ICIT 2015 The 7th International Conference on Information Technology
doi:10.15849/icit.2015.0020 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

Bridging the Gap between Modeling of Mobile Agent-based Systems and Semantic Web using Meta-Modeling and Graph Grammars

Aissam Belghiat[1,2]
[1]Département d'informatique, Université 20 Août 1955-Skikda
Skikda 21000, Algérie
belghiatissam@gmail.com

Allaoua Chaoui[2]
[2]MISC Laboratory, Department of Computer Science, University of Constantine2
Constantine 25000, Algeria
a_chaoui2001@yahoo.com

Ali Aldahoud
Al-Zaytoonah University of Jordan,
P.O. Box 130, Amman 11733,Jordan
aldahoud@zuj.edu.jo

***Abstract*—Recently, the mobile agent-based paradigm has received more attention especially in distributed systems where it provides multiple solutions to several problems which can't be resolved by the object-based paradigm. Unfortunately, this paradigm lacks the interconnection with semantic web standards such as the language OWL (Ontology Web Language) which make it far from profiting from research results and advances in this area. We try in this paper to bridge the gap between the two domains by proposing an integrated approach for modeling mobile agent-based software systems using a transformation of mobile class diagrams into OWL ontologies. The developed approach allows interconnection of mobile agent and Semantic Web technologies can be used in a mobile agent-based application where such interconnection is needed. We use the meta-modeling and graph grammars tool AToM^3.**

Keywords—*M-UML; OWL; MDA; Graph Transformation; AToM^3*

I. Introduction

Mobile agent-based software systems are increasingly very complex; actually the development of such systems is a difficult task since the great number of constraints that are evolved during the development process such as the mobility and security. Modeling and designing of mobile agent-based software systems have received important attention in the last years to deal with previous problems [18]. M-UML [19] has been introduced as an extension of UML [7] for modeling mobile agent-based systems [26]. Researchers have tried to relate mobile agent paradigm to the object oriented paradigm using the standard of object oriented modeling to model mobile agents by introducing to it the appropriate artifacts in order to support the new paradigm.

In other side, the ontologies provide explicit and formal specifications of shared conceptualizations; they are described formally using description logics implemented in OWL language. The knowledge generated from an M-UML diagram during the software development process is a valuable asset in particular in the analysis and design tasks. In order to profit from it, they must be represented and stored in ontologies and will be used for reasoning on mobile agent based software systems.

In this paper, which extends our previous work [20], we propose a set of rules for transforming mobile class diagrams into ontologies described in OWL language in order to profit from the power of ontologies. So, the knowledge described by those diagrams can be reused, shared and linked with other information. The meta-modeling tool AToM^3 is used to create meta-models for mobile class diagram and OWL models. A graph grammar is proposed for automatic transformation between the two formalisms.

The rest of the paper is organized as follows. In Section 2, we present some related works. In Section 3, we present some basic notions about M-UML, OWL, and graph grammars. In Section 4, we describe our approach of transforming M-UML class diagrams to OWL ontologies. In Section 5, we illustrate our approach through an example. Finally concluding remarks and perspectives are presented in Section 6.

II. Related Works

Several works exist in the literature in the context of extracting ontologies from UML diagrams. In [14] the authors have proposed a transformation of UML towards DAML by showing similarities and differences between the two parts of the translation. In [15] the authors have proposed a transformation of a profile UML OUP (Ontology UML Profile) towards an ontology OWL. In [6], the OMG remarks the interest of such subject, it then immediately proposed the ODM which provides a profile for writing RDF and OWL within UML. The ODM also includes partial mappings between UML and OWL. It should be noted that several works are performed as answer to the call of the OMG and gathered in the ODM and it is impossible to evoke all of them here. In [9], an implementation of the ODM using ATL language is presented. In [5], the author has applied a style sheet on a XMI file which represents a model of a class diagram to generate an ontology OWL DL represented as RDF/XML format. In [16], a detailed comparison between UML and OWL has been developed.

On the other hand, $AToM^3$ has been adopted to be a very powerful tool which the meta-modeling and the transformations between formalisms. In fact, in [21] the authors have proposed a formal framework and a tool for the specification and verification of G-Nets models using graph transformation. In [20] the authors have developed an $AToM^3$ based approach for the automatic generation of OWL ontologies from UML diagrams. In [23], the authors have developed an approach for modeling and analysis of mobile agent-based software systems by transforming M-UML statecharts models to nested nets models. In [22], an approach for transforming mobile activity diagrams to nested Petri nets models has been proposed. Also in [1, 2, 3, 17, 18, and 25] we can found treatment and translation of multiple UML diagrams. In these works the Meta modeling allows visual modeling and graph grammar allows the transformation among them.

In contrast to all these previous works, we have the first who think to relate the modeling of mobile agent-based systems and semantic web by translating a profile of UML class diagram for mobile systems into the OWL.

III. Background

The main contribution of the paper is to develop an integrated environment based on meta-modeling and graph grammars for modeling and analysis of mobile agent based software systems which are modeled as a set of mobile class diagrams. We recall here some notions about M-UML and OWL.

A. Mobile Class Diagram

A M-UML Class Diagram [19] has been introduced to describe the static structure of a mobile system by showing all relationships between different types of classes. Mobile objects/agents are created by the instantiation of a mobile class shown with a box (M). A class inherits the mobility by the relation of inheritance while not necessarily true by aggregation relation. An affected class shown with a dashed box (M) is a class that contains methods (behavior) which communicate with other mobile classes. A remote class shown with a dashed box (R) is a class that contains methods (behavior) which communicate with a remote mobile object/agent. A mobile object/agent invokes methods which will be labeled depending on the location of it with either (M) or (R) and a class that includes the two types will show with dashed boxes (M) and (R). Figure 1 [19] shows an example of a mobile class diagram.

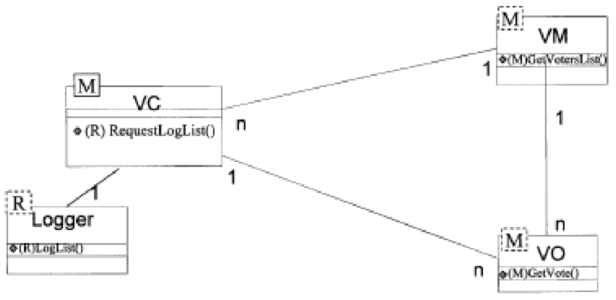


Fig. 1. A mobile class diagram.

