Use Cases, Actions, and Roles

Guy Genilloud(1), William F. Frank(2), Gonzalo Génova(3)

(1,3) Departamento de Informática, Universidad Carlos III de Madrid
Avda. Universidad, 30 – 28911 Leganés, Madrid, Spain
(2) X-Change Technologies Group, 363 7th Avenue, Floor 11, New York, NY 10001
(1) guy.genilloud@ie.inf.uc3m.es, (2) wfrank@xtg.bz
(3) ggenova@inf.uc3m.es

Abstract. Use Cases are widely used for specifying systems, but their semantics are unclear in ways that make it difficult to apply use cases to complex problems. In this paper, we suggest clarifications to use case semantics so that use case modeling can be applied to relate automated systems to business processes and process specifications, particularly in situations where we need to integrate *multiple systems* in support of a business process. We discuss the original intentions of Ivar Jacobson and UML and we find out that use case specifications, whether written in natural language or as interaction diagrams, are misleading as to what is a use case (instance). We consider then a more natural modeling technique, and establish a relation between a use case, a joint action, and a role.

1. Introduction

Use Cases are widely used in software engineering for specifying the observable behavior of systems. However, there is still controversy among practitioners about a number of issues, for example whether or not internal actions of the system should be described at all [1], and what to think of actions by actors [2, 3]. In Section 2, we find answers to these questions by looking at the main definitions of a use case, both by Ivar Jacobson and in UML. While the answers are easy to obtain, they are also surprising because most, if not all, use case specifications are misleading in this respect.

Nothing is said in UML-2 [4] about the semantic relation between a use case and an action in an activity diagram, even though activity diagrams may be used to model business processes. May a use case correspond to such an action? And what would be the correspondence? We answer this question by considering the modeling technique of the RM-ODP, rather than that of UML (see Sections 3 and 4).

2. What is a Use Case?

For understanding the semantic of use case specifications, or in other words, for knowing what is a *use case* (in this paper, we follow ODP and Jacobson's original terminology; so by use case, we mean use case instance), we look at some of the most important definitions and explanations by both UML and Ivar Jacobson. In the quotes below, the emphasis is ours. We start with the definition in UML-2:

“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.” [4]

Before UML, Ivar Jacobson provided several definitions of a use case, and some explanations:

"A use case is a sequence of transactions performed by a system, which yields a measurable result of values for a particular actor." [5]

"A use case is a sequence of transactions in a system, whose task is to yield a result of measurable value to an individual actor of the system." [6]

“Use case: the definition above is really a specific flow of events through the system, that is, an instance. There are great many possible courses of events, many of which are very similar. To make a use case model meaningful, you usually group the courses of events and call each of the groups a use case class. ...” [6]

“In the system: when we say ‘transactions in a system’, we mean that the system supplies the use cases. The actors communicate with the system’s use cases.” [6]

"Transaction: a transaction is an atomic set of activities that are performed either fully or not at all. It is invoked by a stimulus from an actor to the system or by a point in time being reached in the system. A transaction consists of a set of actions, decisions and transmission of stimuli to the invoking actor, or to some other actor(s)." [6]

The main difference between UML’s definition and Jacobson’s is that, before being influenced by UML and the meta-modeling approach to language definition, Ivar Jacobson was not thinking of a use case as a *specification*, but rather as the *real thing* that the system *does*. This point is of little importance for this paper, as is the difference between “a sequence of transactions” and “a set of actions.”

Rather, what matters to us right now is what has remained constant across all definitions and explanations. From the sentences we have underlined, it is very clear that Jacobson and UML intend use cases to include exclusively actions performed by the system, and no other actions performed by actors. In all the above definitions, actions performed by actors are not mentioned, and this cannot be by omission.

Note also that the two sets of definitions share a view of the use case as a *sequence* of actions, rather than as an action. Since most any action is decomposable into smaller groups of actions, and any sequence of actions is considered as such only because the entire sequence is considered to be a manifestation of some higher level action, as the single action of playing a baseball game is a sequence of playing some innings, the sharp distinction between a sequence of actions and an action is only one of viewpoint, and makes inter-viewpoint communications difficult.

