Automatic Generation of Object-Oriented Code from the ReLEL Requirements Model

Andrianjaka Miary Rapatsalahy,
Hajarisena Razafimahatratra,
Thomas Mahatody
Laboratory for Mathematical and Computer Applied to the Development Systems (LIMAD)
University of Fianarantsoa
Fianarantsoa, Madagascar
andrianjaka92@ yahoo.fr,
hajarisena@yahoo.fr,
hasinathomas@hotmail.com

Mihaela Ilie,
Sorin Ilie
Dept. of Computers and Information Technology
University of Craiova
Craiova, România
ela.pirvu@gmail.com,
sorin.ilie@software.ucv.ro

Raft Nicolas Razafindrakoto
Laboratory of Multidisciplinary Applied Research (LRAM)
University of Fianarantsoa
Fianarantsoa, Madagascar
rnraft@gmail.com

***Abstract*—The final executable code should no longer be considered as a central element in a software development process but rather a naturally important component that results from a model transformation. The objective of the MDA (Model Driven Architecture) approach is to lift the lock of software development automation from the CIM (Computation Independent Model) requirements until the code of an application is obtained. Therefore, we have proposed in the framework of MDA an approach that consists of automatically generating object-oriented code from the CIM model represented by ReLEL (Restructuring extended lexical elaborate language). ReLEL is a natural language-oriented model that represents both the client requirements and the conceptual level of a system. However, the MDA framework does not recommend the type of UML model that corresponds to each business activity. Consequently, automating the software development process from the CIM model specified by ReLEL becomes a complex task. Our strategy in this paper includes the instantiation of the ReLEL model in the Praxeme methodology, which models each of the company's concerns, grouped in a homogeneous whole, using the UML (Unified Modeling Language) and which considers the articulation of these aspects by adopting the MDA principle. To do this, we propose to automate the articulation that covers the intentional, semantic, logical, and software aspects of Praxeme. To validate our approach, we measure the coupling and cohesion of the UML class diagram obtained from the Java code generated from this article using the slicing technique. The results show that the coupling is weak, and the cohesion is strong. It can be deduced that the method proposed in this paper can produce a more reliable and efficient system.**

Keywords*—MDA, ReLEL, eLEL, Praxeme methodology, UML language, Atlas Transformation Language (ATL), Acceleo, slicing technique, coupling, cohesion

I. Introduction

The evolution of requirements forces the enterprise information system to remain flexible in these changes. The enterprise therefore needs a methodological framework to control its architecture and govern its transformation [12]. In fact, Praxeme is a way to provide enterprises with tools capable of absorbing change by driving the design of the information system towards a service-oriented approach [6]. As a result, it focuses on the process and technology of deriving the logical model of the future information system from a set of higher-level models derived from the business line itself [13]. The logical model specifies the logical aspect of Praxeme. Thus, it plays a very important role, i.e. an intermediary between various aspects at the level of the business architecture and the technical architecture [22]. This approach allows modelers to build information systems independent of the business and any technical implementation, but also to facilitate structural decisions of the software system [5]. However, automatic localization of logical services in logical modeling is a perilous task [19]. This is due to the dispersion of information at the level of the intentional aspect of Praxeme, which has no dedicated UML diagram to represent the intentions of the enterprise [12, 19, 22]. So, (Razafindramintsa et al., 2016) [17] recommended an approach to represent the intentional aspect of Praxeme with the natural language-oriented requirements model noted eLEL (elaborate Lexicon Extended Language) [14]. As the Praxeme methodology makes use of the MDA approach for automating the UML models representing each aspect of the business, therefore, the automatic localization of the logical service (logical aspect) from eLEL (intentional aspect) becomes obvious [22]. However, in [22], (Rapatsalahy et al., 2020) argue that business logic services automatically localized from eLEL cannot be used in the code (software aspect) framework of object-oriented applications. Since eLEL describes the information that constitutes the logical service of the logical aspect of Praxeme, the problem lies in the intentional aspect of eLEL. Indeed, the attribute types and system method representations of eLEL do not conform to object-oriented conventions [21]. In this paper, our approach consists in using the ReLEL requirements model, which is an extension of eLEL, to specify the intentional aspect of Praxeme and automate the software development process in the MDA framework. To do this, we have proposed transformation rules by applying the model-to-model (M2M) and model-to-text (M2T) transformation techniques. The derivation of the intentional aspect (CIM) into the semantic aspect (Plateform Independent Model or PIM) and then of the semantic aspect into the logical aspect (PIM) is a transformation classified in M2M. Then the translation of the logical aspect into object-oriented code is the classification transformation into M2T. The implementation of the derivation rules in this paper uses the ATL language for M2M and the Acceleo template engine for M2T. Given the choice of technology, the results obtained in the software aspect are packages, classes as well as Java class methods.

II. Related Work

The automated transformation from design to code brings many benefits, such as increased productivity, better portability, reduced delivery time and the elimination of manual programming errors [10]. Praxeme brings a major advantage to the software development process by adopting the MDA approach in its methodology. This means that each aspect of the business advocated by Praxeme is represented by an appropriate UML model, which can then be automatically derived in another aspect. The objective in this paper is to automatically generate a code skeleton that represents the Praxeme software aspect from the intentional aspect specified by the ReLEL requirements model. This approach is solved by the M2M transformation technique as well as the M2T transformation. Consequently, in this section, we will summarize the research literature based on the use of the requirements model in the Praxeme methodology.

In [12], (Biard et al, 2013) introduces an approach that consists in using the Praxeme methodology for business transformation. The researchers start from a pragmatic aspect so that the business objects and their data that make up the semantic aspect of Praxeme can be extracted from the messages exchanged between the different participants. Then, to establish the business and organizational rules in the semantic and pragmatic aspect, they recommend the use of SBVR (Semantics Business Vocabulary Rules) [8] to represent the intentional aspect of Praxeme.

In [17], (Razafindramintsa et al., 2016) present an approach which consists of representing the intentional aspect of Praxeme with the eLEL lexicon, which is a terminological database facilitating the derivation of other aspects of the methodology, such as semantics. To do so, they proposed rules to derive the eLEL natural language-oriented requirements model into a preliminary version of the UML class diagram and the transition state machine which are the components of the semantic aspect of the Praxeme methodology [12]. Since Praxeme absorbs the MDA approach for the articulation of its aspect, the implementation of the derivation rules noted M2M proposed in this paper uses the ATL language.

In [18] (Razafindramintsa et al., 2017) present an approach that deals with the automatic derivation of the pragmatic aspect of Praxeme from the intentional aspect specified by the natural language-oriented requirements model eLEL. Their strategy consists in proposing rules to derive eLEL from the UML diagrams, namely the use case and the activity. Indeed, these two diagrams represent the pragmatic aspect of the Praxeme methodology [18]. As an initial result, they obtained a use case and activity diagram in an XMI (XML Metadata Interchange) format [4]. Then, in [18], (Razafindramintsa et al, 2017) transformed the previous results into a concrete UML diagram. The final result of the derivation process are UML use case and activity diagrams in XMI in a concrete format.

In [19], (Razafindramintsa et al., 2017) present an approach that deals with the automation of the logical service localization process at the time of logical modeling. This approach instantiates the eLEL requirements model in the Praxeme methodology in order to specify the business intent aspect. Then, the main contribution in this paper is the proposal of rules to derive the semantic [17] and pragmatic [18] aspect recently obtained in logical factory, data structure and logical workshop of the Praxeme logical aspect. They used the M2M transformation approach and the ATL language for the implementation of the rules. The results of the derivation are concrete and abstract models (XMI format) such as the logic factory, data structure and logic workshop.

In [22], (Rapatsalahy et al, 2020) present a strategy that focuses on the automatic generation of software components of the Praxeme methodology from the intentional aspect specified by ReLEL. To do so, they proposed rules allowing the automatic translation of software components obtained from ReLEL into software components of the Praxeme methodology according to the M2T approach. For the implementation of the translation rules, they made use of the Acceleo template engine [22].

