Computational Environment to Semi-Automatically Build a Conceptual Model Represented in OntoUML

Joselaine Valaski, Sheila Reinehr, Andreia Malucelli

PPGIa – Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba – PR – Brazil

joselaine.valaski@pucpr.br, sheila.reinehr@pucpr.br,
malu@ppgia.pucpr.br

Abstract. *A conceptual model can be an important instrument to support the software functional requirements elicitation because it promotes better understanding of a domain. However, the representation quality of the conceptual model depends on the expressivity of the language used. OntoUML is a proposed language to solve expressivity problems. Nevertheless, OntoUML models are complicated to build for novice modelers. This study presents an experiment performed in order to semi-automatically build a conceptual model represented in OntoUML. All the experiment steps were executed by a computational environment named ENSURE. The results showed that it is possible to identify 60% of the meaningful concepts.*

1. Introduction

During software development, poor understanding of the business and poor communication between the business specialists and the computing specialists can compromise the quality of the software (Luis et al., 2008). Therefore, especially in its early stages, the use of a common language that enables shared understanding among stakeholders is necessary to aid the smooth flow of information obtained from different sources (Lee and Gandhi, 2005).

The conceptual model is an instrument that enables the use of a common vocabulary and facilitates comprehension and discussion of elements that may appear in the software. However, the suitability of a conceptual modeling notation is based on its contribution to the construction of models that represent reality, thus enabling a common understanding between their human users (Mylopoulos, 1992). One of the most known conceptual metamodel is the Entity–Relationship (ER) model. However, the reason for the popularity of the ER model is also its main weakness. Although the metamodel is simple, which helps the conceptual modelers, it does not present high expressivity. The UML is also a well-known language for building conceptual models, which also presents the same problem of expressivity.

Guided by these matters, Guizzardi (2005) proposed OntoUML, a language used to represent ontology-based conceptual models. As the language is ontology-based, the conceptual models constructed in OntoUML are assumed to be more expressive and represent the real world of the domain more faithfully than other languages of conceptual representation. The constructs proposed in OntoUML prevent the overload and redundancy found in other languages, such as UML. However, as OntoUML is a

more expressive language, it proposes a larger set of constructs that are not easily identified, especially by novice modelers (Guizzardi et al., 2011).

Motivated by these challenges, this study describes an experiment to build semi-automatically a conceptual model represented by OntoUML. All steps described in the Experiment Method Section are executed by a computational environment called ENSURE (ENvironment to SUpport Requirement Elicitation). One of the main goals of ENSURE is to support the extraction of functional requirements of a domain using a conceptual model represented by OntoUML. This paper is organized as follows: in Section 2, the background of the proposal is outlined. Section 3 presents the method of the experiment, while Section 4 presents the results of the experiment. The final considerations and future works related to the proposal are presented in Section 5.

2. Background

This section presents the main concepts related to this study. The concepts are not exhaustively explained due to space limitation. However, they are discussed enough to understand the experiment executed.

2.1 OntoUML

The OntoUML language proposed by Guizzardi (2005) was motivated by the need for an ontology-based language that would provide the necessary semantics to construct conceptual models with concepts that were faithful to reality. The classes proposed in OntoUML are specializations of the abstract classes of the Unified Foundational Ontology (UFO) and extend the original metamodel of UML.

In this study, only the main constructs that make up the object type category will be presented (Guizzardi et al., 2011). In this category, constructs are more closely related to the static conceptual modeling of a domain. The Object Type constructs can be Sortal and Non-Sortal. The Sortal constructs provide identity and individualization principles to their instances, while the Non-Sortal constructs do not supply any clear identification principles. The Sortal constructs are classified as Rigid Sortal and Anti-Rigid Sortal. A Sortal is classified as rigid if it is necessarily applied to all its instances in all possible worlds. A Sortal is said to be anti-rigid if it is not necessarily applied to all its instances. The Rigid Sortal includes the Kind and Subkind categories. A Kind is a Rigid Sortal, and therefore has intrinsic material properties that provide clear identity and individualization principles. The Kind determines existentially independent classes of things or beings and are said to be functional complexes. A Subkind is also a Rigid Sortal that provides the identity principle and has some restrictions established and related to the Kind construct. Every object in a conceptual model must be an instance of only one Kind. There are two sub-categories of Anti-Rigid Sortal: Phases and Roles. In both cases, the instances can change their types without affecting their identity. Whereas during the Phase construct, the changes can take place as a result of changes of intrinsic properties. In the Role construct, the changes take place because of relational properties.

