Abstract

One of the main limitations attributed to Object Orientation (OO) by software engineers is the immaturity of the Object-Oriented Analysis (OOA) process. This article aims to propose a method to formalize this process. This method is based on the use of linguistic information from informal specifications. This information is composed of words which, in turn, denote elements of an OO modelling, such as classes, properties, etc. These words have a particular meaning, and their use in the modelling is usually related with that meaning. So, the objective is to analyse this information from the semantic and syntactic viewpoint and extract, by means of a formal procedure, the components of an OO system. These components are represented by one model that contains the static part of the system and another one that describes system behaviour.

1. INTRODUCTION

In view of the important benefits offered by OO [4][12], the software engineering community has looked back to this paradigm. However, there are still sizable gaps obstructing the way towards the adoption of OO. One of the main limitations is the indefiniteness in the OOA phase [20][10], which can be put down to a shortage of defined methods for identifying the key components of the conceptual models [25][2].

The content of this article is part of a research project¹ that seeks to formalize the OOA phase. For this purpose, we propose the use of the linguistic information contained in the informal requirements of a problem. The benefits of this approach have already been pointed out by several authors [13][9][15][7].

There have been several attempts to put this idea into practice. For example, [6][21][1] assign a common noun to a class; [5][23][3] relate the structure “is_a” with inheritance hierarchies; [19][24] associate verbs with relations between classes. However, the assignations proposed by the above authors are not justified. They merely constitute a set of heuristics or are shown by means of some examples.

A few researchers have tried to bring some formalization into the OOA process. For example, [8] defines assignations between elements of an intermediate language and elements of the conceptual models. This intermediate language is based on functional grammar. It is close to natural language, but not directly natural language.

Our work directly uses natural language structures in the construction of the conceptual models. In this way the process is more generic. Additionally, the method presented is formally justified. We propose a mapping between a subset of structures in the linguistic world, called linguistic patterns, and a subset of components of the conceptual world, called conceptual patterns. The linguistic world studied in our work, is the Spanish language. The justification of the above assignation is based on the equivalence of the linguistic and conceptual pattern representations in the mathematical world, namely in logic and set theory [17][18].

This paper shows the set of activities to be carried out to build the conceptual models on the basis of the above

¹ Researcher Training Grant, Universidad Politécnica de Madrid.
2. OVERVIEW OF THE METHOD

The overview of the method is shown in Figure 1. The method is composed of nine stages. Figure 1 also shows the Requirements Elicitation and Utility Language Validation processes. These processes are not covered by the proposed method but some phases of the method interact with them.

The Requirements Elicitation process must provide a textual description of the system to be built, the requirements document. No information on the syntactic content of this document has been found in the literature, so this is assumed to contain any natural language linguistic expression. To define the proposed mapping between the linguistic and conceptual worlds, we need to have a set of predetermined structures from both worlds. Conceptual world structures, that is conceptual patterns, are defined on the basis of the conceptual models discussed later. As regards linguistic structures, the solution is to transform natural language, which is potentially unlimited, to a restricted natural language [8]. That is, to work with a subset of natural language with defined syntax and semantics by means of what have been termed linguistic patterns. In our method, this sublanguage has been named Utility Language.

The Utility Language Validation process enables users to check that the Utility Language output as a result of method application, represents the same information that they have in mind.

Stages 1 to 3 prepare information contained in the requirements document for conceptual system modelling in the later stages, Stages 4 to 9. During Stage 1, information required for the OOA process, mainly functional requirements, is extracted from the requirements document. This information is debugged in Stage 2, looking for synonyms and polysemies. In Stage 3, we separate information representing data with which the system works, denoted Static Requirements; and information describing system behaviour, named Dynamic Requirements. For more details about these first stages see [17].

The conceptual models to be developed are the Object Model (OM), which sets out the static structure of the problem, and the Behaviour Model (BM), which represents system operation. Now we move on to discuss the activities to be carried out to build these models, during Stages 4 to 9.

3. CONCEPTUAL MODELLING

A small case for vehicle sales is proposed to illustrate the application of the above stages. Suppose after applying the first three stages to a requirements document, the Static Requirements are:

Vendors may be sales employees or companies. Sales employees receive a basic wage and a commission, whereas companies only receive a commission. Each order corresponds to one vendor only, and each vendor has made at least one order, which is identified by an order number. One basic wage may be paid to several sales employees. The same commission may be paid to several sales employees and companies.

and the Dynamic Requirements are:

A monthly payment is made to vendors. When a vendor makes a sale, he/she reports the order to the system. The system then confirms the order to the customer, and orders are delivered to customers weekly.