B. OWL

OWL (Ontology Web Language) is a language for representing ontologies by defining the concepts of a domain and the relations between them what will allow automatic reasoning about the domain knowledge using their formal semantics. OWL1 offers three sublanguages with increasing expression oriented for specific communities of developers and users: OWL Lite, OWL DL, and OWL Full [10] whereas OWL2 defines three new profiles: OWL2 EL, OWL2 QL, and OWL2 RL [13].

C. Graph Grammars

Graph transformation was largely used for the expression of model transformation [4]; particularly, transformations of visual models can be naturally formulated by graph transformation, since the graphs are well adapted to describe the fundamental structures of models [17]. The set of graph transformation rules constitutes what is called the model of graph grammar. A graph grammar is a generalization, for graphs, of Chomsky grammars. Each rule of a graph grammar is composed of a left hand side (LHS) pattern and of a right-hand sided (RHS) pattern. Therefore, the graph transformation is the process to choose a rule among the graph grammar rules, apply this rule on a graph pattern that matches the LHS pattern to produce the RHS pattern, and reiterate the process until no rule can be applied [4]. We have adopted the $AToM^3$ tool [1] which is a visual tool for model transformation to implements our approach. In the next sections, we will discuss how we use $AToM^3$ to meta-model mobile class diagrams and how to generate OWL models by applying a graph grammar.

ICIT 2015 The 7th International Conference on Information Technology
doi:10.15849/icit.2015.0020 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

IV. OUR APPROACH

A. *Overview*

Mobile agent-based software systems are very difficult to design and to implement, although of this, we are urgently and still need this type of software systems to resolve some huge problems in different domains that the object oriented paradigm could not deal with them. We propose in this paper integrated approach M-UML class diagram/OWL ontology for modeling and analysis of mobile agent-based software systems by direct transformation of the mobile class diagram to OWL ontology. A mobile class diagram describes statically and in a very rich way the entities evolved in a mobile agent-based software system and all relationships among them. This ontology generated contains the knowledge extracted from the M-UML diagram and will be used for reuse purpose, knowledge sharing, conversation, integration and reasoning on mobile agent-based software systems.

The architecture of the proposed approach (Figure 2) is based on meta-modeling and graph grammars. For the realization of this application, we have to propose a meta-model of mobile class diagrams that allows us to edit visually mobile class diagrams on AToM^3 canvas. In addition, we have to develop a graph grammar made up of several rules which allows transforming progressively all what is modeled on the canvas towards an OWL ontology in RDF/XML format stored in a disk file. The graph grammar is based on transformation rules; each rule deals with some constructs in the left hand side (LHS) to transform them to others constructs in the right hand side (RHS). For ontology, the choice among OWL profiles is made on OWL DL because it places certain constraints on the use of the structures of OWL such as separation between classes, data types, data type properties, object properties, annotation properties, ontologies properties, individuals, data values, and integrated vocabulary [11][12].

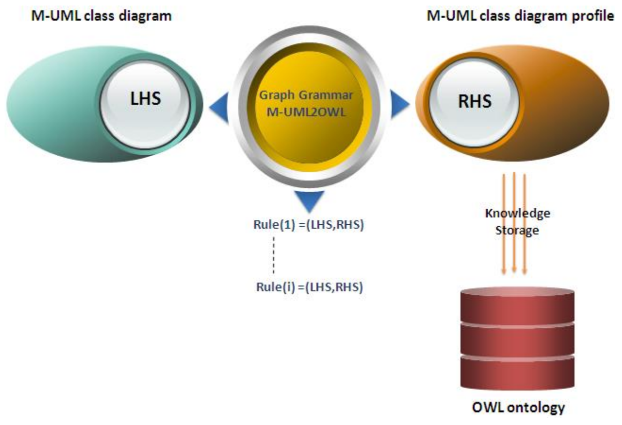


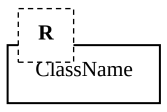
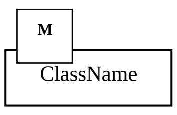
Fig. 2. Architecture of the proposed approach.

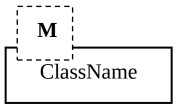
B. *Transformation rules*

We propose a set of rules to transform classes, enumerations, associations, roles, dependencies, association classes, and all the elements of a mobile class diagram that are important to store in the OWL ontology. For lack of space, we have presented class (and their extensions) transformation rules in table 1.

Concerning the transformation of data types, all data types used in M-UML are transformed into XML schema (XSD) data types because OWL uses the majority of the datatypes integrated into XML schema. The calls of these data types are done through data type URI address reference http://www.w3.org/2001/XMLSchema[11]. The instances of the primitive types used in M-UML itself include: Boolean, Integer, String, Unlimited Natural[7].

TABLE I. TRANSFORMATION RULES.

M-UML to OWL	
Remote Class	
 R ClassName	<owl:DatatypePropertyrdf:ID="**isRemote**"> <rdfs:domainrdf:resource="#**Remote-ClassName**"/> <rdfs:rangerdf:resource="http://www.w3. org/2001/XMLSchema#boolean"/> </owl:DatatypeProperty> <owl:Classrdf:ID="**Remote-ClassName**"> <rdfs:subClassOf> <owl:Restriction> <owl:onPropertyrdf:resource= "#**isRemote**"/> <owl:mincardinalityrdf:datatype="http://www.w3. org/2001/XMLSchema#nonNegativeInteger">**1** </owl:mincardinality> </owl:Restriction> </rdfs:subClassOf> </owl:Class>…
A remote class is represented by a data type property and a class OWL. A remote class can contain behaviors affected by multiple remote mobile agent, we use the restriction *mincardinality* to indicate this.	
Mobile Class	
 M ClassName	<owl:DatatypePropertyrdf:ID="**isMobile**"> <rdfs:domainrdf:resource="#**Mobile-ClassName**"/> <rdfs:rangerdf:resource="http://www.w3. org/2001/XMLSchema#boolean"/> </owl:DatatypeProperty> <owl:Classrdf:ID="**Mobile-ClassName**"> <rdfs:subClassOf> <owl:Restriction> <owl:onPropertyrdf:resource= "#**isMobile**"/> <owl:cardinalityrdf:datatype="http://www.w3. org/2001/XMLSchema#nonNegativeInteger">**1** </owl:cardinality> </owl:Restriction> </rdfs:subClassOf> </owl:Class>…
A mobile class is represented by a data type property and a class OWL. A mobile class can contain exactly one property concerning the mobility of their objects.	
Affected Class	