Compared with UML, OntoUML has a larger set of constructs, enabling greater expressivity of conceptual models and avoiding overload and redundancy (Guizzardi, 2005). Nevertheless, OntoUML is more complex to use than the traditional languages, such as UML, especially for novice modelers (Guizzardi et al., 2011). One of the difficulties of constructing a model represented in OntoUML is identifying the correct

construct for a given concept to be represented. In this sense, it is important to develop automatic or semi-automatic mechanisms that help the domain modeler to identify this concept and its correct construct. A linguistic approach with a semantic focus can be applied to aid comprehension of the concepts to be modeled (Castro, 2010).

2.2 Semantic types and Disambiguation

Dixon (2005) proposed a semantic organization for words in classes of meaning known as semantic types. In this proposal, semantic types handle nouns, adjectives, and verbs. Generally, in conceptual modeling, nouns are the semantic types that indicate important concepts in a conceptual modeling. The semantic types can be mapped to the constructs of OntoUML (Castro, 2010) and thereby enable semi-automatic support for their identification using Natural Language Processing (NLP). However, one of the challenges of automatic identification of the semantic type is the disambiguation of the term (Castro, 2010; Leão et al., 2013).

A word can have several meanings and the correct identification of its meaning may depend on the context in which it is used. The task of computationally identifying the meaning of words based on their context is known as Word Sense Disambiguation (WSD) (Pedersen and Kolhatkar, 2009). Techniques and algorithms for disambiguation are available, such as TargetWord, which is applied only in the case of a target word in a sentence, and the AllWord, which is applied to all words in a sentence. An example of a disambiguation technique is WordNet::SenseRelated (Pedersen and Kolhatkar, 2009). WordNet:SenseRelate is based on WordNet, which is a lexical base for the English language. This tool performs the disambiguation of a term found in the base and also identifies the corresponding semantic type. Figure 1 illustrates the use of Semantic types and Disambiguation concepts, where an example of "driver" term from WordNet database is presented. The term "driver" can have different meanings that depend on the context and each one can have different semantic type. In the "driver" example, the term can be associated to three semantic types: person, communication, and artifact. Considering the context, the TargetWord algorithm identifies the correct meaning. When the correct meaning is identified, it is possible to retrieve the associated semantic type.

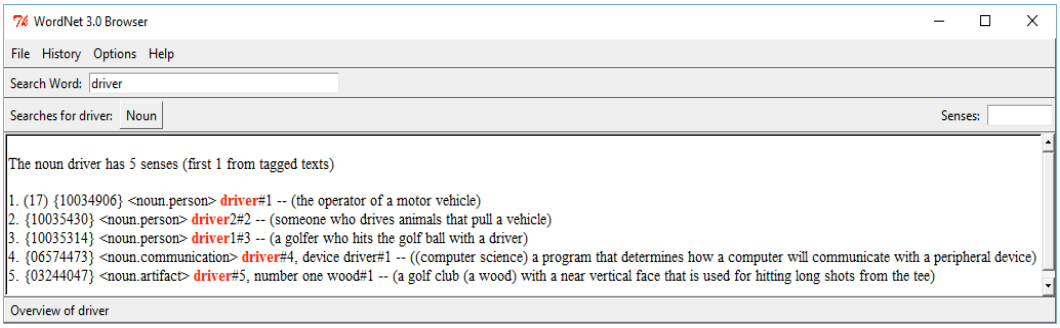

Figure 1. Example of a semantic type identified by WordNet.

2.3 ENSURE

ENSURE is a computational environment developed to support the execution of tasks related to Software Requirement Engineering. This environment contains in the

integration of relevant terms, extraction algorithms, terms disambiguation algorithms, WordNet database, OLED (2015) (OntoUML editor) and others.

This integration and interfaces allow the management of domain texts, identify relevant terms, execute terms disambiguation using WordNet database, build conceptual model, and derive functional requirements of a domain. The main goal of the environment is to support the decision taking in each one of these tasks. The main users comprise students of Requirement Engineering, the novice professionals.

