Modeling Interactions using Role-Driven Patterns

Isabel Díaz[1-2] , Oscar Pastor[2] , Alfredo Matteo[1]
[1] *Universidad Central de Venezuela- Escuela de Computación*
[2] *Universidad Politécnica de Valencia (España) – Dpto. de Sistemas Informáticos y Computación*
{idiaz,opastor}@dsic.upv.es ; {idiaz,amatteo}@kuaimare.ciens.ucv.es

Abstract

Recently, great interest in model-driven approaches that are based on the automatic transformation of models has emerged. In this software development paradigm, the models and their transformations are specified at a high abstraction level supporting evolution, refinement and code generation. One of the benefits that these approaches try to achieve is improving the software quality through the reusability of well proven patterns. In order to achieve this goal, this paper presents an approach that defines patterns to transform Use Case Models into Interaction Models. This approach uses roles to capture the linguistic abstractions on the use case text as well as to describe the corresponding interaction. The roles are defined as model elements to guarantee genericity. In this way, the transformation patterns are independent of a specific domain or a technological platform. The defined patterns have been validated. Some of the validation results are also presented in this paper.

1. Introduction

This proposal treats aspects relative to two of the main activities of the Requirements Engineering [1,2]: the *specification* or precise definition of the system functional requirements and their *analysis* or interpretation. Since Jacobson coined the term "use case", many object-oriented software development methods present the functional requirements specification as a *Use Case Model* [3,4]. A use case describes the communication between an actor and a system for the exchange of information, as well as the actions that must be carried out internally by the system to respond to these information requests. During the early phases of systems development, this specification is generally expressed as a document written in natural language that can adopt several forms [5,6]. In the requirements analysis, the system behaviour described in the use cases is distributed into the *Object Model* and the *Interaction Model*. The Object Model represents the internal components of the system (objects/classes), their relationships and restrictions. The purpose of the Interaction Model is to show how these components exchange information to provide the system with the specified functionality in the use cases. RUP (Rational Unified Process), OOSE (Object-Oriented Software Engineering), FUSION and OO-Method are representative methods that follow this line of work [3,7,8,9].

In general, obtaining the interactions from the use case analysis is a complex task, because it requires a great amount of modelling experience and domain knowledge. To facilitate this process, various techniques and tools have been proposed but they are not exhaustive, precise and practical enough [3,7,8,9,10]. Consequently, the work of the analyst is very tedious and ineffective. In addition, such resources do not explicitly establish the correspondence between the text of a use case and the elements of the interaction resulting from the analysis. As the transition between both activities leaves no trace, the transition becomes irreversible. Thus, it is difficult to determine the impact on the developed system when its functional requirements change.

In this work, a solution alternative to the above mentioned problem is presented. Metamorphosis is an approach conceived to facilitate the modelling of interactions and to establish persistent relationships between the specification of the functional requirements of a system (reflected in a Use Case Model) and its analysis (expressed through an Interaction Model). This approach was created for automatic software production environments but it is also possible to take advantage of this as a manual solution. In addition, Metamorphosis is independent of the system domain, the technological aspects and the software development methods.

To achieve this, Metamorphosis has established a strategic alliance between different focuses of systems

Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering (RE'05)
0-7695-2425-7/05 $20.00 © 2005 IEEE

development: (i) it follows a *linguistic orientation* that uses natural language techniques to support the information extraction from text of a use case [11,12,13]; (ii) it is *model-centred*, and the *model transformations* are defined at high-level of abstraction to separate the system structure and behaviour from its implementation [14]; (iii) it is *role-driven* to provide special meaning to the elements and to facilitate its recognition at model-time [15]; (iv) it is based on *pattern application* to capture the transformation knowledge as reusable and generic solutions [16,17]; (v) it uses the *Unified Modeling Language* (UML), *Object Constraint Language* (OCL) and *Query/Views/Transformation* (QVT) to specify standards models and transformations [18,19].

The main purpose of this paper is to describe how Metamorphosis identifies abstractions in the texts of use cases and how it deduces the corresponding interactions, using role-driven transformation patterns. The paper has six sections. Section 2 explains the transformation architecture of Metamorphosis and how the source and target models are understood in this approach. Section 3 describes the action transformation patterns and their specification using roles. Section 4 presents a validation experience of the designed patterns. Finally, the last two sections are dedicated to the conclusions and references.

2. Transformation Architecture

Figure 1 shows the transformation model of Metamorphosis [14,19]. At the highest level (L_2), the Use Case Linguistic Model is the transformation source and the Interaction Model is the target. At the middle level (L_1), the transformation applies to a Use Case and an Interaction Unit is obtained. In the last hierarchy level (L_0), the transformation converts a Sentence into a Fragment. Each transformation level establishes a dependency relationship by abstraction. These UML relationships link elements that represent a same concept which has been expressed at different abstraction level (a dependency is denoted with a dashed arrow in Figure 1) [4]. In particular, the transformation defined at each level (System, Behaviour Unit and Action) provides a link between the specification model and the analysis model of the system behaviour. The QVT notation represents each transformation between a source model and a target model using the hexagon symbol [19]. Each model is denoted as a UML package (Fig. 1) [4].