Jacobson, like UML, considers the stimuli sent by actors to the system as being events outside of use cases. Likewise for the stimuli received by actors. The actual communication of stimuli is not modeled – they are simply assumed to arrive at the receiver some time after having been sent by the sender (without any action taking place).

2.1. Actions by Actors

The above observation (that a use case includes exclusively actions performed by the system) is likely to surprise many practitioners. Indeed, most if not all use case textual specifications describe actions by actors: for example, “the clerk inputs customer information in the system”, “the customer selects one of the presented options”, “the system manager confirms her entries,” etc (some authors even recommend using multiple columns for textual use case specifications, one column for the actions of the system, the other(s) for the actions of the actors).

Describing actor’s actions is in fact just a way of describing that the system will receive the corresponding stimuli and input values. It is more important to write use case specifications such that they are an easy read for users and stakeholders (untrained in computing science or object modeling), than to write a use case specification so that it fits exactly its intended semantics. The question of whether an actor’s actions of sending information to the system belongs or not to a use case is important to methodologists, and perhaps to analysts or to programmers, but not to their customers.

Some use cases are specified with an interaction diagram, showing the system interacting with its actors. In this case, it is the limitations of the graphical notation that make it necessary to represent actors and their actions of sending and receiving messages. In this case, it is a lot simpler to teach readers of specifications that the only actions relevant to the use case are those of the system, than to extend the interaction diagram notation.

2.2. Internal Actions of the System

Does a use case include internal actions by the system? Of course, it does, or there would be no relevant actions in it.

Following the classical object modeling technique (in which objects communicate exclusively by exchanging messages between them) assumed by UML and by Ivar Jacobson, sending a message is an internal action of the sender, and receiving it is an internal action of the receiver (when an object’s state machine receives/accepts a message as an input event, no other object observes or controls that action). Such actions of the system clearly belong to the use case, but what about other actions, such as recording a customer’s address, or erasing the data of its credit card?

They do. Read the quotes in Section 2.1: both UML and Jacobson speak of a set of actions performed by a system, without insisting that they should be actions of either sending or receiving a message. Again, this cannot be by accident.

In an old Objectory manual (1994), it is written:

“*In the description of a use case, there are descriptions of what happens in the system. The use case description does not define how tasks are performed in the system.*”

This is a clear confirmation that a use case includes actions of changing the system’s state, and that use case specifications describe them. Of course, the level of granularity of the actions should be such no unnecessary details are revealed (or rather, that programmers are not excessively constrained).

It would be a big mistake to omit such actions in the use case description. Think of yourselves as an analyst, who faces the question of whether or not to write, "The system erases the customer's credit card information" (assuming that no confirmation of this fact is to be given to the actor). Do you want to tell programmers that the system must erase credit card information, or leave the decision to them? Therefore, a use case description should include those internal actions of the system whose effects can be observed or inferred by the actors.

Lesley Lamport explains in [7] that it is almost necessary to mention internal actions in a (black box) system specification, and not just the system interactions. While it is possible to write specifications without mentioning any internal actions (Lamport calls such specifications *purely temporal*), it is not desirable to do this: the specifications so obtained would be much more complex. Doing so would of course be totally against the philosophy of use cases, that specifications should be easy to read, even by non-professionals. Indeed, for knowing what the system does in a particular use case, a reader would potentially need to read all the use case specifications, instead of just one.

2.3. The Actor's Task and the Use Case

Jacobson explains his important idea that a use case should yield a result of measurable value to its primary actor:

"A measurable value: this expression is a very important key to finding the correct level of a use case, that is, one that is not too detailed. A use case must help the actor to perform a task that has identifiable value. It may be possible to assess the performance of a use case in terms of price or cost. For example, applying for a loan in a bank is something that is of value for a customer of a bank."[6]

Jacobson explanations are such that the actor alone is performing this task, for which it receives help from the system. The task of the use case is to help the actor, and it is entirely separate from the actor's task. It seems therefore that the responsibility of the actor's task lies squarely with the actor, not at all with the system.