3. Experimental Method

The experiment to build semi-automatically a conceptual model was conducted based on the steps of the method proposed in (Leão et al., 2013; Valaski et al., 2014). The main differences in this experiment are a new heuristic to identify the OntoUML construct, the execution by a computational environment, and the partial building of a conceptual model. Each one of these tasks are executed using the ENSURE.

3.1 Identifying relevant terms

The first step was the selection of the text. In this experiment, the same text applied on Valaski et al. (2014) was used. This decision was taken to facilitate the comparison between the previous and present experiment. The text selected is presented in Table 1 and describes a domain of bus route. The next step was the selection of the relevant terms. The starting point to define the relevant terms is the conceptual model relative to the text. The conceptual model is represented using the ER model (Gemino and Wand, 2005). Table 2 presents the 32 terms identified as relevant. In the context of this experiment, the terms are called "gold terms". They were compared with the terms automatically extracted.

The last step was the selection of algorithm to extract the relevant terms. The topia.termextract version 1.1.0, developed by Python (https://pypi.python.org/pypi/topia.termextract/) was used. This tool was chosen because of its satisfactory results (Valaski et al., 2014) and its use is free. To analyze Topia algorithm results, the metrics *precision* and *recall* Fawcett, T. (2006) were used.

Table 1. Selected text

Text
There are two ways for people to travel with Voyager. Either passengers can make a reservation on a trip, or passengers can show up at the boarding gate without a reservation and purchase a ticket for an unreserved seat. Passengers with a reservation are assigned a reservation date, whereas, passengers without reservations are assigned a boarding date. The name and addresses of all passengers are collected. Telephone numbers are collected where possible. All bus trips are organized into daily route segments. All daily route segments have both a start time and an end time. Each daily route segment. Voyager organizes is classified as a route segment with a segment number, start town, and finish town. Voyager offers a range of trips, and each trip is made up of one or more route segments. For every trip there is a trip number, start town, and finish town. If the trip is organized around a special event, the event name is also associated with the trip. Each daily route segment that Voyager offers is part of a dally trip. A daily trip is undertaken by one or more bus drivers. The name, address, and employee number of all drivers is collected. Voyager also records information about absent drivers. When a driver is absent. Voyager records the absence start date and the details about the absence. The absent driver provides one or more reasons for

being absent and each reason is assigned a detail number and a short description. Voyager also collects information about the buses used for daily trips. Buses have a make, model, and registration number. For buses in use, the average daily kilometers is collected. If a bus requires maintenance, Voyager notes the date on which the bus entered maintenance and records the one or more problems with the bus. Voyager assigns a problem number and a short description for every maintenance problem. Finally, the average cost to repair all problems with a bus in maintenance is also recorded.

Table 2. Gold terms

Terms
Absence; Absence Start Date; Address; Average Daily Kilometers, Average Cost to Repair; Boarding Date; Bus; Daily Route Segment; Daily Trips; Date Maintenance; Description; Details; Driver; Employee; End time; Finish Town; Maintenance Problems; Make; Model; Name; Passengers; Problem; Registration number; Reservation Date; Route Segment; Segment; Event name; Start Time; Start Town; Telephone; Trip; Trip Number

3.2 Identifying OntoUML construct

After selecting the relevant terms using Topia algorithm, the following heuristics were applied:

Rule i: For each relevant term (simple or compound), verify if the last word is number, name, or date. If true, the construct suggested must be Datatype. This rule was built because it was observed in previous experiment (Valaski et al., 2014) that these terms in general are not identified in the semantic database. Furthermore, these terms in general are observed to be related to attributes.

Rule ii: For each relevant term, where Rule *i* is not true, apply the algorithm TargetWord to disambiguate the term. If the TargetWord obtained the term disambiguate, retrieve the associated semantic type. With the semantic type, retrieve the OntoUML construct using the mapping described in Table 3. This mapping was established by partially using the proposal of Castro (2010). Castro used the Dixon (2005) theory to propose some mapping between semantic types and OntoUML construct.

Table 3. Semantic type vs Construct, map.