III. BACKGROUND

This section presents the two main concepts developed in this article. The first describes an overview of the basic concept of the Praxeme methodology. Then the second presents the ReLEL lexicon, a model which allows the representation of the enterprise's intention in the intentional aspect of Praxeme.

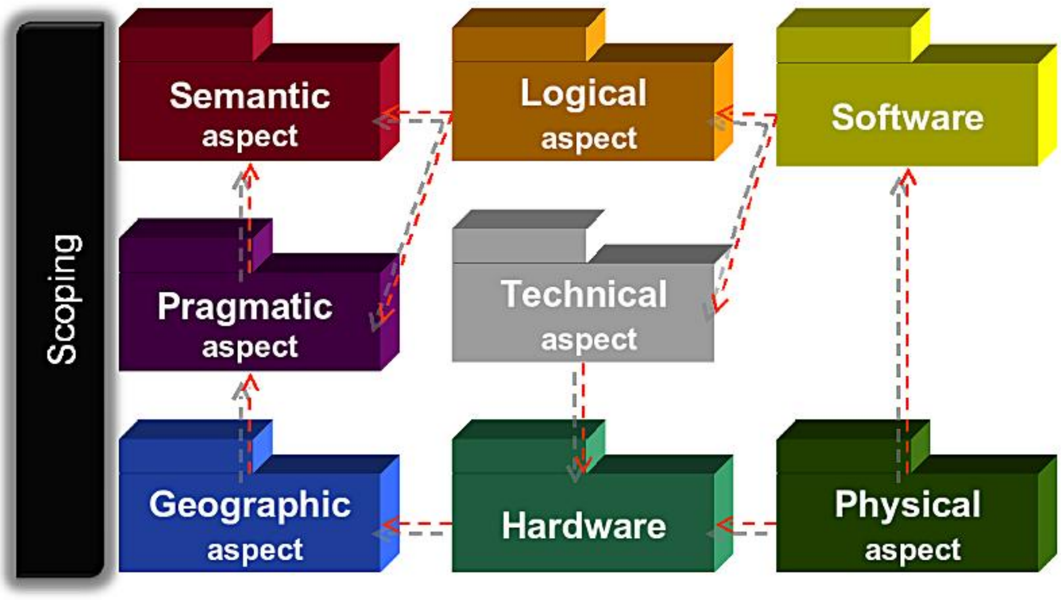


Fig. 1. Enterprise System Topology [22]

A. *Praxeme Methodology*

Praxeme is an enterprise methodology that allows the complexity of a system to be mastered through the principle of separating the responsibilities of the enterprise into a homogeneous whole named "aspect" [22]. To do this, this methodology proposes a framework called the Enterprise System Topology (TSE) that defines several aspects and each of them is represented by a particular model [19]. This article deals with the intentional, semantic, logical, technical and software aspects of the Praxeme methodology. It uses the MDA approach concerning model transformation and service architecture (SOA) to build a system. Fig. 1 represents the Praxeme Enterprise System Topology.

The intentional aspect of Praxeme gathers the terms used exclusively in the enterprise and/or in its field of activity accompanied by their precise definitions [12, 19]. Consequently, in [12], (Biard et al., 2013) support the usefulness of language close to natural language to express the company's intentions so that it is understood by all stakeholders. This is the reason why several research works have focused on the instantiation of the natural language-oriented requirements model in the Praxeme methodology (see Related work section).

The semantic aspect captures the knowledge of the business such as the objects, data and business rules that are the basis of the enterprise. These are the most stable and durable elements, and they are independent of the organization deployed. The semantic business model is developed in terms of packages, modeled through UML class and transition state diagrams. It is obtained by deriving the intentional aspect [17].

Praxeme prioritizes logical modeling through the logical aspect, which is the most important because it ensures the decoupling between the solution designed and the technologies used for its implementation. It uses basic units called logical services to build systems. Indeed, they are the finest components in the service architecture and are used for an elementary response given by the system in relation to an information, action or transformation need [5]. All the information that makes up the logical services should be described in the model that specifies the intentional aspect of Praxeme. Consequently, the logical aspect is obtained from the articulation of the intentional, semantic, and pragmatic aspects [19].

The technical aspect concerns the choice of the technology that should implement the logical aspect (e.g., how to code the services in the logical aspect of Praxeme) [22].

The software aspect covers all software components that automate the part of an action of a system [5]. It is composed of all things found in a software (code, sources, binaries) based on a structured data model [22].

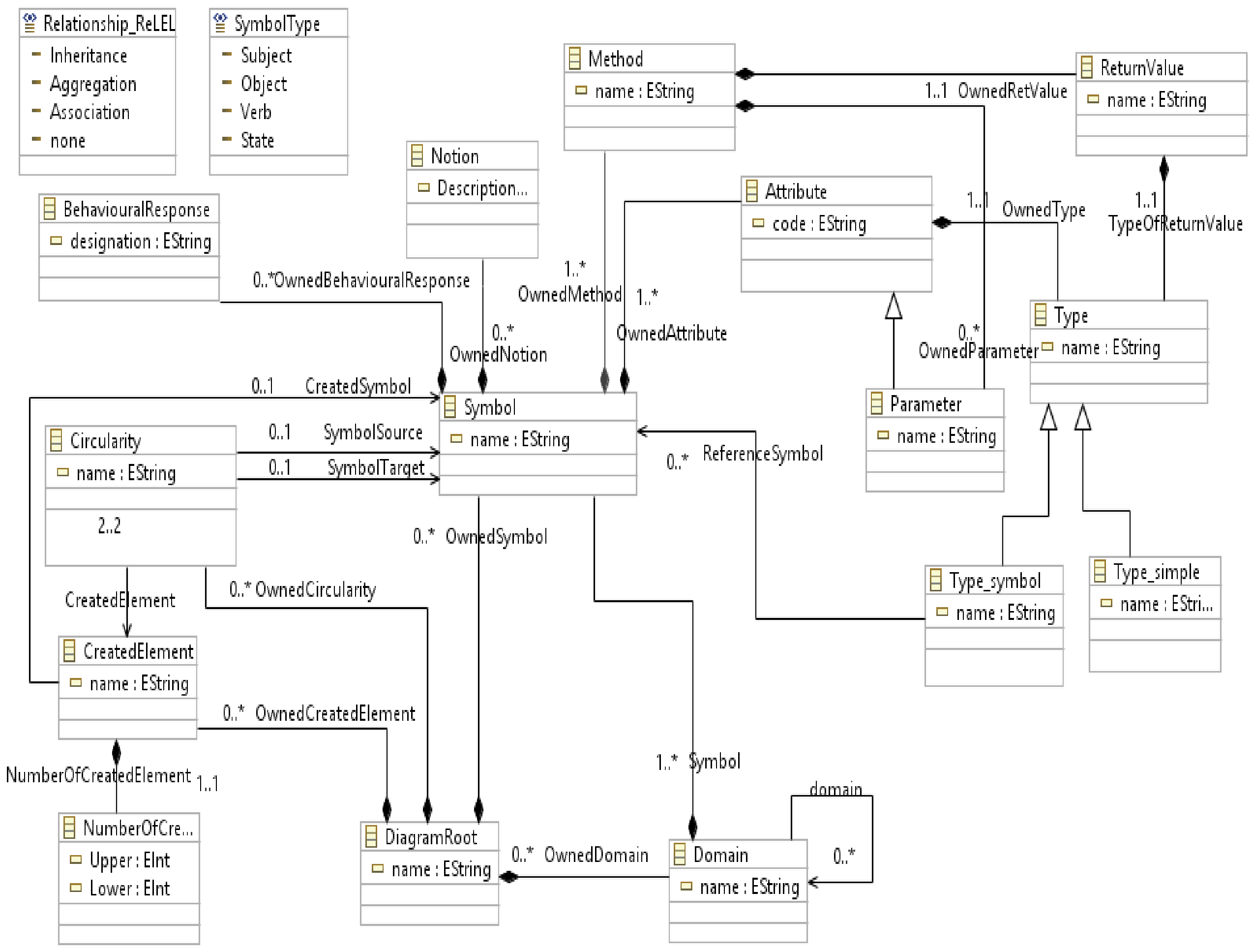


Fig. 2. Simplified ReLEL metamodel [21]

