

Informal Semantics of UML Use Case Diagram

This document describes the informal semantics of UML 2.0 (Unified Modeling Language version 2.0) Use Case diagrams. The document is intended for verification of UML Use Case Diagrams and hence does not describe how to draw a use case diagram. Consequently, the readers should refer to UML manuals or books on UML notations if they would like to know how to draw them.

Use case diagram

A use case diagram of a software system describes the functionalities to be implemented by the software system. It is at a relatively higher level of abstraction compared to the other UML diagrams and will not include implementation details. Hence, it could be considered as a schematic view of the requirements for the software system. Together with the narratives for the use cases, the use case view provides a complete set of requirements for the system.

Notations - Summary

A UML use case diagram is composed of the following symbols:

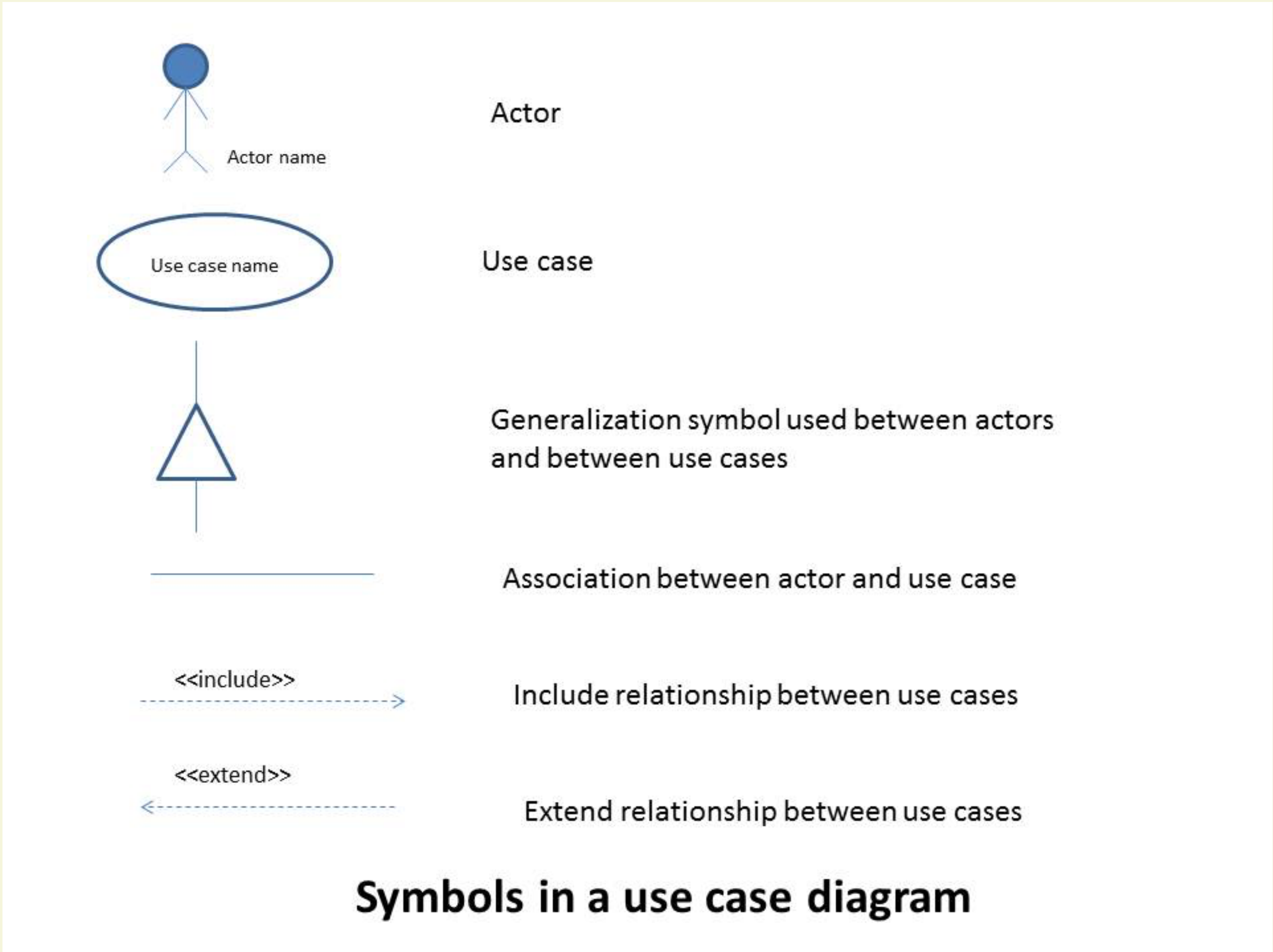


Figure 1. Symbols used in a use case diagram

Informal Semantics

**Actor:** An actor represents an external entity that interacts with a system. Since it is external to the system, the actor itself is not fully modeled by the system. However, in order to design the interactions between an actor and a system, the latter may have a simplified model of the actor. A user of a system is a typical example of an actor. Other types of actors include software systems that are being integrated with the current system (e.g., a network protocol, a database system, a file system), external hardware such as a sensor and so on.

Actors are classified into primary actors (also called active actors) and secondary actors (also called passive actors). Primary actors initiate a use case and hence are somewhat independent. Secondary actors, on the other hand, are used by the system but they do not interact with the system on their own. In other words, secondary actors do not initiate any use case. The symbol for an actor does not differentiate between a primary actor and a secondary actor; the difference must be inferred from the use case descriptions (also called use case narratives).

An actor represents the role of the entity that interacts with the current system, not an instance. For example, an actor may represent a customer of a bank. In this case, every customer of a bank is represented by this one actor rather than a separate actor for each customer. Similarly, there may be another actor representing a manager of the bank. Interestingly, a particular manager of a bank in real world may be a customer of the same bank. That is, the same person plays the role of a customer and a manager. But this instance relationship is not captured by the actor. So, as far as the use case diagram is concerned, there are two different types of actors (manager and customer) in this example.

The following rules are applied to verify actors in a use case diagram.

- [ACT.1] Every actor has a unique name in the diagram. If the diagram includes more than one actor with the same name, they are logically combined into one actor. They might have been represented more than once for simplifying the drawing of the diagram.
