UML Collaboration semantics
A green(?) paper
Trygve Reenskaug
Version of November 8, 1999

991108 *- Added sections on separation of concern, virtualRoles, and abstract interactions.*
991027 *- Bug fixes. Improved conformance with UML terminology and definitions (but UML 1.3 itself is fairly inconsistent). Chapter on "Specialization of a Collaboration" removed. OOram synthesis is a powerful construct, but its possible inclusion in UML should be postponed until the Collaboration fundamentals are clear.*
991019 *- Several clarifications. Object identity defined in terms of message sends. Defined the notion of an instance level collaboration. Defined ClassifierRole in terms of its instances. Used UML notation for state diagrams.*
990920 *- first version*

1 Summary and conclusion

A thread on UML issue2837 originated with a message from Juergen Boldt dated Wed, 11 Aug 1999: "role concept in UML remains rather vague" where he said that

>> So what are roles? It seems that in UML they are not types (or
>> classifiers), but a positive definition (other than the equally vague
>> glossary entry) would seem imperative!

I tried to answer the question by quotes from Egil Andersen's thesis. These quotes were clearly insufficient to answer the question, and there has now been a long and interesting discussion in the thread. I have studied the communications so far and use them as a background for an explanation of what I believe could or should be the semantics of the UML Collaboration.

Here is a summary of the paper:

1. ***The class.*** Any abstraction is based on a choice of what is important and what isn't. The highly successful and useful class abstraction is the result of one such choice. It is useful for modeling many interesting aspects of objects. A notable exception is the modeling of system behavior; the class abstraction deals in sets of objects and is not suited for modeling how an object sends a stimulus (a signal or an invocation of an operation) to another object as part of an overall conversation.
2. ***The instance level collaboration.*** The instance level collaboration shows how objects work together for a common purpose. It models their conversation very nicely, but it is too concrete and specific for all but the simplest cases. I propose an extension of UML 1.3 that defines the abstract syntax of the instance level collaboration and binds it to the existing specification level collaboration in Appendix 1.
3. ***The specification level collaboration.*** The specification level collaboration is more powerful because it is more abstract. It retains the capability of modeling object conversation and adds capabilities for modeling generalized interaction patterns. It also handles cases with cardinality more than one; this is discussed in section 10.
4. ***Separation of concern.*** Three techniques are discussed: The arbitrary filtering permitted for class diagrams; components that separate on services, hiding their complexity; and collaborations that separate on system behavior.

5. ***Higher level collaboration abstractions with virtualRoles.*** Collaborations are often too precice and detailed for overviews and early architecture work. VirtualRoles (stereotype of *package*) permit arbitrary grouping of classifier roles. This hides the detailed object structure without weakining the collaboration semantics.
6. ***Abstract interactions.*** The basic collaboration specifies interactions in terms of message flow. This may be too detailed for overviews and early architecture work. I propose to use UML comments to name interactions and identify their communication paths (AssociationRoles).
7. ***Typing the AssociationEndRoles***. Typing the object references gives improved control over architecture and system design.
8. ***The Object-Role-Class trichotomy.*** Quite simple, really. (I do not discuss the use of the forward slash):
   "The concrete object"
   / "The role it plays in the collaboration"
   : "The class or classes that can implement it"
9. ***System behavior, messages and cardinalities > 1.*** An example illustrates how the behavior of the system as a whole can be composed from descriptions of individual ClassifierRole behavior. I also consider the case where there are more than one object playing a given role.
10. ***Appendix 1: Instance level and specification level collaborations.*** I propose abstract syntax and constraints for the collaboration concept.
11. ***Appendix 2: Glossary.*** Extracts from the UML 1.3 glossary + a few proposals

My conclusion

1. ***The Collaboration abstraction is, and should be, an integral part of UML. It complements the class abstraction and is particularly useful for architectural models on all levels and for modeling patterns and frameworks.***
2. ***The abstract syntax of the specification level Collaboration and its elements in UML 1.3 is basically OK, but the instance level collaboration needs to be added and linked to the specification level to complete the semantics definition.***

Acknowledgements
My sincere thanks to Alistair Cockburn, Sridhar Iyengar, Bran Selic, James Rumbaugh, Jos Warmer, and Gunnar Overgaard for valuable comments.

2 The class

UML 1.3 defines the notion of *class* as follows:

class *[UML 1.3 glossary]*
A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment. See: *interface*.

The UML class abstractions can be described as follows:

1. Objects are encapsulations of data and behavior.
2. Objects are instances of classes.
3. Associations between two or more classes specify links between their instances.
4. An operation describes a service that can be requested from an object to effect behavior.

An example illustrates what this definition covers and doesn't cover. The following class diagram illustrates how a *Person* is the child of a *Man* and a *Woman*:

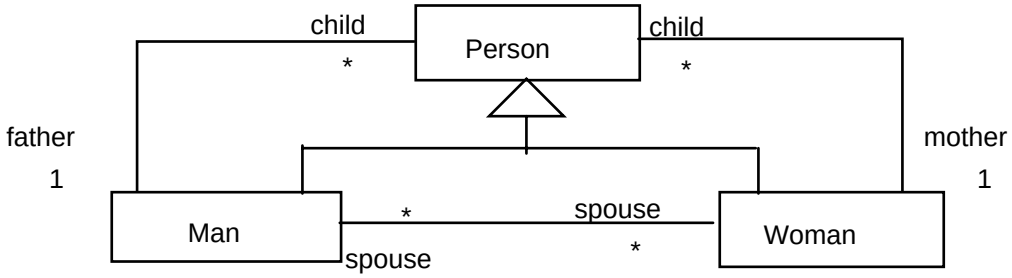


Figure 1. A class diagram

A very powerful model capturing a world of information in three simple classes. The number of instances of class *Person* in the world today exceeds 6 billion. In addition, there are all the people of the past. Should we guess 9 billion instances all told? About half of them are also instances of class *Man*, another half instances of class *Woman*. A person has exactly one *mother* and one *father*; both men and women can have many children. The many-to-many association between *Man* and *Woman* has no moral implications; it is sufficient that a widow or widower may remarry.

What about behavior? The class abstraction models attributes and operations. So it describes what the object knows and the services it offers to its environment. Let's add a *cashInHand* attribute and two simple operations:

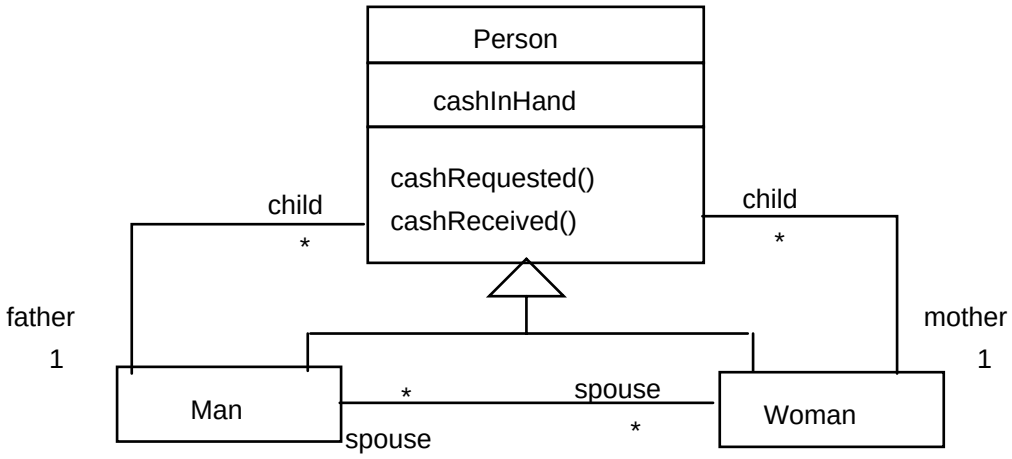


Figure 2. A class diagram with attributes and operations

The class abstraction deals in sets of Instances and sets of Links. This makes it hard to model how an object *sends* a stimulus to another object; some kind of a broadcast message is the closest we can get. In the model above, a person can own cash; receive cash, and be asked for cash. But we cannot describe that a person gives cash to another person because there is no way of identifying the particular receiver among the many billion of possible receivers.

3 The instance level Collaboration

Objects are often described as entities having state, behavior, and identity. Objects send stimuli to other objects in order to trigger their services. The class abstraction hides two of these notions; namely the notion of object identity and the notion of an object sending a stimulus to another object. These two characteristics are closely related: the specification of a message send includes the specification of the identity of the message receiver.

Despite its undisputed success, the class abstraction is based on a *choice* of what is essential and what isn't. It is both possible and useful to make other choices that will lead to other abstractions. In particular, the Collaboration abstraction models object identity and message sending. This makes it useful for modeling the behavior of a system of collaborating objects since we can model both the sending and the receiving messages.

In the Collaboration abstraction, the "essence" of an object is as follows:

1. An object has a unique identity.
2. An object collaborates with other objects by receiving stimuli from them and by sending stimuli to them.
3. An object is encapsulated so that its internal construction is invisible from its outside. The visible behavior consists of the stimuli it can receive from its collaborators and the stimuli that each of them can cause it to send to its collaborators. The state is only visible indirectly: a state changes causes a change in behavior. (e.g. that the object returns a different value in response to a query).

UML 1.3 distinguishes clearly between instance level collaborations and specification level collaborations. But this distinction is only apparent in the textual definitions of semantics and in the notation. *The notion of an instance level collaboration is not shown in any syntax diagram. I have tried to remedy this with a new metamodel diagram given in Appendix 1.*

A collaboration can be defined as follows:

Collaboration *[My proposal]*
A Collaboration describes how a number of objects work together for a common purpose. There are two aspects. The structural aspect is a description of the responsibilities of each object in the context of the overall purpose of the collaboration; and also the links that connect the objects into a communicating whole. The dynamic aspect is a description of how stimuli flow between the objects to achieve the common purpose.