B. Restructuring extended Lexical Elaborate Language

ReLEL is a natural language lexicon that captures meaningful terms in the UofD (universe of discourse) and stores it from a symbol. The UofD includes all information sources and people related to software [1]. Indeed, the ReLEL symbol is a simple coding system that includes four entities such as notion, behavioral response, attribute, and method. The notion relates the denotation of the symbol, that is, the intrinsic and substantial characteristics of the symbol, while the behavioral responses represent the relationship between the described symbol and other symbols or "connotation" [23]. The attribute and method represent the conceptual level of a symbol and its characteristics necessarily depend on the symbol typology. Each ReLEL symbol belongs to one of the following categories, namely subject, object, verb and state. The symbols classified as subject, verb and object are the elements integrating the requirements [9]. The three objectives of ReLEL are the unification of the language for communication with stakeholders, the specification of requirements, and the accurate representation of the conceptual information corresponding to each term in the UofD. Fig. 2 represents the extract of the ReLEL metamodel used in this article.

IV. PROPOSED METHODOLOGY

The approach proposed in this paper classifies the model transformation into two parts. The first one adopts the M2M transformation technique that links the intentional aspect to the semantic aspect and then the semantic aspect to the logical aspect. The model that represents the intentional aspect of the Praxeme methodology captures the customer requirements. Consequently, it is classified as a CIM (Computation Idependant Model) in the MDA approach. The models that specify the semantic as well as the logical aspect are PIMs (Platform Independent Models) since they describe the details of the system without showing the specific characteristics of a particular execution platform or technology. Hence, the basic principle of this M2M approach is to generate the target model namely the semantic (PIM) and logical (PIM) from the source model such as the intent (CIM) as well as the semantic model (Fig. 3). In the transformation, the intentional aspect is represented by the ReLEL requirements model and conforms with its metamodel (Fig. 2). Then the semantic aspect is specified by the class diagram and the UML transition state machine. Finally, the logical aspect is specified by the logical model which conforms to its metamodel (Fig. 4). The second approach adopts the M2T transformation technique which consists in translating the logical model obtained from the intentional aspect represented by the ReLEL model into an object-oriented code. Fig. 3 illustrates the synopsis of the M2M approach proposed in this paper.

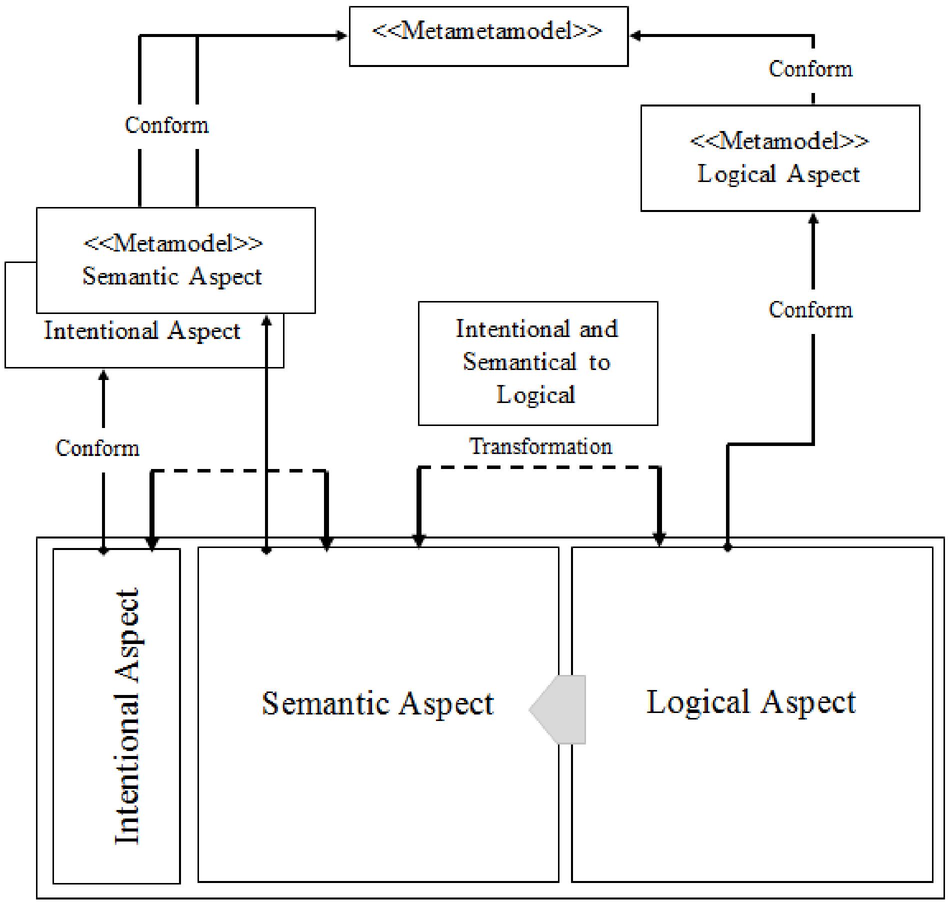


Fig. 3. Overview of the proposed M2M approach

A. Derivation process from the intentional aspect to the semantic aspect of Praxeme

The first part of the derivation process consists of transforming the ReLEL requirements model of the intentional aspect into the UML class diagram of the semantic aspect of the Praxeme methodology. The transformation is based on ten rules described in human language. Table I illustrates the mapping relationship between elements of the ReLEL model and the UML class model. The mapping rules between the ReLEL model and the UML class diagram in Table I only concern the ReLEL symbols of type subject and object. The mapping rules link from the ReLEL model elements (left) to the UML class model elements (right).

TABLE I. RELATIONSHIP BETWEEN ELEMENTS OF THE RELEL MODEL AND THE UML CLASS MODEL

Description of ReLEL Model	**Elements of the ReLEL model**	**UML class model elements**
A domain is a necessary concept for storing ReLEL classified as subject and object	ReLEL Domain	UML package
A ReLEL of subject type corresponds to an actor in the UofD. A ReLEL classified as object represents a significant and passive entity in the UofD.	ReLEL Symbol	UML class
The attributes provide the characteristics of the ReLEL symbol.	ReLEL Atrribute	UML class attribute
The method is an action to access a ReLEL object.	ReLEL Method	UML class operation
A parameter is nothing else than an already existing attribute which is manipulated by the method of a ReLEL object.	ReLEL Parameter Method	UML operation parameter
The return value is a response returned by the method after the manipulation of the ReLEL object.	Return value of a ReLEL method	Return value of a UML class operation
The type of a ReLEL attribute can be either a ReLEL symbol (subject/object) or a simple type (complete, string of characters, etc...).	Type of a ReLEL attribute	Type of a UML class attribute
The type of the return value of a method of a ReLEL object can be either a ReLEL symbol (subject/object) or a simple type (complete, string of characters, etc...).	Type of the return value of a method of a ReLEL object	Type of the return value of an operation of a UML class
The concept of circularity links two ReLEL source and target objects.	Circularity between two ReLEL objects	Association between two different UML classes
The concept of number of elements created defines the minimum and maximum occurrence of an association between two ReLEL symbols.	Number of ReLEL items created	UML cardinality

The second part of the derivation process is the transformation of the ReLEL requirements model of the intentional aspect into a UML transition state diagram of the semantic aspect of the Praxeme methodology. The UML transition state diagram is composed of state, transition and event that triggers the transition. Table II illustrates the mapping relationship between elements of the ReLEL model and the UML transition state model.

