Building a bridge between 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
Email: {acibran, vmola, cpons, wanda}@sol.info.unlp.edu.ar

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.

A collaboration diagram may be presented at two different levels: specification level (syntax) or instance level (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.

1. Introduction

The models constitute the fundamental base of information upon which the problem domain experts, the analysts and the software developers interact. Thus, it is of a fundamental importance that it clearly and accurately expresses the essence of the problem.

Models are constructed using a modelling language (which may vary from natural language to diagrams and even to mathematical formulas).

The success of software development methods, are mainly based on their use of intuitively appealing modelling constructs and rich structuring mechanisms, which are easy to understand, apply and transmit to the customers. However, the lack of precise semantics for the modelling notations used by these methods can lead to several problems such us different interpretations, ambiguities, etc.

On the other hand, formal languages for modelling have a well-defined syntax and semantics. However, their use in industry is not frequent. This is due to the complexity of its mathematical formalisms, which are difficult to understand and communicate.

As a consequence, it has been proposed to combine the advantages of both approaches, intuitive graphical notations on the one hand and mathematically precise formalisms on the other hand, in development tools. The basic idea for this combination is to use mathematical notation in a transparent way, hiding it as much as possible under the hood of graphical notations. This approach has advantages over a purely graphical specification development as well as over a purely mathematical development because it introduces precision of specification into a software development practice while still ensuring acceptance and usability by current developers.

The specification of the Unified Modelling Language (UML) [UML 99] 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 [Araújo 98] translates a subset of UML sequence diagrams into temporal logic formulas. Gehrke et al. sketch a translation of UML collaboration diagrams to Petri nets, Knapp [Knapp 99] proposes a formal semantics of interactions using temporal logic formulas, Övergaard [Övergaard 99] gives a formal definition of the collaboration construct in the UML, the UZ-translator [Waldoke et al.98] produce 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 (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.

1* Authors appear in alphabetical order

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 (syntactic) point of view.

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.

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 proyection 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.

```
ClassifierRol
e
    Classifier
    multiplicity: Integer
    availableContents: ▯ ModelElement
    availableFeatures: ▯ Feature
    base: ▯ Classifier
    ----------------------------------------
    allAssociationRoles: ▯ AssociationRole
    ∀ ar: AsociationRole •
    ((Ö c ³ ar.connections × c.type = self) Û ar ³
    allAssociationRoles)

    allAvailableFeatures:  ▯ Feature
    ∀ p ∊ self.parents •
      (allAvailableFeatures= availableFeatures
    ¿ self.allAvailableFeatures)

    allAvailableContents : ▯ 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 × Ö c ³ self.base × a ³
    c.allFeatures)
```

- *Attributes*

multilplicity refers to the number of Instances playing this particular role in the Collaboration.

- *Associations*

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.

- *Well formedness rules*

[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 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: ▯ ModelElement
    interactions: ▯ Interaction
    ownedAssociationRoles: ▯ AssociationRole
    ownedClassifierRoles: ▯ ClassifierRole
    ownedGeneralizations: ▯ Generalization
    representedClassifier: Classifier
    representedOperation: Operation
    ------------------------------------------------
    [1](representedClassifier = null) xor (representedOperation = null)

    [2]∀ a ∈ ownedAssociationRoles • (∃ aend ∈ a.associationEndRoles •
      (Ö c ∈ ownedClassifierRoles • (aend.base.classifier ∈ c.base)))

    [3]∀ o ∈ ownedAssociationRoles • (o.base.namespace = self.namespace) ∧
    ∀ c ∈ ownedClassifierRoles • (∀ b ∈ c.base • b.namespace =
    self.namespace)

    [4]∀ c ∈ constrainingElements •  (c.base ∈ self.namespace.ownedElements)




    [5]∀ c ∈ ownedAssociationRoles • (c.name  = null  ▯
        (¬ (∃ d ∈  ownedAssociationRoles • (d.base = c.base ∧d ≠  c))) ∧
        ∀ c ∈ ownedClassifierRoles • (c.name  = null  
        (¬ (∃ d ∈ ownedClassifierRoles • (d.base = c.base ∧d ≠  c)))

