Rigorous description of the syntax and semantics of UML Collaborations

María Agustina Cibrán – Vanesa Mola – Claudia Pons – Wanda Marina Russo[1]

Lifia, Universidad Nacional de La Plata-Calle 50 esq.115, 1er.Piso, (1900) La Plata, Buenos Aires, Argentina

Abstract

The specification of the UML in general, and the specification of Collaboration Diagrams in particular, is semi-formal. This lack of precise semantics can lead to several problems such us different interpretations, ambiguities, etc.
In this paper, we propose a formalization of the syntax and semantics of Collaboration diagrams in the formal specification language Object-Z.
During this formalization process, we discovered inconsistencies and ambiguities, which motivated the discussion of some improvement ideas that will be presented in this document.

1. Introduction

It is commonly accepted that a language needs a formal specification. The main motivation for formalization is that users of the language need to understand each other precisely. The lack of accuracy in the definition of the language can cause problems regarding the models expressed in the language, such us different interpretations, ambiguities, etc.
Furthermore, tools like code generators and consistency checkers require that syntax and semantics of the language to be precise.
The Unified modeling Language, UML (1999) is a standard language for modeling and specifying object-oriented systems. The language consists of a set of constructs common to most object-oriented modeling languages.
The specification of UML in general, and the specification of *Collaboration Diagrams* in particular, is semi-formal, i.e. certain parts of it are specified with well-defined languages while other parts are described informally in natural language.
A number of formal semantics of Collaboration in the UML have already been investigated: Araújo (1998) translates a subset of UML sequence diagrams into temporal logic formulas. Gehrke et al. (1998) sketches a translation of UML collaboration diagrams to Petri nets, Knapp (1999) proposes a formal semantics of interactions using temporal logic formulas, Övergaard (1999) gives a formal definition of the collaboration construct in the UML, the UZ-translator (Waldoke et al. 1998) produces Object-Z specifications from Collaboration diagrams.
In this paper, we propose a formalization of the syntax and semantics of Collaboration diagrams in the formal specification language Object-Z. A collaboration diagram may be presented at two different levels: specification level (syntax) or instance level

---

* Authors appear in alphabetical order

(semantics). In our formalization we take into account both levels of abstraction. Moreover, we provide a function (*sem*) that maps a Collaboration into its semantic domain.
During this formalization process, we discovered inconsistencies and ambiguities, which motivated the discussion of some improvement ideas that will be presented in this document.

2. Formalizing the syntax of Collaborations

A Collaboration defines a specific way to use the Model Elements in a Model. It specifies the concepts needed to express how different elements of a model interact with each other from a structural point of view. Also, a Collaboration describes how different kinds of Classifiers and their Associations are to be used in accomplishing a particular task. The same Classifier or Association can appear in several Collaborations, and several times in one Collaboration, each time in a different role. In each appearance it is specified which of the properties of the Classifier or the Association are needed in that particular usage. These properties are a subset of all the properties of that Classifier or Association.
An Interaction defined in the context of a Collaboration specifies the details of the communication that should take place in accomplishing a particular task. A communication is specified with a Message, which defines the roles of the sender and the receiver Instances, as well as the Action that will cause the communication.
We give a formalization in Object-Z of each of the model elements that conforms the syntax of a Collaboration, including its attributes, associations and well formedness rules. In what follows, we show some examples of the formalized classes. For a full description of the formalization refer to www-lifia.sol.info.unlp.edu.ar/~cpons/CollaborationReport.html.

ClassifierRole
A collaboration is not actually defined in terms of classifiers, but ClassifierRoles. A ClassifierRole is a specific role played by a participant in a Collaboration. It defines a projection of a Classifier. A ClassifieRole specifies a restricted view of a Classifier, defined by what is required in a collaboration. It also may have several base Classifiers. In the metamodel, a ClassifierRole specifies one participant in a Collaboration. It defines a set of Features and a set of ModelElements, which are subsets of those available in the base Classifiers, that are used in the role. Given the fact that a ClassifierRole is a kind of Classifier, a Generalization relationship may be defined between two ClassifierRoles.