The layer architecture also establishes dependency relationships between the transformation levels. The transformation relations form between them a hierarchical structure by integration and by refinement. In the hierarchy, the integration is a top-down relationship while that the refinement is a bottom-up relationship. However, they are complementary. The <<integrate>> and <<refine>> stereotypes differentiate the dependency relationships in Figure 1. The refinement permits the reduction of the transformation problem at action level of a use case. The refinement relationships express the convenient split in parts of a model (i.e. a use case can be broken into sentences). Each part is defined as a unit that can be analysed independently on the other parts. The integration represents the suitable combination at relationship target level of the information generated at relationship source level. The integration relationship must guarantee consistency and completeness when the results obtained at each level are combined.

The main purpose of the System Transformation is to integrate the interactions deduced for each use case at the Behaviour Unit Transformation level. The integration must resolve the possible conflicts that are generated so that all partial representations are contained or expressed in the Interaction Model. The Behaviour Unit Transformation also depends on the Action Transformation. The main goal of this level is to integrate the generated information from the sentences of a use case. In addition, it must incorporate into the obtained interaction the information that can only be deduced through either a partial or complete analysis of the use case. For example, two transformation activities performed from the use case text are: to combine the interaction fragments that are deduced from each sentence of a use case until the interaction is completed and to deduce the candidate parameters of each message. The followings subsections explain how the transformation source and target are defined in this approach.

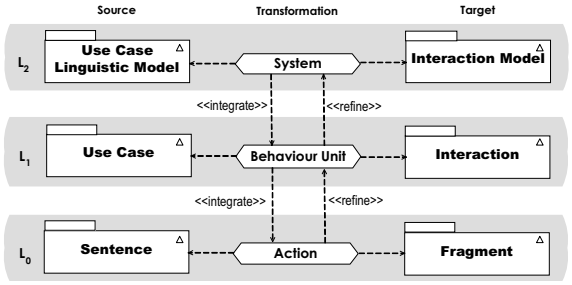


Figure 1. Transformation Model

2.1. Transformation Source

In Metamorphosis, the transformation source is the Use Case Linguistic Model. This model extends the UML concepts with the linguistic properties of a use case text [5,6]. In addition, this adaptation introduces other model elements that enrich the use case definition. The Use Case Linguistic Model specified as a profile of the UML UseCases Package in order to show the extensions carried out [4,20]. This profile focuses on three aspects which are orthogonal but complementary: the conceptual aspect, the syntactic aspect, and the semantic aspect (Fig. 2).

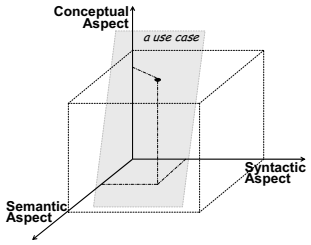


Figure 2. Use case aspects

A partial view of the Use Case Linguistic Profile is shown using UML notation in Figure 3. Each metamodel element is represented as a class by a rectangle. The bounds established among them can be extension, association, aggregation or generalization relationships. The multiplicity specifies how many connections can be established between the instances of the classes bounded by some of these relationships. The LinguisticUseCase metaclass is an extension of UseCase, a UML base metaclass (Fig. 3). Thus, the former metaclass has the properties of the last metaclass and also other properties defined into Use Case Linguistic Profile. This profile shows the *action* metaclass from the perspective of each aspect, as follows:

2.1.1. The conceptual aspect. It describes the use case metaclass and its relationship with other metaclasses. A *use case* is defined as the complete sequence of actions that the system must carry out when interacting with the *actors* to achieve a *goal*. A use case has an only goal and each goal of the system behaviour is described by an only use case. An action may express: (i) a *communication* established between the actor and the system for the exchange of information; (ii) the *behaviour* of the system, in response to the communication established with the actor. The *special actions* allow the constraint, the addition or the repetition groups of actions in a use case.

2.1.2. The syntactic aspect. A use case is viewed as a text composed by *sentences*, each of which represents an action [5]. A sentence has a syntactic structure constituted by *word* groups, according to the *grammatical function* that these groups or *phrases* fulfil in the sentence. Each syntactic structure has a *kernel* that is its principal constituent or head (i.e., the preposition is the head or kernel of a prepositional phrase). A *simple* sentence describes a non-special action. It has a single *subject* and a single *main verb*. The grammatical function of the subject may only be performed by the noun phrase that designates the actor or the system under development. The *predicate* describes what is said about them in the sentence. Generally, the predicate is expressed by noun or prepositional phrases. The main verb of the sentence must be transitive, guaranteeing the presence of a direct object in the sentence. These syntactic characteristics of a simple sentence are consequence of the action concept [6,21]. Finally, *special sentences* have a predefined structure that uses key words (i.e., '`include`', '`extend`' and '`repeat`').

