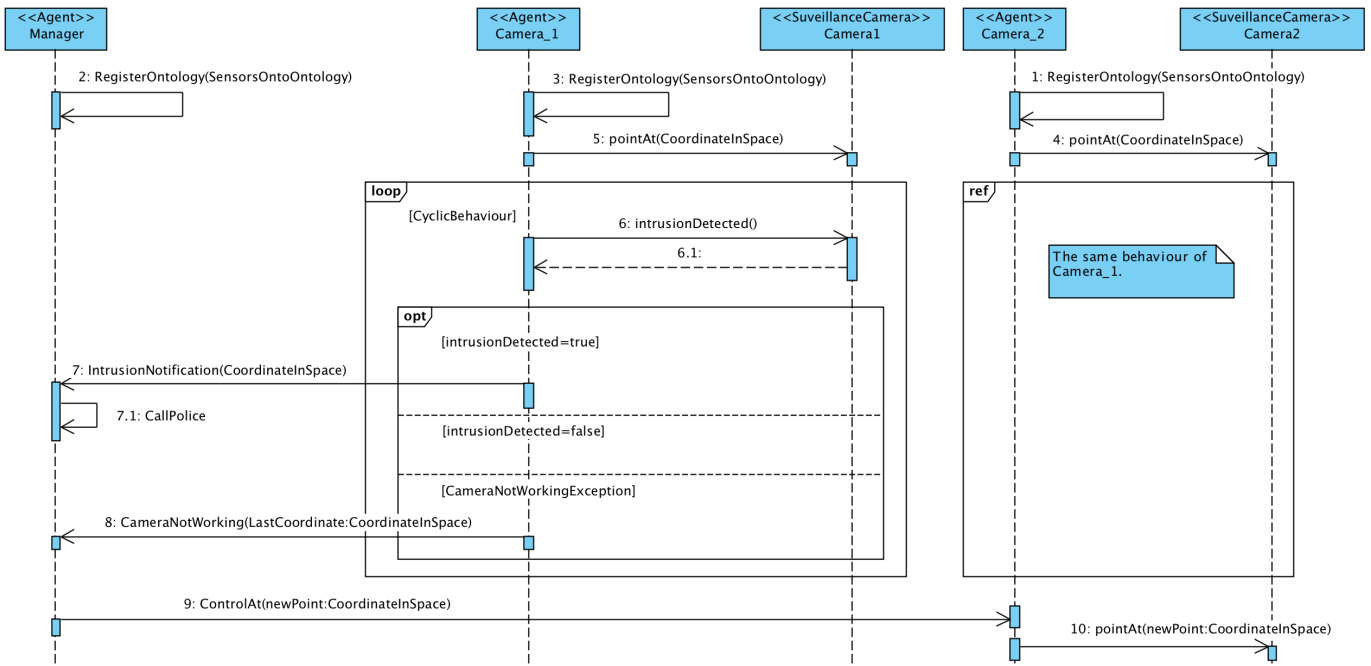


Fig. 4: CyclicBehaviour is always active: after Camera<sub>2</sub> receives the pointAt call from Manager, it starts the loop again, calling the intrusionDetected method. We assume here that method pointAt never throws exception. Note that agents exchange AgentActions each other, while agents manage artifacts thanks to their methods.

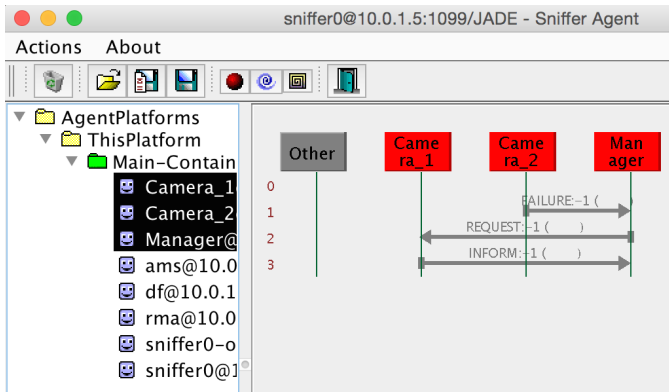


Fig. 5: Screenshot from the Jade Sniffer

is slightly different (we do not have access on the Java object representing the Remote Web Service, while we have direct access to the Java object representing the specific artifact in the MAS). Nevertheless, since OWL-S is an OWL ontology, we could evaluate to integrate it (after understanding how to relate the Jade classes with the OWL-S existing classes) and directly exploiting it in our tool, maybe as a new part (not alternative but parallel to the current one): in this way, we could meet the requirements of a larger MAS designer community. This study is part of the future work.

As further extensions, we plan to add more types in the methods definition (the Array type and other less used Java types, which could be specified by the user in a configuration file to get greater flexibility), and to allow the Java Generics

usage. We would also directly let the user insert the methods code (to be automatically used in the Default concrete classes), and static methods. Finally, since OBG5.0 does not work with the most recent releases of Protégé, we plan to exploit the lessons learned with its design and development, to create a completely new version of OBG5.0 compliant with Protégé 5.2.0.