Below, we study how Static Requirements and Dynamic Requirements are processed to obtain the conceptual models of the application, OM and BM.
3.1. Static Requirements Structuring

Static Requirements are transformed according to the linguistic patterns shown in Table 1 to produce the Preliminary Static Utility Language. Some of the guidelines used to get this language are:

- Substitute pronouns with the noun or noun group which they replace.
- Discard adverbs and modal adjectives (probable, necessary, ...).
- If a modifier refers to several Nominal Syntagmata, that reference will be made explicit in each one.

The Preliminary Static Utility Language for the example concerned would be as follows:

1. **l1.1** Vendors may be sales employees or companies
2. **l2** Sales employees receive a basic wage and a commission
3. **l2** Companies (the adverb only is discarded) receive a commission
4. **l2** Each order corresponds to one vendor (the adverb only is discarded)
5. **l2** Each vendor has made at least one order
6. **l4** An order is identified by an order number
7. **l2** The same commission may be paid to several sales employees and several companies (several has been made explicit with companies)
8. **l2** One basic wage may be paid to several sales employees

3.2. Dynamic Requirements Structuring

Dynamic Requirements will be studied independently for each use case [14]. In one use case, all actions performed by a system are related with each other. These relations must be made explicit and should be described in chronological order, as this will facilitate BM construction. Dynamic Requirements are structured according to the linguistic pattern shown in Table 2, where the subordinate-clause may be formed by the coordination of simple clauses. The outcome is the Preliminary Dynamic Utility Language. Some of the rules applied to each use case to obtain this language are:

- Substitute pronouns with the noun groups they replace.
- Adverbs or adverbial expressions of time, such as “monthly”, “daily”, etc., will be transformed using the conjunction if. For example, “if the end of the month is reached”.

The Preliminary Dynamic Utility Language of the example is composed of the description of two use cases. The first one describes vendor payment, and the other describes order management. Their linguistic structures are:

1. Vendor Payment
   1.1 If the end of the month is reached then the firm pays vendors
2. Order Management
   2.1 If a vendor makes a sale then the vendor reports the order to the system
   2.2 If a vendor reports the order to the system then the system confirms the order to the customer
   2.3 If the system confirms the order to the customer and the end of the week is reached then the company delivers the order to the customer.

Preliminary Static Utility Language and Preliminary Dynamic Utility Language constitute the starting point for building the OM and BM, respectively. Therefore, the analyst must ensure that this information adequately represents what the user has expressed in the requirements document. So, these languages will be presented to the user for revision in the Utility Language Validation process, outputing the Static Utility Language and the Dynamic Utility Language.

In our example, the preliminary languages are assumed to be correct, so they would produce the Static Utility Language and the Dynamic Utility Language.

3.3. Object Model Construction

During this stage the OM is built from the Static Utility Language. This model is based on OMT notation [22]. Figure 2 shows the tasks performed to get it.

3.3.1. Identification of Classes and Relations

The classes and relations of the proposed example are shown in Figure 3. For their determination, each linguistic structure of the Static Utility Language is analysed as follows:

1. Identify the modelling pattern corresponding to the linguistic pattern of each structure, as shown in Table 1. For the justification of these assignations, see [17][18]. The names of the OM classes will be the nuclei of the noun syntagmata and of the complements, and are denoted NS_1, ..., NS_N and C_1, ..., C_N.
2. For any relation that requires an explicit name, this will be the third person singular of the verb in the tense in which it is conjugated. If there is a relation in the model with the same name, add a consecutive number to the name of the last relation. The relations go with a label composed of the number of the source clause, followed by its linguistic pattern identifier.

3. For relations based on patterns of the types \( l_2, l_3 \) and \( l_5 \), check whether there are other relations having the same semantics. If so, add the number indicating the second semantically equivalent clause to the label of each relation.