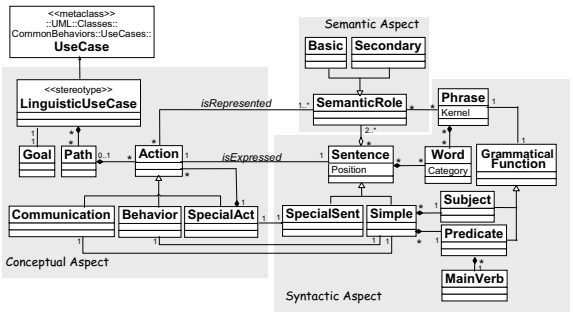


Figure 3. Linguistic use cases

2.1.3. The semantic aspect. An action is expressed in terms of roles. A *semantic role* is an abstract function that fulfils a participant of a sentence in relation to the verbal action [22,23]. The semantic roles are defined independently of the syntactical forms that the action may take. In addition, an action can be described by means of a role set. Metamorphosis has considered these properties of the semantic roles in order to use them in the recognition of the elements that participate in an action at a high-level of abstraction. Thus, the element functions do not depend on the language used to write the use case (language-independent) or on a particular modelled domain (domain-independent). The semantic roles chosen to define the transformation patterns are: (i) general or abstract, in the sense that they are not used for a specific verb; (ii) applicable to action sentences to allow the identification of interaction elements; and (iii) previously known so that they are familiar to the

stakeholders. According to these criteria, Metamorphosis uses two kinds of semantic roles: *basic* and *secondary*. The basic semantic roles always participate in transitive sentences. They can be linked to the syntactic functions of subject (`agent`) and direct object (`object`). The secondary semantic roles can be attached to the basic semantic roles. Generally, they are not requested by the verbal action. These semantic roles can be related to grammatical functions that are fulfilled by prepositional phrases and noun phrases (Table 1). Some secondary semantic roles are: `destination` (receiver/beneficiary of the action), `owner` (what/who possess), `owned` (possessed entity), `instrument` (the means used to execute an action), `state` (entity status/condition), `location` (where an action takes place), `cause` (entity that cause something) and `time` (when the action occurs). The groups of semantic roles are not closed. If other roles are required, they can be incorporated to describe functions that are not specified here.

The Action Concept
The Customer introduces the requested data into the system
The Action Syntax
The Customer [<noun-phrase>]$_{\text{subject}}$ [*introduces* <verb> *the requested data* <noun-phrase>]$_{\text{direct-object}}$ *into the system* <prepositional-phrase>]$_{\text{circumstance}}$ $_{\text{predicate}}$
The Action Semantics
The Customer <agent> *introduces the requested data* <object> *into the system* <destination>

Table 1. Action aspects: an example

2.2. Transformation Target

When a use case is transformed by Metamorphosis, an interaction is obtained. This approach assumes the interaction concept given by UML [4]. An interaction describes the exchange of messages between two or more instances. Specifically, the instances are considered as class objects, and the messages represent operations of the classes. The interactions can be graphically represented by sequence, communication or high level diagrams. The modelling concepts of the UML Interaction Package have been extended in the Interaction Structure Profile [4]. This adaptation allows the definition of the granularity of an interaction and the addition of semantics to its structure. The profile has been constructed to facilitate the interaction specification as product of the transformation. Figure 4 shows a partial view of the profile. Four metaclasses of this profile has been created by extension of UML base metaclasses, such as: (i) ModelInteraction, InteractionUnit and ActionFragment from Interations; (ii) FragmentLine from Lifeline; (iii) FragmentMessage from Message; (iv) ActorClass from Actor; and (v) BorderClass, ControlClass and EntityClass from Class. Each life line of a UML interaction represents an individual participant of an interaction. From the life lines, the messages are sent and received. A message defines a particular communication between two life lines. In Metamorphosis, a life line can play the role of an actor instance or of a class of type: boundary, entity and control. These types of object classes were proposed by Jacobson in [3]. Several interaction modelling techniques use these types of classes in order to construct sequence diagrams [3,7,8].

2.2.1. The granularity of an interaction. Depending on the transformation source, an interaction granule can be: a *fragment* (from action), an *interaction unit* (from use case) or an *interaction model* (from Use Case Linguistic Model). Each granule is delimited by the specification type that gives rise to it. In addition, if a specification type depends on another specification type, the interaction granules generated are dependent on each other. A fragment is obtained when an action of a use case is transformed. In UML terms, a fragment is an interaction that is part of another interaction [4]. It may consist of one or more messages and of one or more instances that send and receive these messages. In our approach, the fragment term is reserved to interactions obtained from actions. When a use case is transformed, an interaction unit (or, simply, an interaction) is deduced. Each interaction unit is the combination of the fragments corresponding to all actions that belong to a complete path of a use case. Thus, it is possible to deduce one ore more interaction units from a use case (one for each complete path or scenario). The fragment combination is achieved by the strict-sequence operator defined by UML [4]. This operator guarantees that the fragments union is carried out in the order of appearance that the actions have in the use case. Finally, a complete and consistent interaction model is obtained at system level. This model integrates the information of all the interactions obtained from the use cases of the system.