But do human actors see things in this way? If we were to ask some actors about the task they are performing, many of them would tell us that their task actually requires them to use the computer – they would not see the computer as just a help tool. With respect to taking responsibility, they would tell us that they do not feel responsible if the system makes incorrect actions, or gives them incorrect values (she is not checking calculations received from the system, and she is not asked to doing it). In many cases, the system's availability is essential for the actor's task to be performed. For all these reasons, it would seem reasonable to say that the actor and the system are performing the task together. But Ivar Jacobson assumed a modeling technique in which the system and its actors perform all their actions alone, and communicate exclusively by exchanging messages. Since a use case consists exclusively of actions by the system, it cannot be a part of the actor's task. However, it can be part of means by which the actor might achieve his goals, if the actor is a primary one (an actor for whose benefit the system exists).

In the second part of this paper, we will look at this same issue from a different perspective, that of the ODP modeling technique [8].

3. Use Cases from an ODP Perspective

3.1. ODP Basic Modeling Concepts

In an ODP model, entities in the real world can be modeled by objects, including the system, no matter if it is an IT centralized system, a distributed IT system, or a business system. The ODP notion of object is much more general than that of most OO specification or programming languages, in particular because an object may participate in all kinds of interactions (see below), instead of just communicating with other objects by exchanging messages.

8.1 Object*: A model of an entity. An object is characterized by its behaviour (see 8.6) and, dually, by its state (see 8.7). An object is distinct from any other object. An object is encapsulated, i.e. any change in its state can only occur as a result of an internal action or as a result of an interaction (see 8.3) with its environment (see 8.2). ... [8]*

8.3 Action*: Something which happens.*

Every action of interest for modeling purposes is associated with at least one object.

The set of actions associated with an object is partitioned into internal actions and interactions. An ***internal action*** *always takes place without the participation of the environment of the object. An* ***interaction*** *takes place with the participation of the environment of the object.*

NOTES

1 - "Action" means "action occurrence". Depending on context, a specification may express that an action has occurred, is occurring or may occur.

2 - The granularity of actions is a design choice. An action need not be instantaneous. Actions may overlap in time. ... [8]

An action of an object is either an *internal action* or an *interaction* of that object. The discriminator between these two categories is rather subtle, because an object may interact with itself [9]. So, for the purpose of this paper, we will adopt a simpler classification. Borrowing the terminology from Catalysis [10], we speak of *localized actions* for those actions which have just one participating object, and of *joint actions* for those actions in which several objects participate (so, every joint action is an interaction, but not every interaction is a joint action). An object which participates in an action is a *participant* in that action.

The ODP modeling technique is such that the actor's task is preferably modeled as a single joint action in which the system is also a participant. This joint action is composite, since it is specified as a configuration of simpler (joint or localized) actions[1]. Communication of information between objects is explained by joint actions of the

[1] In Catalysis, all joint actions are specified atomically (using pre- and post-conditions on the participant states) before being decomposed. However, an "atomic" specification is only valid under the assumption that no other joint action, happening concurrently, interferes with this joint action. We do not propose adopting this practice, as it is not realistic for actions at the granularity of use cases. For example, an actor may receive an urgent call and decide not to complete a task.

communicating objects, either directly between themselves, or with intermediate objects [9, definition 8.8]. See [11] for a discussion and examples of joint actions at the granularity level of use cases.

The ODP modeling technique makes it easy and natural to model the task of a primary actor, but what is a use case? To answer this question, we investigate the notion of role in a joint action.

3.2. Joint Actions and Roles

A fundamental characteristic of ODP interactions, and therefore of joint actions, is that they have roles ("*An object may interact with itself, in which case it is considered to play at least two roles in the interaction.*" [8, Definition 8.3, Note 4]). For any joint action that we imagine, we can always find two or more roles.