TABLE II. MAPPING RELATIONSHIP BETWEEN ELEMENTS OF THE RELEL MODEL AND THE UML TRANSITION STATE MODEL

Description of the ReLEL model elements	**Elements of the ReLEL model**	**Elements of the UML transition state model**
The state classified ReLEL symbol is characterized by attributes that contain values at different times during system execution.	ReLEL classified state	UML transition state diagram
The concept of circularity allows to link two ReLEL objects target, and source classified as a state.	ReLEL classified state related by the concept of circularity	Transition of an event in the UML transition state diagram
A state classified ReLEL object is triggered by the event of another state ReLEL object.	Method of a ReLEL of type state	UML transition state diagram event

B. *Process for deriving the semantic aspect into the logical aspect of Praxeme*

The logical aspect of the Praxeme methodology is a means of opening the system because it leads to the production of software components published as a service in the logical continuation of the web service technology. In this article, we base ourselves more on the logical service called BLS (Business Logic Service). It is described in the logical aspect of Praxeme from the semantic modeling of the methodology. The BLS is obtained from the derivation of the operation belonging to the semantic class. It is in the Business Logic Machine or BLM of the Praxeme logic model which also derives from a semantic class. The BLM is encapsulated in the logical workbench or LW which does not correspond to any element in the upstream model but results from the structuring decision taken by the logical architect during the logical design. Thus, the access to the BLS is either directly between BLM located in the same LW or through the service interface provided by the LW. The relationship between the BLM is only reflected in the usage relationship which is a functional dependency obtained at the time of execution of the BLS service call. Finally, the aggregate of the LW is stored in the logical factory or LF. Fig. 4 represents the logical factory metamodel used in this paper.

Table III illustrates the mapping relationship between elements of the semantic model and the logical model of the Praxeme methodology. The components of the Praxeme logical model resulting from the approach proposed in this article are the logical factory, the logical workshop, the logical business machine, the logical business service, and the interface services.

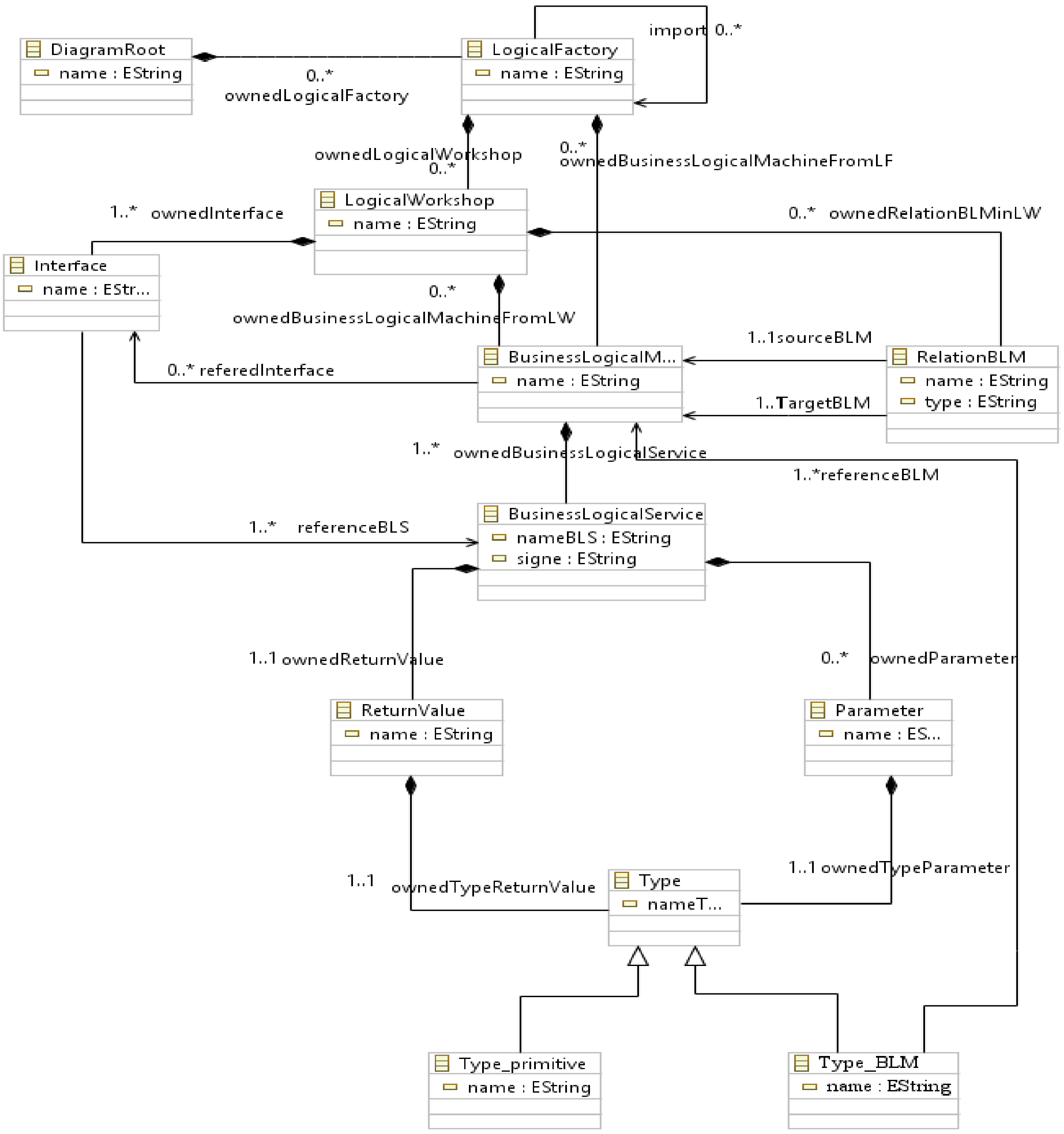


Fig. 4. Overview of the proposed M2M approach

TABLE III. MAPPING RELATIONSHIP BETWEEN ELEMENTS OF THE UML CLASS MODEL AND THE LOGICAL MODEL

Elements of the UML class model (semantic aspect)	**Elements of the logic model (logical aspect)**
UML package	Logic factory or LF
UML Class	Business Logic Machine (BLM)
UML class operation	Business Logic Service (BLS)
UML class operation parameter	BLS parameter
Return value of a UML class operation	BLS return value
Primitive type of a parameter or a return value of a UML class operation	Type primitive of a parameter or return value of a BLS
Class type of a parameter or a return value of a UML class operation	BLM type of a parameter or return value of a BLS

C. *Derivation process from the logical aspect to the Praxeme software aspect*

The derivation of the logical component obtained from ReLEL into software components of the Praxeme software aspect is the last process proposed in this article. Indeed, the technical implementation of the logical components obtained from ReLEL into software components must take into account the technical aspect of the Praxeme methodology [5, 22] Thus, the rules for mapping the logic model into the code skeleton presented in Table IV take into account the Java technology. Fig. 5 illustrates the synopsis of the translation of the logic model into a code skeleton of the Praxeme software aspect that implements the M2T approach in MDA. We used the translational approach that directly translates the PIM classified logic model into object-oriented code [15]. Therefore, for the translation, the Acceleo template engine is used. The principle is that the logical model obtained in the logical aspect of Praxeme is added in the execution chain as a source model. Afterwards, the data that makes up the logical model is extracted into a module (file '.mtl') [16]. The mapping rules for translating logical components into software components proposed in this paper are applicable for other object-oriented technological choices [22].