2.2.2. The interaction semantics. *Roles* are used to specify interaction structures [15,17]. These roles describe the semantic properties of the fragment participants. In Metamorphosis, when an action fragment is described by roles it is called a semantic fragment. A role is an element type of an interaction which has a special meaning in a fragment. For example, the `initiator` role is a type of life line that is responsible for sending the first message of a fragment. In addition,

this message stimulates the receipt instance to send other messages. The `initiator` role can not be played by an actor life line. The interaction roles enable to model generic fragments and to enrich them with a specific semantics.

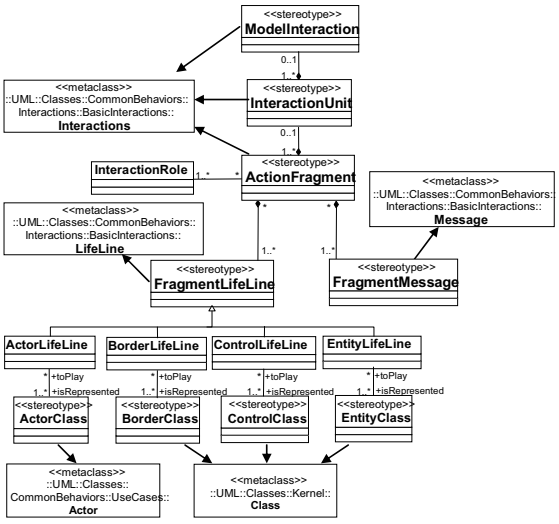


Figure 4. Interaction Structure Profile (view partial)

3. Role-Driven Transformation Patterns

The key point of the Metamorphosis approach is the action transformation. All the levels of the transformation hierarchy depend on the results obtained when a use case sentence is converted into an interaction fragment (Fig. 1). The action transformations have been specified by means of patterns [16]. The purpose of these patterns is to capture the transformation knowledge and reuse them suitably. To describe the action transformation patterns, the UML-Based Pattern Specification Technique was applied [17,24]. This technique is based on the specialisation of UML metamodels. The specialisation is carried out by roles that describe the participants of a pattern. A role defines a subtype of a metamodel class or base metaclass. The roles specify the properties that a model element must have in order to be part of the pattern solution model. In Metamorphosis, the specializations with roles are fulfilled from the Use Case Linguistic Profile and the Interaction Structure Profile.

Figure 5 shows the action transformation model that is applied by Metamorphosis. This model details the last level of the transformation hierarchy that was described in Section 2 (Fig. 1). The action transformation model describes how an action of a use case can be specified using patterns. At model level, the transformation patterns are specified from a semantic perspective. At instance level, these patterns are described from a syntactic perspective. At this level, the patterns can be applied to a particular action in order to obtain a particular interaction fragment. UML dependency relationships have been established in the action transformation model [4]. These relationships show how an immediate upper level is instantiated. The semantic specification of the transformation patterns is carried out at model level. To achieve this, the source and target metamodels are instantiated. The result of the instantiation from the Use Case Linguistic Model is a semantic context. This is a structure that describes a sentence using semantic roles. Similarly, the instantiation from the Interaction Model produces a semantic fragment that is also described by roles. The transformation rules of the patterns are defined at model level. These rules are applied to semantic roles to obtain the corresponding semantic fragments.

At model level, the specification of the transformation patterns is generic, domain-independent and implementation-independent. The patterns are also independent of the language used to write the sentences and their syntactic structure. The semantic fragments are independent of the graphic representation style. Finally, to obtain the wanted solution, the pattern must be applied. This is carried out by instantiation of the specified pattern. Thus, a particular sentence is transformed to a specific interaction fragment. Figure 6 describes the behaviour of a role when the instantiation occur. The `initiator` role is a subtype of the `BorderLifeLine` metaclass from Interaction Structure Profile (see Section 2.2.2). This role is defined at the model level or pattern specification level.

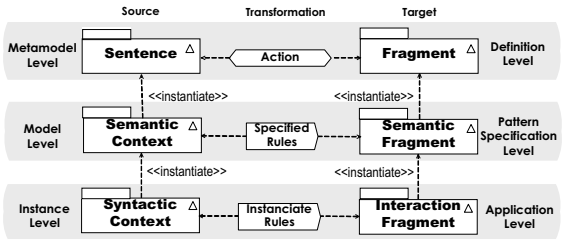


Figure 5. Action Transformation Model

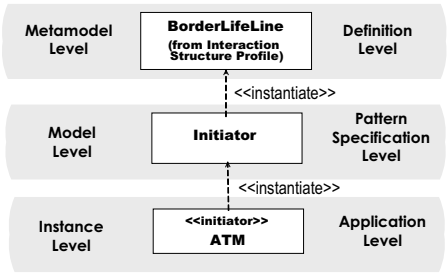


Figure 6. Instantiation of a role

The format of the each action transformation pattern has six elements [16]: (i) *name* identifies the action transformation pattern and distinguishes it from others; (ii) *description* gives a concise explanation about the pattern; (iii) *observations* shown specific patterns, application examples and/or several annotations. The following subsections describe the remaining elements of an action transformation pattern: (iv) *context semantic*; (v) *semantic fragment*; and (vi) *transformation rules*. In addition, an example of the instantiation of a transformation pattern is presented.

