The Emperor's New Use Case

Gonzalo Génova, Juan Llorens

Computer Science Department, Carlos III University of Madrid,
Avda. Universidad 30, 28911 Leganés (Madrid), Spain
{ggenova, llorens}@inf.uc3m.es
http://www.inf.uc3m.es/

Abstract. Use cases are intended to specify system behavior from the user's point of view. In UML, use cases are meta-modeled as classifiers, trying to fit them within the general object-oriented paradigm. Classifiers specify a set of instances, and use case instances are said to be occurrences of emergent behaviors, that is, concrete system-actor interactions. This idea poses some difficulties, since it is not clear how an interaction can have classifier features such as attributes, operations and associations. Therefore, we challenge the notion that use case instances are interactions. On the other side, if we proceed on to the complete specification of system behavior by means of use cases, we reach a notion of use case (a coordinated use of system operations) that is very close to the traditional role with an associated protocol interface, therefore concluding that use cases and protocols are not essentially different things.

Many, many years ago lived an emperor,
who thought so much of new clothes
that he spent all his money in order to obtain them;
his only ambition was to be always well dressed.
…
"But he has nothing on at all," said a little child at last.

Hans Christian Andersen, *The Emperor's New Suit* (1837) [1]

Introduction: in defense of use cases

Jacobson [9] originated the idea of use cases by observing that, despite the huge number of potential executions, most applications are *conceived* in terms of a relatively small number of typical interactions. Consequently, use cases have shown to be very useful to elicit user requirements: the user (or better, the stakeholder) explains in a simple way what he or she expects from the system to be built, by means of describing an interaction with the system, including the information supplied by the user, and the expected system answer. Usually, in this description it is revealed *how the user thinks about the system*, and what the fundamental concepts in the domain are, hence analysis classes are discovered. No doubt, the description of quasi-linear user-system interactions aids in understanding system functional requirements, even though the final system will surely not work in a quasi-linear form.

The next step for the software engineer is to formalize this simple interaction description into a true requirements specification that properly defines the expected system behavior, transforming user requirements into software requirements. If a use case is defined as the specification of a set of interactions, then we are faced with the following questions: Which interactions belong to the use case? What do all these interactions have in common, an executed interaction pattern, or a goal to be achieved?

The less abstract way to specify a use case is through the description of a small set of *typical interactions*, usually in textual form, such as main success scenario and main variant and exceptional behaviors [2]. If we stop the use case specification at this stage, then the interactions that we can say to belong to the use case are those that conform to these few interaction patterns. A more abstract way to specify the use case is by means of a full description of the allowed interactions. This requires a much more elaborated textual form, which in many cases resembles too much the use of low level pseudo-code, with all associated well-known problems; an improvement to this approach is the use of a graphical form to specify the allowed interactions, such as activity or statechart diagrams.

But software engineers cannot stop at this point. Beyond specifying the *interaction pattern*, the crucial point to obtain a true black-box view of the system is the identification of the *interaction goal*, so that any interaction that fulfills the goal will belong to the use case, no matter what the steps followed in the interaction are. The specification of the expected behavior through a *contract* (that is, pre- and post-conditions) is the only way a software engineer can reach the proper level of abstraction needed for a requirements specification. The expected functionality or service is not completely specified without identifying its goal; it is the goal what makes the related behavior coherent [11]. Moreover, the specification of an interaction pattern risks to compromise design issues, by focusing on the interaction pattern, rather than the interaction goal.

In other words, what the user really requires from the system (the true requirement to be elicited) is not the interaction, but the observable result, or goal: system functionality, at an abstract level, is given by the input/output relationship, not by the interaction performed. The typical interaction description is only a very useful method the user has to express in a simple way what he or she expects from the system, from where the requirements engineer has to elicit the true requirements. The interaction is relevant only to *illustrate*, to elicit requirements, but not to *specify* them. We must distinguish between *understanding* a requirement and *specifying* it: a small set of typical stories is not enough to specify the required system function. The *home, sweet home* for requirements engineers can be reached walking through the path of interactions, but stopping midway would leave the task unfinished.