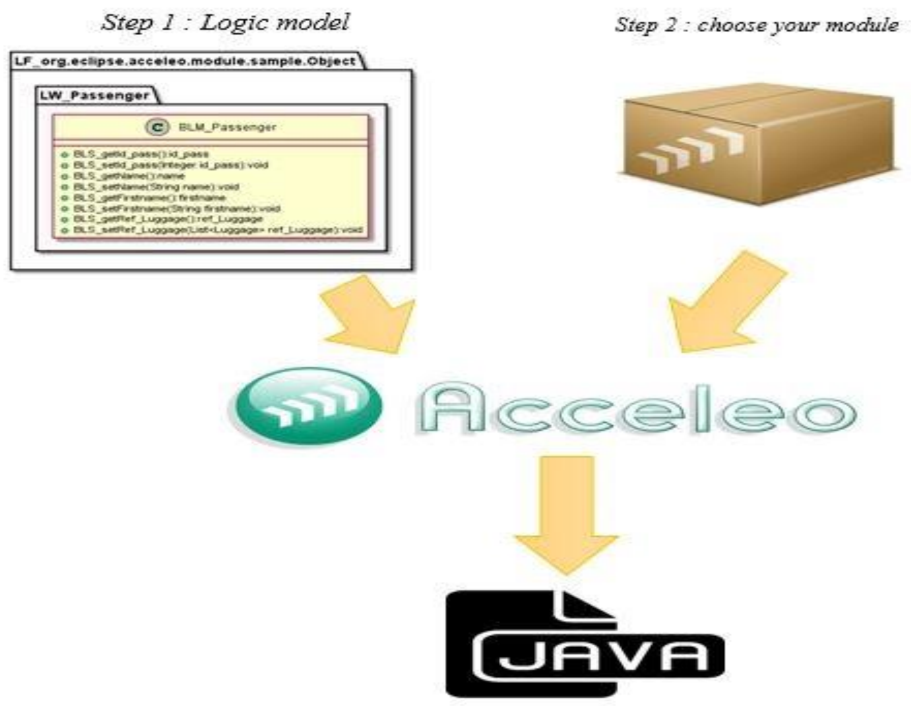

Step 1 : Logic model
LF_org.eclipse.acceleo.module.sample.Object
LW_Passenger
BLM_Passenger
BLS_getId_pass():id_pass
BLS_setId_pass(Integer id_pass):void
BLS_getName():name
BLS_setName(String name):void
BLS_getFirstname():firstname
BLS_setFirstname(String firstname):void
BLS_getRef_Luggage():ref_Luggage
BLS_setRef_Luggage(List<Luggage> ref_Luggage):void
Step 2 : choose your module
Acceleo
JAVA

Fig. 5. Overview of the proposed M2T approach

TABLE IV. MAPPING RELATIONSHIP BETWEEN ELEMENTS OF THE PRAXEME LOGIC MODEL AND JAVA CODE

Elements of the logic model (logical aspect)	**Java elements (software aspect)**
Logic factory or LF	Java package
Business Logic Machine (BLM)	Java class
Business Logic Service (BLS)	Java class method
BLS parameter (prefixed by “SET” [21]	Java class instance variable
Type (BLM or primitive) of a BLS parameter (prefixed by "SET")	Type (class or primitive) of an instance variable of a Java class
Return value of a BLS	Return value of a method of a Java class
Type (BLM or primitive) of the return value of a BLS	Type (class or primitive) of the return value of a Java class method
Parameter of a BLS	Parameter of a method of a Java class
Type (BLM or primitive) of a BLS parameter	Type (class or primitive) of a parameter of a method of a Java class

V. VALIDATION OF THE STRATEGY

To validate the strategy, we introduced the proposed approach on the travel booking process of the agency 'Transpost Malagasy' connecting two major cities in Madagascar [21]. Only an extract of information from the main source, namely the UofD, is presented in this section: "for a reservation, the luggage can be a part of the passenger's equipment" [22]. Fig. 6 shows an extract of the information from the ReLEL symbol classified object "Reservation" in XMI format.

```
<? xml version="1.0" encoding="UTF-8"?>
<MMReLEL:DiagramRoot xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:MMReLEL="http://MMReLEL/1.0" xsi:schemaLocation="http://MMReLEL/1.0../ Metamodele/MM_ReLEL.ecore" nameOfRoot="Trip
reservation">
<ownedDomaine nameOfDomain="term classified object">
<ownedSymbol name="Reservation" Classification="Object">
   <OwnedBehavioral Description="The action applied to it is the validation of the reservation"/>
   <OwnedBehavioral Description="The action applied to it is the listing of reservations "/>
   <OwnedBehavioral Description="The action applied to it is the verification of the payment deadline"/>
   <OwnedBehavioral Description="The action applied to it is the initialization of the status (paid not or paid)"/>
   <OwnedNotion Description="This is an object that allows the customer to book a land trip"/>
   <OwnedNotion Description="A reservation contains one or more places"/>
   <OwnedNotion Description="A reservation has a customer reference"/>
   <OwnedNotion Description="A reservation has a travel reference in order to know the destination, the chosen departure date, the place of departure"/>
   <OwnedNotion Description="The reservation contains the deadline of the date of payment"/>
   <OwnedNotion Description="A reservation is made up of passengers"/>
   <OwnedNotion Description="A reservation has a status"/>
   <OwnedMethod Name="getPassenger" OwnerSymbol="Reservation">
    <OwnedRetValue name="passenger" OwnerSymbol="Reservation">
     <TypeOfReturnValue xsi:type="MMReLEL:Type_Symbol" nameOfType="Type_Symbol" OwnerSymbol="Reservation"
       OwnerType="Method" nameOfType_symbol="Passenger" ReferSymbol="//@ownedDomaine.0/@ownedSymbol.2"/>
    </OwnedRetValue>
   </OwnedMethod>
   <OwnedMethod Name="setPassenger" OwnerSymbol="Reservation">
    <OwnedParameter Code="passenger" Definition="this is a person who takes a seat in a car for a trip."
      OwnerSymbol="Reservation" Size="3" name="passenger">
     <OwnedType xsi:type="MMReLEL:Type_Symbol" nameOfType="Type_Symbol" OwnerSymbol="Reservation"
       OwnerType="Method" nameOfType_symbol="Passenger" ReferSymbol="//@ownedRootSymbol.7"/>
    </OwnedParameter>
   </OwnedMethod>
   <OwnedAttribute Code="passenger" Definition=" this is a person who takes a seat in a car for a trip."
     OwnerSymbol="Reservation" Size="3">
    <OwnedType xsi:type="MMReLEL:Type_Symbol" nameOfType="Type_Symbol" OwnerSymbol="Reservation"
      nameOfType_symbol="Passenger" ReferSymbol="//@ownedDomaine.0/@ownedSymbol.2"/>
   </OwnedAttribute>
 </ownedSymbol>
</ownedDomaine>
</MMReLEL:DiagramRoot>
```

Fig. 6. Extract of information from the ReLEL symbol object "reservation"

Then we applied the derivation rules (intentional, semantic, logical) and the ATL transformation described in this paper (Tables I, II, III) on this case study. Thus, we automatically obtained as a result of the derivation and transformation the logical model represented by Fig. 6 (only the extract of the logical model is presented in this paper). Fig. 7 illustrates the logical components of the logical aspect of Praxeme, namely the logical factory prefixed by the letter 'LF' which stores a logical workshop called 'LW_reservation' in which a logical business machine called 'BLM_Reservation' is encapsulated as well as its service interface [22] Indeed, 'BLM_Reservation' stores all the services related to this business logic machine. It can thus be directly accessed by another logical business machine which is in the same logical workshop 'LW_Reservation' or from its service interface called 'ServiceInterfaceReservation'.