- [ACT.2] An actor must be represented by its unique symbol shown in the notations given in Figure 1.
- [ACT.3] Every actor must have at least one association (refers to a connection to a use case; to be discussed shortly) in the diagram. Otherwise, the actor will stand alone which is not useful.
- [ACT.4] An actor can have at most one association with a use case. Stated otherwise, an actor cannot be connected to the same use case more than once.
- [ACT.5] An actor may be connected with another actor through a generalization relationship.

**Use Case:** A use case represents a functionality (typically a requirement) that is expected to be implemented by the system. The details of a use case, other than its unique name, are not represented visually in the diagram; such details are given in use case narratives. Depending on the application domain and the choice made by the designer, a use case may be broken down into several use cases which are connected through <<include>> or <<extend>> relationships (described later in this document). Use cases are generally initiated by primary actors. Secondary actors such as a database are used by the system through a set of use cases. An association between a use case and an actor represents a two-way communication. Hence, for every initiation of a use case by a primary actor, the latter must get a response back. Similarly, for every association between a secondary actor and a use case, the communication starts from a use case and the secondary actor is expected to respond back to the initiation.

The following rules are applied to verify use cases in a use case diagram.

- [UCASE.1] Every use case must have a unique name in the diagram. If there is more than one use case with the same name in the diagram, then they are logically combined into one.
- [UCASE.2] A use case must be represented by an ellipse (the symbol for use case) as shown in the notations given in Figure 1.
- [UCASE.3] A use case must be connected to at least one actor or to at least one use case. Otherwise, the use case may stand alone which is not useful. If it is connected to an actor, the connection must be an association. If it is connected to another use case, the connection must be a <<include>> relationship or an <<extend>> relationship.
- [UCASE.4] A use case can have at most one relationship with another use case. That is, there cannot be more than one <<include>> or <<extend>> relationships between two use cases.

**Generalization:** This represents a relationship between actors or between use cases. If two actors are related through this relationship, then the actor (or use case) at the tail end of the arrow (connected to the base of the triangle) is a specialized version of the actor (or use case) at the other end. Sometimes, the actor (or use case) at the base end (connected to the base of the triangle) is called a specialized version of the actor (or use case) at the other end. The meaning of the generalization is that the specialized version has every feature of the general version and may have more. Notice that the two actors (or the two use cases) connected through the generalization relationship do not have any communication between them. In other words, this relationship is only used by the designer for simplification of coding during implementation.