    [6]∀ r ∈ ownedAssociationRoles • (∃ r2 ∈
    self.generalization.parent.ownedAssociationRoles •
              (r.name = r2.name  r2 =r1  r2  r1.allParents))  ∧
        ∀ r ∈ ownedClassifierRoles • (∃ r2 ∈
    self.generalization.parent.ownedClassifierRoles •
             (r.name = r2.name  r2 =r1  r2  r1.allParents))

    [7](representedClassifier  null ) 
        ∀ i ∈ interactions • (i.messages[0].receiver.base =
    self.representedClassifier)

    [8](∀ gr ∈ self.generalization.parent.ownedClassifierRoles •
      (∃ o ∈ ownedClassifierRoles • o=gr ∨ o   gr.allParents  ))

    [9](∀ parentInt ∈ self.generalization.parent.interactions •
          (∃ int ∈ interactions • parentInt.messages  º  int.messages))
```

- *Associations*

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.

- *Well formedness rules*

[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 or their specializations.

[9] A Collaboration specializing another one must contain at least all the messages that are present in the parent among its interactions.

Interaction

An Interaction specifies the communication between Instances performing a specific task. Each Interaction is defined in the context of a Collaboration.

An Interaction contains a set of Messages specifying the communication between a set of Instances conforming to the ClassifierRoles of the owning Collaboration.

An Interaction defined in the context of a Collaboration specifies the details of the communications that should take place in performing 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. The order of the communications is also specified by the Interaction.

Syntactically, an Interaction is represented as a context and a set of Messages. These messages are the ones sent and received to carry out the specific task.

```
Interaction
    ModelElement
    ─────────────────────────────────────────
    context: Collaboration
    messages: ▯ Message
    ─────────────────────────────────────────
    [1]∀ i ∈ dom messages • (message[i].sender ∈ context.ownedClassifierRoles) ∧
                                  (messages[i].receiver ∈
    context.ownedClassifierRoles)

    [2]∀ i,j ∈ messages • (i=j ∨i.activator ≠ j.activator) ∨ (i ∈ j.predecessors ∨j ∈
    i.predecessors)

    [3]∀m ∈ messages • (m.interaction = self)

    [4]∀m ∈ messages • (m.sender = m.activator.receiver)
```

- *Associations*

context specifies the Collaboration which defines the context of the Interaction.

messages is a set of Message which specifies the communication in the Interaction.

- *Well-Formedness rules*

[1] Every sender and receiver of the messages conforming the interaction must be in the context of this Interaction.

[2] Every pair of messages with the same activator can always be compaired in the predecessor order.

[3] All the messages in the set *messages* must be in the same interaction that is being described.

[4] A message sender is its activator receiver.

Message

A Message defines a particular communication between Instances that is specified in an Interaction. The Message defines the role of the sender and the receiver Instances, as well as the Action that will cause the communication. Moreover, the Message also specifies which messages should have been received and sent before the current one, as well as the expected response of the receiver, which should be in conformance with the specification of the corresponding operation of the receiver.

In the metamodel a Message defines one specific kind of communication in an Interaction; it defines a particular usage of a Request in an Interaction. A communication can be e.g. raising a Signal, invoking an Operation, creating or destroying an

Instance. The Message specifies not only the kind of communication, but also the roles of the sender and the receiver, the dispatching Action, and the role played by the communication Link. Furthermore, the Message defines the relative sequencing of Messages within the Interaction.

```
Messag
e
    action: Action
    activator: Message
    communicationConnection: AssociationRole
    predecessors: seq(Message)
    receiver: ClassifierRole
    sender: ClassifierRole
    interaction: Interaction
    ----------------------------------------------------------------
    [1](∃m ∊ interaction.messages • (m = self))

    [2](∀p ∊ predecessors • (p.interaction = self.interaction)) Ù activator.interaction =
    self.interaction

    [3](∀p ∊ predecessors • (p.activator = self.activator))

    [4](∀p ∊ predecessors • (p µ self))

    [5]activator µ self

    [6]sender ∊ interaction.context.ownedClassifierRoles ∧ receiver ∊
    interaction.context.ownedClassifierRoles