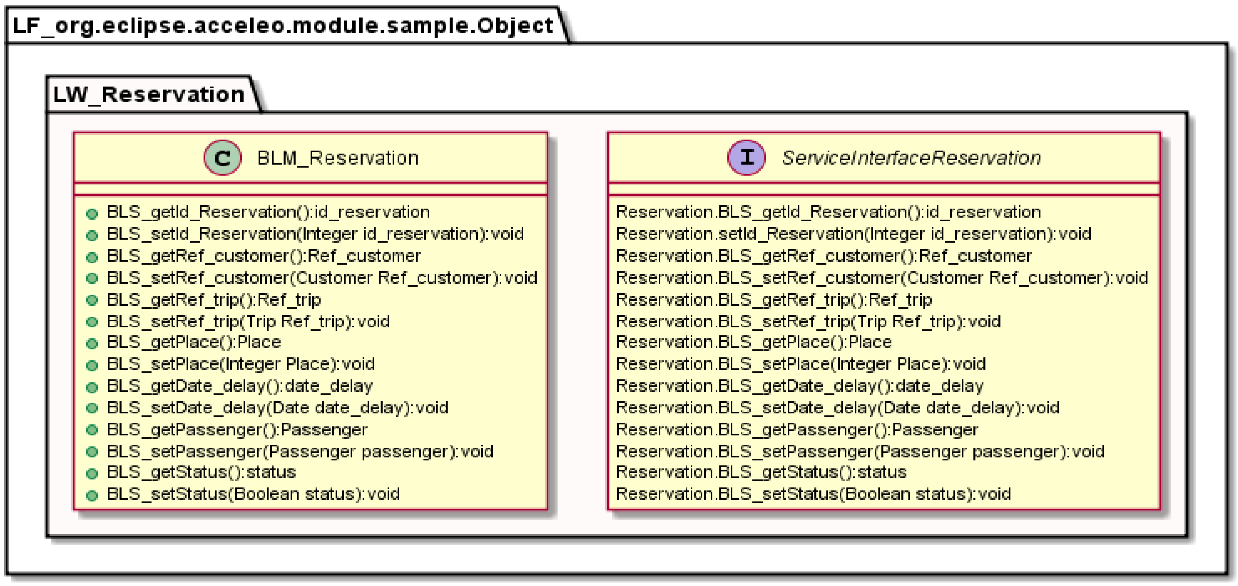
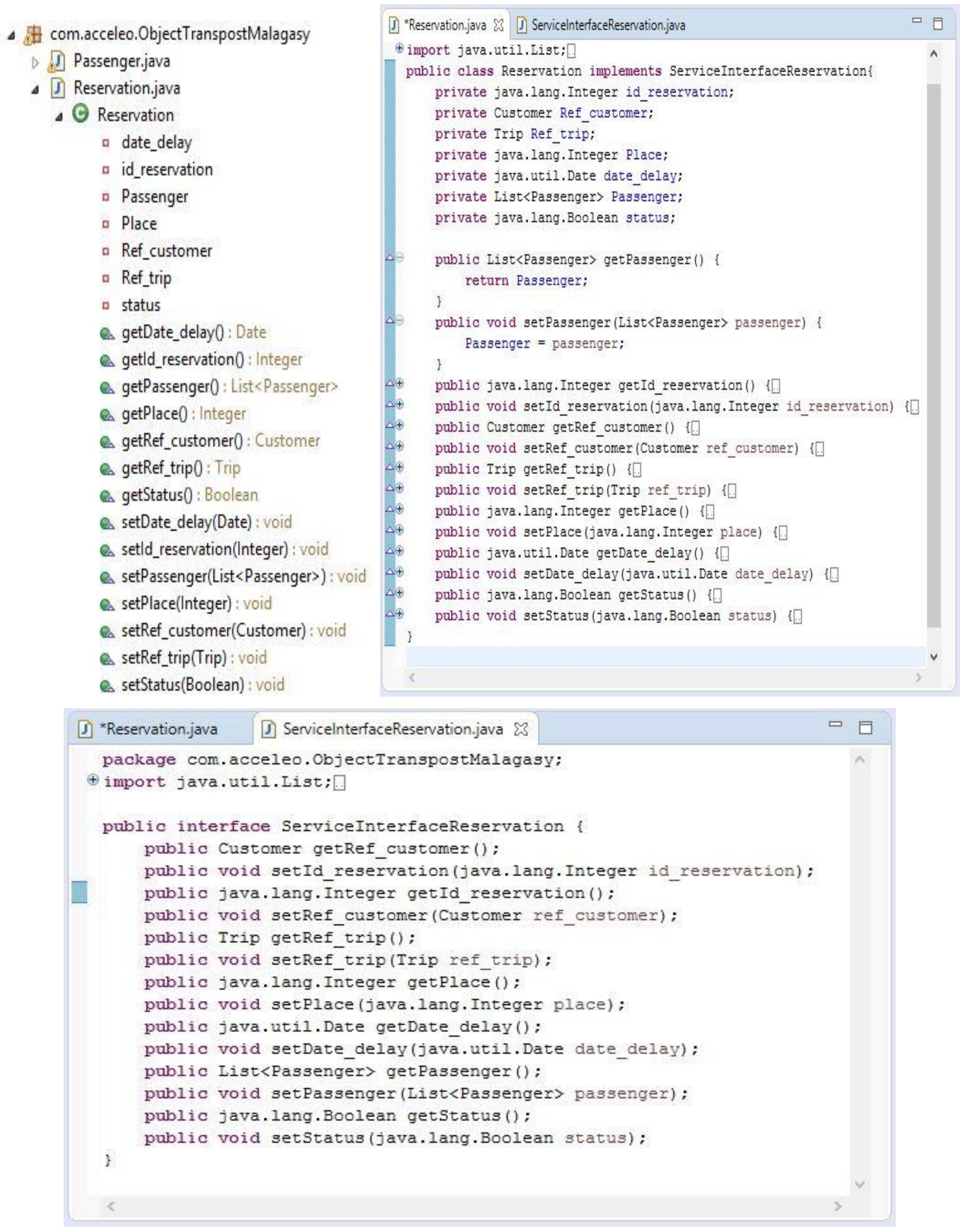

LF_org.eclipse.acceleo.module.sample.Object
LW_Reservation
C BLM_Reservation
BLS_getId_Reservation():id_reservation
BLS_setId_Reservation(Integer id_reservation):void
BLS_getRef_customer():Ref_customer
BLS_setRef_customer(Customer Ref_customer):void
BLS_getRef_trip():Ref_trip
BLS_setRef_trip(Trip Ref_trip):void
BLS_getPlace():Place
BLS_setPlace(Integer Place):void
BLS_getDate_delay():date_delay
BLS_setDate_delay(Date date_delay):void
BLS_getPassenger():Passenger
BLS_setPassenger(Passenger passenger):void
BLS_getStatus():status
BLS_setStatus(Boolean status):void
I ServiceInterfaceReservation
Reservation.BLS_getId_Reservation():id_reservation
Reservation.setId_Reservation(Integer id_reservation):void
Reservation.BLS_getRef_customer():Ref_customer
Reservation.BLS_setRef_customer(Customer Ref_customer):void
Reservation.BLS_getRef_trip():Ref_trip
Reservation.BLS_setRef_trip(Trip Ref_trip):void
Reservation.BLS_getPlace():Place
Reservation.BLS_setPlace(Integer Place):void
Reservation.BLS_getDate_delay():date_delay
Reservation.BLS_setDate_delay(Date date_delay):void
Reservation.BLS_getPassenger():Passenger
Reservation.BLS_setPassenger(Passenger passenger):void
Reservation.BLS_getStatus():status
Reservation.BLS_setStatus(Boolean status):void