## REFERENCES

- [1] T. Gruber, "Ontology," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer, 2009, pp. 1963–1965.
- [2] A. Taweel, "Semantic interoperability-enabled architecture for connected health services," in *Reshaping medical practice and care with health information systems*. IGI Global, 2016, pp. 246–265.
- [3] J. C. Mandel, D. A. Kreda, K. D. Mandl, I. S. Kohane, and R. B. Ramoni, "SMART on FHIR: a standards-based, interoperable apps platform for electronic health records," *Journal of the American Medical Informatics Association*, vol. 23, no. 5, pp. 899–908, 2016.
- [4] S. Bhalla, S. Sachdeva, and S. Batra, "Semantic interoperability in electronic health record databases: Standards, architecture and e-health systems," in *International Conference on Big Data Analytics*. Springer, 2017, pp. 235–242.
- [5] V. Peristeras, "Semantic standards: Preventing waste in the information industry," *IEEE Intelligent Systems*, vol. 28, no. 4, pp. 72–75, July 2013.
- [6] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmaja, and K. Wasielewska, "Semantic interoperability in the Internet of Things: An overview from the INTER-IoT perspective," *Journal of Network and Computer Applications*, vol. 81, pp. 111–124, 2017.
- [7] S. Jabbar, F. Ullah, S. Khalid, M. Khan, and K. Han, "Semantic interoperability in heterogeneous IoT infrastructure for healthcare," *Wireless Communications and Mobile Computing*, vol. 2017, 2017.
- [8] G. Bajaj, R. Agarwal, P. Singh, N. Georgantas, and V. Issarny, "A study of existing ontologies in the IoT-domain," *CoRR*, vol. abs/1707.00112, 2017.
- [9] V. Jirkovský, "Semantic integration in the context of cyber-physical systems," czech Technical University in Prague, Doctoral Thesis. [Online]. Available: <https://dspace.cvut.cz/handle/10467/73622>