    [7]communicationConnection ∊ interaction.context.ownedAssociationRoles

    [8](communicationConnection <> null) =>
        (∃ assEndRole1, assEndRole2 ∊ communicationConnection.allConnections •
            assEndRole1.type = sender ∧ assEndRole.type = receiver)

    [9]action ∊ CallAction Û action.operation ∊ receiver.availableFeature
```

- *Associations*

action specifies the Action which causes the Stimulus to be sent according to the Message.

activator specifies the Message which invokes the operation whose associated behavior (method) causes the dispatching of the current Message.

communicationConnection refers to the AssociationRole played by the Links used in the communications specified by the Message. This specifies the knowledge relationship of the Message's sender and receiver.

receiver specifies the ClassifierRole of the Instance that receives the communication specified by the Message and reacts to it.

predecessors refers to the set of Messages whose completion enables the execution of the current Message. All of them must be completed before execution begins. Empty if it is the first message in a method.

sender is the ClassifierRole of the Instance that invokes the message and possibly receives a response.

interaction is the Interaction to which the Message belongs. This defines the context in which the communication defined by the message takes part.

- *Well-Formednes Rules*

[1] The current Message is one of the messages that the associated Interaction contains.

[2] and [3] All predecessors and activator must be contained in the same Interaction as the current Message.

[4] Predecessors must have the same activator as the Message.

[5] A Message cannot be predecessor of itself.

[6] A Message cannot be the activator of itself.

[7] The sender and receiver must participate in the Collaboration which defines the context of the Interaction.

[8] The communicationConnection of the Message must be an AssociationRole in the context of the Message's Interaction, i.e., it must participate in the Collaboration which defines that context.

[9] The sender and receiver roles must be connected by the associationRole which acts as the communicationConnection of the Message.

[10] If the Action associated to the Message is a CallAction, the corresponding operation must belong to the available features of the Message´s receiver.

AssociationEndRole

An AssociationEndRole specifies an endpoint of an association as used in the collaboration.
In the metamodel an AssociationEndRole is part of an AssociationRole and specifies the connection of an AssociationRole to a ClassifierRole. It is related to the AssociationEnd, declaring the corresponding part in an Association.

```
AssociationEndRole
    AssociationEnd
    multiplicity: £
    availableQualifiers: ¡ Attribute
    base: AssociationEnd
    associationRole: AssiciationRole
    type: ClassifierRole
    [1](self.base.type = self.type.base) Ú
     (self.base.type ³ self.type.base.allParents)

    [2]availableQualifiers º base.qualifiers

    [3]self.isNavigable Û self.base.isNavigable

    [4]type ∊ associationRole.collaboration.ownedClassifierRoles

    [5]base ∊ associationRole.base.connections
```

- *Attributes*

multiplicity specifies the number of LinkEnds playing this role in a Collaboration

- *Associations*

availableQualifiers is the subset of the qualifiers of the corresponding AssociationEnd (base) that are used in the Collaboration. availableQualifiers is a set of qualifier Attributes for the end that are used in this context.

base represents the AssociationEnd which the AssociationEndRole is a projection of.

associationRole is the AssociationRole to which the AssociationEndRole is part of.

type refers to the ClassifierRole connected to this end of the AssociationRole.

- *Well-Formednes Rules*

[1] The type of the ClassifierRole must conform to the type of the base AssociationEnd.

[2] The qualifiers used in the AssociationEndRole must be a subset of those in the base AssociationEnd.

[3] In a collaboration an associationEndRole may only be used for traversal if it is allowed by the base associationEnd.

AssociationRole

In UML an AssociationRole describes the role being played by an Association in a Collaboration; it specifies a restricted view of the Association. An AssociationRole is defined between a subset of the ClassifierRoles that represent the Classifiers associated by the base Association.