ClassifierRole

Classifier

multiplicity: Integer
availableContents: ℙ *ModelElement*
availableFeatures: ℙ *Feature*
base: ℙ *Classifier*

---

allAssociationRoles: ℙ *AssociationRole*
∀ *ar: AsociationRole* •
((Ö c ³ ar.connections ˣ c.type = self) Û ar ³ allAssociationRoles)

allAvailableFeatures: ℙ *Feature*
∀ *p* ∈ *self.parents* •
(allAvailableFeatures= availableFeatures ¿ self.allAvailableFeatures)

allAvailableContents : ℙ *ModelElement*
∀ *p* ∈ *self.parents* •
(allAvailableContents= availableContents ¿ self.allAvailableContents)

---

[1] ∀ *ar* ∈ *allAssociationRoles* •
(Ö c ∈ *self.base* • *(Ö a* ∈ *c.allAssociations* • *ar.base = a))*

[2] (∀ *f* ∈ *allAvailableFeatures* • *Ö c* ∈ *self.base* • *f* ∈ *c.allFeatures)* ∧
(∀ *f* ∈ *allAvailableContents* • *Ö c* ∈ *self.base* • *f* ∈ *c.allContents)*

[3] isEmpty(self.allFeatures)

[4] (self.generalization.parent <> null) Û
(Õa ³ self.generalization.parent.allFeatures ˣ Ö c ³ self.base ˣ
a ³ c.allFeatures)

The Object-Z schema above describes the following Attributes, Associations and Well formedness rules:
multiplicity refers to the number of Instances playing this particular role in the Collaboration.
availableContents represents the subset of ModelElements contained in the base Classifier which is used in the Collaboration.
availableFeatures represents the subset of Features of the base Classifier which is used in the Collaboration.
base is the Classifier which the ClassifierRole is a view of.

[1] The AssociationRoles connected to the ClassifierRole must match a subset of the Associations connected to the base Classifiers.
[2] The features and contents of the ClassifierRole must be subsets of those of the base Classifiers.
[3] A ClassiferRole does not have any Features of his own.
[4] If two ClassifierRoles is a specialization of another one, all the features contained in the parent must be included in at least one of the Classifiers that conform to the base of the child ClassifierRole.

Collaboration

In UML, a Collaboration describes how an Operation or a Classifier is realized by a set of Classifiers and Associations. It defines the communication between the objects involved in that Collaboration, specifying a view of a model of Classifiers.
The schema Collaboration inherits from Namespace, which means that it can contain ModelElements with a name designating a unique element within it. Collaboration also inherits from GeneralizableElement because it may specify a task, which is a specialization of the task of another Collaboration.

Collaboration

GeneralizableElement
NameSpace

constrainingElements: ℙ *ModelElement*
interactions: ℙ *Interaction*
ownedAssociationRoles: ℙ *AssociationRole*
ownedClassifierRoles: ℙ *ClassifierRole*
ownedGeneralizations: ℙ *Generalization*
representedClassifier: Classifier
representedOperation: Operation

[1] (representedClassifier = null) xor (representedOperation = null)

[2] $\forall$ *a* $\in$ *ownedAssociationRoles* $\bullet$ ($\exists$ *aend* $\in$ *a.associationEndRoles* $\bullet$
($\ddot{O}$ *c* $\in$ *ownedClassifierRoles* $\bullet$ (*aend.base.classifier* $\in$ *c.base*)))

[3] $\forall$ *o* $\in$ *ownedAssociationRoles* $\bullet$ (*o.base.namespace = self.namespace*) $\wedge$
$\forall$ *c* $\in$ *ownedClassifierRoles* $\bullet$ ($\forall$ *b* $\in$ *c.base* $\bullet$ *b.namespace = self.namespace*)

[4] $\forall$ *c* $\in$ *constrainingElements* $\bullet$ (*c.base* $\in$ *self.namespace.ownedElements*)