This is reflected in how the definition of a use case has evolved in the UML[1], from the notion of "typical usage" to the notion of "specified usage". In UML 1, a use case is defined as the specification of "a sequence of actions, including variants, that the entity can perform, interacting with actors of the entity" [UML1, p. 2-132]. This

[1] In this article we compare versions 1.5 (March 2003) and 2.0 (August 2003) of the UML Specification [12, 13]. These documents will be quoted for clarity as "UML1" and "UML2", followed by page number.

definition is accompanied, some pages below, by the following "note", that does not strictly pertain to the definition: "A pragmatic rule of use when defining use cases is that each use case should yield some kind of observable result of value to (at least) one of its actors. This ensures that the use cases are complete specifications and not just fragments" [UML1, p. 2-140]. In UML 2 this recommendation has been integrated in the definition, yielding a much more refined and rigorous statement: "A use case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system" [UML2, p. 519][2].

In the next Sections we will try to go deeper in the notion of use case, as it has been formalized in the UML Specification.

1. Use case instances

Since Jacobson introduced them in his OOSE method [9], and specially after their adoption by the UML, use cases have proliferated in the Software Engineering industry as a means "to capture the requirements of a system, that is, what a system is supposed to do" [UML2, p. 511]. The UML 2, which in many aspects is so different from UML 1, has introduced few changes about use cases, apart from some minor clarifications. Less than 20 pages in the use cases Chapter are devoted to use cases and use case diagrams in the 640-pages Superstructure document [UML2, pp. 511-528], from which only four pages deal specifically with the notion of a use case [UML2, pp. 519-522]. The most significant improvement is the explicit introduction of the *subject*, represented as a system boundary, which is "the system under consideration to which the use cases apply" [UML2, p. 511], which may be "a physical system or any other element that may have behavior, such as a component, subsystem or class" [UML2, p. 519]. This notion of *subject* is meta-modeled as a classifier to which the use case is meta-associated [UML2, p. 512] (see Figure 1).

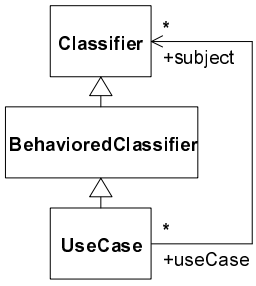

Figure 1. Use case and Subject in the UML 2 metamodel (extracted from Figures 312 and 401)

[2] Anyway, Jacobson et al. did not ignore either the importance of the observable result: "A use case is a set of transactions performed by a system, which yields an observable result of value for a particular actor" [10].

Since the beginning of OOSE [9], Jacobson et al. tried to conceptualize use cases within the general object-oriented paradigm. Therefore, in all versions of UML, use cases have been classifiers in the metamodel, that is, each use case is a specification of a set of instances[3]; in other words, a use case specifies the features (*intension*) that all its instances must conform to (*extension*). What are the instances of a use case? UML 2 gives an explicit answer to this question: "An instance of a use case refers to *an occurrence of the emergent behavior* that conforms to the corresponding use case type. Such instances are often described by *interaction specifications*" [UML2, p. 511]. Later on, we find: "a use case is the specification of *a set of actions* performed by a system", and "an execution of a use case is an *occurrence of emergent behavior*" [UML2, p. 520]. This idea was even more clearly expressed in UML 1: "A use case instance is the performance of a sequence of actions specified in a use case" [UML1, p. 2-133]. Summing up, a use case specifies a behavior, and its instances are concrete behaviors, or concrete sequences of actions.