	<owl:DatatypePropertyrdf:ID="**isAffected** "> <rdfs:domainrdf:resource="#**Affected-ClassName**"/> <rdfs:rangerdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/> </owl:DatatypeProperty> <owl:Classrdf:ID="**Mobile-ClassName**"> <rdfs:subClassOf> <owl:Restriction> <owl:onPropertyrdf:resource= "**#isAffected**"/> <owl:mincardinalityrdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">**1** </owl:mincardinality> </owl:Restriction> </rdfs:subClassOf> </owl:Class>…
An affected class is represented by a data type property and a class OWL. An affected class can contain behavior (methods) that is affected (communicating with) by mobile objects, we use the restriction *mincardinality* to indicate this.	

C. *Meta-model of UML Class diagram*

To build M-UML class diagrams in AToM^3, we have to define a meta-model for them. Our meta-model is composed of two classes and four associations developed by the meta-formalism (CD_classDiagramsV3), and the constraints are expressed in Python [8] code (Figure 3).

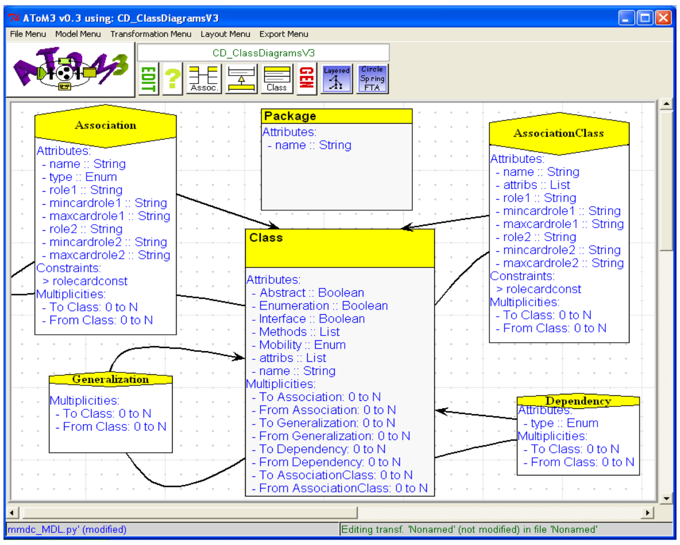


Fig. 3. Mobile class diagram meta-model

After building our meta-model, it remains only its generation. The generated meta-model comprises the set of classes modeled in the form of buttons which are ready to be employed for a possible modeling of a class diagram. Figure 4 shows an example of a mobile class diagram of a mobile voting system [19] modeled in our mobile class diagram environment.

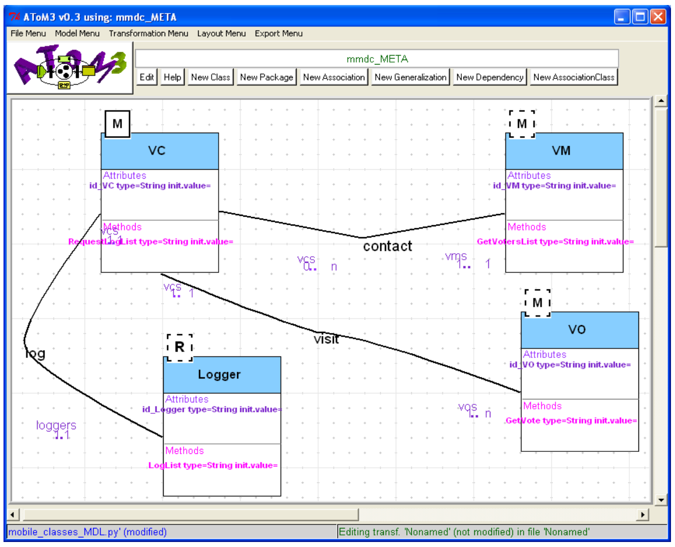

AToM3 v0.3 using: mmdc_META
File Menu Model Menu Transformation Menu Layout Menu Export Menu
mmdc_META
Edit Help New Class New Package New Association New Generalization New Dependency New AssociationClass
M
VC
Attributes
id_VC type=String init.value=
Methods
RequestLogList type=String init.value=
M
VM
Attributes
id_VM type=String init.value=
Methods
GetVotersList type=String init.value=
contact
vcs
0..n
vms
1..1
visit
M
VO
Attributes
id_VO type=String init.value=
Methods
GetVote type=String init.value=
R
Logger
Attributes
id_Logger type=String init.value=
Methods
LogList type=String init.value=
log
loggers
1..1
vos
0..n
mobile_classes_MDL.py' (modified)
Editing transf. 'Nonamed' (not modified) in file 'Nonamed'

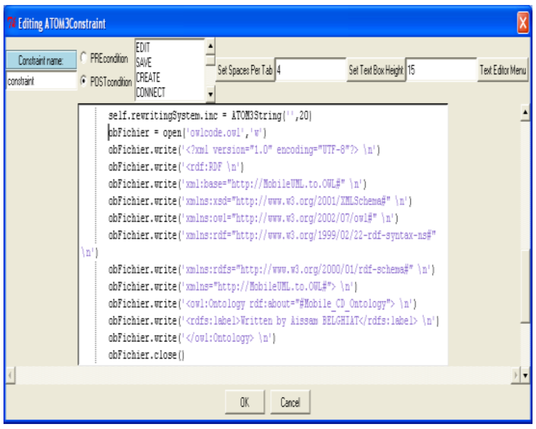
Fig. 4. Generated tool for mobile class diagram.

D. *The Proposed Graph grammar*

To perform the transformation between class diagrams and OWL ontologies, we have proposed a graph grammar composed of an initial action, ten rules, and a final action. For lack of space, and because we used python code to specify the transformation in the condition and action of each rule, we have not presented all the rules.

Initial Action: Ontology header

Role: In the initial action of the graph grammar, we create a file with sequential access in order to store generated OWL code. To do that, we used Python. We begin by writing the ontology header which is fixed for all generated ontologies (Figure 5).