[5] $\forall$ *c* $\in$ *ownedAssociationRoles* $\bullet$ (*c.name = null* $\triangleright$
($\neg$ ($\exists$ *d* $\in$ *ownedAssociationRoles* $\bullet$ (*d.base = c.base* $\wedge$ *d* $\neq$ *c*))) $\wedge$
$\forall$ *c* $\in$ *ownedClassifierRoles* $\bullet$ (*c.name = null* $\triangleright$
($\neg$ ($\exists$ *d* $\in$ *ownedClassifierRoles* $\bullet$ (*d.base = c.base* $\wedge$ *d* $\neq$ *c*)))

$$
\begin{aligned}
&[6]\ \forall\, r \in ownedAssociationRoles \bullet (\exists\, r2 \in \\
&self.generalization.parent.ownedAssociationRoles \bullet \\
&\qquad (r.name = r2.name \ \Rightarrow\ r2 = r1 \vee r2 \in r1.allParents)) \ \wedge \\
&\quad \forall\, r \in ownedClassifierRoles \bullet (\exists\, r2 \in \\
&\qquad self.generalization.parent.ownedClassifierRoles \bullet \\
&\qquad\qquad (r.name = r2.name \ \Rightarrow\ r2 = r1 \vee r2 \in r1.allParents))
\end{aligned}
$$

$$
\begin{aligned}
&[7]\ (representedClassifier \neq null) \ \Rightarrow \\
&\quad \forall\, i \in interactions \bullet (i.messages[0].receiver.base = self.representedClassifier)
\end{aligned}
$$

$$
\begin{aligned}
&[8]\ (\forall\, gr \in self.generalization.parent.ownedClassifierRoles \bullet \\
&\ (gr \in ownedClassifierRoles \vee \exists\, o \in ownedClassifierRoles \bullet gr \in o.allParents))
\end{aligned}
$$

$$
\begin{aligned}
&[9]\ (\forall\, parentInt \in self.generalization.parent.interactions \bullet \\
&\qquad (\exists\, int \in interactions \bullet parentInt.messages \ ^{o}\ int.messages))
\end{aligned}
$$

The Object-Z schema above describes the following Attributes, Associations and Well formedness rules:
constrainingElements represents a set of objects that add extra constraints to the Collaboration.
interactions contains the interactions defined in the Collaboration.
ownedAssociationRoles and *ownedClassifierRoles* define the sets of AssociationRoles and ClassifierRoles involved in the Collaboration.
ownedGeneralizations represents the generalizations between them.
representedClassifier or *representedOperation* refers to the ModelElement being represented by the Collaboration.
[1] The Collaboration represents either a Classifier or an Operation and for that reason only one of the variables representedClassifier and representedOperation is not null.
[2] Every AssociationRole associates only ClassifierRoles that are included in the Collaboration.
[3] All ClassifierRoles and AssociationRoles in the Collaboration must be associated to Classifiers and Associations in the Namespace owning the Collaboration.
[4] All constraining elements that restrict a Collaboration must be contained in its same NameSpace.
[5] If two ClassifierRoles or AssociationRoles have no name within the Collaboration then they have different base.
[6] A role (AssociationRole or ClassifierRole) with the same name as that of one in a parent of the Collaboration must be a specialization of that role.
[7] All Interaction Diagrams within the Collaboration (in the case of representing a Classifier) begin with a message sent to the representedClassifier. We added this restriction so as to impose an order among messages.

[8] A Collaboration specializing another one must include all the ClassifierRoles contained in the parent Collaboration (possibly specialized).
[9] A Collaboration specializing another one must contain at least all the messages that are present in the parent among its interactions.

3. Formalizing the semantics of Collaborations

3.1. Formal definition of the semantic domain

A Collaboration may be presented at two different levels: specification level (syntax) or instance level (semantics).
A diagram presenting the collaboration at the specification level will show ClassifierRoles and AssociationRoles, while a diagram at the instance level will show Instances and Links conforming to the roles in the collaboration.
Note that there is a clear correspondence between the metaclasses at the syntax level and the metaclasses at the semantics level; that correspondence represents a sort of instantiation relation. For example, Association and Link, Classifier and Instance, etc.

Instance
An Instance defines an entity to which a set of operations can be applied and which has a state that stores the effects of the operations.
In the metamodel, an Instance is connected to at least one Classifier, which defines its structure and behavior.
The current state of the instance is implemented by a set of attribute values and a set of links. Both sets must match the definitions of its Classifiers. Instance is an abstract metaclass.