Fig. 7. Praxeme logical component

Afterwards, we applied the rules for translating the logical component into a Praxeme software component (Table IV) to this case study, using Acceleo as the template. Finally, we obtained as a result Java software component such as package, class, interface and class method. Fig. 8 shows the generated Java component.



```
com.acceleo.ObjectTranspostMalagasy
  Passenger.java
  Reservation.java
    Reservation
      date_delay
      id_reservation
      Passenger
      Place
      Ref_customer
      Ref_trip
      status
      getDate_delay() : Date
      getId_reservation() : Integer
      getPassenger() : List<Passenger>
      getPlace() : Integer
      getRef_customer() : Customer
      getRef_trip() : Trip
      getStatus() : Boolean
      setDate_delay(Date) : void
      setId_reservation(Integer) : void
      setPassenger(List<Passenger>) : void
      setPlace(Integer) : void
      setRef_customer(Customer) : void
      setRef_trip(Trip) : void
      setStatus(Boolean) : void
```

```java
import java.util.List;
public class Reservation implements ServiceInterfaceReservation{
    private java.lang.Integer id_reservation;
    private Customer Ref_customer;
    private Trip Ref_trip;
    private java.lang.Integer Place;
    private java.util.Date date_delay;
    private List<Passenger> Passenger;
    private java.lang.Boolean status;

    public List<Passenger> getPassenger() {
        return Passenger;
    }
    public void setPassenger(List<Passenger> passenger) {
        Passenger = passenger;
    }
    public java.lang.Integer getId_reservation() {
    public void setId_reservation(java.lang.Integer id_reservation) {
    public Customer getRef_customer() {
    public void setRef_customer(Customer ref_customer) {
    public Trip getRef_trip() {
    public void setRef_trip(Trip ref_trip) {
    public java.lang.Integer getPlace() {
    public void setPlace(java.lang.Integer place) {
    public java.util.Date getDate_delay() {
    public void setDate_delay(java.util.Date date_delay) {
    public java.lang.Boolean getStatus() {
    public void setStatus(java.lang.Boolean status) {
}
```

```java
package com.acceleo.ObjectTranspostMalagasy;
import java.util.List;

public interface ServiceInterfaceReservation {
    public Customer getRef_customer();
    public void setId_reservation(java.lang.Integer id_reservation);
    public java.lang.Integer getId_reservation();
    public void setRef_customer(Customer ref_customer);
    public Trip getRef_trip();
    public void setRef_trip(Trip ref_trip);
    public java.lang.Integer getPlace();
    public void setPlace(java.lang.Integer place);
    public java.util.Date getDate_delay();
    public void setDate_delay(java.util.Date date_delay);
    public List<Passenger> getPassenger();
    public void setPassenger(List<Passenger> passenger);
    public java.lang.Boolean getStatus();
    public void setStatus(java.lang.Boolean status);
}
```

Fig. 8. Generated Java component

A better quality of the source code favors the updating of the program, i.e. its evolution [20].A modular system is easy to maintain because of its components with high cohesion and low coupling [20, 22]. For that reason, in this paper we measured the coupling and cohesion of the UML class diagram generated from the Java classes 'Reservation.java' and 'passenger.java' using the slicing technique [2, 11, 22]. Class slicing is defined as a decomposition technique that removes class components that are not relevant for computation from the slicing criterion [11].

A. *Identify the Headings Computing class coupling based on dependency graph slices*

Coupling measures the information flow between two classes [11]. The computation is based on the dependency of attributes and its methods [22]. Fig. 9 represents the class dependency graph or "CDG" based on the reservation information section.

Table V illustrates the result of the computation of the coupling between reservation and passenger and vice versa.

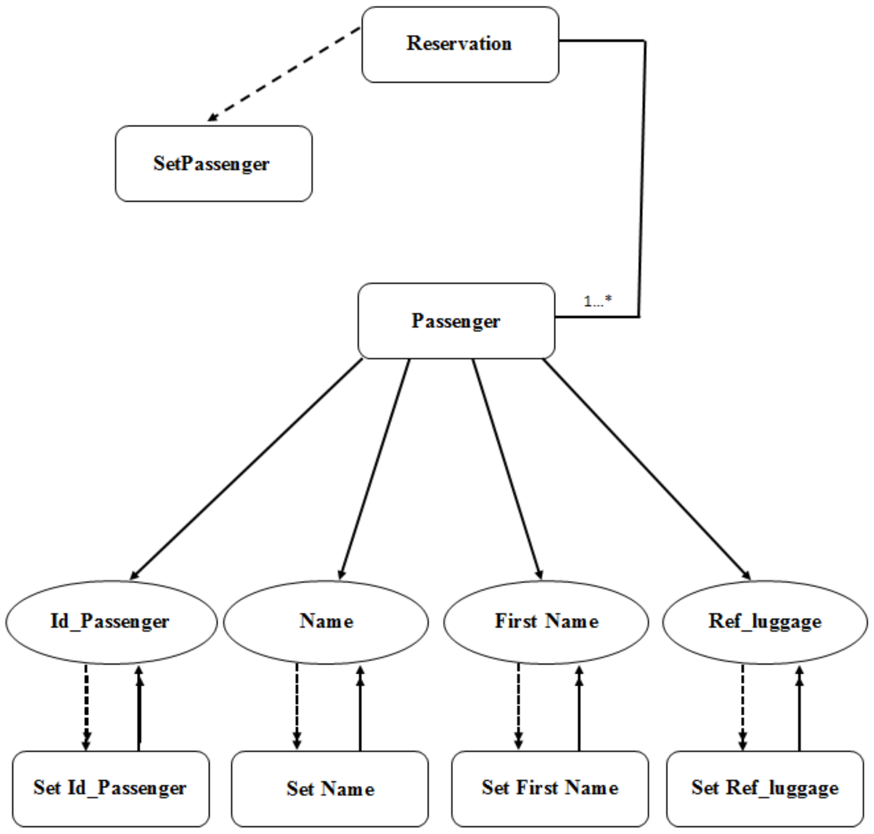


Fig. 9. CDG based on a reservation information slice

TABLE V. COUPLING BETWEEN RESERVATION CLASS AND PASSENGER AND VICE VERSA

	Coupling (Reservation, passager)	Coupling (Passager, reservation)
Total	0.125	0

B. *Calculation of the class cohesion based on the dependency graph slice*

Cohesion is an important factor in software design, indicating that the system has been well partitioned into components with strong internal relationships between attributes, method, and class [11]. The slice for the reservation class is defined as follows: Slicing Criterion (Reservation, id_Reservation).

Table VI shows the result of the cohesion calculation of the reservation class.

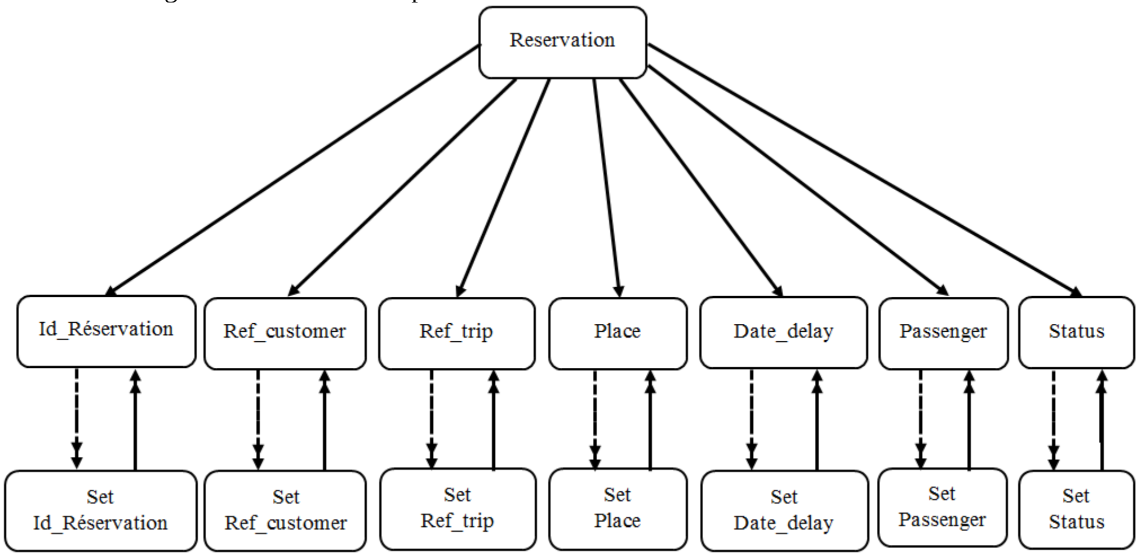


Fig. 10. CDG reservation class based on a slice

TABLE VI. COHESION OF THE RESERVATION CLASS

Nc	Dep_ D(n)	DRC(C)
Id-reservation	0.928	
Ref_customer	0.928	
Ref_trip	0.714	
Place	0.642	
Date_delay	0.785	
Passenger	0.785	
Status	0.785	
Set Id_reservation	0.928	
Set Ref_customer	0.928	
Set Ref_trip	0.714	
Set Place	0.642	
Set Date_delay	0.785	
Set Passenger	0.785	
Set Status	0.785	
DRC(C) = 1/Nc Σ Dep_ D(n)		**0.8**

The slice for the Passenger class is defined by the following slicing criterion: Slicing Criterion (Passenger, Id_passenger). Table VII shows the result of the calculation of the cohesion of the Passenger class.