The following rules are applied to verify generalization relationships in a use case diagram.

- [GEN.1] The generalization relationship can be used between a pair of actors or between a pair of use cases.
- [GEN.2] There is at most one generalization relationship between a given pair of actors or a given pair of use cases. However, one actor (or one use case) may have generalization relationship with any number of actors (or use cases).
- [GEN.3] If used between use cases, the two use cases connected by the generalization relationship must not have any other relationship between them. In other words, they are independent of each other. The generalization relationship only helps the designer in coding.
- [GEN.4] The generalization relationship does not have a label.

**Association:** This represents a two-way communication between an actor and a use case and hence is a binary relation. Since it is a two-way communication, for every use case initiated by a primary actor, the actor must get a response back from the use case. Similarly, for every communication between a use case and a secondary actor (initiated by a use case), the secondary actor must send a response back to the use case. An association may include cardinality information which indicates how many instances of use cases or actors participate in the communication. See the rules below for details of cardinality.

The following rules are applied to verify associations in a use case diagram.

- [ASSO.1] An association can exist only between an actor and a use case.
- [ASSO.2] An association does not have a label.
- [ASSO.3] Between a given actor and a use case, there can be at most one association.
- [ASSO.4] An association may have a cardinality which may be included on either end of the association. It is represented by one of the following format: (i) a single integer, (ii) a range of integers (n..m, n >= 0 and m >= 0 and n < m), and (iii) a '\*' symbol indicating zero or more.

**Include:** This is a special type of relationship between two use cases. If a use case A includes another use case B, then the implementation of A requires the implementation of B in order to complete its task. However, B is independent on its own. That is, B does not need to know anything about A. B can also be included in any other use case.

The following rules are applied to verify <<include>> relationships in a use case diagram.

- [INCL.1] An <<include>> relationship can exist only between use cases.
- [INCL.2] Every <<include>> relationship is represented by a straight line connection and must also have the label "<<include>>".
- [INCL.3] If a use case A includes another use case B, then the <<include>> relationship arrow points from A to B.
- [INCL.4] If a use case A includes a use case B, then there cannot be any other relationship between A and B.
- [INCL.5] If a use case A includes a use case B, and if B includes another use case C, then technically, A includes C. That is, the <<include>> relationship is transitive. However, the connection between A and C is not shown in the diagram.

**Extend:** This is another special type of relationship between two use cases. If a use case B extends another use case A, then the implementation of A may conditionally include the implementation of B in order to complete its task. That is, A may complete the task without B in some situations. But depending on the condition stated, A may require B. B in this case is dependent on A and cannot exist on its own. For this reason, B cannot extend more than one use case. The use case narrative of A will include the execution step at which it requires B; this point is called an *extension point*.