```
AssociationRole
    Association
    base:Association
    collaboration:Collaboration
    multiplicity: (£,£)
    connections:i AssociationEndRole
    [1] #connections >= 2

    [2] (∀c ∈ connections • c.base ∈ self.base.connections)

    [3] #connections <=#(base.connections)

    [4] (∀ i ∈ connections • i.base ∈ self.base.connections)

    [5] (∃ i, j ∈ self.connections • i.base=j.base ⇒i=j)
```

- *Attributes*

multiplicity represents the range of Links (associated to the base Association) that play this role in the Collaboration.

- *Associations*

base refers to the Association it represents (its base Association).

collaboration represents the Collaboration in which the AssociationRole is contained.

connections represents the AssociationEndRoles that the represented AssociationRole connects.

- *Well formedness rules*

[1] The cardinality of connections must be greater or equal to 2, i.e.: an AssociationRole associates at least two AssociationEndRoles.

[2] All AssociationEndRoles that the AssociationRole connects, must have this AssociationRole as its base. They must conform to the AssociationEnds of the base Association.

[3] The number of AssociationEndRoles in connections must be less than or equal to the number of AssociationEndRoles of the connections of the base Association. This can be seen more clearly in the following rule.

[4] The variable connections represents a subset of the AssociationEndRoles connected to the base Association.

[5] There are not two AssociationEndRoles in connections with the same base Association.

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 instanciation 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 behaviour.

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: i AttributeLink
    linkEnds: i LinkEnd
    classifiers: i Classifier
    persistent: (transitory, persistent)
    opossiteLinkEnds: Instance § i 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 × Õ le ³ oppositeLinkEnds(self) × a.name ≠ le.name
```

- *Attributes*

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.

- *Associations*

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.

- *Well formedness rules*

[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 instanciation 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.

```
Stimul
us
    ModelElement
    arguments: seq(Instance)
    communicationLink: Link
    message: Message
    receiver: Instance
    sender: Instance
    [1]#(arguments) = #(message.action.actualArguments) Ù
        Õi ³ dom arguments  ×( Ö c: Classifier ×
        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
```

- *Associations*

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.

- *Well formednes rules*

[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.

Action

An action is a specification of a computable/executable statement and is realized by sending a message to an object or modifying a link or a value of an attribute. After executing an action, the state of the model may be changed.

In the metamodel an Action may be part of an ActionSequence and may contain a specification of a target as well as a specification of the actual arguments.

Actions are always executed within the context of an instance, so the target set expression and the argument expressions are evaluated within an instance.

```
Action
    ModelElement
    IsAsynchronous: Boolean
    Recurrence: IterationExpression
    script: ActionExpression
    target: ObjectSetExpression
    actualArguments: seq(Argument)
```

- *Attributes*

isAsynchronous indicates if a dispatched stimulus is asynchronous or not.

recurrence specifies an expression stating how many times the Action should be performed.

script is an ActionExpression describing the effects of the Action.

target is an ObjectSetExpression which determines the target of the Action. When it is executed, it resolves into zero or more specific instances that are the intended target of the Action.

- *Associations*

actualArguments refers to a sequence of expressions which determines the actual arguments (a set of instances) needed when evaluating the Action. These instances are the actual arguments of e.g. the stimulus being dispatched by the action, i.e. the instances passed with a signal or he instances used in an operation invocation.

AttributeLink

In UML an AttributeLink represents a piece of state of an Instance, which holds the value of an attribute.

```
AttributeLink
    ModelElement
    value: Instance
    attribute: Attribute

    [1] (∃ v ∈ value.classifier.allParents • v = attribute.type)
```

- *Associations*

value contains the Instance which is the value of the AttributeLink

attribute corresponds to the attribute from which the AttributeLink originates, i.e.: an attribute of that Instance.

- *Well formedness rules*

[1] The type of the *instance* must match the type of the *attribute*. The reason for this restriction is that *instance* is an istantiation of the *attribute*, so the type of the former must be the same (or a subtype) of the last one.

CallAction

A CallAction is an Action in which a stimulus is created that causes an operation to be invoked on the receiver. A CallAction can be synchronous or asynchronous, indicating whether the operation is invoked synchronously or asynchronously.

```
CallAction
    Action
    operation: Operation
    [1] #(self.actualArguments) = #(self.operation.parameters) Ù
          Õ i ³ dom actualArguments ×
                satisfyType(self.actualArguments[i],
    self.operation.parameters[i].type)
```

- *Attributes*

isAsynchronous (inherited from Action) indicates if a dispatched operation is asynchronous or not.
(false indicates that the caller waits for the completion of the execution of the operation; true indicates that the caller does not wait for the completion of the execution, it continues immediately).