Semantic type	OntoUML Construct
Act	Relator
Artifact	Kind
Cognition	Kind
Communication	Relator
Location	Kind
Person	Role
Possession	Kind

Rule iii: For each relevant term, where Rules *i* and *ii* were not true, apply the following rules: if compound term (two or more words), verify if it is the suggested construct for each word individually. If suggested individually, the construct is either Kind or Relator; suggest the construct Relator. Example: **Bus Trips**, this term does not

exist in the WordNet database. However, the terms bus and trip exist. When the TargetWord algorithm was executed, the construct Kind was identified to the bus term and the construct Relator to the trip term. In this example, rule *iii* must suggest the construct Relator to the term **Bus Trips**. Rule *iii* was also extracted based on the observation of previous experiments. The simple terms were found, but the compound terms were not. This rule is a suggestion to try solving cases where the terms are not found in the WordNet database. It is important to emphasize that despite the application of these three rules, some terms are not suggested by the environment. The obtained results of this step were compared with previous results (Valaski et al., 2014).

3.3 Building conceptual model represented in OntoUML

Using the relevant terms, where the corresponding constructs were identified, the conceptual model was built. The OLED editor (2015) was used to build the model. In this experiment, the model was limited to represent only the elements without their relationship. This is a challenge to be addressed in future works. A complete model was manually built to compare the results obtained between the manual and automatic processes. The precision and recall metrics were used to calculate the accuracy and completeness of the results.

4. Experiment Results

The results of the experiment are presented according to the steps described in Section 3. All steps are executed automatically by ENSURE.

4.1 Identifying relevant terms

Figures 2 and 3 present the execution sequence to identify the relevant terms. Figure 2 shows the text selection, which was previously added in ENSURE, and Figure 3 shows the results of Topia algorithm execution. The algorithm processing returned 31 relevant terms, which are listed in Table 4.

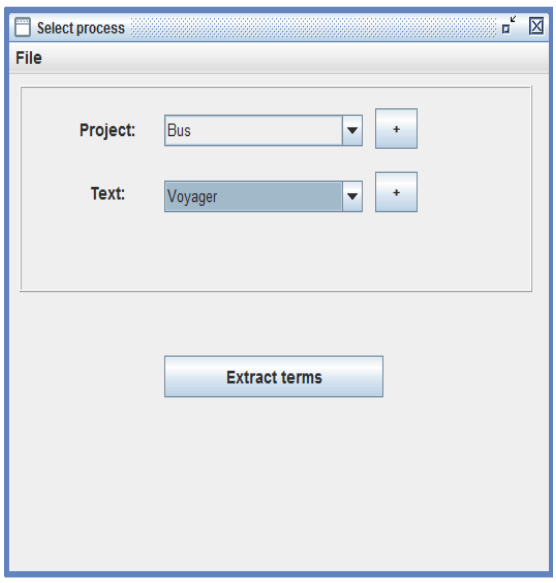

Figure 2. Step 1: Text selection

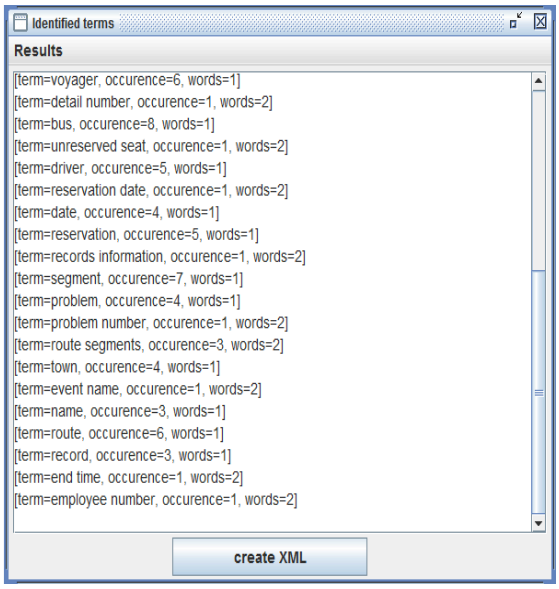

Figure 3. Step 2: Topia results

Table 4. Extracted relevant terms by Topia vs gold terms

Topia terms	Gold terms (exact)	Gold terms (partial)
Bus	Bus	
Bus drivers	-	Bus; driver
Bus trips	-	Bus; trips
Date	-	Boarding date
Detail number	-	Details
Driver	Driver	
Employee number	-	Employee
End time	End time	
Event name	Event name	
Maintenance	-	Maintenance problems
Maintenance problems	Maintenance problems	
Name	Name	
Number	-	Registration number
Passenger	Passenger	
Problem	Problem	
Problem number	-	Problem
Record	-	-
Records information	-	-
Registration number	Registration number	
Reservation	-	Reservation date
Reservation date	Reservation date	
Route	-	Route segment
Route segment	Route segment	
Segment	Segment	
Segment number	-	Segment
Telephone numbers	-	Telephone
Town	-	Finish town; Start town
Trip	Trip	
Trip number	Trip number	
Unreserved seat	-	-
Voyager	-	-