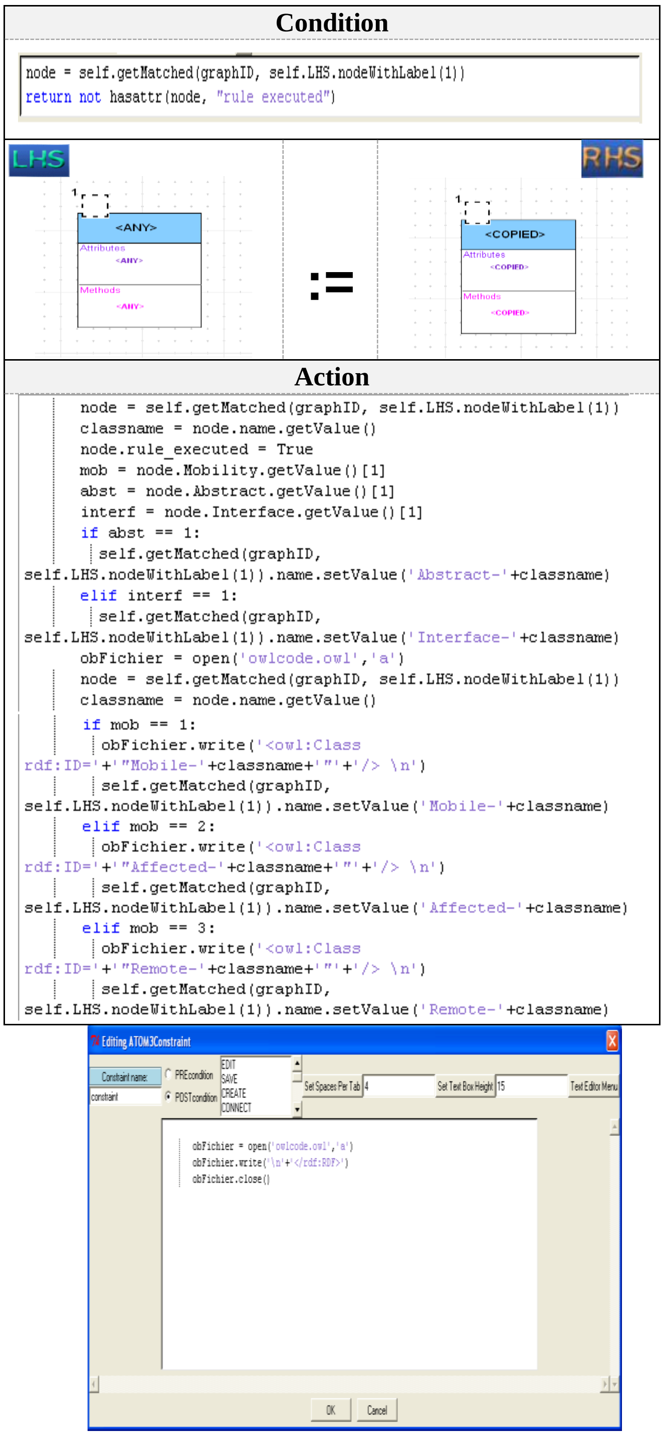
Constraint name: constraint
POSTcondition
Set Spaces Per Tab: 4
Set Text Box Height: 15

```
self.rewritingSystem.inc = ATOM3String('',20)
obFichier = open('owlcode.owl','w')
obFichier.write('<?xml version="1.0" encoding="UTF-8"?> \n')
obFichier.write('<rdf:RDF \n')
obFichier.write('xml:base="http://MobileUML.to.OWL#" \n')
obFichier.write('xmlns:xsd="http://www.w3.org/2001/XMLSchema#" \n')
obFichier.write('xmlns:owl="http://www.w3.org/2002/07/owl#" \n')
obFichier.write('xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
\n')
obFichier.write('xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" \n')
obFichier.write('xmlns="http://MobileUML.to.OWL#"> \n')
obFichier.write('<owl:Ontology rdf:about="#Mobile_CD_Ontology"> \n')
obFichier.write('<rdfs:label>Written by Aissam BELGHIAT</rdfs:label> \n')
obFichier.write('</owl:Ontology> \n')
obFichier.close()
```

Fig. 5. Ontology header definition.

Final Action: **the end of ontology**

Role: In the final action of the graph grammar, we end our ontology. So, we will have to open our file and to add '</rdf:RDF>' (Figure 6).



Condition

```
node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
return not hasattr(node, "rule executed")
```

Action

```
    node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
    classname = node.name.getValue()
    node.rule_executed = True
    mob = node.Mobility.getValue()[1]
    abst = node.Abstract.getValue()[1]
    interf = node.Interface.getValue()[1]
    if abst == 1:
      self.getMatched(graphID,
self.LHS.nodeWithLabel(1)).name.setValue('Abstract-'+classname)
    elif interf == 1:
      self.getMatched(graphID,
self.LHS.nodeWithLabel(1)).name.setValue('Interface-'+classname)
    obFichier = open('owlcode.owl','a')
    node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
    classname = node.name.getValue()
    if mob == 1:
      obFichier.write('<owl:Class
rdf:ID='+'"Mobile-'+classname+'"'+'/> \n')
      self.getMatched(graphID,
self.LHS.nodeWithLabel(1)).name.setValue('Mobile-'+classname)
    elif mob == 2:
      obFichier.write('<owl:Class
rdf:ID='+'"Affected-'+classname+'"'+'/> \n')
      self.getMatched(graphID,
self.LHS.nodeWithLabel(1)).name.setValue('Affected-'+classname)
    elif mob == 3:
      obFichier.write('<owl:Class
rdf:ID='+'"Remote-'+classname+'"'+'/> \n')
      self.getMatched(graphID,
self.LHS.nodeWithLabel(1)).name.setValue('Remote-'+classname)
```

Constraint name: constraint

Set Spaces Per Tab: 4

Set Text Box Height: 15

```
obFichier = open('owlcode.owl','a')
obFichier.write('\n'+'</rdf:RDF>')
obFichier.close()
```

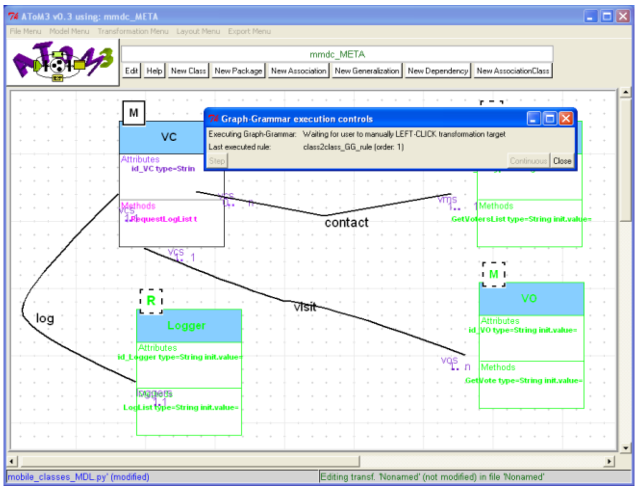
Fig. 6. End of ontology.