### Table 1. Assignment between linguistic and conceptual patterns for the OM

<table>
<thead>
<tr>
<th>LINGUISTIC PATTERN</th>
<th>CONCEPTUAL PATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 ) CLASSIFICATION</td>
<td>Nominal Syntagma bottom-up_classification_verb Complement ([\text{Nominal Syntagma}]_n) and/or Nominal Syntagma</td>
</tr>
<tr>
<td>( l_2 ) TOP-DOWN</td>
<td>Nominal Syntagma top-down_classification_verb Complement ([\text{Complement}]_n) or Complement</td>
</tr>
<tr>
<td>( l_3 ) MULTIPLE</td>
<td>Nominal Syntagma multiple_classification_verb Complement ([\text{Complement}]_n) and Complement</td>
</tr>
<tr>
<td>( l_4 ) COMPLEMENT ENUMERATION</td>
<td>Nominal Syntagma general_verb Complement ([\text{Nominal Syntagma}]_n) and/or Nominal Syntagma</td>
</tr>
<tr>
<td>( l_5 ) COMPLEMENT COMPOSITION</td>
<td>Nominal Syntagma component_composition_verb Complement ([\text{Nominal Syntagma}]_n) and/or Nominal Syntagma</td>
</tr>
<tr>
<td>( l_6 ) CONTENT COMPOSITION</td>
<td>Nominal Syntagma container_composition_verb Complement ([\text{Nominal Syntagma}]_n) and/or Nominal Syntagma</td>
</tr>
<tr>
<td>( l_7 ) IDENTIFICATION</td>
<td>Nominal Syntagma identifies Complement</td>
</tr>
<tr>
<td>( l_8 ) ADJACENT COMPLEMENTS</td>
<td>Nominal Syntagma general_verb ([\text{Nominal Syntagma}]_n) and/or Nominal Syntagma</td>
</tr>
</tbody>
</table>

### Figure 2. Object Model Construction Tasks

### Figure 3. Classes and Relation

#### 3.3.2. Determination of Cardinalities

The cardinalities of the relations are given by the determiners of the nouns. For their identification, the following rules are applied:

1. Determine the cardinality of each individual relation. Several rules have been proposed depending on the source linguistic pattern [17]. The cardinality will be added to the arrow of the relation, following [22].

2. Unify relations with the same semantics. That is, the first number of the label of the first relation is equal to the third number of the label of the second relation. The cardinalities of the resulting relation are the cardinalities of the ends of the arrows of the relations to be unified. The name of the final relation will be the name of the relation of higher cardinality.

3. Relations whose labels contain only the source clause and the linguistic pattern are unchanged in the model, except for relations based on the structure type \( l_4 \), where the direction indicators are deleted from the line that represents the association.

For the example considered, the resulting relations are
shown in Figure 4.

3.3.3. Reexamination of Inheritance

This task seeks to increase common characteristics sharing in order to create Optimized Inheritance Hierarchies. If all the subclasses in an inheritance hierarchy have a relation of equal semantic with the same class, the associations between the subclasses and the common class are substituted by a relation between the highest superclass in the hierarchy and the aforesaid class. The final relation will have the least restrictive cardinality and the name corresponding to the relation whose cardinality has prevailed.

In the example, the above can be applied to the classes “Sales employee”, “Company” and “Commission”. The Optimized Inheritance Hierarchy is shown in Figure 5.

3.3.4. Attribute Selection

During this task, the OM Class Attributes will be identified. Thus far, all we have are classes and relations. Each relation of aggregation and binary association in the model is studied in order to extract attributes. If one class only participates in the aforesaid relation, that class will be an attribute of the other class in the relation (for n-ary relations see [17]). The resulting OM is shown in Figure 6.

3.3.5. Object Model Verification

The above OM must be checked for completeness in this task. The model will be complete if there are no directional relations. Failure to comply with this rule indicates that the method will have to be reiterated from the Requirements Elcitation process. The OM of Figure 6 contains no directional relations.

3.4. Behaviour Model Construction

The BM is constructed in this stage from the Dynamic Utility Language. This model describes overall system behaviour in terms of events and operations as in J. Martin’s approach [16]. The tasks and outputs generated are detailed below.

3.4.1. Identification of Events, Operations and Control Conditions

Each linguistic structure in the Dynamic Utility Language is studied as below:

1. Check that the main clause represents a change of status in a system object. Determine which basic operation (creation, deletion, classification, declassification, change or information [17]) this status change refers to, indicating the prestatus and poststatus, that is, object states before and after the operation.