One of the points that is not clear about this notion is whether the actions specified in the use case are system actions, actor actions, or both. Generally, it seems they are "actions performed by the system" [UML2, p. 520], or actions that "the subject can perform in collaboration with one or more actors" [UML2, p. 519], that is, system actions issued by an actor's message. This can be considered equivalent to an interaction, a sequence of messages interchanged between actor and system, which includes not only system actions, but, at least, also the actor's action to send a message. Does the use case describe isolated system behavior, or does the use case describe the actor-system collaboration? In any case, it is explicitly stated that internal actions of the system or the actor, which are not visible to one another, should not be included in the use case description: "use cases define the offered behavior of the subject without reference to its internal structure" [UML2, p. 519], "it is not possible to state anything about the internal behavior of the actor apart from its communications with the subject" [UML2, p. 520]. This contrasts, however, with recognized textual techniques to describe use cases, which include these internal actions [15], because they are useful to understand the interaction.

On the other side, UML gives another different, only implicit, answer to the same question, what are the instances of a use case? If we look at the instances that play the *role* specified by the use case, then the instances of a use case are the instances of the subject it applies to (remember the subject is a classifier itself). That is, if a use case specifies a behavior, then a classifier that realizes or implements this behavior may be said to be a *subtype* of the use case, and any instance of this classifier is an indirect instance of the use case, much in the same way as an instance of a classifier realizing an *interface* is an indirect instance of the interface [16]. We can put it in another way: since the use case specifies (the behavior of) a subject, therefore the instances (that play the role) of the use case are those of the specified subject, which is the physical system or element with behavior (a component, a subsystem or a class). That a use case specifies a role is stated at least in two places, where it is shown that the use case does not type the interaction, but one of the participants: "The behavior of a use case … may also be described indirectly through a Collaboration that uses the use case and

[3] "A classifier is a classification of instances — it describes a set of instances that have features in common" [UML2, p. 61].

its actors as the classifiers that type its parts" [UML2, p. 519]. "Use cases and actors may represent roles in collaborations" [UML2, p. 522]. Summing up, use cases have no direct instances; they only specify a behavior (a role) that can be realized (played) by instances of other classifiers, which can be considered as *indirect instances* of the use case.

And here is the contradiction (see Figure 2). On the one side, the *explicit notion* that a use case specifies a set of *interactions*; on the other side, the *implicit notion* that a use case specifies a set of *entities* (that play a role in an interaction).

- First notion: the use case specifies an interaction between actor and system, that is, a collaborative system-actor behavior. The use case types the interaction. Use case instances are occurrences of emergent behavior, that is, concrete system-actor interactions.
- Second notion: the use case specifies the behavior of the system, that is, the role the system plays in the interaction. The use case types the system, whereas the actor types the external agent interacting with the system. Use case instances are the instances of the subject the use case applies to, that is, any concrete system that conforms to the behavior specified.

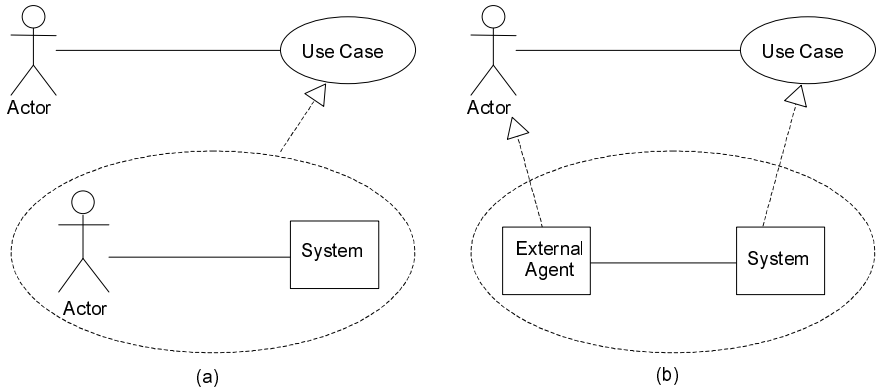

Figure 2. Contradiction: does the use case specify a system-actor interaction (a), or does it specify the role played by the system within the interaction (b)?

Obviously, these two notions cannot be simultaneously true. We are going to show several arguments to support the second, implicit notion, against the first, explicit one.