Unfortunately, the RM-ODP explanations of the concept of role let much to be desired (they might be changed in the next revision of the standard). So we provide here a different explanation[2]: actions are decomposed in different parts that are played (in principle) by different objects; a ***role*** (of an action) is that part of an action that is played by an object, i.e., the contribution of that object to the action. Thus, a role is a part of an interaction, in a non-recursive decomposition of that action[3]. This decomposition is *non-recursive* in that the parts obtained, the roles, are not all actions. It is different from the typical *recursive* decomposition of an action, which yields sub-actions. We call the decomposition of an action that yields its roles the *role decomposition* of the action.

A role decomposition may be applied to an action seen as a whole, in which case it simply yields the roles of the action. For example, in an action in which a message is communicated, there is a sender and a receiver.

A role decomposition may also be applied to an action seen as a configuration of simpler actions (sub-actions) with constraints between them (e.g., the sequential constraint that an action occurs before another). In this case, the decomposition yields the relevant roles of the joint sub-actions, and the relevant localized sub-actions. As for the constraints between actions, they remain. For example, the role of accepting a proposal includes receiving the proposal (a role in a joint action), reading it (a localized action), and giving back a notification of acceptance to the counter-party (a role in another joint action).

A *role* is therefore a subset of the behavior of an object, that is, a role is a collection of more primitive roles (of joint actions) and localized actions with a set of constraints between them. The *behavior of an object* is then the union of all its roles. It is also a collection of roles and localized actions with a set of constraints between them. The difference between role and behavior is that a role is a contribution to some joint action, whilst no such constraint applies to the behavior of an object.

[2] The concept of role, being *primitive*, cannot be defined on the basis of other concepts, but it can be explained (much like the RM-ODP does for objects and actions).

[3] We would define a role of a link in the same way.

3.2.1. Difference with the Behavior of an Object in the RM-ODP

The interested reader should note that the following definition in the RM-ODP is flawed.

8.6 Behaviour (of an object): *A collection of actions with a set of constraints on when they may occur. ... [8]*

The problem is in fact obvious (so much that one may make the definition right without even thinking about it). For knowing what an object does, that is, its behavior, it is not enough to know all the actions in which this object participates. One must also know which roles the object performs in those actions. For example, consider a joint action in which two people get married, in front of two witnesses (as illustrated in Fig. 2). If you were only told that a person participated in the action, you would not know whether he or she got married or was simply a witness.

3.3. Use Cases and Actors are Roles

Having explained the concepts of joint action and role, we are now in a position to provide an alternate definition of a use case, which assumes the ODP modeling technique:

Use case: The system role in a joint action (among the system, its actors, and possibly additional participants) that is intended by the system's designer, and expected by some actor or other stakeholder, when circumstances are appropriate, to yield a particular observable result that is of value to one or more actors or other stakeholders of the system.

Our definition makes it clear that IT systems, not just people, perform some of the actions of a business process (remember that the original vision is for a use case to help an actor perform a task, which might be an action in a business process). Likewise, IT systems (or in fact their providers), rather than their actors, may need to assume the responsibility when an action fails to provide the expected results. In other words, business processes are not composed of actors' actions (performed with the "help" of the system), but of actor-system joint actions.

Defining a use case as a role of a joint action is also interesting because it makes the notion of actor easier to understand. In UML-1.5, an actor was defined as "*A coherent set of roles that users of use cases play when interacting with these use cases. An actor has one role for each use case with which it communicates.*" But UML 1.5 did not provide one compatible, good explanation of the concept of role. And it failed to point out that a use case is a role of the system [3].

No real progress has been made in UML-2 with respect to the concepts of actor and role. In fact, the new definition of actor in UML-2 speaks of just one role for an actor, which makes the notion of an actor harder to understand. The definition may still be considered to be correct, if one sees this single role as a composite of the actor's roles in the use cases in which it participates. For further explanations of the concept of actor, see [12] and [11].

4. Applications to Business Processes

Support for a business process by IT systems becomes particularly clear, when one understands the idea that the IT system is performing roles (use cases) in the actions.