3.1. Semantic Context

Each action transformation pattern is applicable to a semantic context. A semantic context describes the features of a sentence in terms of semantic roles and the relationships established among these. It allows the expression of the abstract properties that its components perform with regard to the verbal action, in a generic way, independently of the syntactic structure of the sentence (see Section 2.1.3). A semantic context defines a family of sentence types.

In Metamorphosis, a *semantic context* (SC) is specified using: (i) a *name* that identifies the semantic contex; (ii) a *description* in natural language; (iii) a logic formula that formally describes the SC; and (iv) a *graphic representation* that shows the role-driven specialization of the Use Case Linguistic Profile. Table 2 shows a formula that describes the Owning Chain Semantic Context. Its logic formula is specified as $SC=<\alpha, \mu, \psi>$, where α and μ are sets of variables and constants respectively, and ψ is a set of functions applicable to SC terms. These functions have two objectives: (i) to determine which semantic roles participate in the sentence, their properties and the relationship types established among them; (ii) to establish the conditions that must be met by a sentence to be transformed into the corresponding interaction fragment. If these conditions are fulfilled, then the transformation rules can be applied. In addition, a informal description of the Owning Chain SC is also presented.

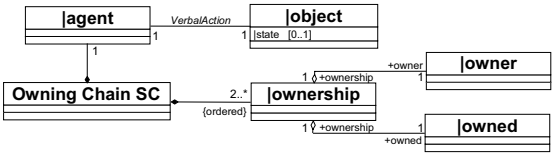
Owning Chain Semantic Context
$\forall V,A,O,St,Or,Od:$ $\exists n>1$ / NumberOf(Ownership)=n $\wedge$ Action(Verb:V,Agent:A,Object:O,Destination:_)$\wedge$ (StateSet(Object:O,State:St)$\vee$ StateSet(Object:O,State:?))$\wedge$ ($\forall i=1..n$ $Ownership_i$($Owner_i$:Or,$Owned_i$:Od)$\wedge$ $Owned_i$:O $\wedge$ $Owned_i$:$Owner_{i-1}$);
Description
This context describes actions that have an active entity that initiates or controls the action (the agent) and a passive entity on which such action falls (the object). The action does not have a destination role. The state change undergone by the object (as a consequence of the action performed) may or may not have been explicitly expressed in the sentence. The simple sentence must satisfy more than one Ownership relationship between an owner entity and an owned entity. The NumberOf function is used to determine the quantity of Ownership relationships that are present in the sentence.
Graphic Representation


Table 2. Semantic context of a transformation pattern

Finally, Table 2 shows a partial view of the graphic representation of the Owning Chain SC. The roles that participate in the semantic context and its restrictions are presented in a UML class diagram. The notation of roles is based on the Metarole-Based Modeling Language (RBML) [17,24]. To indicate that the classes of the diagram represent roles, the <<role>> stereotype was introduced. Other alternative is using a straight slash placed in front of the role name ("|roleName"). The Owning Chain diagram indicates that the agent and object roles only can participate once in the semantic context specification. The state role is an attribute of the object role which may or may not to be present in the context. It must have two or more Ownership relationships established in the sentence.

If a simple sentence satisfies the conditions described in Table 2, then it can be inferred that its semantic

context is the one described by the Owning Chain SC. Table 3 shows an example in which this situation occurs.

'The Sales System calculates the total amount of the restaurant customer bill'
NumberOf(Ownership)=3 ∧ StateSet(Object:'the total amount',State:?) ∧ Action(Verb:'calculates', Agent:'The Sales System', Object:'the total amount') ∧ $Ownership_1$($Owner_1$:'of the bill', $Owned_1$:'the total amount') ∧ $Ownership_2$($Owner_2$:'of the customer', $Owned_2$:'of the bill') ∧ $Ownership_3$($Owner_3$:'of the restaurant', $Owned_3$:'of the customer') ∧ $Owned_1$:'the total amount' ∧ $Owner_1$:$Owned_2$ ∧ $Owner_2$:$Owned_3$;

Table 3. Evaluation of a semantic context

3.2. Semantic Fragment

Other element of a pattern is the description of the semantic fragment that is obtained by transformation. The result of the transformation of a sentence is a fragment. This fragment is part of the complete interaction that describes the use case to which the sentence belongs. A semantic fragment (SF) is a generic structure that specifies a particular form of interaction. The action transformation patterns present their semantic fragments by means of a sequence diagram since it is easy to understand. This representation type is well-known by the stakeholders and its use is not restrictive. The corresponding role model is also presented [17,24]. This role model describes the participants of a semantic fragment. An interaction role specifies the properties that a modelled element of an Interaction Structure Profile must have to form part of a fragment. Thus, an interaction role is a subtype of a metaclass of this profile.

Figures 7A and 7B present the specification of the Domino Effect SF. Its graphic representation is presented in Figure 7A. The expression |init:|Initiator indicates that the lifeline role init is played by an instance of an Initiator class. Similarly, the lifeline role brid[i] is played by the i^{th} Bridge instance in the set of bridges; the lifeline role perf is played by an instance of a Performer class. In the interaction fragment, the messages are synchronic. A first message is sent by an Initiator instance to a Bridge instance. Attach represents an operation type that relates a sender instance to a receiver instance. This message induces the receiving instance to send another message to another Bridge instance and so on, until a Bridge instance sends a message to a Performer instance. This message activates the Update operation that changes the Performer instance state.