2. Use case features

First of all, let's look at the features of a use case. As any other classifier, a use case can have structural and behavioral features: "operations and attributes are shown in a compartment within the use case" [UML2, p. 522]. The meaning of use case attributes and operations is not clearly explained in the UML Specification. Nothing at all is said in the use cases Chapter of version 2. Apparently, however, the intention is more or less to represent the state of the specified subject, and the messages it can

answer[4]. This interpretation is not contained in the UML Specification, but we can find it in the old UML Reference Manual [14] and other places [17]. The Reference Manual is useful here because it reflects the intention of the original authors. It is not part of the UML Specification, and it corresponds roughly to version 1.3, but, where it has not been explicitly ammended, we can consider the original intention is still valid: "The attributes are used to represent the *state* of the use case – that is, the progress of executing it. An operation represents a piece of work the use case can perform. It is not directly callable from the outside, but may be used to describe the effect of the use case on the system. The execution of an operation may be associated with the receipt of a *message* from an actor. The operations act on the attributes of the use case, and indirectly on the system or class that the use case is attached to" [14, p. 489].

In other words, use case attributes represent the state of the system that takes part in the interaction; and use case operations represent actions performed by the system within the context of the interaction. This is perfectly clear for *an entity* that realizes the use case (the subject): it must provide attributes and operations to implement the features specified in the use case. But what is the sense of *an interaction*, a collaboration among objects, having attributes and operations? The answer to this question might be related to the introduction of two new metaclasses in UML 2: **Behavior** and **BehavioredClassifier** (see Figure 3).

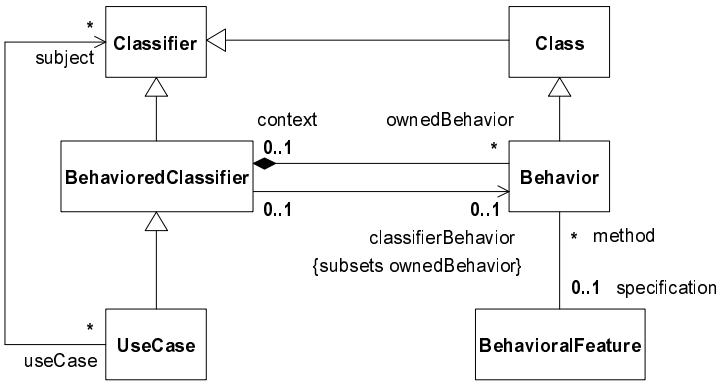

Figure 3. Behavior and BehavioredClassifier in the UML 2 metamodel (extracted from Figures 312 and 401)

The new **Behavior** metaclass is defined as follows: "Behavior is a specification of how its context classifier changes state over time" [UML2, p. 379]. "Instantiating a behavior is referred to as *invocating* the behavior, an instantiated behavior is also called a behavior *execution*" [UML2, p. 379]. For **BehavioredClassifier**, instead, we do not have a proper definition, only a rather poor description that does not say what it is, only what it has: "A classifier can have behavior specifications defined in its namespace. One of these may specify the behavior of the classifier itself" [UML2, p.

[4] This is supported also by the way some authors map use cases to system operations [15].

383]. This corresponds to the two meta-associations represented in Figure 3, between **BehavioredClassifier** and **Behavior**. That is, a **BehavioredClassifier** is a kind of **Classifier** that can own one or more **Behavior**s. In other words, a behaviored classifier is rather an ordinary classifier with ordinary instances, only it can own behaviors[5].

If a use case is "the specification of a set of actions performed by a system" [UML2, p. 519], and a use case instance is "an occurrence of emergent behavior" [UML2, p. 520], then a use case resembles greatly a **Behavior**, the instances of which are behavior executions. However, **UseCase** is not a subtype of **Behavior** in the UML 2 metamodel, but a subtype of **BehavioredClassifier**. *Why?* This manifests again the contradiction exposed above (see Section 1): on the one side, **UseCase** is explicitly defined as a behavior specification; on the other side, **UseCase** subtypes **BehavioredClassifier**, that is, it represents a kind of ordinary classifier that owns behaviors. Is a use case a behavior, or does a use case own a behavior?