A business process is a configuration of localized and joint actions (e.g., a sequence of actions). Typically, people are involved in many actions, and more and more they perform their tasks by using an IT system. It is even possible that one person uses several systems in support of a single joint action.

Unfortunately, some notations for business process modeling work under the assumption that all actions are localized to just one participant (e.g., UML activity diagrams with swimlanes). They do not support representing joint actions with their multiple participants, much less indicating which specific role each participant performs in the joint action[4] (see Section 4.2). So a specifier faces a problem whenever she wants to describe an action that happens to have multiple participants (e.g., a primary actor, a system and a secondary actor). She may decide to aggregate all the action participants as a group, but she may end up having too many groups, some of which unnatural. She may represent only one action participant and omit others (in particular the system that helps the primary participant perform her task). Or she may decompose the action into sub-actions, such that all actions are localized, and participants (supposedly) communicate by exchanging messages). She would therefore go at a much lower level of abstraction than she originally intended to. In any case, these notations make it unnecessarily difficult to relate use cases to the actions of the business process (i.e., in Jacobson's terms, to the tasks of the primary actors).

The problems that we mentioned here are compounded when one wants to examine how several systems might be used jointly for supporting a same actor in a same business process. Indeed, one needs then not just to relate each system's use cases to the business process, but also to relate them to use cases of other systems. The following method, based on the concept of joint action, solves all these problems: first specify the business process (once and for all) as a configuration of joint actions, then find the participants in each action, and find their roles (or contributions in the action). Use cases are then given by the roles of the systems in the joint actions.

4.1. Making Use of the Use Case Specification Technique

The use cases graphical notation may be used to support our approach, as Wegmann and Genilloud showed in [11]. Fig. 1, taken from that paper, can be produced using most UML case tools on the market. Ovals represent joint actions, and actors represent participants. The role of a participant is represented using a rolename of the association between the actor and the use case. The related mutiplicity indicates how many roles of the type (say sender) might be in the joint action.

[4] Such notations may nevertheless have a concept of role, much like the use cases notation has the concept of actor (see Section 3.3). But they lack support for the more fundamental concept of a joint action's role.

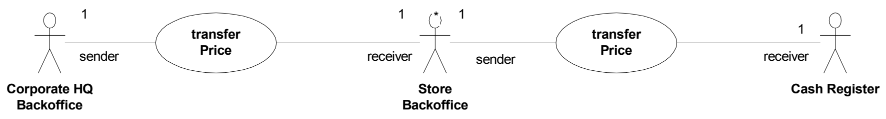

Fig. 1. The Use Case Notation may be used for representing joint actions and roles.

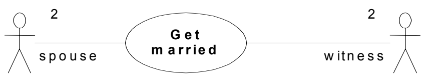

Fig. 2. A joint action with four roles of two types, spouse and witness.

Joint actions may be specified textually, just like use cases. The difference is that the (sub-)actions of all participants must be mentioned, not just those in which the system participates. The use case specification(s) may then be obtained by projection of this joint action specification, i.e., by removing all the actions in which the system does not participate.

4.2. Improving the Activity Diagrams Notation

UML activity diagrams might be improved for supporting joint actions. It should be possible to indicate that actions have multiple participants, each playing one or more roles in the action. For a participant in an action, it should be possible to indicate the type of role that it performs, with a name (much like a role name of an association in UML).

We stress that this name should denote a type of role rather than a role (a role is not a type, but a kind of part of a joint action). It is indeed possible for several participants to perform roles of the same type (for example, a "Get married" joint action with two identical roles of spouse, and two identical roles of witness). If we were to name roles rather than their types, the notation might be unnecessarily restrictive and unnatural. For example, UML insists that all rolenames of associations be different, which rules out the possibility of having symmetric associations in a model [13]. See [12] for a discussion of the relation between the notions of role, type of role, and type of object.