- *Associations*

operation refers to the operation that will be invoked when the Action is executed.

- *Well formedness rules*

[1] All the actualArguments of this action have a correspondence in number and type with the parameters specified in the associated operation.

DataValue

DataValue is a kind of Instance which has no identity. Moreover, a data value cannot change its state; all operations that are applicable to a data value are queries and do not cause any side effects.

A data value cannot change its data type.

```
DataValue
  Instance
  --------------------------------------------
  [1] #classifiers = 1

  [2] Ö c: DataType × classifiers = {c}

  [3] self.slots = ¸
```

- *Well formedness rules*

[1] A DataValue originates from exactly one classifier.

[2] The classifier from which the DataValue originates must be a DataType.

[3] A DataValue has no AttributeLinks.

Link

In UML a Link describes a connection between instances. It represents the instantiation of an Association.

```
Link
  ModelElement
  connections: Seq(LinkEnd)
  association: Association
  [1] #connections >=2

  [2] ∀ c ∈ connections • (∃ aend ∈ association.connections •
     (c.associationEnd = aend))




  [3] ∀ link2 ∈ self.association.connections •
        ((#self.connections = # link2.association.connections) ∧
          (∀ i ∈ [0..self.connections.size] •
             self.connections(i). instances =
  link2.association.connections(i).instances)
                                                          □    link2=self )

  [4]  (□ le1,le2 □ connections • le1.name=le2.name □ le1=le2)
```

- *Associations*

connections contains the set of LinkEnds that are instances of the AssociationEnds corresponding to that Association.

association describes the Association from which this Link is instance

- *Well formedness rules*

[1] The size of connections must be greater or equal to two.

[2] The set of LinkEnds in *connections* must match the set of AssociationEnds corresponding to the Association in *association*. This is because *connections* contains LinkEnds that must be instances of the AssociationEnds of the Association.

[3] There are not two Links of the same Association that connects the same Instances in the same way.

[4] There are not two LinkEnds in *connections* with the same name.

LinkEnd

In UML a LinkEnd describes an end point of a Link. It represents an instantiation of an AssociationEnd.

```
LinkEnd
    ModelElement
    instances: ℙInstance
    associationEnd: AssociationEnd
    [1] ∀ i ∈ instances •
         (∃ p ∈ i.classifier.allParents • p = associationEnd.type)

    [2] min (associationEnd.multiplicity) <= #instances <= max
    (associationEnd.multiplicity)
```

- *Associations*

instance represents the Instance that the LinkEnd connects.

associationEnd is the AssociationEnd from which this LinkEnd in an instance.

- *Well formedness rules*

[1] The type of *instances* must match the type of *associationEnd*; i.e.: the type of *instance* must be the same (or a subtype) of *associationEnd*.

[2] The multiplicity of the variable *instance* must match the multiplicity specified by *associationEnd*.

LinkObject

A link object is a link with its own set of attribute values and to which a set of operations may be applied.
In the metamodel LinkObject is a connection between a set of Instances, where the connection itself may have a set of attribute values and to which a set of operations may be applied. It is a child of both Object and Link.

```
LinkObject
```

```
Object, Link

[1] self.association ³ self. classifiers

[2] Ö ac: AssociationClass × asociation = ac
```

- *Well formedness rules*

[1] The Association must be between the Classifiers of the LinkObject. It is possible because the Association is an AssociationClass and therefore, a Classifier.

[2] The Association must be a kind of AssociationClass.

Object

An object is an instance that originates from a class.

In the metamodel, an Object is a subclass of Instance and it originates from at least one Class.

If an Object has multiple classes, it will have all the features declared in all of these classes, both the structural and the behavioral ones.

The set of classes may be modified dinamically. This means that the set of features of the Object may change during its life-time.