Interestingly, "a classifier behavior is always a *definition* of behavior and not an *illustration*. It describes the *sequence of state changes* an instance of a classifier may undergo in the course of its lifetime" [UML2, p. 380]. It seems here that, even though a behavior can be described in different textual and graphical ways [UML2, p. 519], the UML Specification recognizes that the most suitable way to specify system behavior is through state machines. This confirms that a small set of typical stories is not enough to specify a required system function.

A last word on **Behavior**. It subtypes **Class**, therefore it can have attributes as well as operations. The UML Specification, which says nothing about use case features, says very little about behavior attributes and operations, too: "When a behavior is invoked, its attributes and parameters (if any) are created and appropriately initialized" [UML2, p. 381]. That is, a behavior attribute is similar to something that would be called a *local variable* in a more traditional terminology. Even less is said about behavior operations: we only know that a behavior can respond to events [UML2, p. 381], but it seems these events are defined in the context object, not in the behavior itself; therefore, the meaning of behavior operations remain obscure, so that they do not serve to clarify the possible meaning of use case operations, if use cases are still to be considered behaviors.

Our second argument for preferring a use case to specify a set of entities instead of a set of interactions has to do with another kind of structural feature, which is the use case-actor association.

3. Use case-actor associations

We all are used to the familiar representation of relationships between use cases and actors in use case diagrams, like that of Figure 4. We naturally interpret the line

[5] In fact, one wonders why a **Classifier** owning **BehavioralFeature**s needs to be specialized as a **BehavioredClassifier** to own **Behavior**s. This implies that an ordinary **Classifier** cannot own a **Behavior**; instead, it must be specialized first. But this is strange, since the connection between **Classifier** and **Behavior** already exists: "a behavioral feature is implemented (realized) by a behavior" [UML2, p. 382] (see Figure 2).

that connects the **Client** actor with the **Withdraw money** use case as "the client requires that the ATM system provides a service or function to withdraw money". This relationship is represented as a solid line, which is the usual UML graphical symbol for an association. In fact, it formally *is* a binary association [UML2, p. 520], which can have some of the usual association adornments, such as multiplicity or navigability markers[6].

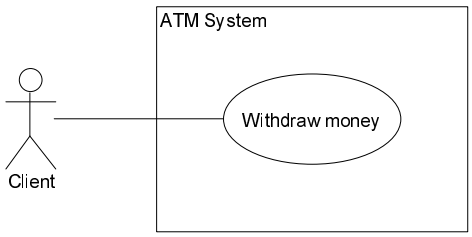

Figure 4. Simple use case diagram for an ATM System.

In the general sense, an association defines a semantic relationship "between classifiers" [UML1, p. 2-19], or, more recently, "between typed instances" [UML2, p. 81]. In both versions of the UML, the association specifies a set of tuples, "relating instances of the classifiers" [UML1, p. 2-19], or, in the new version, "whose values refers to typed instances" [UML2, p. 81]. The instances of the association, that is, the tuples, are called *links* in UML[7]. Therefore, a use case-actor association is supposed to specify a set of links between use case instances and actor instances.

There has been a subtle modification in the interpretation of use case-actor associations in passing from UML 1 to UML 2: "There may be associations between use cases and actors, meaning that the instances of the use case and the actor communicate with each other" [UML1, p. 2-137]. "Use cases may have associated actors, which describes how an instance of the classifier realizing the use case and a user playing one of the roles of the actor interact" [UML2, p. 520]. Note that in UML 1 the actor instance communicates with the *use case instance*, whereas in UML 2 the actor instance interacts (and is linked) with the *instance of the classifier realizing the use case*, that is, an instance of the subject. This difference is of great importance for

[6] Use cases "may have other associations and dependencies to other classifiers, e.g. to denote input/output, events and behaviors" [UML2, p. 522], but "two use cases specifying the same subject cannot be associated since each of them individually *describes a complete usage of the subject*" [UML2, p. 520]. Even though UML 2 has retained Include and Extend relationships, this statement should be enough to discard included or extending use cases as true use cases, since they usually are supposed to be mere fragments, not *complete usage specifications*. However, we are not going to insist on this point. The interested reader is referred to a previous article by the authors [4].