5. Summary and Conclusions

In this paper, we set out to answer the question of what is a use case. We had in mind settling two issues often debated by use case methodologists and practitioners. We found that Jacobson's and UML definitions of a use case provide authoritative answers: (1) all the system's actions, not just those of sending or receiving messages, should be described (to some extent) in a use case specification; and (2), only the actions of the system belong to a use case, and none of the actors.

Common use case textual specifications do not reflect the second answer, but it is all for the better. Specifications in natural language should describe the actors' actions of entering data, selecting an option, etc., as a matter of convention (it is clear enough that the system receives those data), and so that they can be easily read by system users and other stakeholders. Likewise, interaction diagrams should describe the actors sending and receiving messages, because doing otherwise would require complicating the current notation.

In the second part of the paper, we considered use cases in the light of the ODP modeling technique. That is, we replaced the message passing hypothesis with joint actions [10], in which several participants may change state. This modeling technique is more natural than that of UML, since joint actions can model actions in natural language, given by verbs. All joint actions come with multiple roles; e.g., the buy action has the roles of buyer, seller, the sold entity, and its counterpart.

We then explained *a use case* as *a role in a joint action*, an important idea when one is attempting to relate IT systems to business processes, or to integrate multiple systems in support of a business process.

We showed that the use cases specification technique (i.e., the use cases graphical notation together with textual specifications) is applicable to modeling joint actions, and we indicated how the activity diagrams notation should be extended in support of the same cause.

6. Acknowledgements

We acknowledge the contributions of Joaquin Miller and Alain Wegmann to the years of discussions that led to this paper. We are also grateful to the anonymous reviewers for their valuable comments and suggestions.

7. Bibliography

1. Anderson, B. *Formalism, technique and rigour in use case modelling*. in *UML2004 Workshop on Open Issues in Industrial Use Case Modeling*. 2004. Lisbon, Portugal.
2. Isoda, S. *On UML2.0's Abandonment of the Actors-Call-Use-Cases Conjecture*. in *UML2004 Workshop on Open Issues in Industrial Use Case Modeling*. 2004. Lisbon, Portugal.
3. Génova, G. and J. Llorens. *The Emperor's New Use Case.* in *UML2004 Workshop on Open Issues in Industrial Use Case Modeling*. 2004. Lisbon, Portugal.
4. OMG, *Unified Modeling Language: Superstructure (version 2.0)*. 2004, OMG.
5. Jacobson, I., et al., *Object-Oriented Software Engineering--A Use Case Driven Approach*. 1992, Reading, Massachusetts: Addison-Wesley, 1992. 524.

6. Jacobson, I., M. Ericsson, and A. Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology*. ACM Press Books. 1995: Addison-Wesley. 347 pp.
7. Lamport, L., *A simple approach to specifying concurrent systems.* Communications of the ACM, 1989. **32**(1): p. 32-45.
8. ISO/IEC and ITU-T, *Open Distributed Processing - Basic Reference Model - Part 2: Foundations*, in *Standard 10746-2, Recommendation X.902*. 1995.
9. Genilloud, G. and G. Génova. *On Interactions in the RM-ODP*. in *submitted to the Workshop on ODP for Enterprise Computing (WODPEC 2005)*. 2005. Enschede, The Netherlands.
10. D'Souza, D.F. and A.C. Wills, *Objects, Components and Frameworks With UML : The Catalysis Approach*. Addison-Wesley Object Technology Series. 1998: Addison-Wesley. 912.
11. Wegmann, A. and G. Genilloud. *The Role of "Roles" in Use Case Diagrams*. in *Third International Conference on the Unified Modeling Language (UML2000)*. 2000. York, UK: Springer-Verlag.
12. Genilloud, G. and A. Wegmann. *A Foundation for the Concept of Role in the RM-ODP*. in *4th International Enterprise Distributed Object Computing Conference (EDOC 2000)*. 2000. Makuhari, Japan: IEEE Computer Society.
13. Génova, G., *Interlacement of structural and dynamic aspects in UML associations (Ph. D. Thesis)*. 2003, Carlos III University of Madrid: Madrid, Spain.