Rule 1: Class transformation
Name: class2class
Priority: 1
Role: This rule transforms an M-UML class (all type of classes) towards an OWL class (cf. Table 2). In the condition of the rule we test if the class is already transformed. If not yet, we reopen the OWL file to add the adequate OWL code of this class in the action of the rule.

TABLE II. TRANSFORMATION OF DIFFERENT TYPES OF M-UML CLASSES.

V. EXAMPLE

Let us apply our approach on the mobile class diagram illustrated in figure 4which is borrowed from [19]. It models a mobile voting system, where a mobile agent VC (vote collector) gets a list of voters from a stationary agent VM (vote manager) and visits the VO's (voters) stations those already have the list of candidates to collect votes and return them to the VM that mandated the VC in action. It should be noted that this example does not claim to be exhaustive but it gathers most important elements of a mobile class diagram such as: mobile class, affected class, remote class, association, attributes and different types of methods. By applying our graph grammar on this example, we have first obtained the intermediate graphs shown in figure7.



Executing Graph-Grammar: Waiting for user to manually LEFT-CLICK transformation target
Last executed rule: class2class_GG_rule (order: 1)

mobile_classes_MDL.py' (modified)
Editing transf. 'Nonamed' (not modified) in file 'Nonamed'

Fig. 7. Intermediate graph

Then we have obtained the graph of figure8 after the termination of execution of the graph grammar.

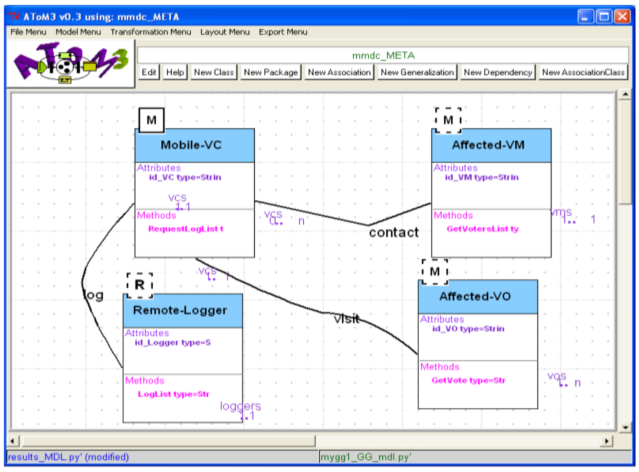
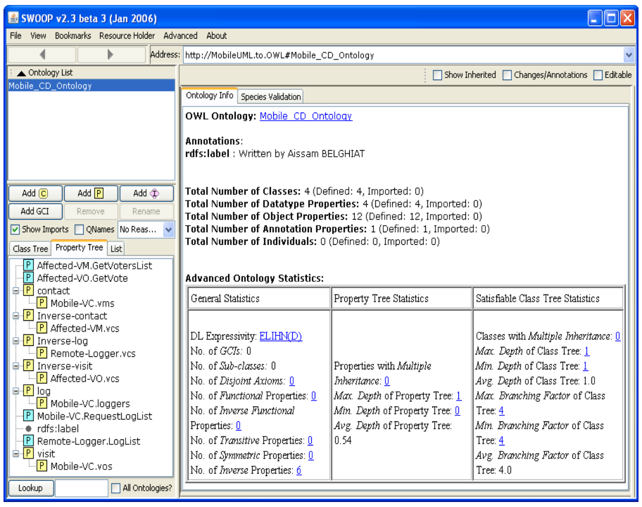


Fig. 8. Class diagram after execution

In parallel, there is an automatic generation of the file containing OWL code stored on hard disc validated and visualized using SWOOP [24] (Figure 9, 10, 11):



SWOOP v2.3 beta 3 (Jan 2006)

File View Bookmarks Resource Holder Advanced About

Address: http://MobileUML.to.OWL#Mobile_CD_Ontology

Ontology List: Mobile_CD_Ontology

Show Inherited Changes/Annotations Editable

Ontology Info | Species Validation

OWL Ontology: Mobile_CD_Ontology

Annotations:
rdfs:label : Written by Aissam BELGHIAT

Total Number of Classes: 4 (Defined: 4, Imported: 0)
Total Number of Datatype Properties: 4 (Defined: 4, Imported: 0)
Total Number of Object Properties: 12 (Defined: 12, Imported: 0)
Total Number of Annotation Properties: 1 (Defined: 1, Imported: 0)
Total Number of Individuals: 0 (Defined: 0, Imported: 0)

Advanced Ontology Statistics:

General Statistics	Property Tree Statistics	Satisfiable Class Tree Statistics
DL Expressivity: ELIHN(D) No. of *GCIs*: 0 No. of *Sub-classes*: 0 No. of *Disjoint Axioms*: 0 No. of *Functional* Properties: 0 No. of *Inverse Functional* Properties: 0 No. of *Transitive* Properties: 0 No. of *Symmetric* Properties: 0 No. of *Inverse* Properties: 6	Properties with *Multiple Inheritance*: 0 *Max. Depth* of Property Tree: 1 *Min. Depth* of Property Tree: 0 *Avg. Depth* of Property Tree: 0.54	Classes with *Multiple Inheritance*: 0 *Max. Depth* of Class Tree: 1 *Min. Depth* of Class Tree: 1 *Avg. Depth* of Class Tree: 1.0 *Max. Branching Factor* of Class Tree: 4 *Min. Branching Factor* of Class Tree: 4 *Avg. Branching Factor* of Class Tree: 4.0