Instance

ModelElement

slots: ¡ AttributeLink
linkEnds: ¡ LinkEnd
classifiers: ¡ Classifier
persistent: (transitory, persistent)

opossiteLinkEnds: Instance § ¡ LinkEnd

opossiteLinkEnds (instance) =
{linkEnd | Õ l:Link ×
Ö le ³ l.connections ×
instance ³ le.instances ∧ linkEnd ³ (l.connections – {le})

[1]Õ s ³ slots × (Ö c ³ classifiers × (s.attribute ³ c.allAttributes))
[2]Ö s1, s2 ³ slots × (s1.name = s2.name ⇒ s1 = s2)
[3]Õ c ³ classifiers × (Õ a ³ c.allAtributes × (Ö s ³ slots × (s.attribute = a)))
[4]Õ le ³ linkEnds × (Ö c ³ self.classifiers × le.associationEnd ³

c.associationEnds)
[5]Õ s ³ slots ˣ Õ le ³ oppositeLinkEnds(self) ˣ a.name ≠ le.name

The Object-Z schema above describes the following Attributes, Associations and Well-formedness rules:
persistent (tagged value) denotes the permanence of the Instance state, marking it as transitory, which means that its state is destroyed when the Instance is destroyed, or persistent, i.e., its state is not destroyed when the Instance is destroyed.
slots represents the set of AttributeLinks that holds the attribute values of the Instance.
linkEnds is the set of LinkEnds of the connected Links that are attached to the Instance.
classifiers is the set of Classifiers that declared the subset of the Instance.
[1] The AttributeLinks must match the declarations in the Classifiers.
[2] There must not exist name conflict between two slots.
[3] All the attributes defined in the Classifiers, must have a corresponding AttributeLink (slot) associated with the Instance.
[4] All the LinkEnds known by the Instance must correspond to AssociationEnds defined in the Instance Classifiers.
[5] There must not exist name conflict between the slots and the opposite LinkEnds known by the Instance.

Stimulus
A Stimulus is, in a way, an instantiation of a Message. It includes features like sender and receiver, which are also specified in the underlying message, but now from the Instances point of view. It also includes arguments, which are all instances (specified in the Action (CallAction) attached to the Message).
In the metamodel, a Stimulus is a communication, a Signal sent to an Instance, or an invocation of an Operation. It can also be a request to create an Instance, or to destroy an Instance. The reception of a Stimulus originating from a CallAction by an Instance causes the invocation of an Operation on the receiver. The receiver executes the method that is found in the descriptor of the class that corresponds to the Operation.

Stimulus

ModelElement

arguments: seq(Instance)
communicationLink: Link
message: Message
receiver: Instance
sender: Instance
[1]#(arguments) = #(message.action.actualArguments) Ù
Õi ³ dom arguments x(Ö c: Classifier x
type(self.message.action.actualArguments[i]) = c Ù
satisfyType(self.arguments[i], c))
[2]self.message.receiver.base ³ self.receiver.classifiers Ù
self.message.sender.base ³ self.sender.classifiers
[3]#(self.receiver.classifiers) = 1 Ù #(self.sender.classifiers) = 1

The Object-Z schema above describes the following Attributes, Associations and Well-formedness rules:
arguments refers to the sequence of Instances being the arguments of the Message Instance.
communicationLink refers to the link which is used for communication.
message refers to the Message which is the base of this Stimulus, i.e. this Stimulus is an "instance" of that *message.*
receiver is the Instance which receives the Stimulus.
sender is the Instance which sends the Stimulus.
[1] There is a correspondence in order, number and types between the stimulus' parameters and the related Action's arguments.
[2] The sender and receiver (Classifiers) of the Message from which this Stimulus is an instance of, must specify the sender and receiver of this Stimulus (Instances).
[3] Due to the fact that the sender (receiver) of the underlying Message is specified by an only one Classifier, the sender (receiver) of this Stimulus must be an instance of that only Classifier. This was stated in order to maintain a consistent relationship between Stimulus and Message.

3.2. Definition of the Semantics Mapping ("sem")

One of the aspects of the specification of UML that is not clear, is the relation between the structural and the behavioral elements.
The following schema describes the function *sem* which performs a mapping from a *Collaboration* into a triple formed by a set of *Stimulus* sequences, a set of *Objects* and finally a set of *Links*. That triple represents all possible instantiations or systems that conform to the specification described by the *Collaboration*.

In a Collaboration, messages appear in a partial order given by their activators and predecessors. A message must be preceded by its activator and its predecessors, but the predecessors may have no order among them. So that the auxiliary function *messageSequences* returns all the possible orders in which Messages can appear in an Interaction.

This order is maintained in the sequences of *Stimuli* returned by the main *sem* function, and in this case it represents all possible orders of *Stimuli* during program execution.