TABLE VII. COHESION OF THE PASSENGER CLASS

Nc	Dep_ D(n)	DRC(C)
Id_passenger	0.875	
Name	0.875	
First Name	0.875	
Ref_luggage	0.875	
Set id_pass	0.875	
Passenger	0.875	
Set Name	0.875	
Set First Name	0.875	
Set Ref_luggage	0.875	
DRC(C) = 1/Nc Σ Dep_ D(n)		**0.875**

C. *Performance analysis*

The coupling and cohesion result "R" represents the correlation coefficient and the coefficient of the linear model [7, 11, 22]. "R" can be between 0.8 and 1.0 (indicating a strong association), or 0.5 and 0.8 is a medium association (if not weak or even non-existent). Accordingly the result of the coupling between the reservation and passenger class which is equal to 0.125 indicates a weak association. Then the result for the CDG of the reservation class R equal to 0.8 and the passenger equal to 0.875 indicates a strong cohesion. Obviously, these results allow us to deduce that the source code obtained from the method proposed in this article is easy to maintain, reusable and reliable, which allows us to obtain a more reliable and efficient software system.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed to automatically generate object-oriented code representing the software aspect of the Praxeme methodology from the intentional aspect specified by the ReLEL model [21]. To do so, we have proposed rules allowing the articulation of the following aspects of Praxeme: intentional, semantic, logical, technical and software. The process of transformation conforms to the MDA approach and is done in two steps, namely the model to model (M2M) which links the intentional, semantic, and logical aspects as well as the model to text (M2T) which links the logical aspect to the software aspect. As a final result of the transformation process, we have obtained Java software components namely packages, classes and class methods. Therefore, we can conclude that we have lifted the lock of software development automation from the ReLEL requirements model to the code of an application. Then, to validate our strategy, we analyzed the performance of the method proposed in this paper by measuring the coupling and cohesion of the UML class diagram obtained from the source code generated in this approach using the slicing technique [3, 11]. The results show that the coupling is weak, and the cohesion is strong, and it can be inferred that our method can produce a maintainable system. As a work perspective, we propose an automatic implementation of the Java code skeleton obtained in this approach from a UML state machine diagram.

REFERENCES

[1] J.C.S.P. Leite, G.D.S. Hadad, J.H. Doorn and G.N. Kaplan, “A scenario construction process”, Requirements Eng. Journals, vol. 5, no. 1, pp. 38-61, July, 2000.

[2] Y. Zhou, L. Wen, J. Wang, Y. Chen, H. Lu and B.X. DRC, “A dependence relationships based cohesion measure for classes”, In Proc. Of the 10th Asia-Pacific Software Engineering Conference Software Engineering Conference (ASPEC), Washington, DC, USA, 2003.

[3] Y.G. Gueheneuc, “ A reverse engineering tool for precise class diagrams”, Proc. Conf. Centre for Advanced Studies on Collaborative Research, pp. 28-41, 2004.

[4] X. Blanc, “MDA en action: Ingénierie logicielle guide par les modèles”, Eyrolles, 2005.

[5] D. Vauquier, “Modus la méthodologie PRAXEME, Guide general”, 2006.

[6] D. Vauquier, “Modus la méthodologie PRAXEME, Guide de l’aspect logique”, 2007.

[7] T.M. Meyers and D. Binkley, “An Empirical Study of Slice based Cohesion and Coupling Metrics”, ACM TOSEM, 17/1, 2007.

[8] OMG, "Semantics of Business Vocabulary and Business Rules", 2008.

[9] N. Niu and S. Easterbrook, “Extracting and modeling product line functional requirements”, in 16th IEEE International Requirements Engineering Conference, pp. 155-164, 2008.

[10] M. Nassar, A. Anwar, S. Ebersold, B. Asri, B. Coulette, Kriouile, “A Code Generation in VUML profile: a Model Driven Approach”, in 7th International Conference on Computer Systems and Applications, Rabat, Morroco, May 10-13, pp. 412-419, 2009.

[11] A. Kumar and S.K. Khalsa, “Determine cohesion and coupling for class diagram through slicing techniques”, IJACE, vol. 4, no. 1, pp. 19-24, 2012.

[12] T. Biard, M. Brigant and J.P. Bourey, “Explicitation et structuration des connaissances pour la transformation de l’entreprise : Les apports de la méthodologie Praxeme”, CIGI, 2013.

[13] S. Lamyae, T.Abdennebi,“Overall design approach for urbanized information systems: Application of the method Praxeme”, in 2014 Third IEEE International Colloquium in Information Science and Technology (CIST), IEEE, Tetouan, Morocco, pp. 18-23, 2014.

[14] J.L. Razafindramintsa, T. Mahatody and J.P. Razafimandimby, “Elaborated Lexicon Extended Language with a lot of conceptual information”, International Journal of Computer Science Engineering and Applications (IJCSEA), vol. 5, no. 6, Dec, 2015.

[15] H. Benouda, M. Azizi, R. Esbai and M. Moussaoui, “Modeling and code generation of Android applications using acceleo”, International Journal of Software Engineering and Its Applications, vol. 10, no. 3, pp. 83-94, 2015.

[16] I. B. Kara, “Design and Implementation of the ModelicaML Code Generator Using Acceleo 3.X.”, 2015.

[17] J.L. Razafindramintsa, T. Mahatody and J.P. Razafimandimby, “Deriving Semantic Aspect of the Praxeme Methodology from Elaborate Lexicon Language”, in Proc. 20th Int. Conf. Syst. Theory Control and Computing (ICSTCC), Sinaia, Romania, Oct. 13-15, pp. 842-847, 2016.

[18] J.L. Razafindramintsa, T. Mahatody and J.P. Razafimandimby, “Pragmatic aspect automatic derivation of the Praxeme methodology from an elaborated Natural Language”, in Proc. 1st RAAI, Bucharest, Romania, June. 19-20, 2017.

[19] J.L. Razafindramintsa, T. Mahatody, S.M. Simionescu and J.P. Razafimandimby, “Logical services automatic location from eLEL”, in Proc. 21st Int. Conf. Syst. Theory Control and Computing (ICSTCC), Sinaia, Romania, pp. 849-854, 2017.

[20] H. Razafimahatratra, T. Mahatody, J.P. Razafimandimby and S.M. Simionescu, “Automatic detection of coupling type in the UML sequence diagram”, 21st International Conference on System Theory, Control and Computing, Sinaia, Romania, pp. 635-640, 2017.

[21] M.A Rapatsalahy, J.L Razafindramintsa, T. Mahatody, S. Ilie and R.N. Razafindrakoto, “Restructuring extended Lexical elaborate Language”, in 23rd Int. Conf. Syst. Theory Control and Computing (ICSTCC), Sinaia, Romania, pp. 266-272, 2019.

[22] M.A. Rapatsalahy, H. Razafimahatratra, T.Mahatody, M. Ilie, S. Ilie, and R.N. Razafindrakoto, “Automatic generation of software components of the Praxeme methodology from ReLEL”, in 24th International Conference on System Theory, Control and Computing (ICSTCC), pp. 843-849, 2020.

[23] M. Urbieta, L. Antonelli, G. Rossi, and J.C.S. do Prado Leite, “The impact of using a domain language for an agile requirements management”, Information and Software Technology, 2020.