The meta-language of UML semantics is English, so I need to add:

structure *[Webster]*
2b: something arranged in a definite pattern of organization <a rigid totalitarian structure>
4b: arrangement or interrelation of parts as dominated by the general character of the whole <economic structure>

Consider a CollaborationInstance with three objects: My father, my mother, and myself. The names of these objects are Bjarne, Gina, and Trygve as shown in this figure:

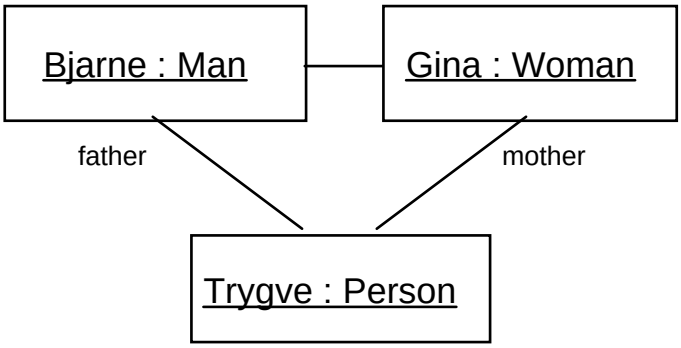


Figure 3. An instance level collaboration

It is now meaningful for me to ask my mother for some cash and get it:

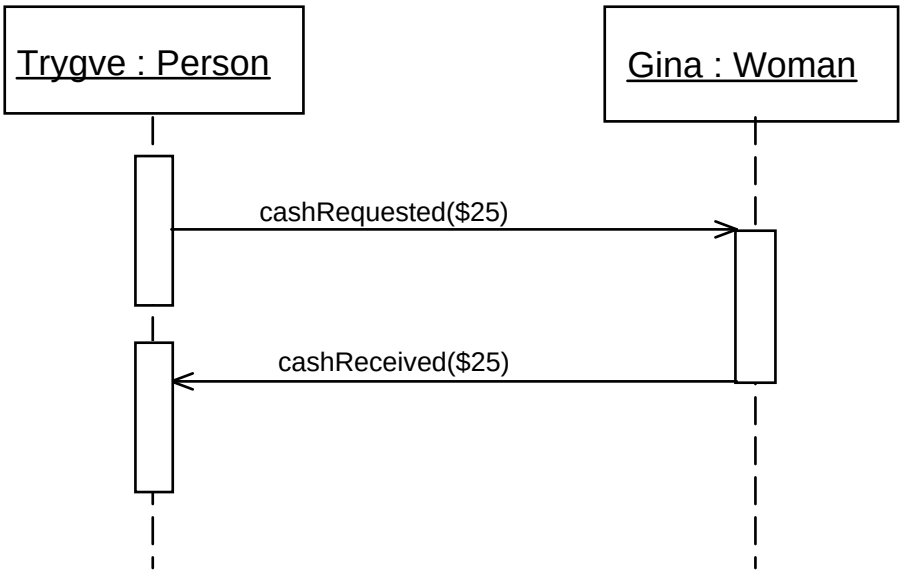


Figure 4. An instance level sequence diagram

This is beautifully concrete. There is no doubt as to who I ask for money, and the money is payed to me, not my brother. An instance level Collaboration can be very illuminating for describing the architecture and behavior of simple object structures. But it is too specific for more complex systems. For example, strictly interpreted the above model applies to me and my mother only. We would need a new model to show how my brother can get money from our mother. The *specification level abstraction* lifts the power of describing system behavior to a more general level.

4 The specification level Collaboration

We generalize the idea of a collaboration by representing the objects by *ClassifierRoles*, naming them by the name of the *position* they hold in the object structure.

position *[Webster]*
4a: the point or area occupied by a physical object <took her position at the head of the line>
5a: relative place, situation, or standing <is now in a position to make important decisions on his own>

Our simple family example again:

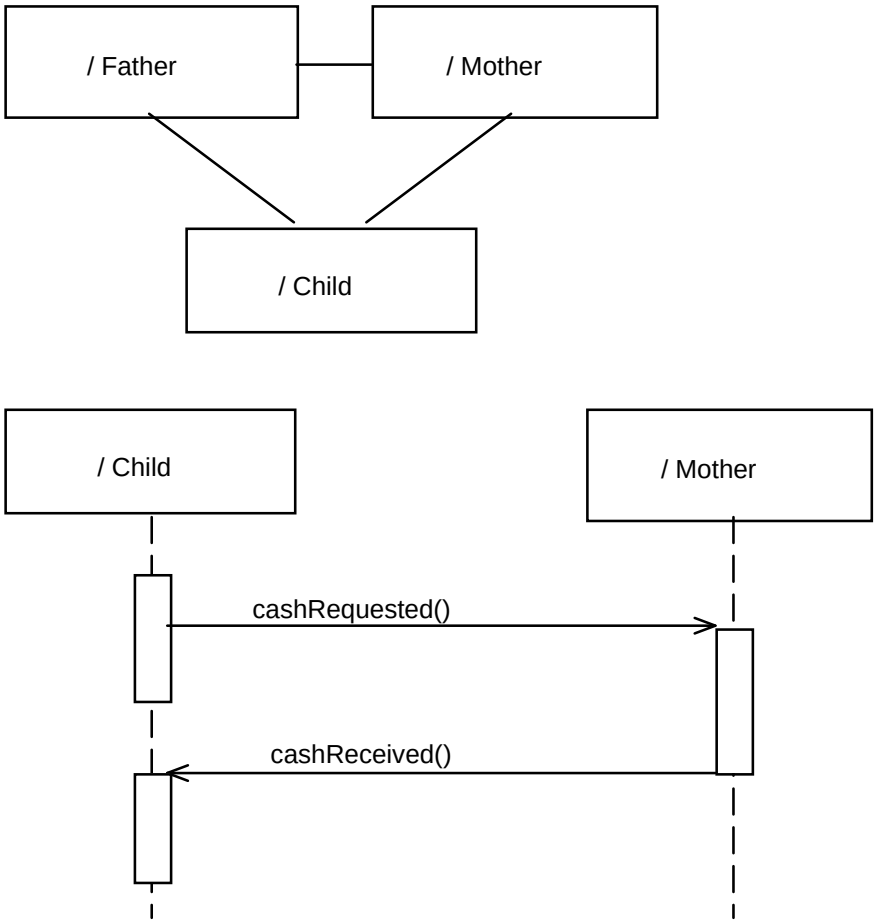


Figure 5. A specification level collaboration with sequence diagram

The object has been replaced by a slot that can hold an object; it names the role that the object plays when it occupies this position. The slots are called *ClassifierRoles,* they form a pattern that is instantiated again and again: Even at my ripe old age I play the Role of Child in relation to my Mother and Father. I also play the role of Father in relation to my daughter. etc. etc.

Notice that the parts have very little meaning when isolated from the whole. Indeed, in UML 1.3 the ClassifierRoles form a composition aggregate under the Collaboration; they are considered meaningless outside this context.

Instantiating class *Man* gives an instance of *Man*.

In contrast, it is meaningless to instantiate a ClassifierRole in isolation. The role gets its meaning from its position in the Collaboration and can only be instantiated in this context. For example, instantiating the */Father* role involves the following:

1. Select a suitable factory object.
2. Ask the factory object for an instance that has the features needed to play the role.
3. Connect the resulting object into the collaboration structure. For example, this may involve linking the father object to the child and mother objects, as well as linking the mother and child objects to the father object.

The definition of a Collaboration still stands:

Collaboration *[My proposal]*
A Collaboration describes how a number of objects work together for a common purpose. There are two aspects. The structural aspect is a description of the responsibilities of each object in the context of the overall purpose of the collaboration; and also the links that connect the objects into a communicating whole. The dynamic aspect is a description of how stimuli flow between the objects to achieve the common purpose.

We also need to define how the model elements on the specification level relate to the elements on the instance level:

ClassifierRole *[My proposal]*
A named slot for an object participating in a specification level Collaboration. Object behavior is represented by its participation in the overall behavior of the Collaboration. Object identity is preserved through this constraint: "In an instance of a collaboration, each ClassifierRole maps onto at most one object."

Note that the constraint is asymmetrical. An object can play several roles in one or more Collaborations.

The whole is greater than the sum of its parts. So the focus is on the collaboration as a whole because it has a value that exceeds the collective value of the objects taken separately. The important questions that can be asked from a collaboration are questions such as "What does it achieve?", "What are its objects?", "What are their responsibilities?", and "How do they interact?". A question that is of secondary importance is "How are its objects constructed?". This question is answered by giving a reference to the appropriate class or classes. Conversely, a class description can include a reference to relevant ClassifierRoles and their Collaborations.

A ClassifierRole may be thought of a a slice of a Class. But this is an oversimplification; it ignores its essential relationship to the Collaboration as a whole. A better metaphor is to think of a ClassifierRole as specifying a slice of an object; namely its AttributeLinks, Links, and behavior that it needs as a participant in the collaboration.

5 Separation of concern

In the last chapter of *A discipline of programming* from1976, Dijkstra said that:

"To my taste the main characteristic of intelligent thinking is that one is willing and able to study in depth an aspect of one's subject matter in isolation, for the sake of its own consistency, all the time knowing that one is occupying oneself with only one of the aspects. ...

"The other aspects have to wait their turn, because our heads are so small that we cannot deal with them simultaneously without getting confused. This is what I mean by 'focusing one's attention upon a certain aspect'; it does not mean completely ignoring the other ones, but temporarily forgetting them to the extent that they are irrelevant for the current topic. ...

"Such separation, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts that I know of. ...

"I usually refer to it as 'separation of concerns', because one tries to deal with the difficulties, the obligations, the desires, and the constraints one by one. ...