There are certain restrictions that ensures the correctness of the result of the *sem* function:

It is guaranteed by rule [1] in AssociationEndRole that all Link returned by *sem* will connect objects defined by ClassifierRoles in the same Collaboration. This is ensured by the fact that the returned set of objects contains all the possible instantiations of Classifiers that are base for the ClassifierRoles of the Collaboration.

There can only exist Stimuli between objects that [a] have an association between them, or [b] one of them has created the other one, or [c] one of them knows the other via parameter passing. This rule is defined in the *sem* schema.

The semantics mapping conforms a bridge between structural and semantic elements of the Semantics package of the UML specification. We can test the correctness of a system just by checking that its elements (object, links, stimuli) are in the range of the function *sem*.

$$sem: Collaboration \rightarrow \mathbb{P}\, Seq(Stimulus) \times \mathbb{P}\, Object \times \mathbb{P}\, Link$$

$$sem\ col = (traces, objects, links)$$

$$traces = \bigcup_{i \in col.\mathrm{int}eraction} sem(i)$$

$$objects = \{object: Object \bullet \exists\, cr \in col.ClassifierRoles \bullet object \in sem(cr)\}$$

$$links = \{link: Link \bullet \exists\, ar \in col.associationRoles \bullet link \in sem(ar)\}$$

$$\forall\, trace \in traces \bullet$$
$$(\forall\, i \in dom\ trace \bullet$$
$$[a]((\exists\, l \in links \bullet \exists\, le1, le2 \in l.connections \bullet$$
$$(trace[i].sender \in le1.instances \mid trace[i].receiver \in le2.instance))$$
$$\vee$$
$$[b](\exists\, j \in dom\ trace \bullet (j < i \mid trace[j].action.isCreateAction$$
$$\mid trace[j].sender = trace[i].sender$$
$$\mid trace[j].action.instantiation \in trace[i].receiver.classifiers)$$
$$\mid (\nexists\, h \in dom\ trace \bullet h<j \mid h>i \mid trace[h].action.isDestroyAction \mid$$
$$trace[h].receiver = trace[i].receiver))$$
$$\vee$$
$$[c](\exists\, tr \in traces \bullet \exists\, j \in dom\ t \bullet$$
$$(tr[j].receiver = trace[i].sender \mid trace[i].receiver \in tr[j].arguments))))$$

sem:: ClassifierRole f P*(Object)*

sem cr = {o:Object | cr.base ∈ *o.allClassifiers}*

sem:: AssociationRole f P*(Link)*

sem ar = {l: Link | ar.base ∈ *l.allAssociations}*

sem:: Message f P*(Stimulus)*

sem mes = {st: Stimulus | st.receiver ∈ *sem(mes.receiver)* ∧ *st.sender* ∈ *sem(mes.sender)* ∧
st.dispatchAction = mes.action ∧ *(*Õ *i* ∈ *dom st.arguments* • *st.arguments[i]* ∈ *sem(mes.action.aguments[i]))* ∧
(mes.communicationConnection µ *null* Û *st.communicationLink* ∈ *sem(mes.communicationConnection))}*

sem:: Interaction f P*(seq Stimulus)*

sem int = {trace: seq(Stimulus) | Ö *l*∈ *messageSequences(int)* •
*(*Õ *i* ∈ *dom l* • *trace[i]* ∈ *sem(l[i]))* ∧ *(*Õ *i,j*∈ *dom l* • *(l[j].activator = l[i]* Û *trace[j].sender = trace[i].receiver))* }

messageSequences:: Interaction f P*(seq Message)*

messageSequences int = {mesSeq| ran(mesSeq) = int.messages ∧
(Õ *i,j*∈ *dom mesSeq* • *mesSeq[i] = mesSeq[j]* Û *i = j)*
(Õ *i,j*∈ *dom mesSeq* • *mesSeq[i].activator = mesSeq[j]* Û *j < i)* ∧
(Õ *i,j*∈ *dom mesSeq* • *mesSeq[j]* ∈ *mesSeq[i].predecessors* Û *j < i)}*

satisfyType:: Instance x *Classifier* f *Boolean*

*satisfyType ins cl = true, if (*Õ *f* ∈ *cl.allFeatures* • *(* ∃ *c* ∈ *ins.classifiers*• *f* ∈ *c.allFeatures))*
= false, otherwise