All Objects originating from the same class are structured in the same way, although each of them has its own set of attribute links.

```
Object
Instance

[1] Õc ³ classifiers × Ö cl: Class × c = cl
```

- *Well formedness rules*

[1] All Classifiers must be a kind of Class.

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 instanciations 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 it´s activator and it´s 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 mantained 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 garranteed 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 instanciations 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*.

Semantics mapping schema

```
sem: Collaboration□ □ Seq(Stimulus)□ □ Object □ □Link
--------------------------------------------------------------------------------
sem col = (traces, objects, links)


traces =¿ sem(i)
          i ∈ interactions

objects = {object: Object • □ cr ∈ col.ClassifierRoles • cr.base ∈  object.classifiers}

links = {link: Link • □ ar ∈ col.associationRoles •  link.association = ar.base}
--------------------------------------------------------------------------------
Õ trace ∈ traces •
(Õ i  ∈ dom trace •
[a]((□  l ∈ links • □ le1,le2 ∈ l.connections •
              (trace[i].sender ∈  le1.instances □ trace[i].receiver ∈ le2.instance))
□
[b]( □ j ∈ dom trace • (j < i □ trace[j].action.isCreateAction □ trace[j].sender = trace[i].sender
              □ trace[j].action.instanciation ∈ trace[i].receiver.classifiers )
          □   (Ø□ h ∈ dom trace • h<j □ h>i  □ trace[h].action.isDestroyAction □
                           trace[h].receiver = trace[i].receiver))
□
[c]( □ tr ∈ traces • □ j ∈ dom t •
       (tr[j].receiver = trace[i].sender □ trace[i].receiver ∈  tr[j].arguments))))
```

```
sem:: Message □ □(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 □ □(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)) }
```

```
sem:: Argument □ □ Instance
--------------------------------------------------------------------------------
sem arg = {instance: Instance | satisfyType(instance, arg.value)}
```

```
sem::AssociationRole □ □ Link
--------------------------------------------------------------------------------
sem assocRole = { link:Link  |  link.association =assocRole}
```

```
messageSequences:: Interaction □ □(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 □ Classifier □ Boolean
--------------------------------------------------------------------------------
satisfyType ins cl = true, if  (Õ f ∈ cl.allFeatures • ( ∃ c ∈ ins.classifiers• f ∈ c.allFeatures))
                    = false, otherwise
```

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:

$$\begin{aligned}
&(\forall\ \text{gr} \in \text{self.generalization.parent.ownedClassifierRoles} \bullet \\
&\quad (\exists\ \text{o} \in \text{ownedClassifierRoles} \bullet (\text{o} = \text{gr}\ \text{Ú}\ \text{o} \in \text{gr.allParents}))
\end{aligned}$$

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:

$$\begin{aligned}
&(\forall\ \text{parentInt} \in \text{self.generalization.parent.interactions} \bullet \\
&\quad (\forall\ \text{parentMesSeq} \in \text{messageSequences(parentInt)} \bullet \\
&\qquad (\exists\ \text{int} \in \text{interactions} \bullet (\exists\ \text{mesSeq} \in \text{messageSequences(int)} \bullet \\
&\qquad\quad (\forall\ \text{i} \in \text{dom(parentMesSeq)} \bullet (\exists\ \text{j} \in \text{dom(mesSeq)} \bullet \\
&\qquad\qquad\qquad \text{parentMesSeq[i]} = \text{mesSeq[j]}\ ))))))
\end{aligned}$$

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:

$$\begin{aligned}
&(\text{self.generalization.parent} <> \text{null})\ \text{Û} \\
&\quad (\text{Õa}\ ^3\ \text{self.generalization.parent.allFeatures} \times \text{Ö c}\ ^3\ \text{self.base} \times \text{a}\ ^3\ \text{c.allFeatures})
\end{aligned}$$

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 seams 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 semantics 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. UML defines that a ClassifierRole "is" a Classifier (because ClassiferRole inherits from Classifier). It implies that an Instance could originate from a ClassifierRole (or a set of ClassifierRoles). This fact is an example of the lack of precision of UML: it would not be desirable to have the possibility to instanciate ClassifierRole because it is not the aim of that metaclass. A ClassifierRole only must represent the use of a Classifier in a Collaboration, and therefore, it would not be necessary to inherit from Classifier.