We will explore three techniques for separation of concern that are more or less available in UML:

1. *Filtered class diagrams offer unspecified separation.* A class diagram can show a filtered view of an arbitrary selection of classes.
2. *Components separate on services.* A component offers certain services (operations) to its clients while it hides the realizations of these services.
3. *Collaborations separate on system behavior.* One collaboration model describes how a system of interacting components realize one or more operations or use cases.

5.1 Filtered class diagrams offer unspecified separation

For an interesting system, the complete class diagram will be very complex. It is therefore permissible to show arbitrarily filtered views of the diagram. Classes, features, and associations may be hidden in order to highlight some interesting aspect of the system

In UML 1.3 *Semantics*, section 2.3.4, we find:

¤ *Every time a word coinciding with the name of some construct in UML is used, that construct is referenced.*

So it is quite simple, if somewhat tedious, to find out the full meaning of a term such as *classifier.* Start on the first page and read the complete document carefully. Collate all the places where the term is used, and you have the complete description.

Filtered class diagrams is the weakest technique for separation of concern. When possible, it should be replaced by one or both of the other techniques.

5.2 Separation of concern with components

There are many definitions of components such as Enterprise Java Beans. To my mind, a component is essentially a "super-object"; an encapsulated realization of a service:

1. A Component has a single access point (an object ID) and offers a well-defined interface to its clients.
2. A Component is reused by cloning
3. A Component does not make assumptions about its clients
4. A Component plays a standardized role within a container
5. Tools are used to deploy components and compose systems

Components serve the separation of concern to the extent that they hide complexity and offer well-defined interfaces to their clients.

The following figure illustrates two chained components, the second one being itself composed from three components.

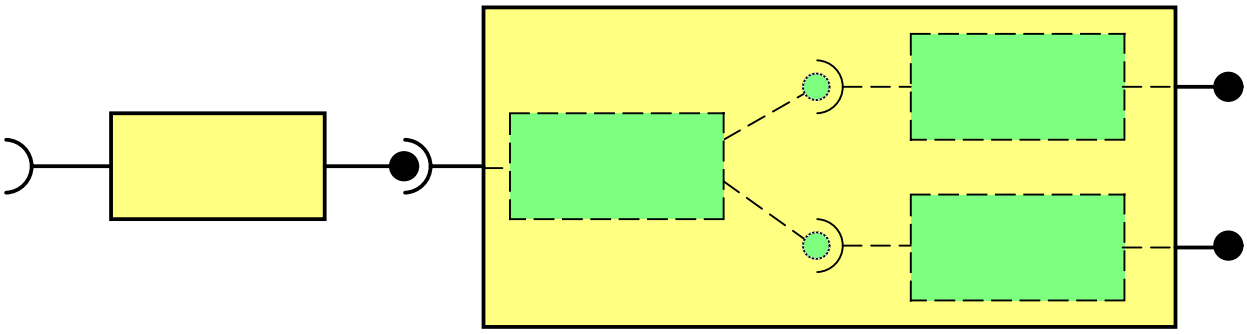


Figure 6. The object nature of components

Standards for shipping objects are still immature. We therefore usually package components in a more primitive form. An example is to use a Java .jar-file containing the necessary classes and other resource files together with rules for their instantiation.

One can think of the component technology as a horizontal separation of concern as illustrated in the following figure.

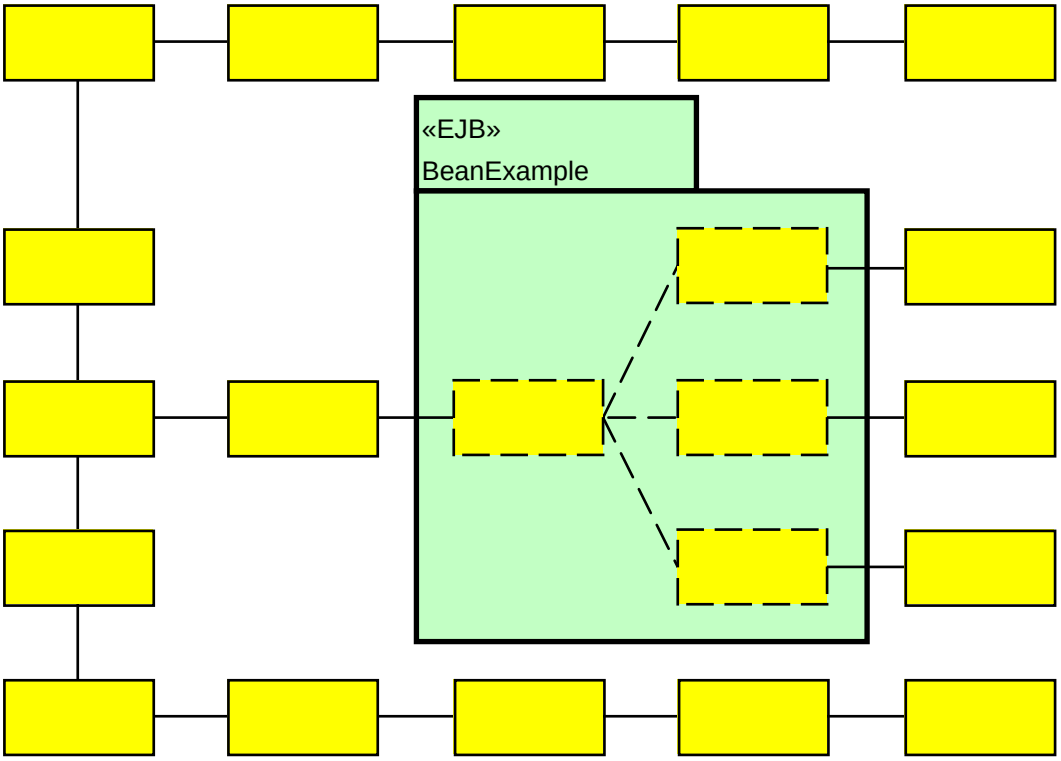


Figure 7. Horizontal separation of concern by object clustering in the horizontal plane.

Work is in progress to find more precise ways of representing components in UML. Two possible starting points are the *UML component* and the *UML subsystem.* For our purposes, the subsystem is the best choice because it is a child of classifier. It can therefore be the base of a ClassifierRole and our semantics is still valid.

5.3 Separation of concern with collaborations

The collaboration models how a society of collaborating objects realizes certain behavior. The separation of concern is achieved by letting each collaboration specify the part of the total behavior and the state that supports this behavior:

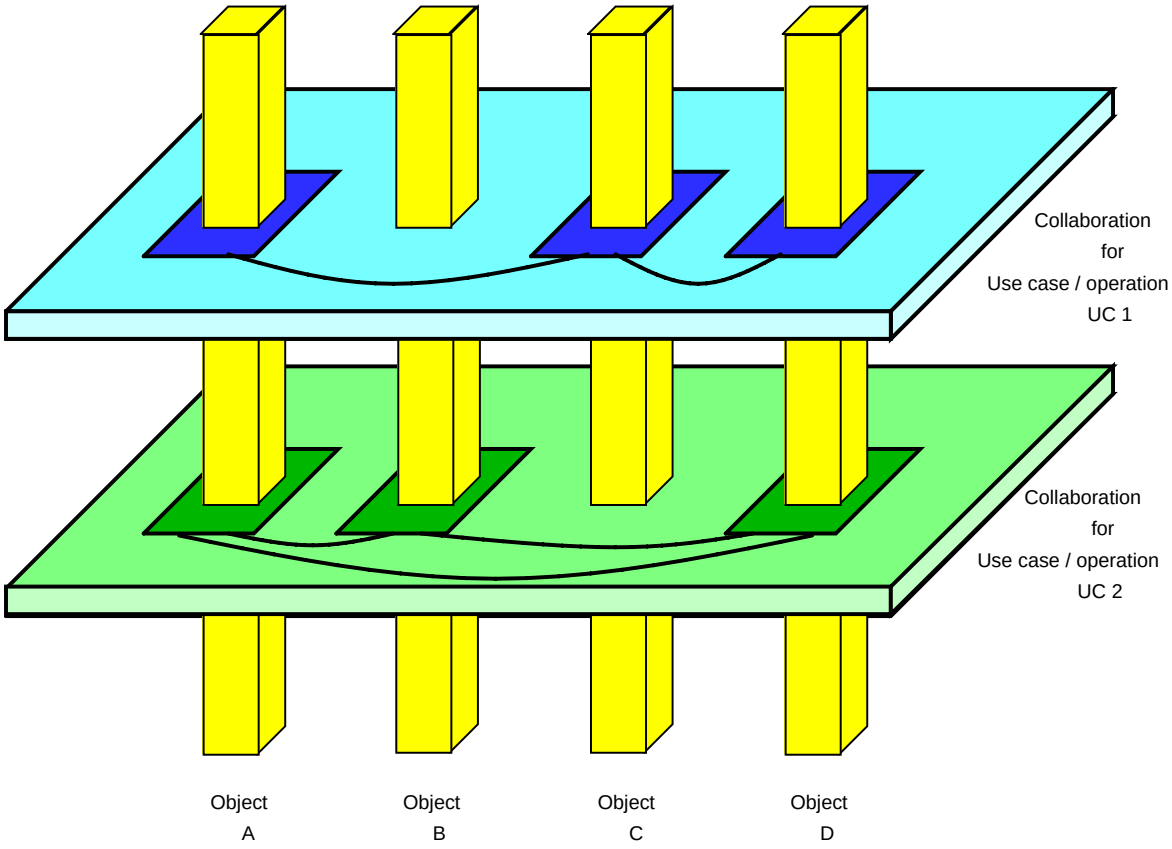


Figure 8. Vertical separation of concern by collaborations

6 Higher level collaboration abstractions with virtualRoles

Consider the graph of an activity network as shown in this example:

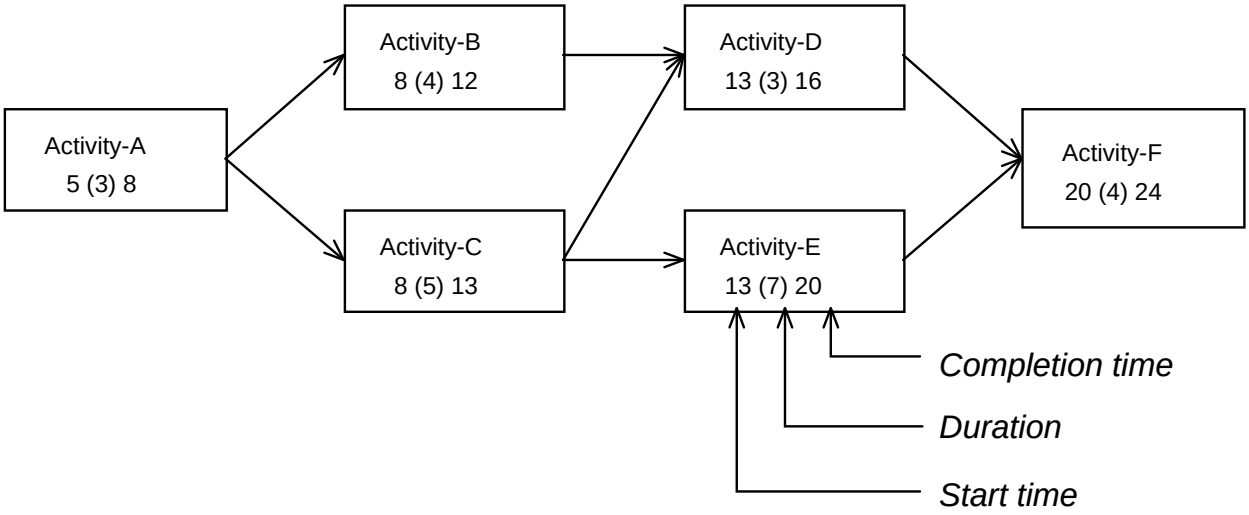


Figure 9. An activity network.

A program implementing such networks can take many forms. We assume an object oriented solution where we separate the presentation from the model. Here are three possible designs:

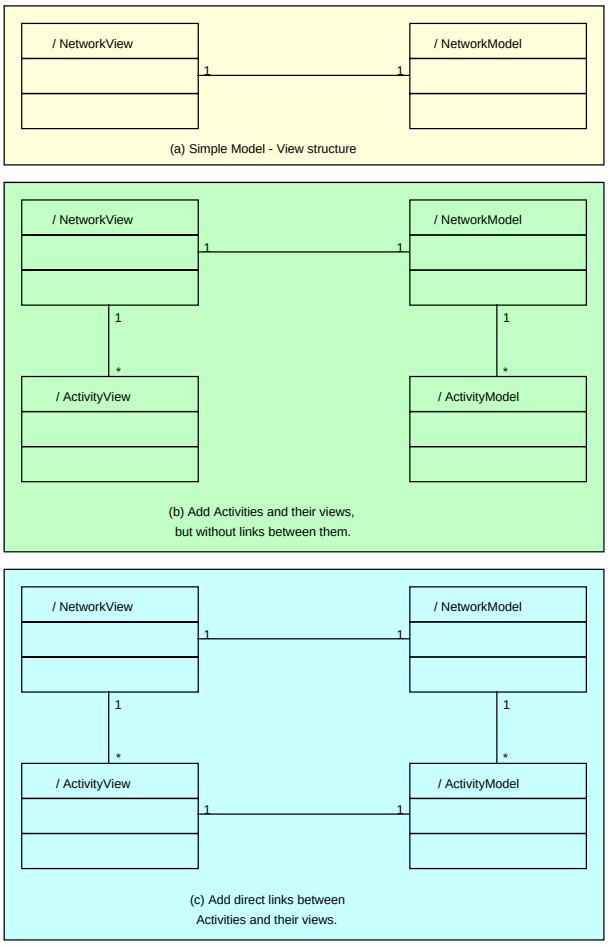


Figure 10. Three designs for the activity network program.

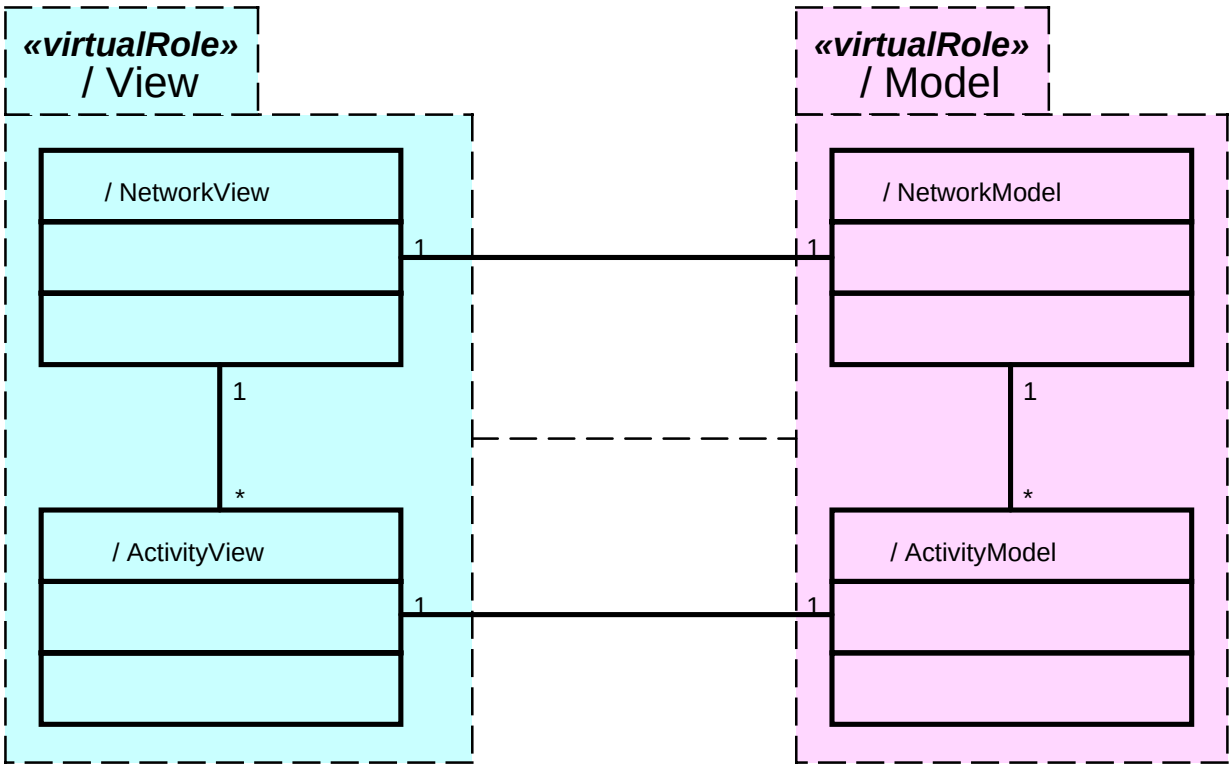
Many other solutions are possible. What they have in common is that the choice is a design decision that we would probably like to postpone to a late stage in the design process.

So how do we express a collaboration model where this decision is left open?

A solution is to adapt the OOram notion of a *virtualRole.* A virtualRole is a symbol that represents an arbitrary set of roles without in any way altering the semantics of the model. In UML, we can use the *package* meta-object and constrain it to contain a structure of ClassifierRoles. We could, for example, define a stereotype of package called ***«virtualRole».*** The above three design solution and many others can then be represented by a single collaboration as shown below.

Figure 11. Abstract solution with virtualRoles that hide design details.

The three original collaborations are different expansions of this model. Example (c) is expanded below. We see that each virtualRole here happens to be expanded into two ClassifierRoles. Further, the line connecting the two virtualRoles is expanded into two AssociationRoles.



(c) Direct links between
Activities and their Views.

Figure 12. Example of expanded model

In conclusion: VirtualRoles permit us to postpone decisions about the detailed object structure without in any way weakening the collaboration semantics.

7 Abstract interactions

The notion of a *message* is defined as a formal entity in UML 1.3. Revisiting the example in the previous example; we see that determining the exact protocols and interactions is an even more detailed decision than determining the objects. I suggest using the UML *comment* meta-object to indicate a composite message interaction. This comment symbol shall be associated with the AssociationRole or AssociationRoles that are involved in the interaction. Here is an example:

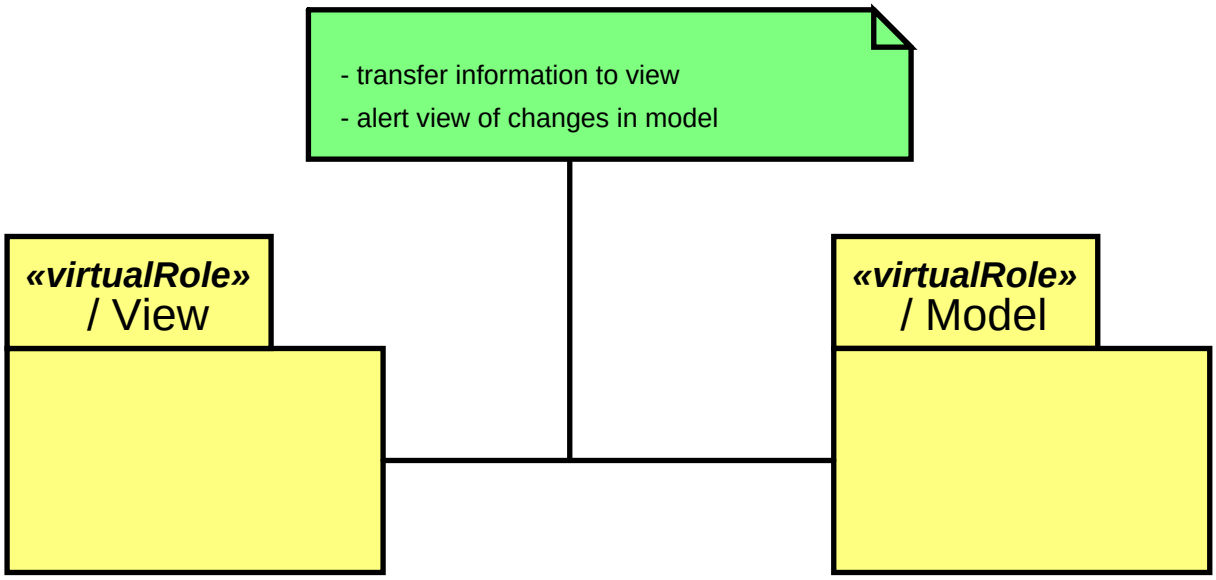


Figure 13. Abstract solution with virtualRoles that hide design details.