sem:: Argument f P *Instance*

sem arg = {instance: Instance | satisfyType(instance, arg.value)}

sem::AssociationRole f P *Link*

sem assocRole = { link:Link | link.association =assocRole}

4. Improvements to UML

4.1. Enhancements

An important point to talk about is inheritance between Collaborations. UML specifies that two Collaborations can be related through a Generalization relation, but hardly defines what it means, i.e., its semantics.
In the UML metamodel, it is defined that a Collaboration "is" a GeneralizableElement. So, it is possible to talk about the concept of inheritance between Collaborations. But UML does not formally define its semantics; UML does not establish the rules that must exist between a parent Collaboration and its child.
This lack of semantic precision leads to different interpretations and ambiguities that emerge due to different points of view of developers. Because of that, we consider the necessity to define, in a formal way, the semantics of the Generalization relation between Collaborations, through the inclusion of the following well formedness rules:

(∀ gr ∈ self.generalization.parent.ownedClassifierRoles •
(gr ∈ ownedClassifierRoles ∨ ∃ o ∈ ownedClassifierRoles • gr e o.allParents))

This rule states that a Collaboration specializing another Collaboration must include all the ClassifierRoles contained in the parent Collaboration or their specializations.
Another rule to mention is the one that states that, when a Collaboration is a specialization of another one, it must contain all message sequences that are present in the parent:

(∀ parentInt ∈ self.generalization.parent.interactions •
(∀ parentMesSeq ∈ messageSequences(parentInt) •
(∃ int ∈ interactions • (∃ mesSeq ∈ messageSequences(int) • parentMesSeq º mesSeq))))

Thus, with the inclusion of these rules, the semantics of the Generalization relation between Collaborations is formally defined.
The same happens with the Generalization relation between ClassifierRoles.
UML specifies that two ClassifierRoles can participate in a Generalization relation because they are GeneralizableElements, but nothing states about the meaning of that relation. So, we consider valuable to define in a precise way, the semantics of inheritance between ClassiferRoles, though the inclusion of the following well formedness rule:
(self.generalization.parent <> null) Û
(Õa ³ self.generalization.parent.allFeatures x Ö c ³ self.base x a ³ c.allFeatures)

The rule states that if two ClassifierRoles are related through a generalization relation, each of the features contained in the parent must be included in at least one Classifier that conforms the base of the child ClassifierRole.

4.2. Inconsistencies and suggestions

1. UML specifies that a LinkEnd connects **only one** Instance that participates in the correspondent Link. However, in the AssociationEnd metaclass an attribute called multiplicity is declared, which defines the valid range (minimal and maximal cardinality) of the number of Instances that could be connected through a LinkEnd.
Because of that, we consider that a LinkEnd must know the set of Instances that could be connected to it. So, we suggest that the range of Instances connected by a LinkEnd must be **1..n**.

2. UML seems to be very flexible at design time; for example, in UML is possible to define an Instance with more than one Classifier defining its structure and behavior.
However UML limits that flexibility in other cases. For example, the Operation and Action metaclasses define parameters and arguments respectively whose types are specified by only one Classifier. This would imply that it would not be possible to use Instances that originates from a set of Classifiers, as Operation parameters.
The same happens with the use of Messages: in the Message metaclass the sender and receiver are specified by one Classifier each. This means that it would not be possible to use Instances that originates from a set of Classifiers, to exchange messages! This contradiction is not desired because it introduces ambiguity and lack of precision in the language constructions. This is a situation in which it is necessary to evaluate and decide what is better depending on the case: whether to have a lot of flexibility and less clarity at design time, or reduce flexibility and gain precision. We propose the following function as a solution to the ambiguity problem:
satisfyType::Instance x Classifier -> Boolean
Given an Instance **i** and a Classifier **c,** this function determines if **i** can be consider "instance" of **c**.
So, **satisfyType(i,c) = true** if **i** has, at least, all the features (structural and behavioral) defined in **c**.
Due to this function, it is possible to use Instances that originates from a set of Classifiers, as Operation parameters: if we have a function that requires a parameter of type **c** and we want to invoke it using an Instance **i** as the argument, we only need to verify that **satisfyType (i,c)** is **true**.