UML also restricts ClassifierRoles from adding new attributes of its own. So, the fact that ClassifierRole inherits from Classifier is useless.

We suggest that ClassifierRole should not inherit from Classifier.

The same fact can be observed between AssociationRole and Association. AssociationRole should not inherit from Association either.

7. There are some well-formedness rules that are incorrect. 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).

8. UML specifies an attribute in Collaboration called ownedElements that represents the set of ClassifierRoles, AssociationRoles and Generalizations used in the Collaboration. We consider that it would be clearer to split up that attribute in three specific ones called ownedAssociationRoles, ownedClassifierRoles and ownedGeneralizations that represent the AssociationRoles, ClassifierRoles and Generalizations respectively. Thus, we are using the expressive power of the Object-z typing system.

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 make the semantics of UML more accurate and precise, avoiding ambiguity and different interpretations of the language. Here we list the rules that were discovered and expressed in Object-z in the formalization section:

AssociationEndRole:

[4] The ClassifierRole that is connected through the AssociationEndRole to the corresponding AssociationRole, must participate in the Collaboration to which the AssociationRole belongs.

[5] The base AssociationEnd must be included in the endpoints set of the AssociationRole's base.

AssociationRole:

[1] The cardinality of connections must be greater or equal to 2, i.e.: an AssociationRole associates at least two AssociationEndRoles.

[3] The number of AssociationEndRoles in connections must be less than or equal to the number of AssociationEndRoles of the connections of the base Association. This can be seen more clearly in the following rule.

[4] The variable connections represents a subset of the AssociationEndRoles connected to the base Association.

[5] There are not two AssociationEndRoles in connections with the same base Association.

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.

[5] If two ClassifierRoles or AssociationRoles have no name within the Collaboration then they have different base.

[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 Collaboration must include specializations of all the classifierRoles in the parent.

ClassifierRole

[4] 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.

CallAction:

[1] All the actualArguments of this action have a correspondence in number and type with the parameters specified in the associated operation.(UML only takes into account the number concordance).

Link:

[1] The size of connections must be greater or equal to two.

[4] There are not two LinkEnds in connections with the same name.

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 and receiver of the underlying Message are specified by an only one Classifier, the sender and receiver of this Stimulus must be an instance of and only of that Classifier. This was stated in order to maintain a consistent relationship between Stimulus and Message.

[4] Due to the fact that an only one Classifier specifies the sender and receiver of a message, the sender and receiver of the Stimulus must be instances of and only of that Classifier. This rule makes the relation between Stimulus and Message consistent.

Instance:

[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.

Interaction:

[1] Every sender and receiver of the messages conforming the interaction must be in the context of this Interaction.

[2] Every pair of messages with the same activator can always be compared in the predecessor order.

[3] All the messages in the set messages must be in the same interaction that is being described.

[4] A message sender is its activator receiver.

Message:

[1] The current Message is one of the messages that the associated Interaction contains.

[6] A Message cannot be the activator of itself.

[10] If the Action associated to the Message is a CallAction, the corresponding operation must belong to the available features of the Message´s receiver.

5. Concluding remarks

The specification of UML is half-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.

Another advantage of our formalization is that once a system is specified in UML, we can test its correctness just by checking that it respects our formal specification of UML in Object-Z.

6. References

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

[Övergaard 99] Gunnar Övergaard, 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, Colorado, USA,. Lecture Notes in Computer Science 1723, Springer. October 1999.

[Knapp 99] Alexander Knapp, 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, Colorado, USA,. Lecture Notes in Computer Science 1723, Springer. October 1999.

[UML 99] 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 et al. 98] S.Waldoke, C. Pons, C.Paz Mezzano and M. Felder, A Formal Approach to Practical Object Oriented Analysis and Design, Proceedings of Argentinean Symposium on Object Orientation, ed: SADIO (Sociedad Argentina de Informática e Inv. Operativa), Buenos Aires, 1998.