A partial view of a specialized Interaction Structure Profile is displayed in Figure 7B. The brid and perf are fragment lifelines that represent entity classes in the system domain. init is a fragment lifeline that represents a border class or a control class. There are two kinds of messages in the Domino Effect Interaction SF: Attach and Update. Update appears in the fragment only once, whereas Attach is applied at least twice. Furthermore, the Attach operation has no parameters, but the Update operation may have them.

3.3. Transformation Rules

The transformation rules describe how the participants of a semantic context are turned into elements of a fragment. The rules of an action transformation pattern are presented in Table 4. These rules are applicable to sentences whose semantic context is described by the Owning Chain SC (Table 2). By applying these rules, a fragment is obtained with the generic form of the Domino Effect SF (Figures 7A and 7B). The left side of a rule corresponds to the fragment elements. The right side of a rule indicates how to identify these elements. The rules recognize roles in the semantic context to identify the interaction elements. To do this, some functions are applied on the semantic roles.

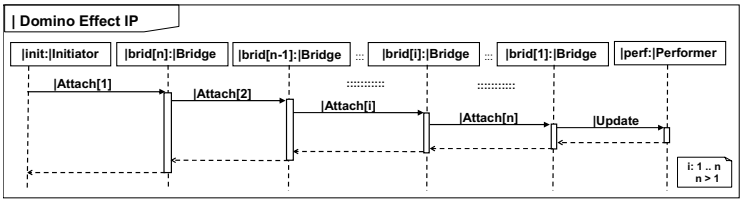


Figure 7A. Graphic representation of a semantic pattern (partial view)

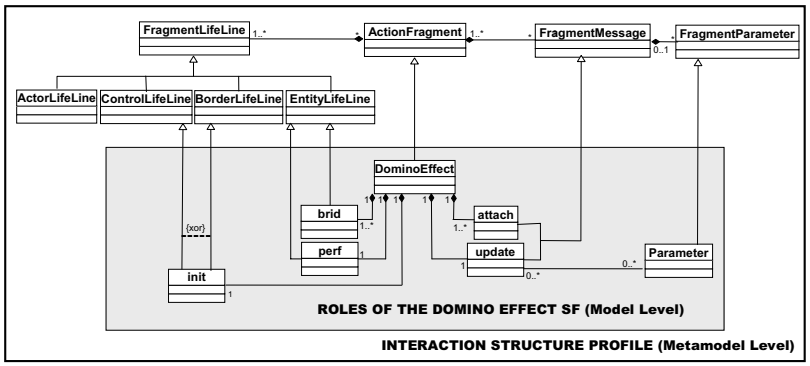


Figure 7B. Roles model of a semantic fragment (partial view)

DominoEffectIP ← OwningChainSC		
|init:|Initiator	←	Agent
|brid[i]:|Bridge	←	<Kernel(Owner-j)>^{Norm} ∀i=1..(n-1) ∧ ∀j=(n-1)..1
|perf:|Performer	←	<Kernel(Owner-1)>^{Norm}
|Attach[i]	←	Owner-j ~ Kernel(Owner-j) ∀i=1..(n-1) ∧ ∀j=n..1
|Update	←	Sequence(Verb,<Object>^{Norm}, State) ∨ Sequence(Verb;<Object>^{Norm})

Description

The Initiator instance is the Agent of the sentence. The Performer instance is deduced from the Owner kernel contained in the first Ownership relationship of the use case sentence. The Kernel function extracts the most important constituent of a semantic role. Each Bridge instance is extracted from the kernel of the remaining Owner instances. To do this, the Owners are considered in inverse order of appearance. The i^{th} Attach operation is obtained using the j^{th} Owner constituents as follows: the difference function (~) takes all the components from the Owner that does not belong to its kernel. The signature of the Update operation is deduced by a Sequence function. This function constructs a label with the principal Verb, the Object, and the Object State (if it is explicit in the sentence). The normalization function allows the expression of the canonical form of the role kernels.

Table 4. Tranformation Rules: An Example

3.4. Instantiating Transformation Patterns

The instantiation of an action transformation pattern allows going from the model level to the instance level (Figure 5). This process supposes that the action has been enriched with its semantic and syntactic information (roles and grammatical structure). Basically, the instantiation process can be performed in three steps. The *first step* is to recognize the semantic context of the sentence that is to be transformed. This is done by comparing the semantic context of the sentence with those that belong to the specified transformation patterns. This consists in evaluating all the semantic contexts of the patterns until that one of these evaluations is satisfied. The *second step* is to identify the transformation rules. This is immediate because each semantic context defines an action transformation pattern. The *third step* is to apply these rules. When this is done, an interaction fragment is obtained. To illustrate the instantiation of a transformation pattern, the sentence given in Table 3 will be considered. This sentence satisfies the `Owning Chain Semantic Context`. The pattern that corresponds to this semantic context has the transformation rules given in Table 4. The corresponding interaction fragment was also described in Figures 7A and 7B. However, to carry out the pattern instantiation the syntactic structure of the roles that participate in the sentence must be known. Thus, the `Agent` and `Object` roles are linked to noun-phrases. The `Owner` and `Owned` roles are expressed by prepositional-phrases.

