JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2002

Vol. 2, No. 2, March–Aprile 2003

Specifying use case behavior with interaction models

José Daniel García
Jesús Carretero
José María Pérez
Félix García
Rosa Filgueira,
Computer Architecture Group, Universidad Carlos III de Madrid

Functional requirements for information systems can be modeled through use cases. Furthermore, use case models have been successfully used in broader contexts than software engineering, as systems engineering. Even if small systems may be modeled as a set of use cases, when large systems requirements are modeled with a plain use case model several difficulties arise. Traditionally, the behavior of use cases has been modeled through textual specifications. In this paper we present an alternate approach based on interaction modeling. The behavior modeling has two variants (one for UML 1.x and one for UML 2.0). We also integrate our behavior modeling with standard use case relationships.

1 INTRODUCTION

Use cases have been used as a mean of capturing the requirements of a software system by several methods as Objectory [1], Catalysis [2] OML [3] or Unified Process [4]. Use cases have also been used successfully in contexts broader than software development, as systems engineering [5, 6, 7].

To model functional requirements as use cases, those requirements are represented as a set of use cases, where each use case is the specification of a set of transactions between the system and the external actors, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system [8].

However most techniques based on use cases do not easily handle the existence of branches and loops in the the use case logic [9]. This is because interaction diagrams are primarily oriented towards a simple, linear sequence of interactions between the actors and the major classes of the system.

Use cases are an inadequately structured mechanism for representing functional requirements of a system. However, some structuring mechanisms have been proposed such as the use of goals [10, 11, 12, 13, 14, 15], use cases grouping, following some clustering criteria (e.g. main actor or feature), into use case packages [4, 14], or the partitioning of the system into smaller collaborating subsystems [16].

Besides, UML offers two standard mechanisms for structuring use cases: *extension* and *inclusion*. These mechanisms may be used to share common behavior between different use cases [8].

In this paper we present an alternate approach to use case textual specifications based on interaction modeling. The proposed behavior modeling has two variants: one for UML 1.x and another one for UML 2.0. The basic idea is the use of a combination of activity diagrams and interaction diagrams (UML 1.x) or an interaction overview diagram (UML 2.0). Because few tools are currently available which fully support UML 2.0 both variants are presented.

This paper is organized as follows: section 2 discusses different alternatives for specifying use cases behavior; section 3 proposes a way of modeling use case behavior with structured interaction models; section 4 studies the impact of standard use case relationships on behavior models; section 5 presents a proposal for use case level stereotypes; finally, section 6 summarizes the conclusions of this paper.

2 USE CASE SPECIFICATIONS

A use case is the specification of a set of transactions between the system and the external actors, thus, it may be described in terms of the interactions between the system and the external actors. In a use case specification the core element is the events flow. Usually, the use case is described as a set of flow of events, where there is one main flow of events (or basic path) and a set of alternative paths.

Typically, the use case specification is written in free text format [1, 10, 11, 4, 14], although some kind of structured text or formal specification may also be used. However, the specification of individual use cases in natural languages, such as English provides ample room for miscommunication and misunderstandings [9]. Some sort of structuring may be obtained by means of a swim lane diagram with a lane for the system and a lane for each actor [17].

As an example, we show the textual description for the use case *Withdraw Funds* for an *ATM System*.

The basic events flow for the use case *Withdraw Funds* is shown below:

1. The account holder identifies himself to the ATM.
2. The ATM requests the account holder authentication.
3. The account holder provides his authentication.
4. The ATM authenticates the holder's identity with the holder's bank.
5. The bank confirms the holder's identity authentication.

6. The ATM requests for an operation from the account holder.
7. The account holder selects the operation "*Withdraw Funds*".
8. The ATM requests for an amount.
9. The account holder provides with an amount.
10. The ATM requests the holders's bank for funds withdrawal authorization.
11. The holders's bank authorizes the withdrawal.
12. The ATM provides a receipt.
13. The account holder takes the receipt.
14. The ATM releases the holder's identification means.
15. The account holder takes his identification (if the identification was a physical object).
16. The ATM dispenses the money.
17. The account holder takes the money.

We show the flow of events for *Withdrawal denial due to insufficient funds* as an example of an alternate events flow:

1. The account holder identifies himself to the ATM.
2. The ATM requests the account holder authentication.
3. The account holder provides his authentication.
4. The ATM authenticates the holder's identity with the holder's bank.
5. The bank confirms the holder's identity authentication.
6. The ATM requests the account holder to select an operation.
7. The account holder selects operation "*Withdraw Funds*".
8. The ATM requests for an amount.
9. The account holder provides with an amount.
10. The ATM requests the holders's bank for funds withdrawal authorization.
11. The holder's bank denies the withdrawal due to insufficient funds.
12. The ATM notifies the account holder about denial.

13. The ATM releases the holder's identification means.
14. The account holder gets his identification (if the identification was a physical object).