Add C | Add P | Add I | Add GCI | Remove | Rename

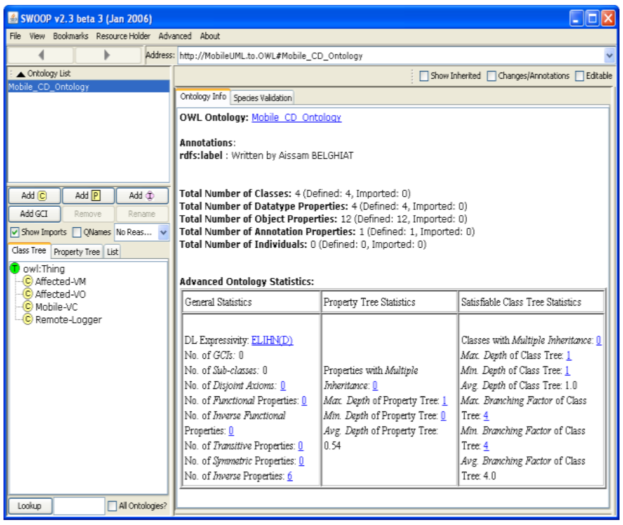
Show Imports QNames No Reas...

Class Tree | Property Tree | List

```
P Affected-VM.GetVotersList
P Affected-VO.GetVote
P contact
    P Mobile-VC.vms
P Inverse-contact
    P Affected-VM.vcs
P Inverse-log
    P Remote-Logger.vcs
P Inverse-visit
    P Affected-VO.vcs
P log
    P Mobile-VC.loggers
P Mobile-VC.RequestLogList
● rdfs:label
P Remote-Logger.LogList
P visit
    P Mobile-VC.vos
```

Lookup All Ontologies?

Fig. 9. The OWL ontology classes.



SWOOP v2.3 beta 3 (Jan 2006)

File View Bookmarks Resource Holder Advanced About

Address: http://MobileUML.to.OWL#Mobile_CD_Ontology

Ontology List: Mobile_CD_Ontology

Show Inherited Changes/Annotations Editable

Ontology Info | Species Validation

OWL Ontology: Mobile_CD_Ontology

Annotations:
rdfs:label : Written by Aissam BELGHIAT

Total Number of Classes: 4 (Defined: 4, Imported: 0)
Total Number of Datatype Properties: 4 (Defined: 4, Imported: 0)
Total Number of Object Properties: 12 (Defined: 12, Imported: 0)
Total Number of Annotation Properties: 1 (Defined: 1, Imported: 0)
Total Number of Individuals: 0 (Defined: 0, Imported: 0)

Advanced Ontology Statistics:

General Statistics	Property Tree Statistics	Satisfiable Class Tree Statistics
DL Expressivity: ELIHN(D) No. of *GCIs*: 0 No. of *Sub-classes*: 0 No. of *Disjoint Axioms*: 0 No. of *Functional* Properties: 0 No. of *Inverse Functional* Properties: 0 No. of *Transitive* Properties: 0 No. of *Symmetric* Properties: 0 No. of *Inverse* Properties: 6	Properties with *Multiple Inheritance*: 0 *Max. Depth* of Property Tree: 1 *Min. Depth* of Property Tree: 0 *Avg. Depth* of Property Tree: 0.54	Classes with *Multiple Inheritance*: 0 *Max. Depth* of Class Tree: 1 *Min. Depth* of Class Tree: 1 *Avg. Depth* of Class Tree: 1.0 *Max. Branching Factor* of Class Tree: 4 *Min. Branching Factor* of Class Tree: 4 *Avg. Branching Factor* of Class Tree: 4.0

Add C | Add P | Add I | Add GCI | Remove | Rename

Show Imports QNames No Reas...

Class Tree | Property Tree | List

```
owl:Thing
    C Affected-VM
    C Affected-VO
    C Mobile-VC
    C Remote-Logger
```

Lookup All Ontologies?

Fig. 10. The OWL ontology properties.

VI. CONCLUSION

The objective of this work is to develop an integrated environment for modeling and analysis of mobile based software systems by the transformation of M-UML class diagrams to OWL ontologies.The approach has been implemented using the AToM^{3}tool. For the realization of this application we have developed a meta-model for M-UML class diagrams, and a graph grammar named "M-UML2OWL" composed of several rules which enables us to transform a mobile class diagram to an OWL ontology stored in a hard disk file. The generated ontology will be used for reuse purpose, knowledge sharing, conversation, integration and reasoning on mobile agent-based software systems.

In future work, we plan to generalize the extraction of OWL ontologies from others M-UML diagrams since they represent different aspects of the systems. We plan also to realize this transformation using other graph transformation tools such as Triple Graph Grammars [27] which provide bidirectional transformations.

REFERENCES

[1] AToM^3. Home page: http://atom3.cs.mcgill.ca.2002.

[2] J. D. Lara, H. Vangheluwe, "Computer aided multi-paradigm modeling to process petri-nets and statecharts," International Conference on Graph Transformations (ICGT), Lecture Notes in Computer Science, vol. 2505, pp. 239-253, Springer-Verlag, Barcelona, Spain, 2002.

[3] J. D. Lara, H. Vangheluwe, "Meta-modeling and graph grammars for multi-paradigm modeling in AToM^3," Software and Systems Modeling, Vol. 3, pp. 194-209, Springer-Verlag, Special Section on Graph Transformations and Visual Modeling Techniques, 2004.

[4] G. Karsai, A. Agrawal, "Graph Transformations in OMG's Model-Driven Architecture," Lecture Notes in Computer Science, Vol 3062, 243-259, Springer Berlin /Heidelberg, juillet 2004.

[5] Sebastian Leinhos, http://diplom.ooyoo.de, 2006.

[6] OMG, "Ontology Definition Metamodel", V1.0, http://www. omg. org/spec/ODM/1.0, May 2009.

[7] OMG, "OMG Unified Modeling Language, Infrastructure, v2.3",http://www.omg.org/spec/UML/2.1.2 /Infrastructure/ PDF, May 2010.

[8] Python. Home page: http://www.python.org.

[9] SIDo Group, “ATL Use Case - ODM Implementation (Bridging UML and OWL),”http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/ , 2007.