In the example sentence, the obtained interaction contains three domain instances: `restaurant`, `customer` and `bill` (Fig. 8). These objects were recognized from the `noun-phrase(kernel)` content in each `of-prepositional-phrase (Owner)`. The canonical form of these noun phrases was verified. The name given to the border instance (`Initiator`) was extracted from the `Agent` (`SS: Sales System`). The three synchronous messages that were sent and received by these instances were identified. The order of the messages was reversed with regard to the order of appearance of the `of-prepositional-phrase` elements in the sentence. The second and first messages allowed the referencing of their respective receiving instances (`restaurant` and `customer`). The third

message was responsible for activating the operation execution identified as `calculates the total amount` in the `bill` instance (a `performer` instance).

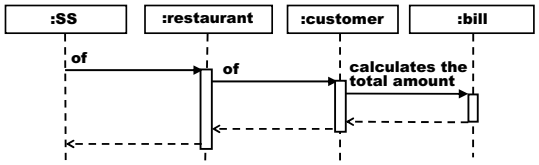


Figure 8. Fragment obtained by Metamorphosis

4. Validation of Transformation Patterns

One important goal of our work is to create a catalogue of well-proven patterns that can be integrated into several frameworks as support to the modelling interactions. The first step given in this direction was given. This task consisted of designing the transformation patterns based on the direct observation of a sample of sequence diagrams. These diagrams were obtained from the use cases of some academic and commercial information systems. A matching between sentences and fragments was made.

Then, an activity was devised to validate the specified patterns [13]. The validation process was carried out both manual and automatically. To do this, the Use Case Model and the Interaction Model of several study cases were constructed (without applying the patterns). For each system, both models were exhaustively revised by expert analysts in order to reach a consensus on the results obtained manually. The automatic validation was supported by a tool developed by way of a prototype. This tool was integrated into OO-Method, an automatic software production environment that supports requirements specification and analysis [9,25]. The prototype developed uses a parser and a grammar to transform sentences into fragments. Later, it combines these fragments to obtain the corresponding sequence diagram.

The sequence diagrams generated using the prototype were then compared with the sequence diagrams obtained manually to determine differences and similarities. This task was carried out by the group of experts. Two-hundred eighty use cases were analyzed with a total of 3524 sentences, only 412 of these sentences were special (conditionals, iterations, etc.). The manually obtained fragments were compared with the fragments generated automatically for each sentence of the use cases. The main objective of this comparison was to establish whether the automatically generated fragments were the ones expected by the analysts. This implied determining whether both fragments were equal, equivalent or different. We considered two fragments to be equal when they were composed of the same instances and the same messages. Two fragments to be equivalent if both represented the same interaction, even though the instances and messages were not the same. If the fragments were neither equal nor equivalent, we considered them as different. Using these criteria, 65% of the transformation patterns were equal, 28% were equivalent and only 7% were categorized as different. This experience allowed us to establish which of the transformation patterns had to be improved or rejected. It was also possible to identify new transformation patterns of semantic contexts that were not considered by the ones designed initially.

Similar validation experiments are being replicated. Currently, the catalogue of Metamorphosis contains 25 validated patterns to transform use case actions written in Spanish. The strategy has allowed us to establish the limitations of the transformation patterns designed initially and then to improve and enrich them.

5. Conclusions

This paper describes the Metamorphosis transformation strategy. A transformation is applicable at three levels: at the system level, at the use case level, and at the action level. The action level is the key to the defined strategy. The action transformation is specified by patterns. A transformation pattern describes how to transform semantic contexts into semantic fragments. The transformation of a sentence is based on the use of a set of catalogued patterns. The result of transforming a sentence is an interaction fragment. The convenient combination of all the fragments, deduced from the sentences of a use case, allows obtaining a complete interaction diagram.

A transformation, at action level, establishes a correspondence between the semantic roles participating in a sentence and those performed by the elements of an interaction. The semantic roles fulfil three important functions in the transformation strategy: (i) they enable the recognition of the elements to be transformed according to their performance in a sentence and not by their acquired grammatical structure; (ii) they add semantics to the elements of an interaction without linking them to a particular domain; (iii) they allow the expression of the transformation rules at a high level of

abstraction; (iv) the pattern specification does not depend on the language used to write the use cases or on the resources used to model the obtained interaction.

Lastly, this paper briefly describes an experiment designed to validate the transformation patterns. Although the results can be considered positive, more replicas of this experiment should be performed. Presently, the research work follows three lines of action. The first line attempts to establish a permanent validation strategy of the catalogue of the transformation patterns, with the purpose of guaranteeing its timely tailoring and expansion. This endeavours to enable identifying the new semantic contexts, design their corresponding semantic fragment, and verify that such correspondence is valid. A second line of action is oriented towards complementing the transformation patterns with structural information. It seeks to expand the scope of these patterns to also allow deducing the object model related to the sentence. The third line of action is related to the design of an evolution strategy of the models obtained. It seeks to guarantee the consistency, when these models undergo changes, as well as the traceability between the different elements composing them.