[7] For the purpose of this article, we can consider both definitions equivalent, although the substitution of "classifier" by "typed instance" is surely important for a more specific work on the semantics of associations. Regarding the problems of identifying "link" and "tuple", the interested reader is referred to other works [3, 5, 18].

our argument. In both cases, an actor instance represents a concrete external agent playing a certain role as it interacts with the system; that is, an actor represents some kind of *entity*. However, the meaning of a use case instance is less clear.

3.1. Use case-actor associations in UML 1

Let's concentrate first in the UML 1 interpretation [6], where a use case instance is the performance of a sequence of actions specified in a use case [UML1, p. 2-133]. That is, a use case instance is not an *entity*, but a *system's execution*. Consequently, a link between a use case instance and an actor instance is not a link between two entities, but a link between an entity and the execution of another entity (the system).

In addition to specifying a set of links, an association *specifies also a possibility of communication* [5, 7]: that is, the linked instances know each other and can communicate, according to the properties specified by the association, by sending messages through the links. A message is the way objects have to require and provide services from one another, that is, to communicate. Therefore, a link between a use case instance and an actor instance allows them to communicate and interact [UML1, p. 2-137, 3-96].

Interactions are represented in UML by means of interaction diagrams, which represent instances and messages interchanged through links along time. Interaction diagrams are widely used to describe the *scenarios* (use case instances) belonging to a given use case. Consider the simple interaction diagram in Figure 5, where an instance of the **Client** actor communicates with an instance of the **ATM System** class (which, at a certain level of abstraction, is a perfectly legitimate abstraction of the whole system): the communication consists of withdrawing some amount of money from a certain account.

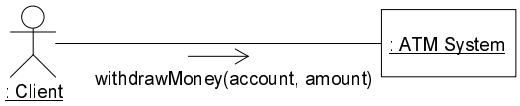

Figure 5. Simple collaboration diagram showing a money withdrawal.

At first sight, it might seem that the link in Figure 5 between the **Client** instance and the **ATM System** instance, which supports the message **withdrawMoney(account, amount)**, is an instance of the association in Figure 4 between the **Client** actor and the **Withdraw money** use case, but not at all! Instead, it is an instance of an association between the **Client** actor and the **ATM System** class. We haven't any proper way in UML 1 to represent a link between a use case instance and an actor instance, probably because these links do not exist at all[8]. Moreover, if the use case instance represents the system-actor interaction, then we need a link

[8] A use case diagram is a specialized form of class diagram, that allows only two types of classifiers: actors and use cases. Therefore, it seems we need a specialized form of object diagram that allows only these two types of instances. What is the instance diagram corresponding to a use case diagram? This special diagram is not acknowledged in UML.

between the system and the use case instance, in addition to the link between the actor and the use case instance.

What messages can be sent through a use case-actor association? None. In UML 1 a use case instance is not an *entity* that provides services, it is not an entity that answers messages: it is merely an execution of a behavior (of another entity). It has no sense saying that "one entity communicates with one execution". Instead, we should say: "one executing entity communicates with another executing entity, and this is a behavior".

Furthermore, the receiver of the message cannot be the use case instance. If the use case instance is defined as the "performance of a use case, initiated by a message instance from an instance of an actor" [UML1, p. 2-137], it is clear that it does not exist prior to the message reception, therefore it cannot be the message receiver. Things are much simpler: the message is not sent to the use case instance, but to the system itself, and the behavior initiated is what we call a use case instance.