For example, the linguistic structure of the first use case would be analysed as follows:

Main clause: “the firm pays vendors”
Basic operation: Change
Prestatus: Vendor not paid
Poststatus: Vendor paid

If the clause were not to generate a basic operation, it would have to be decomposed into further clauses,
according to Dynamic Utility Language constraints, until each one generates a basic operation on an object.

2. Represent the modelling in terms of events and operations, as shown in Table 2, adding the number of the source structure to each operation. For the justification of this assignation, see [17].

3. Determine the control conditions for the activation of each operation. These control conditions will be the logical expression met by the clauses that constitute the trigger events of each operation.

In the case of the above linguistic structure, the control condition would be:

“the end of the month is reached”

<table>
<thead>
<tr>
<th>LINGUISTIC PATTERN</th>
<th>CONDITION</th>
<th>If subordinated_clause then main_clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCEPTUAL PATTERN</td>
<td>sub1</td>
<td>sub2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>subn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mc</td>
</tr>
</tbody>
</table>

Table 2. Equivalence between the linguistic pattern and the conceptual pattern for the BM

In Table 2, sub1, ..., subn are abbreviations representing the different subordinate clauses that may make up the subordinate clause, and mc is the abbreviation representing the main clause.

After analysing these linguistic structures, we will unify all the conceptual patterns, representing them, in a single diagram that will constitute the BM. For the construction of this diagram, it is important to check whether each event associated with each operation is an external or an internal event. An event will be internal if there is an operation in the model whose effect is equivalent to that event. In this case, the event will be represented as an event resulting from the corresponding operation. Otherwise, the event will be an external event and will be represented as the result of an external operation. The resulting BM for the example is shown in Figure 7.

3.4.2. Verification of the Behaviour Model

It is important to check that all operations belonging to the same use case are interrelated. If not, there are trigger events of unconnected operations that have not been defined explicitly. Operations that belong to the same use case are numbered starting with the same number.

The model output by this stage, the Verified BM, will be equivalent to the input model, but will not include the numbering of internal operations. Thus, the Verified BM of the example would be equivalent to the one shown in Figure 7 without the numbering.

<table>
<thead>
<tr>
<th>Method/Class</th>
<th>Report order system</th>
<th>Confirm order customer</th>
<th>Send order customer</th>
<th>Pay vendor</th>
</tr>
</thead>
</table>

3.5. Object Model and Behaviour Model Integration

This stage seeks to determine the methods of the OM classes. For this purpose, a matrix M is built, whose rows are the classes of the OM and whose columns are the internal operations of the BM. The element mij will be marked if the method j performs a basic operation on the class i. If there are several marks in the column corresponding to a method, they must all correspond to classes of the same branch of an inheritance hierarchy. In this case the method will belong to the highest class in the aforesaid hierarchy, whose subclasses will all be marked. Otherwise, the operation is not a basic operation.

Table 3 shows the matrix of classes and methods of the example. The resultant OM is shown in Figure 8, with all its components, classes, attributes, relations, cardinalities and methods.
Table 3. Classes and methods matrix

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Sales employee</th>
<th>Company</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

3.6. Verification of the Object and Behaviour Models

The models making up the conceptual modelling of the information system will be verified in this stage. For this purpose, it is necessary to check that in the OM, there is a relation between the classes whose methods are intercommunicated through events in the BM. The aforesaid classes will interchange messages through these relations.

According to this rule, no change is required in the conceptual models of the example.

The drawbacks of the method are related to the incoherence, ambiguity, inconsistency and other deficiencies in the requirements specification. The method works on the assumption that the requirements document is correct. Nevertheless, problems may be detected during the different verification tasks. Consequently, the method can be applied iteratively.

Another consideration that should be pointed out is that additional effort is required to get the Utility Languages. Nevertheless, this effort is rewarded, as the analyst can obtain the conceptual models directly from these languages.

It should be noted that the Utility Languages are expressed in natural language, and they do not restrict the meaning of the user specifications, as shown in the example. This is an important property as it facilitates communication with the user when validating these languages.

ACKNOWLEDGMENTS

I would like to thank prof. Natalia Juristo, supervisor of my PhD Dissertation, for her comments on this paper and for her support during the elaboration of the Dissertation. Also I am very grateful to prof. R. van de Riet for his help and encouragement.

REFERENCES