In the detailed design, these two functions will have to be expanded into two interactions.

8 Typing the AssociationEndRoles

The object *Links* on the instance level collaboration are mapped onto *AssociationRoles* on the specification level. Similarly, the *LinkEnds* are mapped onto *AssociationEndRoles*. The *AssociationEndRole* is a child of an *AssociationEnd*. In the Core package, an *AssociationEnd* can be *typed* with a *classifier*, e.g., an *interface*.

The practical consequence of this is that we may type the reference that a *ClassifierRole* has to a collaborator. This gives much finer grained control than merely typing the *ClassifierRoles* themselves.

Here is the cash transfer example with Interfaces added to the specification level collaboration:

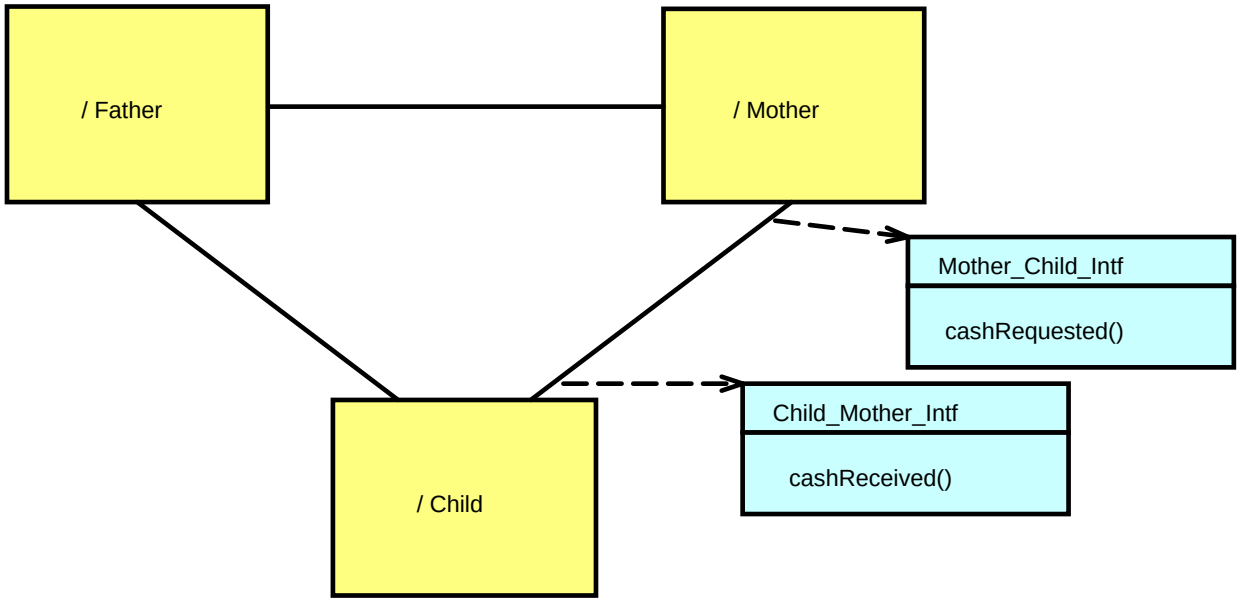


Figure 14. Typing the AssociationEndRoles

This model is more specific than the class diagram in section 2 because it says that only a */Child* can ask for cash from its */Mother*. It also says that any class implementing the */Mother* role must understand *cashRequested()*. Likewise, any class implementing the */Child* role must understand *cashReceived()*. The classes of section 2 are satisfactory, but other solutions are possible.

As another example, consider a visual hierarchy in the Java *awt* and *swing* libraries. The Collaboration on the specification level has two ClassifierRoles: /Container and /Part. The /Container role may be realized by *java.awt.Container* or any of its subclasses, while /Part may be realized by *java.awt.Component* or any of its subclasses. (*Container* is subclass of *Component*, so this realization permits recursion). Notice the "may be realized by", any class that fulfills the obligations of the ClassifierRoles can be used to realize them.

(It is not a good idea to include the class names in a general Collaboration model that defines the mechanism because it unduly restricts the implementation. But it may be useful to include them in the documentation of a particular program.)

It is often useful and instructive to distinguish between the stimuli flowing down the hierarchy from the stimuli that are designed to flow up towards the root. A stimulus that typically flows down the hierarchy is the display command (*print(Graphics g)*), and an architectural rule could be to restrict its use to being sent from the container and down. A stimulus that typically flows up the hierarchy is *invalidate()*, it marks this component and all parents above it as needing to be laid out.

The point here is not the particular decisions about who is allowed to send what, but that it illustrates the such considerations can be useful for determining system architecture and design.

9 The Object-Role-Class trichotomy

An object plays a ClassifierRole in a CollaborationSpecification. If we "open up" the ClassifierRole, we will find that it can be realized by one or more classes. As an example, consider the following instances of the *family* Collaboration above:

1. Bjarne /Father : Man & Gina /Mother :Woman & Trygve /Child : Man
2. Trygve /Father : Man & Bjorg /Mother :Woman & Johan /Child : Man
3. Trygve /Father : Man & Bjorg /Mother :Woman & Borghild /Child : Woman

Here are the corresponding Collaboration diagrams. First the "pure" CollaborationSpecification:

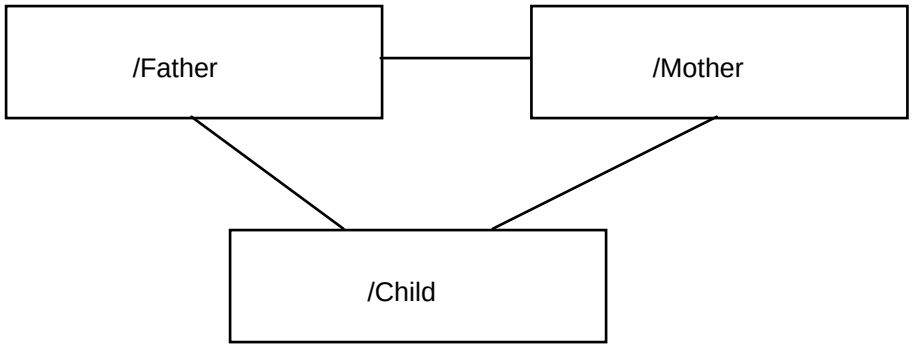


Figure 15. A "pure" specification level collaboration

We next add information about Role realization. *Note that this does not add essential information to the Collaboration as such assuming the Collaboration included interface information.*

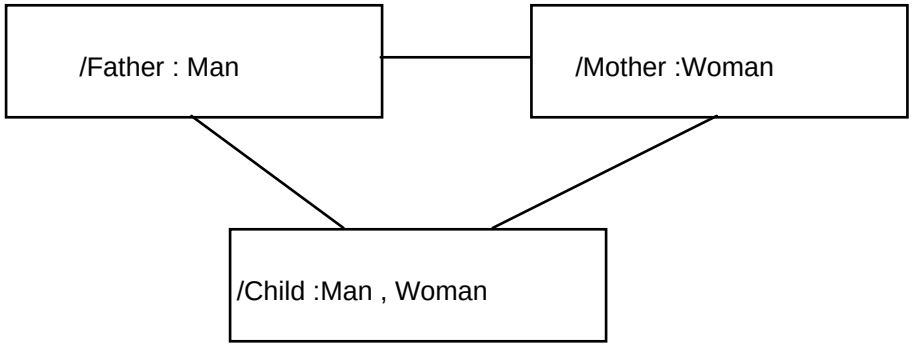


Figure 16. Include specification of class or alternative classes

So Father is of class *Man,* Mother of class *Woman*, and Child of either class.

We finally look at a few instances of this Collaboration. We have, somewhat arbitrarily, decided to include the role and class names:

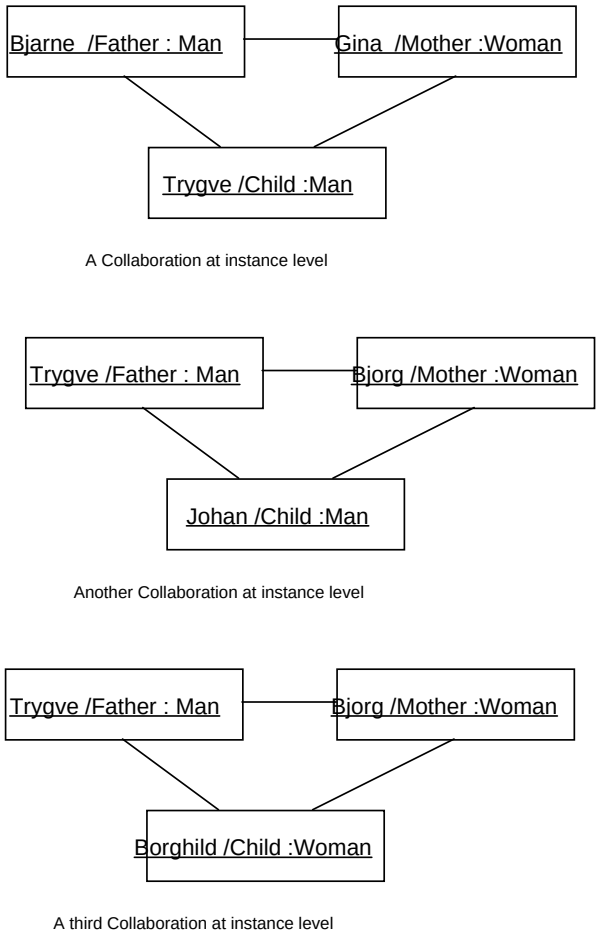


Figure 17. Instance level collaboration with specification of ClassifierRole and class

The object/role/class trichotomy is easy to explain. We can use a theater metaphor to explain the difference between role (ClassifierRole) and actor (object) and his capabilities (class). Or, usually even better, we can use a business organization metaphor: John can play the role of *SystemArchitect* in a certain project and belong to the class of *Programmers*. He may, of course, also play other roles in this or other projects.