The results presented in Table 4 show that among the 31 terms identified by Topia, 14 terms have exact correspondence with the gold terms, while 17 terms do not have exact correspondence. On the other hand, the results presented in Table 5 show that among the 32 gold terms, 14 terms have exact correspondence with the relevant terms extracted by Topia, while 18 terms do not have exact correspondence. Based on these results, the precision (formula 1) and recall (formula 2) metrics were calculated. To proceed with the calculation, the following auxiliary variables are used: CT (total of correct terms returned by Topia); nCT (total of not correct terms returned by Topia); and CGn (total of gold terms do not returned by Topia).

$$Precision = CT/(CT + nCT) = 14/(14 + 17) = 0.4516 \quad (1)$$

$$Recall = CT/(CT + CGn) = 14/(14 + 18) = 0.4375 \quad (2)$$

Table 5. Gold terms vs Relevant Topia terms.

Gold terms	Topia terms (exact)	Topia term (partial)
Absence	-	-
Absence start date	-	-
Address	-	-
Average daily	-	-
Average cost to repair	-	-
Boarding date	-	Date
Bus	Bus	
Daily route segment	-	Route segment
Daily trips	-	Trip
Date maintenance	-	Maintenance
Description	-	-
Details	-	Detail number
Driver	Driver	
Employee	-	Employee number
End time	End time	
Finish town	-	Town
Maintenance problems	Maintenance problems	
Make	-	-
Model	-	-
Name	Name	
Passengers	Passenger	
Problem	Problem	
Registration number	Registration number	
Reservation date	Reservation date	
Route segment	Route segment	
Segment	Segment	
Event name	Event name	
Start time	-	-
Start town	-	Town
Telephone	-	Telephone numbers
Trip	Trip	
Trip Number	Trip Number	

The precision metric shows that Topia algorithm has an accuracy of 45.16%, while the recall metric shows that Topia algorithm has a completeness of 43.75%. The results were considered reasonable mainly because only the terms with exact correspondence were considered. On the other hand, if terms with partial correspondence (third column Tables 4 and 5) were also considered, better results were obtained. Topia extracted 27 terms with partial correspondence with gold terms (CT), while only 4 terms (Record; Records information; Unreserved seat) do not have partial correspondence (nCT). Only 9 terms (Absence; Absence start date; Address; Average daily kilometers; Average cost to repair; description; make; model; Start time) from the gold list do not have partial correspondence (CGn). The precision (formula 3) and recall (formula 4) metrics were recalculated.

$$Precision = CT/(CT + nCT) = 27/(27 + 4) = 0.8709 \quad (3)$$

$$Recall = CT/(CT + CGn) = 27/(27 + 9) = 0.75 \quad (4)$$

If partial terms are considered to collaborate with the identification of the relevant concepts of a model, Topia had an accuracy of 87.09% and completeness of 75%.

4.2 Identifying OntoUML construct

The heuristic described in Section 3.2 was applied with the list of relevant terms extracted by Topia. Figures 4 and 5 show the execution sequence to identify the construct. Figure 4 shows the XML file created. An XML file is required to execute the TargetWord algorithm described in Section 3.2, rule *ii*. In this file, the relevant terms are tagged in the original text. The TargetWord algorithm performs the disambiguation using the WordNet database and returns the corresponding semantic type. Figure 5 presents the disambiguation result and the construct suggested according to the mapping proposed in Table 3.