6. References

[1] Van Lamsweerde A., *Requirements Engineering in the Year 2000: A Research Perspective*, Proc. of the 22nd Conference on Software Engineering (ICSE 2000), pp. 5-19. ACM Press.

[2] Nuseibeh B., Easterbrook S., *Requirements Engineering: A Roadmap*, Proc. of the 22nd Conference on Software Engineering (ICSE 2000), Conference on The Future of Software Engineering, pp. 37-46. ACM Press.

[3] Jacobson I., Christerson M., Jonsson P., Övergaard G., *Object-Oriented Software Engineering. A Use Case Driven Approach*, Addison-Wesley, 1992.

[4] Object Management Group, *Unified Modeling Language: Superstructure Specification*, Version 2.0, August 2003, http://www.omg.org/uml.

[5] Rolland C., Ben-Achour C., *Guiding the Construction of Textual Use Case Specifications*. Data & Knowledge Engineering 25(1998), 125-160. Elsevier Science.

[6] Díaz I., Losavio F., Matteo A., Pastor O., *A Specification Pattern for Use Cases*, Information & Management, Vol. 41/8 (2004), pp. 961-975, Elsevier Science B.V.

[7] Rational Software Corporation®, *The Rational Unified Process (2003)*, http://www.rational.com/rup.

[8] Coleman D., Arnold P., Bodoff S., Dollin Ch., Gichrist H., *Object-Oriented Development. The Fusion Method.* Prentice-Hall, Object-Oriented Series, 1994.

[9] Insfrán E., Pastor O., Wieringa R., *Requirements Engineering-Based Conceptual Modeling*, Requirements Engineering, 7(2), 61-72, Springer-Verlag, March 2002.

[10] Regnell B., *Requirements Engineering with Use Cases: a Basic for Software Development*. PhD Thesis, Dept. of Communication Systems, Lund University, Sweden, 1999.

[11] Burg J.F.M., Van de Riet R.P., *Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modeling*, Proc. of the 2nd International Workshop on Applications of Natural Language to Information Systems (NLDB'96), The Netherlands, 1996.

[12] Flield G., Kop C., Mayerthaler W., Mayr H., Winkler C.. *Linguistic Aspects of Dynamics in Requirements Specifications*, IEEE Proc. of the 11th International Workshop on Databases and Expert Systems Applications (DEXA 2000), LNCS Springer-Verlag, pp. 83-90.

[13] Díaz I., Moreno L., Fuentes I., Pastor O., *Integrating Natural Language Techniques in OO-Method*, Proc. of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2005), LNCS Springer-Verlag, February 2005.

[14] Object Management Group, *MDA Guide*, Version 1.01. Jun 03. http://www.omg.org/uml.

[15] Kim D.K., France R., Ghosh S., Song E., *A Role-Based Metamodeling Approach to Specifying Design Patterns*, Proc. of 27th IEEE Annual International Computer Software and Applications Conference (COMPSAC), Texas, 2003.

[16] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns. Elements of Reusable Object-Oriented Software,* Professional Computing Series, Addison-Wesley, 1992.

[17] France R., Kim D., Ghosh S., Song E., *A UML-Based Pattern Specification Technique*, IEEE Transactions on Software Engineering, 30(3), March 2004, pp. 193- 206.

[18] Object Management Group, *OCL 2.0*, October 2003. http://www.omg.org/uml.

[19] Tata Consultancy Services, *Query/Views/Transformation RFP*, Version 1.1, August, 2003, http:// qvt.org.

[20] Nakatani T., Urai T., Ohmura S., Tamai T., *A Requirements Description Metamodel for Use Cases*, IEEE Proc. of the 8th Asia-Pacific Software Engineering Conference (APSEC 2001), pp. 251-258.

[21] Ben Achour C., Rolland C., Maiden N.A.M., Souveyet C, *Guiding Use Case Authoring: Results of an Empirical Study*, Proc. of the Fourth IEEE International Symposium on Requirements Engineering (RE 1999), pp. 36-43.

[22] Fillmore Ch., *The Case for Case*, In Universals in Linguistic Theory, ed. By Bach & Harms. New York: Holt, Rinehart & Winston, 1968.

[23] Gildea D., Jurafsky D., *Automatic Labeling of Semantic Roles*, Computational Linguistics, 28(3): 245-280, 2002.

[24] France R., Ghosh S., Song E., Kim D., *A Metamodeling Approach to Pattern-Based Model Refactoring*, IEEE Software, Special Issue on Model Driven Development, Vol. 20, Nº 5, Sep/Oct 2003, pp. 52- 58.

[25] Pastor O., Gómez J., Insfrán E., Pelechano V., *The OO-Method Approach for Information Systems Modeling: from Object-Oriented Conceptual Modeling to Automated Programming*, Information Systems 26 (2001): 507-534.