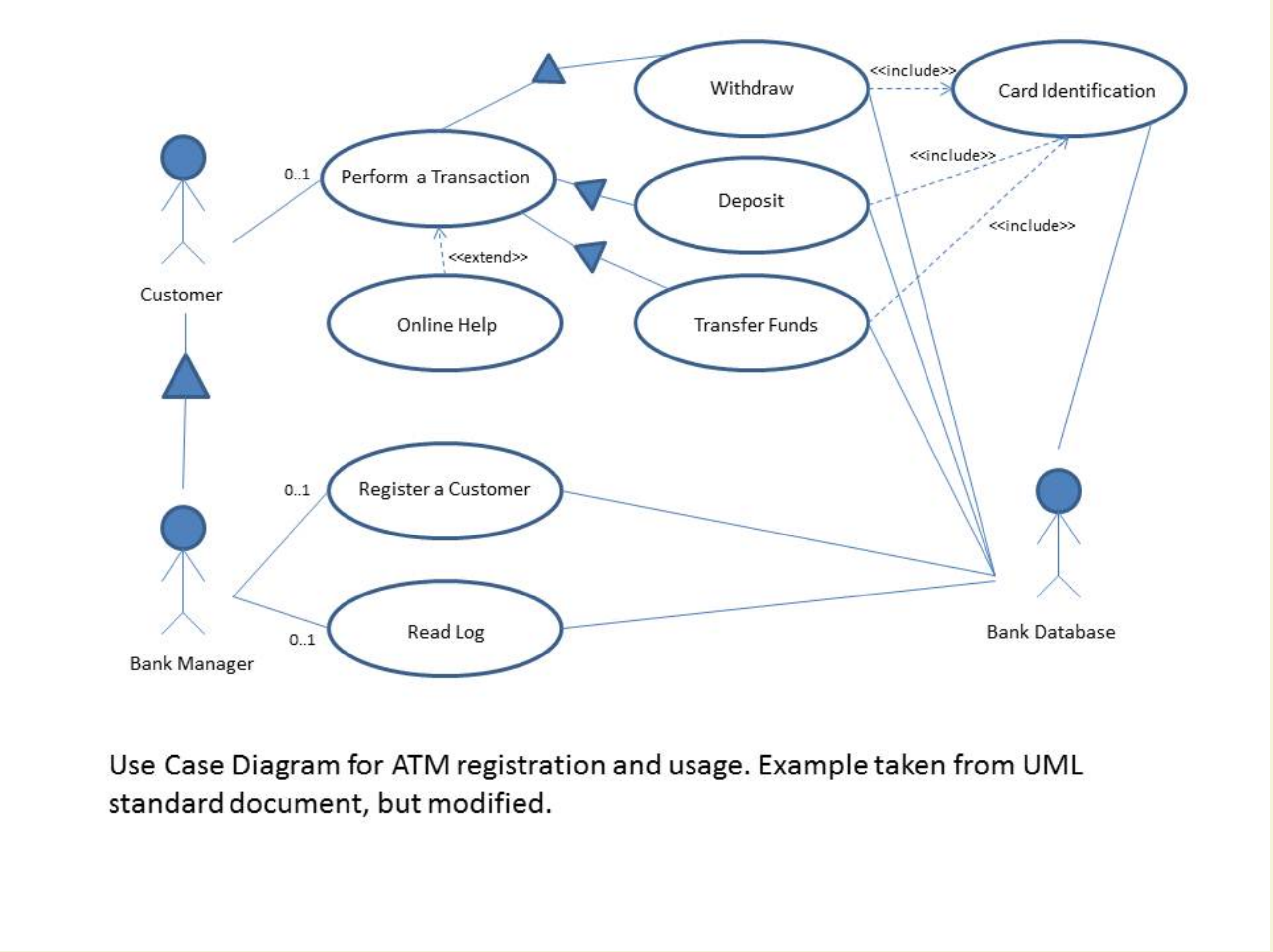
The following rules are applied to verify <<extend>> relationships in a use case diagram.

- [EXTN.1] An <<extend>> relationship can exist only between use cases.
- [EXTN.2] Every <<extend>> relationship is represented by a straight line connection and must also have the label "<<extend>>".
- [EXTN.3] If a use case B extends another use case A, then the <<extend>> relationship arrow points from B to A. Notice that the arrow is reversed between <<include>> and <<extend>>. So, it is appropriate to say that B extends A or A is extended by B.
- [EXTN.4] If a use case B extends a use case A, then there cannot be any other relationship between A and B.
- [EXTN.5] If a use case C extends a use case B, and if B extends another use case A, then technically, C extends A. That is, the <<extend>> relationship is transitive. However, the connection between A and C is not shown in the diagram.

References:

- Hans-Erik Erikson *et al.*, *UML 2 Toolkit*, Wiley Publishing Company, 2004, ISBN: 0-471-46361-2.
- Russ Miles and Kim Hamilton, *Learning UML 2.0*, O-Reilly, 2006, ISBN: 0-596-00982-8.
- Alan Dennis et al., *System Analysis & Design with UML*, Wiley, 2012, ISBN: 978-1118037423.
- Bernd Bruegge and Allen H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns and Java* (3rd edition), Prentice Hall, 2009, ISBN: 978-0136061250.
- UML Standard documents published at the Object Management Group (OMG) web site: [www.omg.org/spec/UML](http://www.omg.org/spec/UML).

Sample Use Case Diagram:



Use Case Diagram for ATM registration and usage. Example taken from UML standard document, but modified.

Figure 2. Use Case Diagram for ATM Transactions