- [10] M. N. Huhns and M. P. Singh, "Ontologies for agents," *IEEE Internet computing*, vol. 1, no. 6, pp. 81–83, 1997.
- [11] J. Hendler, "Agents and the semantic web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 30–37, 2001.
- [12] V. Mascardi, J. A. Hendler, and L. Papaleo, "Semantic web and declarative agent languages and technologies: Current and future trends - (position paper)," in *DALT*, ser. Lecture Notes in Computer Science, vol. 7784. Springer, 2012, pp. 197–202.
- [13] K.-t. Ng and Q. Lu, "Improve the service quality of multi-agent system: Ontology management," in *Proc. of AAMAS*. ACM, 2003, pp. 1078–1079. [Online]. Available: <http://doi.acm.org/10.1145/860575.860804>
- [14] D. Briola, A. Locoro, and V. Mascardi, "Ontology agents in FIPA-compliant platforms: A survey and a new proposal," in *Proc. of WOA 2008*, 2008, pp. 68–75.
- [15] S. M. Deen and K. Ponnampertuma, "Dynamic ontology integration in a multi-agent environment," in *Proc. of AINA 2006*, vol. 1, April 2006.
- [16] R. Brena and H. G. Ceballos, "Combining local and global access to ontologies in a multiagent system," *Journal of Advanced Computational Intelligence*, vol. 9, no. 1, 2005.
- [17] J. Euzenat, "Crafting ontology alignments from scratch through agent communication," in *Proc. of PRIMA*, ser. LNCS, vol. 10621. Springer, 2017, pp. 245–262.
- [18] T. Klapiscak and R. H. Bordini, "JASDL: A practical programming approach combining agent and semantic web technologies," in *Proc. of DALT*, ser. LNCS, vol. 5397. Springer, 2008, pp. 91–110.
- [19] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.
- [20] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
- [21] A. Ricci, M. Viroli, and A. Omicini, "The A&A programming model and technology for developing agent environments in MAS," in *Proc. of PROMAS*, ser. LNCS, vol. 4908. Springer, 2007, pp. 89–106.
- [22] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.
- [23] F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing multiagent systems: The Gaia methodology," *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 3, pp. 317–370, 2003.
- [24] A. Freitas, R. H. Bordini, and R. Vieira, "Model-driven engineering of multi-agent systems based on ontologies," *Applied Ontology*, vol. 12, no. 2, pp. 157–188, 2017.
- [25] F. Bergenti, E. Iotti, S. Monica, and A. Poggi, "Agent-oriented model-driven development for jade with the jadel programming language," *Comput. Lang. Syst. Struct.*, vol. 50, no. C, pp. 142–158, Dec. 2017. [Online]. Available: <https://doi.org/10.1016/j.cl.2017.06.001>
- [26] V. Mascardi, D. Briola, M. Martelli, R. Caccia, and C. Milani, "Monitoring and diagnosing railway signalling with logic-based distributed agents," in *Proc. of the International Workshop CISIS'08*, E. Corchado, R. Zunino, P. Gastaldo, and Á. Herrero, Eds. Springer Berlin Heidelberg, 2009, pp. 108–115.
- [27] D. Briola, V. Mascardi, M. Martelli, G. Arecco, R. Caccia, and C. Milani, "A prolog-based mas for railway signalling monitoring : Implementation and experiments," in *Proceedings of the Workshop on Objects and Agents, WOA'08*, 2008.
- [28] D. Bonino and F. Corno, "DogOnt - ontology modeling for intelligent domotic environments," in *The Semantic Web - ISWC 2008*, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 790–803.
- [29] A. Locoro, V. Mascardi, D. Briola, M. Martelli, M. Ancona, V. Deufemia, L. Paolino, G. Tortora, G. Polese, and R. Francese, "The Indiana MAS Project: Goals and Preliminary Results," in *Proceedings of the 13th Workshop on Objects and Agents Milano, Italy, September 17-19, 2012*, G. V. Flavio De Paoli, Ed., vol. CEUR Vol-892. CEUR, 2012. [Online]. Available: <http://ceur-ws.org/Vol-892/paper10.pdf>
- [30] Mascardi, V., Briola, D., et al., "A holonic multi-agent system for sketch, image and text interpretation in the rock art domain," *Int. J. of Innovative Computing, Information and Control*, vol. 10, no. 1, pp. 81–100, 2014.
- [31] D. Briola, V. Deufemia, V. Mascardi, and L. Paolino, "Agent-oriented and ontology-driven digital libraries: the IndianaMAS experience," *Software: Practice and Experience*, vol. 47, no. 11, pp. 1773–1799, 2017.
- [32] D. Briola, V. Deufemia, V. Mascardi, L. Paolino, and N. Bianchi, "Ontology-driven processing and management of digital rock art objects in IndianaMAS," in *Proc. of EuroMed 2014*, 2014, pp. 217–227.
- [33] M. Leotta, S. Beux, V. Mascardi, and D. Briola, "My MOoD, a multimedia and multilingual ontology driven MAS: Design and first experiments in the sentiment analysis domain," in *Proc. of ESSEM workshop*, ser. CEUR-WS.org, vol. 1351, 2015.
- [34] D. Briola, "Agents and ontologies for a smart management of heterogeneous data: The IndianaMAS system," in *Intelligent Distributed Computing IX*, P. Novais, D. Camacho, C. Analide, A. El Fallah Seghrouchni, and C. Badica, Eds. Springer International Publishing, 2016, pp. 25–36.
- [35] M. Bozzano, D. Briola, D. Leone, A. Locoro, L. Marasso, and V. Mascardi, "MUSE: MULTilinguality and SEMantics for the citizens of the world," *Studies in Computational Intelligence*, vol. 446, pp. 97–102, 2013.
- [36] D. Briola, V. Mascardi, M. Martelli, R. Caccia, and C. Milani, "Dynamic resource allocation in a MAS: A case study from the industry," *Proceedings of the 10th Workshop on Objects and Agents, WOA'09*, 2009.
- [37] Á. F. Moreira, R. Vieira, R. H. Bordini, and J. F. Hübner, "Agent-oriented programming with underlying ontological reasoning," in *Proc. of DALT*, ser. LNCS, vol. 3904. Springer, 2005, pp. 155–170.
- [38] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language," in *Proc. of MAAMAW*, ser. LNCS, vol. 1038. Springer, 1996, pp. 42–55.
- [39] V. Mascardi, D. Ancona, M. Barbieri, R. H. Bordini, and A. Ricci, "Cool-AgentSpeak: Endowing AgentSpeak-DL agents with plan exchange and ontology services," *Web Intelligence and Agent Systems*, vol. 12, no. 1, pp. 83–107, 2014.
- [40] A. Freitas, A. R. Panisson, L. Hilgert, F. Meneguzzi, R. Vieira, and R. H. Bordini, "Integrating ontologies with multi-agent systems through CArtAgO artifacts," in *WI-IAT (2)*. IEEE Computer Society, 2015, pp. 143–150.
- [41] M. Tomaiuolo, F. Bergenti, A. Poggi, and P. Turci, "OWLBeans - from ontologies to Java classes," in *Proc. of WOA'04*. Pitagora Editrice Bologna, 2004, pp. 116–125.
- [42] M. Tomaiuolo, P. Turci, F. Bergenti, and A. Poggi, "An ontology support for semantic aware agents," in *Agent-Oriented Information Systems III*, M. Kolp, P. Bresciani, B. Henderson-Sellers, and M. Winikoff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 140–153.
- [43] C. van Aart, R. Pels, G. Caire, and F. Bergenti, "Creating and using ontologies in agent communication," in *Proc. of OAS*, 2002.
- [44] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with JaCaMo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747 – 761, 2013, special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) & Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.
- [45] N. Shah, R. Iqbal, A. James, and K. Iqbal, "Exception representation and management in open multi-agent systems," *Information Sciences*, vol. 179, no. 15, pp. 2555 – 2561, 2009.