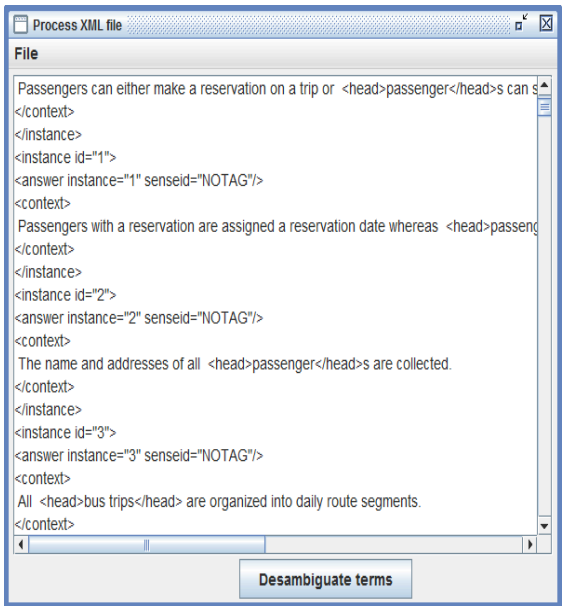

Figure 4. Step 3: XML file created

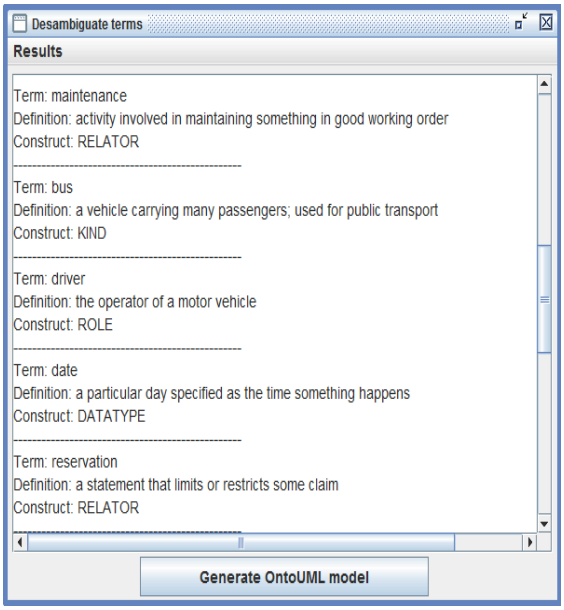

Figure 5. Step 4: Disambiguation results

Among the 31 relevant terms extracted by Topia, the environment suggested construct to 28 terms, which comprise about 90% accuracy. Considering these 28 terms, 13 had the construct suggested as Datatype (rule *i*), 12 had the construct suggested using the disambiguation process (rule *ii*), and 3 had the construct suggested using semantic type assigned by the simple terms (rule *iii*). In the experiment described in Valaski et al. (2014), among the 31 identified relevant terms, only 12 terms had suggested construct, comprising about 40% accuracy. The new heuristic contributed to identify mainly terms associated to Datatype construct. It also contributed to identify the construct of terms not found in the WordNet database, such as Bus Trips, Route Segment, and Maintenance Problems. These terms exemplified situations where the compound terms are not found but the simple terms are found. It is important to state that this experiment

only indicates the possibility to explore this kind of association. Other experiments with a diversity of text must be executed to confirm the validation of rule *iii*.

4.3 Building conceptual model represented in OntoUML

The conceptual model was generated both automatically and partially using the relevant terms list and the construct suggested. The conceptual model was built on OLED editor. Figure 6 shows the results of Figure 5 execution. The automatic identification of the relationship among terms is not yet possible on ENSURE. This is a challenge to be addressed in future work.