10 System behavior, messages and interfaces

We should now have a clear picture of a Collaboration modeling a structure of interacting objects. But how do they interact?

UML offers two ways of modeling the avalanche of messages that flow from the sending objects to the receiving objects in a Collaboration: By Interactions or by State Machines. There is no magic; every message has a sender and a receiver. (Or, more precisely: The only magic is in the methods that are not first class citizens of a Collaboration model).

An example illustrates the importance of studying collaboration behavior. Consider the *instance level collaboration* shown to the right in the figure below. We can clearly associate

a state diagram with each of the objects. If these diagrams show stimulus sends as well as stimulus receives, we can match the stimulus sends in one object with the corresponding stimulus receives in the receiving object. We can then check to see that the collaboration as a whole is consistent by checking that the receiving objects are in the appropriate state when receiving the stimuli, and that they will subsequently send expected stimuli to the other objects. (Notice that when considering the behavior of a collaboration, we ignore any behavior that is outside this context).

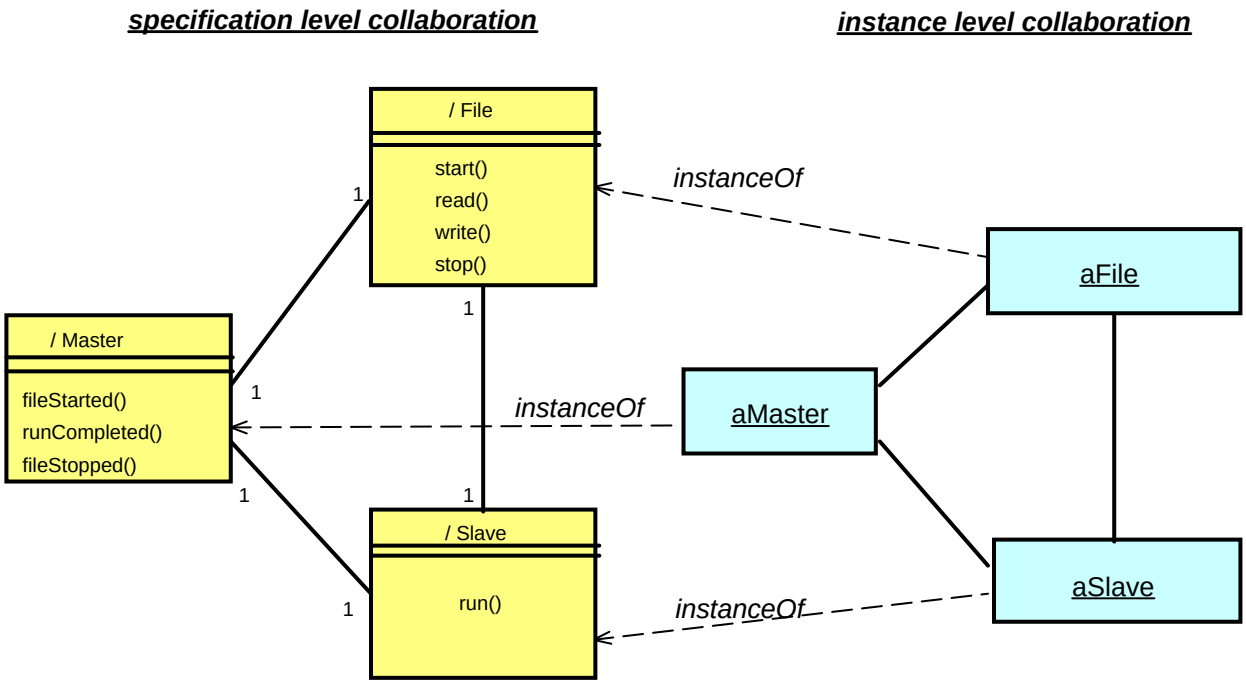


Figure 18. Specification and instance level collaborations example

A message sequence chart for opening the file, editing it, and finally closing it could be as follows:

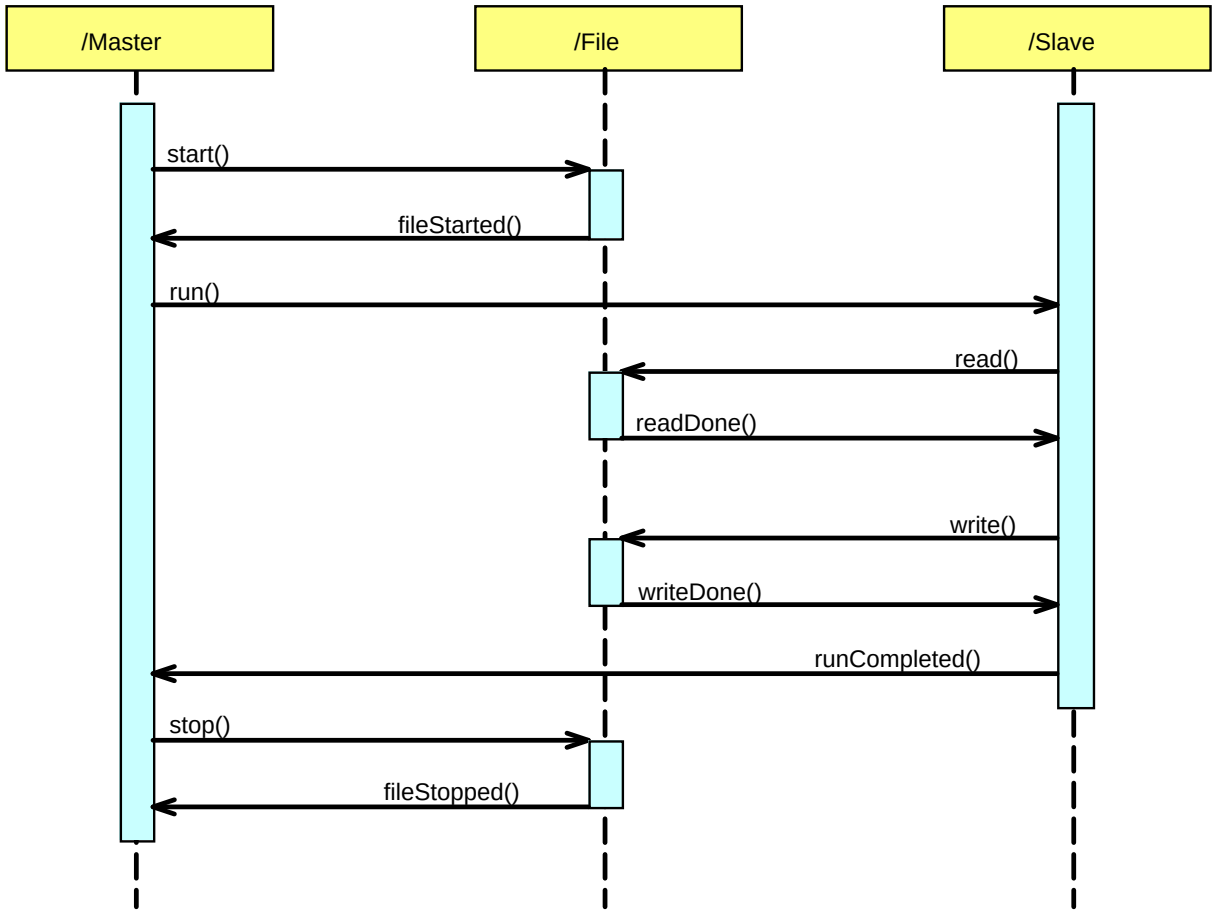


Figure 19. Specification level sequence diagram

This diagram is sufficient for this simple behavior; the file is opened, used, and closed. For more complex behavior, we may need to use state machine diagrams to convince ourselves that the system works properly in all situations. We first create state machines for each ClassifierRole:

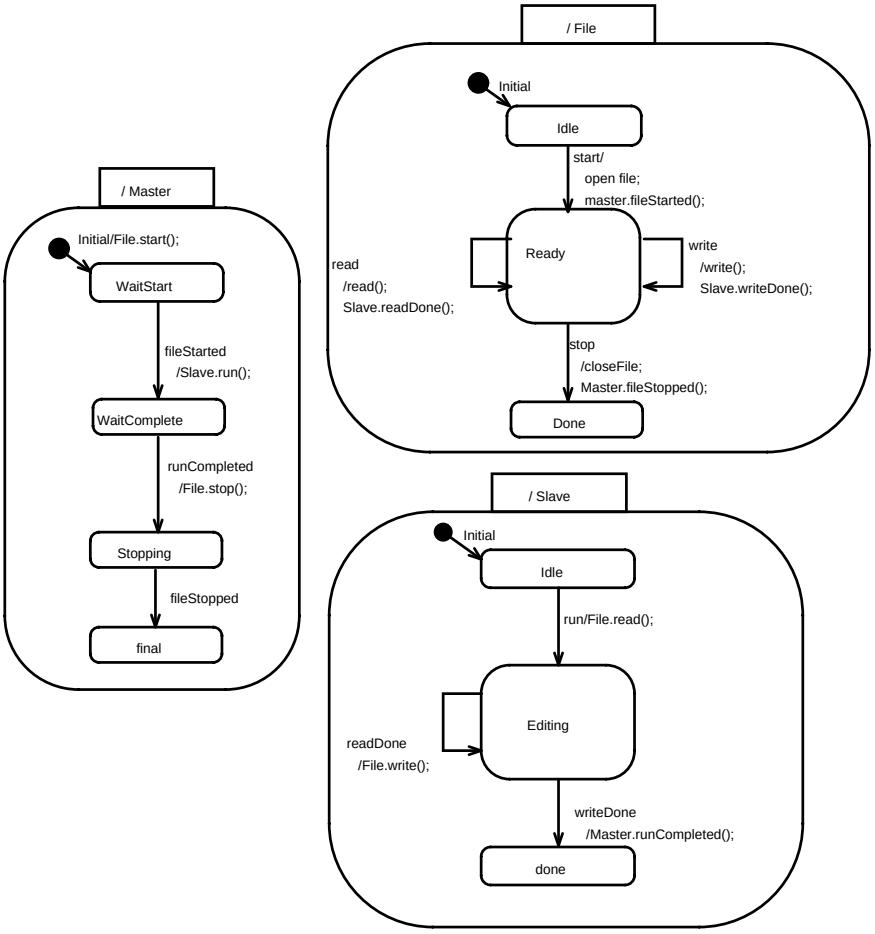


Figure 20. State machines for each of the three ClassifierRoles.

We now reach the crux of the system behavior. We create an overall state machine description of the collaboration as a whole by joining the message sends with the corresponding message receives. We get the following thread of execution:

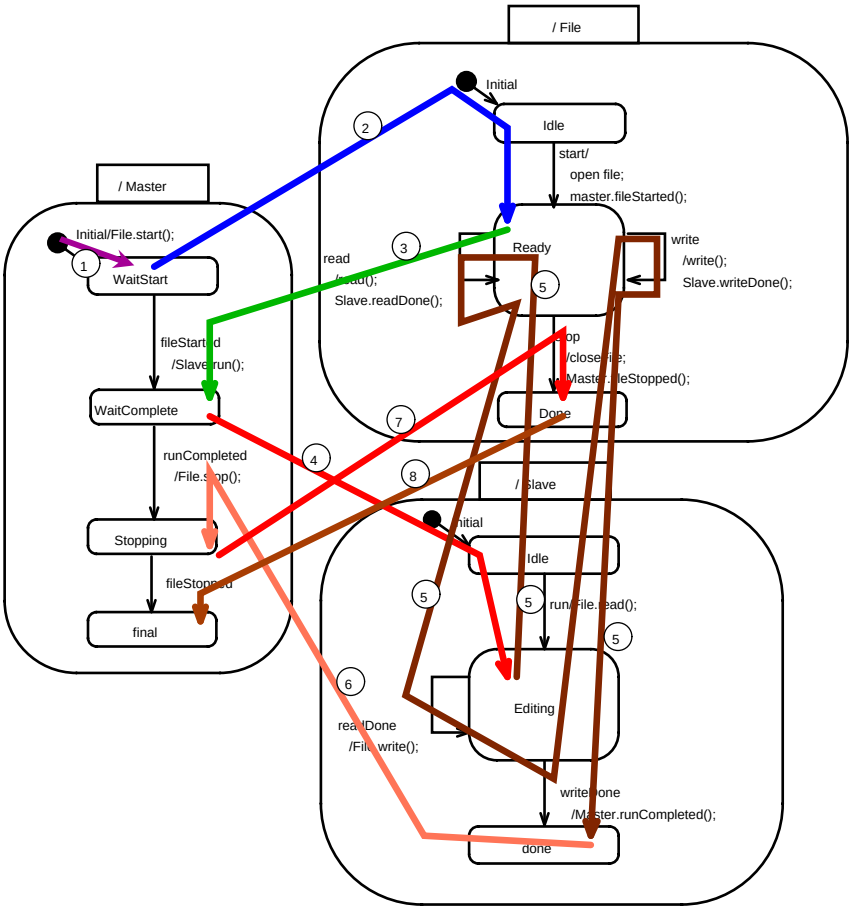


Figure 21. The thread of execution

More formally, we can derive a state diagram for the collaboration as a whole. Each state is defined as the combined states of the three roles; we use the notation *role1-state1 & role2-state2 & ...* . We show every change in the combined state, and the actions associated with the transitions. The result is the following diagram:

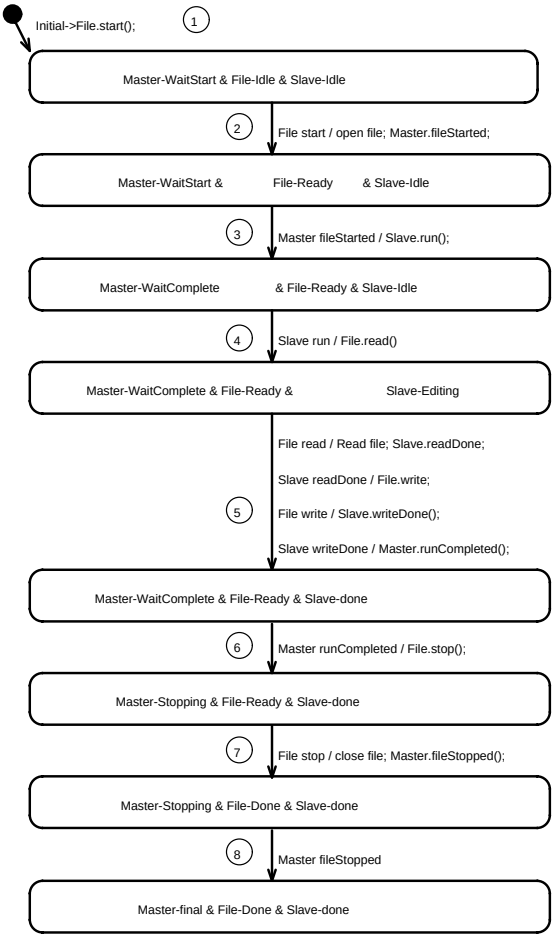


Figure 22. The state machine for the whole system.

This is a trivial example. More complex systems can easily lead to state explosion, making it impractical to draw the above diagram. We are somewhat helped by the separation of concern caused by restricting ourselves to the states that are relevant to the purpose and processes of the a single collaboration. We can get further help by using a special version of the Harel state charts that were proposed by Egil Andersen in his doctoral thesis on role modeling [*ftp://ftp.nr.no/pub/egil/ConceptualModelingOO.ps.gz*]. It can also be argued that we are skating on very thin ice if we cannot oversee the behavior of our system.

In any case, a tool can generate the system behavior from the individual ClassifierRole behaviors and use it to check system consistency. So we now have three levels of consistency checking:

1. The normal typing system can be used by the compiler to check that objects only send messages that will be understood by the receiver.
2. Typed AssociationEnds can be reflected in correspondingly typed references. (The Java language permits typing references with interfaces).
3. A set of state machines that specify the behavior of individual objects can be automatically checked for mutual consistency by a suitable tool.

Up to this point we have only considered very simple cases where there was a one-to-one mapping from role to instance. Now consider that there are any number of slave objects:

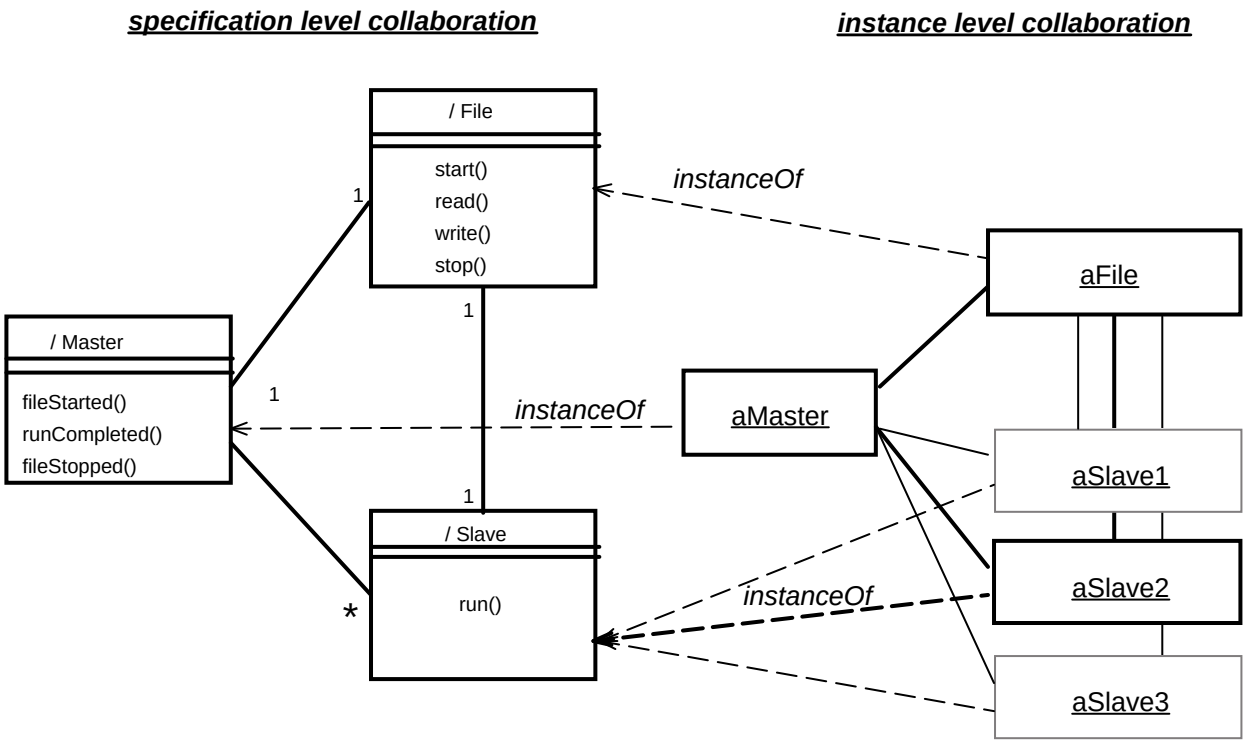


Figure 23. Multiple instances are represented by an archetypical example.