All this manifests a severe confusion in the definition of use cases as classifiers and use case-actor relationships as associations in UML 1: the actor does not really interact with the use case [8]; the actor interacts with the system, and the representation of this interaction is what we call a use case. That is, the concept of a use case includes both the system and the actor. Therefore, a use case diagram should not show use cases related with actors, but rather *a system related with actors through use cases*. In this sense, it can be thought that a use case is some kind of property of an association between the actor and the system, such as a system interface, if not the system-actor association itself (see Figure 6).

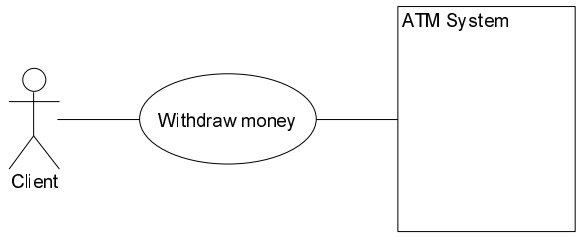

Figure 6. Imaginary use case diagram notation showing an actor associated with a system *through* a use case: the use case becomes equivalent to an association.

3.2. Use case-actor associations in UML 2

Let's come now to UML 2, where this problem might be solved more easily. In the new version, as we have already argued, there exist two contradictory notions of use case: a behavior, or something that owns a behavior; a set of interactions, or a set of entities that may play a role in an interaction specification.

If we adopt the first notion, which is equivalent to that of UML 1, then the problem is not solved. But the modified definition of use case-actor association in UML 2 supports the second notion: "Use cases may have associated actors, which describes

how an instance of the classifier realizing the use case and a user playing one of the roles of the actor interact" [UML2, p. 520]. That is, the actor instance does not interact with the *use case instance* any more (in the UML 1 sense), but with with the *instance of the classifier realizing the use case*, that is, with an instance of the subject (which may be considered an indirect instance of the use case, as we have already argumented, see Section 1).

The conceptual change needed here is the explicit recognition that a use case does not specify a set of interactions, but a role that may be played by the subject. This would clarify the meaning of the use case-actor association, since roles are a very natural concept in the context of associations. And this is precisely what makes a use case nearly the same concept as a role with an associated protocol interface, releasing UML of unnecessary complexities by collapsing two concepts into one.

Conclusion: use cases are protocols

Jacobson's original notion of use case as a description of typical system usages, or system-actor interactions, has demostrated to be very fruitful in requirements elicitation. This process cannot stop in the illustration of behavior, but has to go deeper into its full specification, including pre- and postconditions, and use case goals. Accordingly, the UML 2 notion of use case goes far beyond behavior illustration, into behavior specification, which requires the specification of system states and recognized events, that is, a state machine.

In spite of the usefulness of interaction descriptions, the formalization of use cases as classifiers in UML has some obscure points, especially regarding the concept of use case instance. Two contradictory notions of use case still coexist in UML 2: "set of interactions" vs. "set of entities"; "behavior" vs. "role with behavior". If the first notion is kept, then the metamodel should be changed to make **UseCase** subtype of **Behavior**, not of **BehavioredClassifier**, and the meaning of use case features (attributes, operations, and associations) should be clarified. If the second notion is adopted, as we suggest, then the metamodel may be kept as it is, but it should be recognized explicitly that *a use case is the specification of a role played by the subject it applies to*; a use case would not have direct instances, but the instances of the subject could be considered its indirect instances. The usual notation for use case diagrams does not really need to change.

Summing up, in our view a use case resembles more and more a *role*, the behavior of which is specified through a *protocol interface* with an associated state machine, a concept that has been explicitly introduced in UML 2 with the same purpose as use cases, namely, to specify system or subsystem usages [UML2, p. 455]. In other words, a use case is *a coordinated use of system operations* invoked through messages from the actors. A properly defined use case is not a different thing from our old friend, the protocol. All other is invisible clothing, like in the old fable.

"But he has nothing on at all," said a little child at last. "Good heavens! listen to the voice of an innocent child," said the father, and one whispered to the other what the child had said. "But he has nothing on at all," cried at last the whole people [1].