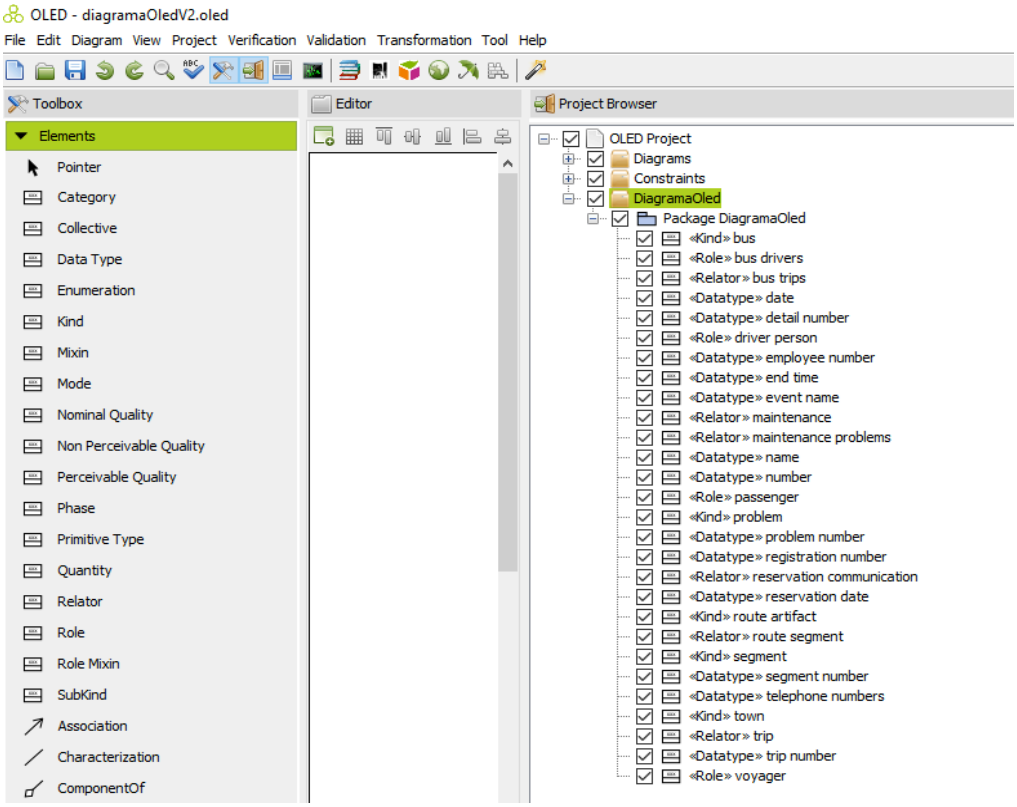

Figure 6. Identified automatic elements

The conceptual model in Figure 7 is manually built corresponding to the processed text (bus route) to analyze the result of this step. This model represented only the elements related to Kind, Role, and Relator (12 elements). The ER conceptual model presented in (Gemino and Wand, 2005) was also used to validate the model built. The elements associated to Datatype construct were not presented as they were considered less representative in this experiment. The model in Figure 7 was named as "gold model".

Considering only elements associated to Kind, Role, and Relator construct, the environment automatically identified 15 elements (Bus, Bus drivers, Bus Trips, Driver, Maintenance, Maintenance Problems, Passengers, Problem, Reservation, Route, Route Segment, Segment, Town, Trip, Voyager). Among of them, nine are present in the gold model (gray elements). Only the terms Bus drivers, Bus Trips, Maintenance, Route, Segment, and Voyager do not have exact correspondence in the gold model. However, all terms, except Voyager, have partial correspondence in the gold model. That is, if the

exact concept is not identified, it is believed that the partial concept can help in finding the exact concept. Among the elements represented in the gold model, only three terms were not identified by the environment, namely, Person, Daily Trip, and Daily Route Segment. However, the partial terms Trip and Route Segment were found. Considering these results, the precision (formula 5) and recall (formula 6) metrics are calculated.

$$Precision = 9/(9+6) = 0.60 \quad (5)$$

$$Recall = 9/(9+3) = 0.75 \quad (6)$$

In this context, considering only the more representative elements (Kind, Role, Relator) of the gold model, the environment had an accuracy of 60% and completeness of 75%. The quality and complexity of the text directly influence the accuracy of the results.