[10] D. L. McGuinness, F. V. Harmelen, “OWL Web Ontology Language-Overview,” http://www.w3.org/TR/ 2004/REC-owl-features-20040210/. W3C Recommendation 10 February 2004.

[11] M. K. Smith, C. Welty, D. L. McGuinness, “OWL Web Ontology Language–Guide”, http://www.w3.org/TR/2004/REC-owl-guide-2004 0210. W3C Recommendation 10 February 2004.

[12] M. Dean, G. Schreiber, S. Bechhofer, F.V. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein, “OWL Web Ontology Language-Reference,”, http://www.w3.org/TR/2004/ REC-owl-ref-20040210. W3C Recommendation 10 February 2004.

[13] W3C OWL Working Group, “OWL 2 Web Ontology Language Document Overview,” http://www.w3.org/TR/2009/REC-owl2-overview-20091027. W3C Recommendation 27 October 2009.

[14] K. Baclawski, M. K. Kokar, P. A. Kogut, L. Hart, J. Smith, W. S. Holmes, J. Letkowski,M. L. Aronsonl, “Extending UML to Support Ontology Engineering for the Semantic Web,” (pp. 342-360). Springer Berlin Heidelberg. 2001.

[15] D. Gašević, D. Djurić, V. Devedžić, V. Damjanović, “Converting UML to OWL Ontologies,” In : *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, p. 488-489. 2004.

[16] K. Kiko, C. Atkinson, “A Detailed Comparison of UML and OWL,” Technical Report, Reihe Informatik, TR-2008-004,2008.

[17] R.Bardohl,H. Ehrig, J. De Lara, G. Taentzer,"Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation,"Lecture Notes in Computer Science 2984, pp.: 214-228. 2004.

[18] H. Mouratidis, J. Odell, G. Manson, “Extending the Unified Modeling Language to Model Mobile Agents,”Proceedings Agent Oriented Methodologies Workshop, Annual ACM Conference on Object Oriented Programming, Systems, Languages (OOPSLA), Seattle – USA, 2002.

[19] K. Saleh, C. El-Morr, “M-UML: an extension to UML for the modeling of mobile agent-based software systems,” Journal of Information and Software Technology, ELSEVIER, Vol 46, 2004, pp. 219–227.

[20] A. Belghiat, M. Bourahla, “An Approach based AToM3 for the Generation of OWL Ontologies from UML Diagrams,”International Journal of Computer Applications (0975 – 8887) Volume 41– No.3, March 2012.

[21] E. Kerkouche and A. Chaoui, “A Formal Framework and a Tool for the Specification and Analysis of G-Nets Models Based on Graph Transformation,”International Conference on Distributed Computing and Networking -CDCN’09-, LNCS 5408, pp. 206–211, Springer-Verlag Berlin Heidelberg, India, 3-6 January, 2009.

[22] F. Guerrouf, A. Chaoui, A. Aldahoud, “A graph transformation approach of mobile activity diagram to nested Petri nets,” IJCAET 5(1): 44-57 (2013).

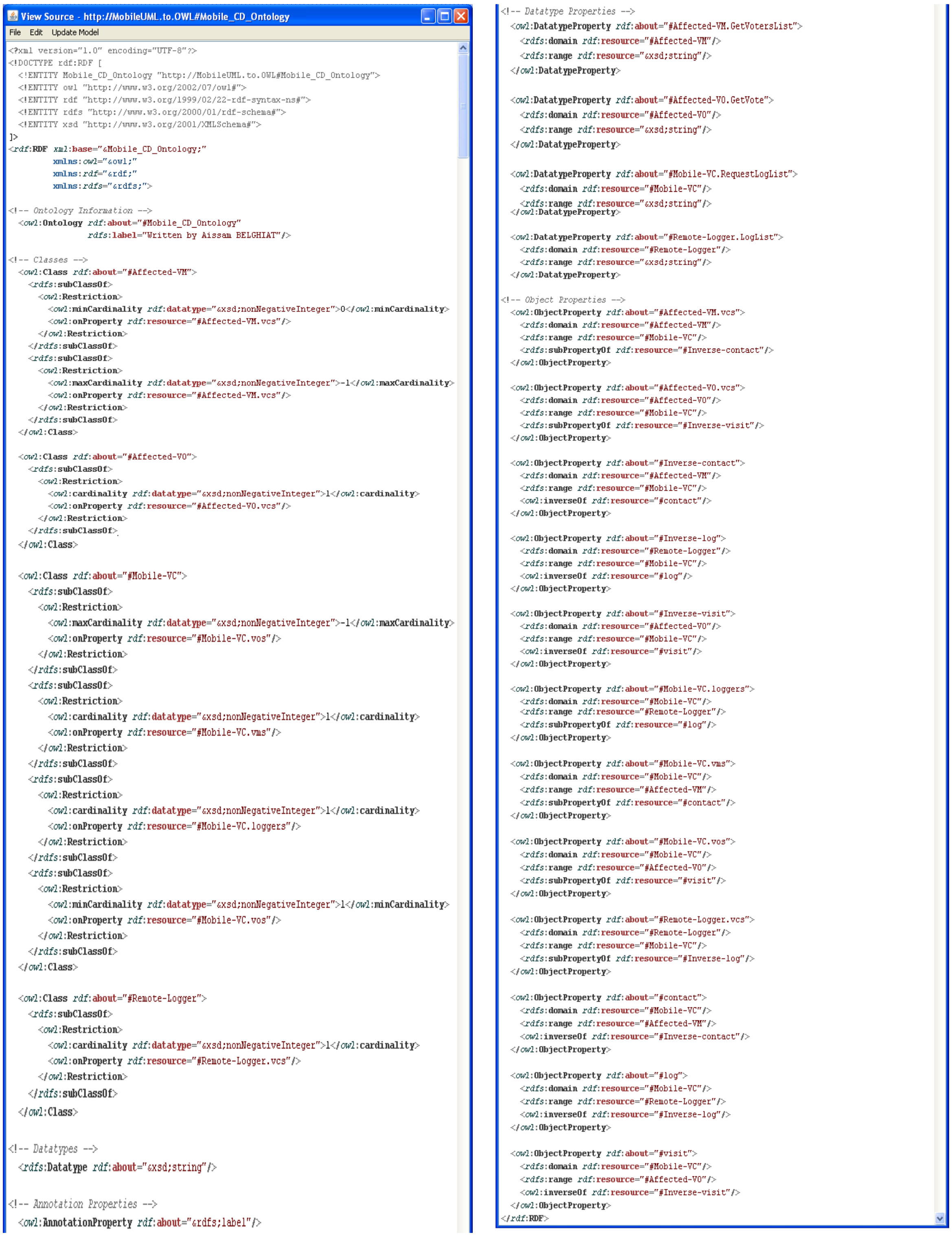
[23] M. R. Bahri, A. Hettab, A. Chaoui, and E. Kerkouche, “Transforming Mobile UML Statecharts Models to Nested Nets Models using Graph Grammars: An Approach for Modeling and Analysis of Mobile Agent-Based Software Systems,” in SEEFM ‘09, IEEE Computer Society Washington, pp.33-39, 2009.