The definition of a collaboration still holds. So one of the slaves is promoted to represent them all. In the above figure, we have designated **aSlave2** to represent the set. The specification means that the other slaves are to behave consistently; that any slave could have been chosen to represent them all; but that once selected, it must represent them throughout the argument. It also means that collaborators waiting for some message from the slave must wait until it has received the message from all the slaves. The complete definition of a Collaboration will then be as follows:

Collaboration *[My proposal]*
A Collaboration describes how a number of objects work together for a common purpose. There are two aspects. The structural aspect is a description of the responsibilities of each object in the context of the overall purpose of the collaboration; and also the links that connect the objects into a communicating whole. The dynamic aspect is a description of how stimuli flow between the objects to achieve the common purpose.

ClassifierRole *[My proposal]*
A named slot for an object participating in a specification level Collaboration. Object behavior is represented by its participation in the overall behavior of the Collaboration. Object identity is preserved through this constraint: "In an instance of a collaboration, each ClassifierRole maps onto at most one object."

If there can be more than one instance corresponding to a given ClassifierRole, one of these instances is selected to represent them all. The other instances are constrained to behave in a way corresponding to the selected representative.

The net result is that the above message sequence chart and state diagrams are still valid and need not be changed in any way.

App. 1 Instance level and specification level collaborations

UML 1.3 makes a clear distinction between collaborations at the instance level and at the specification level:

"A collaboration may be presented at two different levels: specification level or instance level. A diagram presenting the collaboration at the specification level will show classifier roles and association roles, while a diagram at the instance level will show instances and links conforming to the roles in the collaboration." [Section 2.10.4 on page 2-112].

"A collaboration diagram can be given in two different forms: either at specification level (the diagram shows ClassifierRoles, AssociationRoles, and Messages) or at instance level (the diagram shows Objects, Links, and Stimuli). The former presents the roles and their structure as defined in the underlying Collaboration, while the latter focuses on instance that conforms to the roles in the Collaboration." [Intro to Part 8: Collaboration Diagrams on p. 3-109]

"A collaboration diagram shows a graph of either Objects linked to each other, or ClassifierRoles and AssociationRoles; it may also include the communication stated by an Interaction. A collaboration diagram can be given in two different forms: at instance level or at specification level; it may either show Instances, Links, and Stimuli, or show ClassifierRoles, AssociationRoles, and Messages ..." [Section 3.65.2 Notation on page 3-111]

A slight difficulty is the use of *object* and *instance* as synonyms. They are different according to the Core package. *Object* is defined as one of the subclasses of *Instance*, the others being *DataValue*, *ComponentInstance* and *NodeInstance*. I have not tried to resolve this question here.

More important is that there is no abstract syntax that supports the instance level collaboration. Since I believe it to be essential to the proper understanding of collaborations, I suggest a definition here.

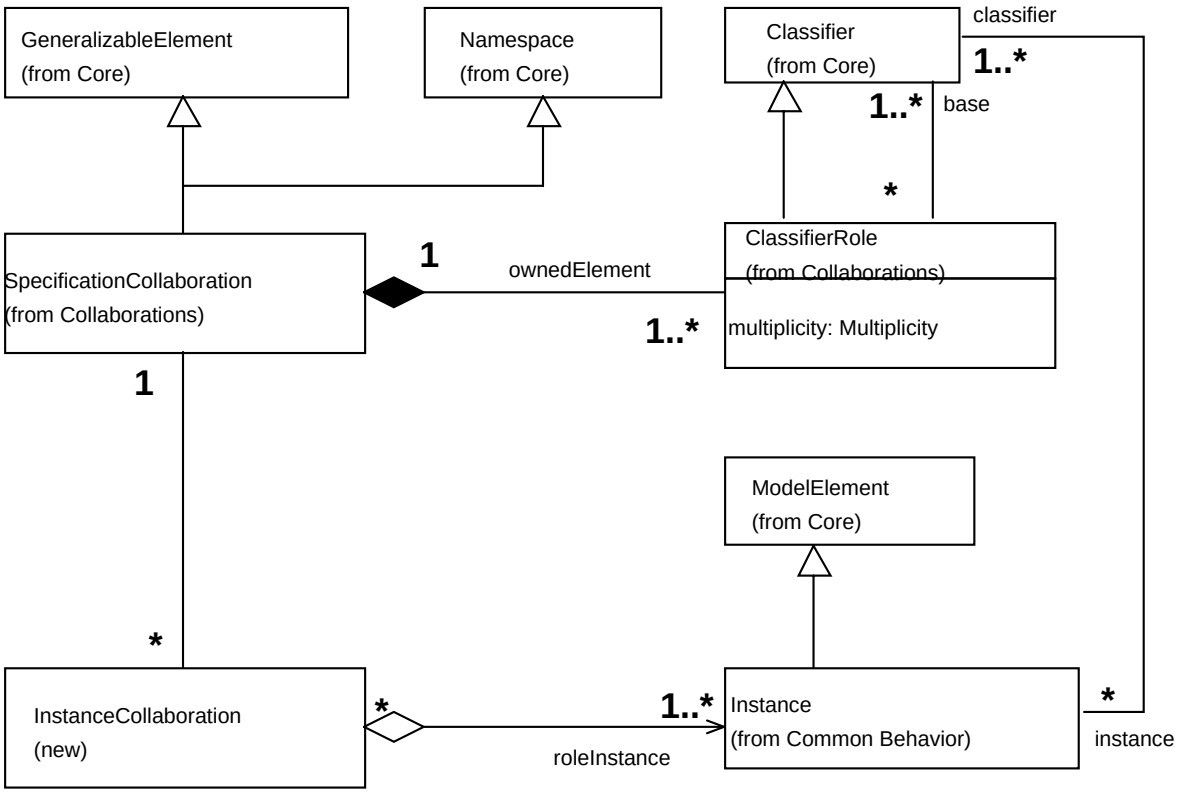


Figure 24. UML 1.3 extension for the instance level collaboration

Compared to UML 1.3, *InstanceCollaboration* is added because it is needed to define the Collaboration semantics and it is used in UML 1.3 semantics and notation.

Well-Formedness rules

"In an instance of a collaboration, each classifier role maps onto at most one instance of the classifier."

Expressed in OCL:

```
context InstanceCollaboration inv:
    self.specificationCollaboration.ownedElement->
      forAll( c : ClassifierRole |
        c.instance->select(i : Instance |
            self.roleInstance->includes(i))->size <= 1)
```

App. 2 A short glossary

1. **action** *[UML 1.3 glossary]*
   The specification of an executable statement that forms an abstraction of a computational procedure. An action typically results in a change in the state of the system, and can be realized by sending a message to an object or modifying a link or a value of an attribute.

2. **association** *[UML 1.3 glossary]*
   The semantic relationship between two or more classifiers that specifies connections among their instances.
3. **class** *[UML 1.3 glossary]*
   A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment. See: *interface*.
4. **collaboration** *[UML 1.3 glossary]*
   The specification of how an operation or classifier, such as a use case, is realized by a set of classifiers and associations playing specific roles used in a specific way. The collaboration defines an interaction. See: interaction.

   collaboration *[my proposal]*
   describes how a number of objects work together for a common purpose. There are two aspects. The structural aspect is a description of the responsibilities of each object in the context of the overall purpose of the collaboration; and also the links that connect the objects into a communicating whole. The dynamic aspect is a description of how stimuli flow between the objects to achieve the common purpose.
5. **instance** *[UML 1.3 glossary]*
   An entity to which a set of operations can be applied and which has a state that stores the effects of the operations. See: object.
6. **interface** *[UML 1.3 glossary]*
   A named set of operations that characterize the behavior of an element.
7. **message** *[UML 1.3 glossary]*
   A specification of the conveyance of information from one instance to another, with the expectation that activity will ensue. A message may specify the raising of a signal or the call of an operation.
8. **object** *[UML 1.3 glossary]*
   An entity with a well-defined boundary and identity that encapsulates state and behavior. State is represented by attributes and relationships, behavior is represented by operations, methods, and state machines. An object is an instance of a class. See: class, instance.
9. **operation** *[UML 1.3 glossary]*
   A service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.
10. **reference** *[UML 1.3 glossary]*
    1. A denotation of a model element.
    2. A named slot within a classifier that facilitates navigation to other classifiers.
    Synonym: *pointer*.

11. **role** *[UML 1.3 glossary]*
    The named specific behavior of an entity participating in a particular context. A role may be static (e.g., an association end) or dynamic (e.g., a collaboration role).

    ClassifierRole *[My proposal]*
    A named slot for an object participating in a specification level Collaboration. Object behavior is represented by its participation in the overall behavior of the Collaboration. Object identity is preserved through this constraint: "In an instance of a collaboration, each ClassifierRole maps onto at most one object."

    If there can be more than one instance corresponding to a given ClassifierRole, one of these instances is selected to represent them all. The other instances are constrained to behave in a way corresponding to the selected representative.

12. **signal** *[UML 1.3 glossary]*
    The specification of an asynchronous stimulus communicated between instances. Signals may have parameters.