3. It would be necessary to add the attribute called **multiplicity** in the Association metaclass that defines the minimal and maximal number of Links that could exist of that Association (for consistency with the attribute multiplicity in AssociationRole).

4. UML clearly defines, as we said earlier, a clear correspondence between the metaclasses at the syntax level and the metaclasses at the semantic level. That correspondence represents a sort of instantiation relation. For example, Association and Link, Classifier and Instance, etc. In the same way, it would be possible to deduce that Action is the metaclass associated with Stimulus, because a Stimulus knows an Action and both belongs to different levels. But on the other hand, a Stimulus knows a sender and a receiver that correspond to the attributes sender and receiver defined in the Message metaclass. This means, in a way, that Stimulus represents the semantic metaclass of both, Action and Message, introducing ambiguity in the metamodel. Instead of knowing an Action, we consider that a

Stimulus should know the Message that defines it, because its semantics corresponds with the instantiation of a Message. With this approach, Action would be accessible anyway through the Message. This modification maintains the fact that each semantic metaclass corresponds to one syntax metaclass, and contributes to the clarity and legibility of the UML metamodel.

5. Another important question to talk about is the fact that both, the semantic level metaclasses and syntax level metaclasses, inherit from ModelElement. At this point, the two levels are not differentiated. In our opinion, it would be desirable the existence of a metaclass in the semantic level, for instance DataElement, that would correspond to ModelElement. In this way, all the metaclasses at the semantic level would inherit from the new metaclass DataElement.

6. There are some well-formedness rules that are incorrect in the UML document. For instance:

- The function hasSameSignature was designed in order to formalize the rule number [3] of Instance in the UML document (this rule was misplaced in that document).
- Rule number [2] corresponding to LinkEnd was taken from Instance because it was misplaced.
- Rule number [4] in Instance was stated in terms of the LinkEnds, because an Instance does not know the Links in which it participates (in UML the rule was defined in terms of Links).

4.3. Well-formedness rules discovered during formalization

During the formalization, we discovered new restrictions that UML does not take into account or cannot express, and that we consider useful to include. Due to the use of a formal language, we can express them. These rules (see Object-Z schemas in the paper) make the semantics of UML more accurate and precise, avoiding ambiguity and different interpretations of the language.

5. Concluding remarks

The specification of UML is semi-formal, i.e. certain parts of it are specified with well-defined languages while other parts (such as the semantics of UML constructs) are described informally in natural language.

Our aim is to formalize the syntax and semantics of UML collaboration diagrams using the formal language Object-Z. The main motivation for formalization is that users of the language need to understand each other precisely. The lack of accuracy in the definition of the language can cause problems regarding the models expressed in the language, such as inconsistent interpretations and ambiguities. Furthermore, tools like code generators and consistency checkers require that syntax and semantics of the language to be precise.

Our research has permitted the discovering of many inconsistencies and ambiguities of the UML definition, motivating the discussion of some improvement ideas.

References

Araújo, João, Formalizing Sequence Diagrams, In Luís Andrade, Ana Moreira, Akash Deshpande and Stuart Kent, editors, Proc. OOPSLA´98 Wsh. Formalizing UML. Why? How? Vancouver, (1998).

Gehrke, T. Goltz, U. and Wehrheim, H. The dynamic models of UML: towards a semantics and its applications in the development process. Technical report 11/98, Universität Hildesheim, (1998).

Knapp, Alexander, A formal semantics for UML interactions, <<UML>>´99 - The Unified Modeling Language. Beyond the Standard. R.France and B.Rumpe editors, Proceedings of the UML´99 conference, USA,. Lecture Notes in Computer Science 1723, Springer. (October 1999).

Övergaard, Gunnar, A formal approach to collaborations in the UML, <<UML>>´99 - The Unified Modeling Language. Beyond the Standard. R.France and B.Rumpe editors, Proceedings of the UML´99 conference, USA,. Lecture Notes in Computer Science 1723, Springer. October (1999).

The Unified Modeling Language (UML) Specification – Version 1.3, (July 1999). UML Specification, revised by the OMG, http://www.rational.com/uml/index.jtmpl

Waldoke, S., Pons, C., Paz Mezzano,C. and Felder, M., A Formal Approach to Practical O-O Analysis and Design, Proceedings of Argentinean Symposium on Object Orientation, Buenos Aires, (1998).