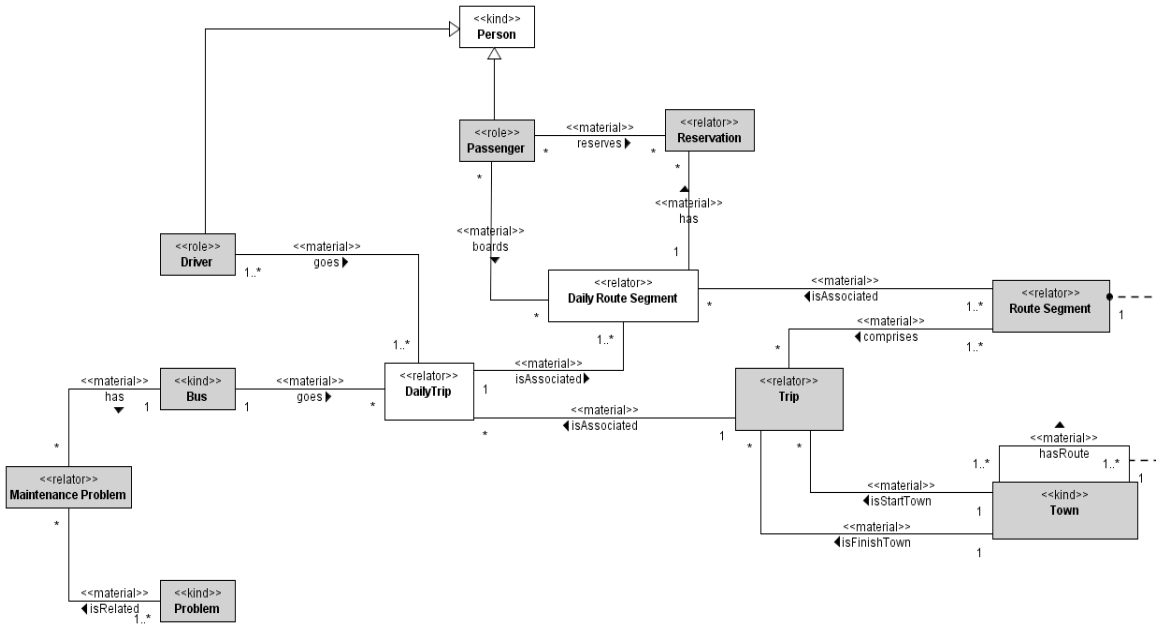

Figure 7. OntoUML gold model.

5. Conclusion

An experiment was performed to demonstrate the semi-automatically building of a conceptual model represented in OntoUML. The execution was performed through a computational environment called ENSURE. The initial results were considered satisfactory because through a domain text, it was identified 60% of the meaningful concepts. However, a number of challenges need to be addressed to improve the used methods and the environment. Some of these main challenges are: availability of algorithm and tools to process natural language with better results and availability of terms in semantic database to perform disambiguation. Another challenge is to identify automatic relationship among the elements in a conceptual model.

ENSURE is an environment built to promote Requirement Elicitation using a conceptual model represented in OntoUML as support. From the conceptual model, the environment suggests functional requirements of a domain. New implementations are

currently executed to improve the building of the conceptual model and the extraction of its functional requirement.

References

Castro, L. (2010) “Abordagem Linguística para Modelagem Conceitual de Dados com Foco Semântico”, Msc Dissertation, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, Brazil.

Dixon, R.M. (2005) “A Semantic Approach to English Grammar”, 2nd ed. Oxford University Press, USA.

Fawcett, T. (2006) “An introduction to ROC analysis”. Pattern Recognition Letters, 27, 861–874.

Gemino, A. and Wand, Y. (2005) “Complexity and clarity in conceptual modeling: Comparison of mandatory and optional properties”, Data & Knowledge Engineering, 55(3), pp 301–326.

Guizzardi, G. (2005) “Ontological Foundations for Structural Conceptual Models”, Telematica Institut Fundamental Research Series 15, Universal Press.

Guizzardi, G., Graças, A. and Guizzardi, R.S.S. (2011) “Design Patterns and Inductive Modeling Rules to Support the Construction of Ontologically Well-Founded Conceptual Models in OntoUML”, 3rd Workshop on Ontology Driven Inf. Systems (ODISE 2011), London.

Leão, F., Revoredo, K. and Baião, F. (2013) “Learning Well-Founded Ontologies through Word Sense Disambiguation”, in proceeding of: 2nd Brazilian Conference on Intelligent Systems (BRACIS-13), 2013

Lee, S.W. and Gandhi, R. (2005) “Ontology-based active requirements engineering framework”, in Engineering Conference, APSEC.

Luis, J., Vara, D. and Sánchez, J. (2008) “Improving Requirements Analysis through Business Process Modelling : a Participative Approach”, 1, pp 165–176

Mylopoulos, J. (1992) “Conceptual modeling and Telos, In P. Loucopoulos and R. Zicari, editors, Conceptual modeling, databases, and CASE. Wiley.

OLED (2015) “Ontouml-lightweight-editor” https://code.google.com/p/OntoUML-lightweight-editor/, acessado em: 2015 04.

Pedersen, T. and Kolhatkar, V. (2009) “WordNet :: SenseRelate:: AllWords: a broad coverage word sense tagger that maximizes semantic relatedness”, Human Language Technologies.

Valaski, J., Reinehr S. and Malucelli, A. (2014). “Environment for Requirements Elicitation Supported by Ontology-Based Conceptual Models: A Proposal”. In Proceedings of the 2014 International Conference on Software Engineering Research and Practice (SERP'14), ISBN 1-60132-286-0, Las Vegas, USA, p. 144–150.