[24] SWOOP. Home page: http://www.mindswap.org /2004/SWOOP.

[25] E. Kerkouche, A. Chaoui, E. Bourennane, and O. Labbani, “Modeling and verification of dynamic behaviour in UML models, a graph transformation based approach,“ proceedings of SEDE’2009, Las Vegas, Nevada, USA, 22-24 June 2009.

[26] A. Belghiat, A. Chaoui, M. Maouche, M. Beldjehem, “Formalization of Mobile UML Statechart Diagrams using the π-calculus: An Approach for Modeling and Analysis,”In G. Dregvaite and R. Damasevicius (Eds.): ICIST 2014, CCIS 465, pp. 236–247. Springer.

A. Schurr, “Specification of Graph Translators with Triple Graph Grammars,” In G. Tinhofer, editor, WG’94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science, volume 903 of Lecture Notes in Computer Science (LNCS), pages 151–163, Heidelberg, 1994. Springer Verlag.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY Mobile_CD_Ontology "http://MobileUML.to.OWL#Mobile_CD_Ontology">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&Mobile_CD_Ontology;"
         xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
  <owl:Ontology rdf:about="#Mobile_CD_Ontology"
                rdfs:label="Written by Aissam BELGHIAT"/>

<!-- Classes -->
  <owl:Class rdf:about="#Affected-VM">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">0</owl:minCardinality>
        <owl:onProperty rdf:resource="#Affected-VM.vcs"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">-1</owl:maxCardinality>
        <owl:onProperty rdf:resource="#Affected-VM.vcs"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#Affected-VO">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#Affected-VO.vcs"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#Mobile-VC">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">-1</owl:maxCardinality>
        <owl:onProperty rdf:resource="#Mobile-VC.vos"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#Mobile-VC.vms"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#Mobile-VC.loggers"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:onProperty rdf:resource="#Mobile-VC.vos"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#Remote-Logger">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        <owl:onProperty rdf:resource="#Remote-Logger.vcs"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

<!-- Datatypes -->
  <rdfs:Datatype rdf:about="&xsd;string"/>

<!-- Annotation Properties -->
  <owl:AnnotationProperty rdf:about="&rdfs;label"/>
```

```
<!-- Datatype Properties -->
  <owl:DatatypeProperty rdf:about="#Affected-VM.GetVotersList">
    <rdfs:domain rdf:resource="#Affected-VM"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#Affected-VO.GetVote">
    <rdfs:domain rdf:resource="#Affected-VO"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#Mobile-VC.RequestLogList">
    <rdfs:domain rdf:resource="#Mobile-VC"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#Remote-Logger.LogList">
    <rdfs:domain rdf:resource="#Remote-Logger"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

<!-- Object Properties -->
  <owl:ObjectProperty rdf:about="#Affected-VM.vcs">
    <rdfs:domain rdf:resource="#Affected-VM"/>
    <rdfs:range rdf:resource="#Mobile-VC"/>
    <rdfs:subPropertyOf rdf:resource="#Inverse-contact"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Affected-VO.vcs">
    <rdfs:domain rdf:resource="#Affected-VO"/>
    <rdfs:range rdf:resource="#Mobile-VC"/>
    <rdfs:subPropertyOf rdf:resource="#Inverse-visit"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Inverse-contact">
    <rdfs:domain rdf:resource="#Affected-VM"/>
    <rdfs:range rdf:resource="#Mobile-VC"/>
    <owl:inverseOf rdf:resource="#contact"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Inverse-log">
    <rdfs:domain rdf:resource="#Remote-Logger"/>
    <rdfs:range rdf:resource="#Mobile-VC"/>
    <owl:inverseOf rdf:resource="#log"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Inverse-visit">
    <rdfs:domain rdf:resource="#Affected-VO"/>
    <rdfs:range rdf:resource="#Mobile-VC"/>
    <owl:inverseOf rdf:resource="#visit"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Mobile-VC.loggers">
    <rdfs:domain rdf:resource="#Mobile-VC"/>
    <rdfs:range rdf:resource="#Remote-Logger"/>
    <rdfs:subPropertyOf rdf:resource="#log"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Mobile-VC.vms">
    <rdfs:domain rdf:resource="#Mobile-VC"/>
    <rdfs:range rdf:resource="#Affected-VM"/>
    <rdfs:subPropertyOf rdf:resource="#contact"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Mobile-VC.vos">
    <rdfs:domain rdf:resource="#Mobile-VC"/>
    <rdfs:range rdf:resource="#Affected-VO"/>
    <rdfs:subPropertyOf rdf:resource="#visit"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#Remote-Logger.vcs">
    <rdfs:domain rdf:resource="#Remote-Logger"/>
    <rdfs:range rdf:resource="#Mobile-VC"/>
    <rdfs:subPropertyOf rdf:resource="#Inverse-log"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#contact">
    <rdfs:domain rdf:resource="#Mobile-VC"/>
    <rdfs:range rdf:resource="#Affected-VM"/>
    <owl:inverseOf rdf:resource="#Inverse-contact"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#log">
    <rdfs:domain rdf:resource="#Mobile-VC"/>
    <rdfs:range rdf:resource="#Remote-Logger"/>
    <owl:inverseOf rdf:resource="#Inverse-log"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="#visit">
    <rdfs:domain rdf:resource="#Mobile-VC"/>
    <rdfs:range rdf:resource="#Affected-VO"/>
    <owl:inverseOf rdf:resource="#Inverse-visit"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Fig. 11. The generated OWL ontology.