References

1. Hans Christian Andersen. *The Emperor's New Suit (Keiserens nye Klæder)*, 1837 (available at http://www.andersen.sdu.dk/, http://hca.gilead.org.il/, etc.)
2. Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.
3. Guy Genilloud. "Informal UML 1.3 - Remarks, Questions, and some Answers". *UML Semantics FAQ Workshop* (held at ECOOP'99), Lisbon, Portugal, June 12, 1999.
4. Gonzalo Génova, Juan Llorens, Víctor Quintana. "Digging into use case relationships". *The Fifth International Conference on the Unified Modeling Language-UML'2002*, September 30-October 4 2002, Dresden, Germany. Lecture Notes in Computer Science 2460, Springer 2002, pp. 115-127.
5. Gonzalo Génova. *Entrelazamiento de los aspectos estático y dinámico en las asociaciones UML*. PhD Thesis, Universidad Carlos III de Madrid, 2003.
6. Gonzalo Génova, Juan Llorens. "On the Nature of Use Case-Actor Relationships", *Upgrade-The European Journal for the Informatics Professional*, vol. V, no. 2, April 2004.
7. Gonzalo Génova, Juan Llorens, José Miguel Fuentes. "UML Associations: A Structural and Contextual View", *Journal of Object Technology*, 3(7): 83-100, Jul-Aug 2004, (http://www.jot.fm/issues/issue_2004_07/article1).
8. Sadahiro Isoda. "A Critique of UML's Definition of the Use Case Class". *The Sixth International Conference on the Unified Modeling Language-UML'2003*, October 20-24, 2003, San Francisco, California, U.S.A. Lecture Notes in Computer Science 2863, Springer 2003, pp. 280-294.
9. Ivar Jacobson, M. Christerson, P. Jonsson, G. Övergaard, *Object-Oriented Software Engineering: a Use Case Driven Approach*, Addison Wesley, 1992.
10. Ivar Jacobson, Martin Griss, P. Jonsson. *Software Reuse: Architecture Process and Organization for Business Success*. Addison-Wesley, 1997.
11. Pierre Metz. "Against Use Case Interleaving", *The Fourth International Conference on the Unified Modeling Language-UML'2001*, October 1-5, 2001, Toronto, Ontario, Canada. Lecture Notes in Computer Science 2185, Springer 2001, pp. 472-486.
12. Object Management Group. *Unified Modeling Language Specification*, Version 1.5, March 2003 (Version 1.3, June 1999. Version 1.4, September 2001).
13. Object Management Group. *Unified Modeling Language Superstructure Specification*, August 2003, ptc/03-08-02.
14. James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
15. Shane Sendall, Alfred Strohmeier. "From Use Cases to System Operation Specifications". *The Third International Conference on the Unified Modeling Language-UML'2000*, October 2-6, 2000, York, United Kingdom. Lecture Notes in Computer Science 1939, Springer 2000, pp. 1-15.
16. Friedrich Steimann. "A Radical Revision of UML's Role Concept". *The Third International Conference on the Unified Modeling Language-UML'2000*, October 2-6, 2000, York, United Kingdom. Lecture Notes in Computer Science 1939, Springer 2000, pp. 194-209.
17. Perdita Stevens. "On Use Cases and Their Relationships in the Unified Modelling Language". *4th International Conference on Fundamental Approaches to Software Engineering-FASE 2001*. Genova, Italy, April 2-6, 2001. Lecture Notes in Computer Science 2029, Springer 2001, pp. 140-155.
18. Perdita Stevens. "On the Interpretation of Binary Associations in the Unified Modelling Language", *Journal on Software and Systems Modeling*, 1(1):68-79, 2002. A preliminar version in: Perdita Stevens. "On Associations in the Unified Modeling Language". *The Fourth International Conference on the Unified Modeling Language-UML'2001*, October 1-5, 2001, Toronto, Ontario, Canada. Lecture Notes in Computer Science 2185, Springer 2001, pp. 361-375.