In UML, a use case may be formally specified using an interaction model, with one object representing the system (as a black box) and one interacting object representing every participating actor [18, 19]. Some work has been done in the direction of deriving such interaction models from the textual specification [20]. Besides, use case paths [17] may be used for generating complete, unambiguous and verifiable requirement specifications.

Using interaction diagrams (sequence or collaboration) for the specification implies the development of one diagram for the basic flow of events and one diagram for each alternate events flow.

This approach presents several disadvantages:

- One interaction diagram is needed for every flow of events, even if they are rather similar. The requirements engineer will be doing a repetitive work.
- A change in the use case specification may require the modification of a set of interaction diagrams, repeating the modification in a each diagram. This repetitive work is not cost-effective and it may lead to inconsistencies derived from errors.
- In UML 1.x, modeling loops and alternatives in sequence diagrams is difficult.

3 MODELING USE CASE BEHAVIOR WITH STRUCTURED INTERACTION MODELS

As it was presented in the previous section, modeling each event flow of a use case independently with an interaction diagram creates several problems. In this section we present a general method for modeling use case behavior.

Although UML 2.0 has already been released, most tools currently support UML 1.x. Adoption of UML 2.0 by tools is not being as fast as it would be desirable. That is why we present our solution customized for both UML 1.x and UML 2.0

The UML artifacts to be used are different in UML 1.x and UML 2.0. Any of these two representations may allow the generation of textual specifications of the use case from its behavior model. Automatic textual specifications may be useful for interacting with users and other stakeholders with no or little understanding of UML.

In the following subsections we describe our proposal for building the use case behavior model for each version of UML.

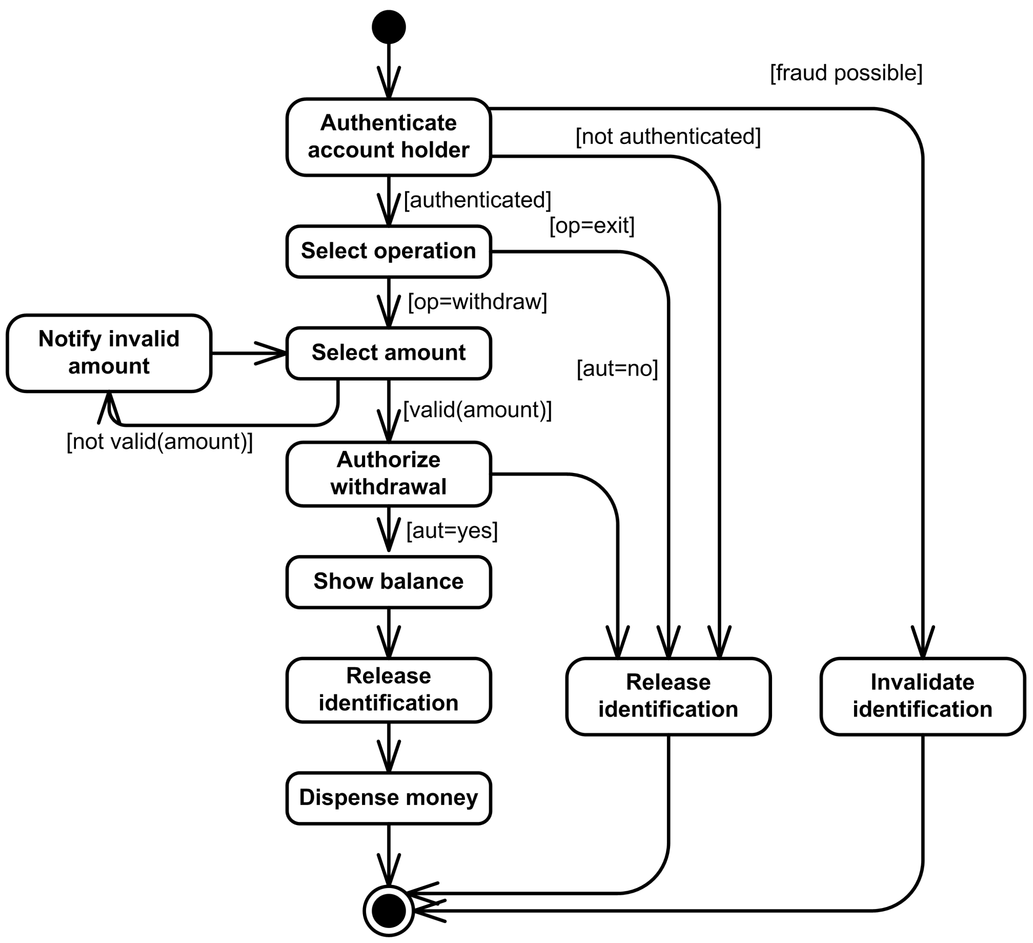

Figure 1: Activity diagram showing the flow of control for transactions of the use case *Withdraw Funds*.

UML 1.x

UML 1.x refers to the full version 1 of the UML family (specially to versions 1.4 [21] and 1.5 [22]).

UML 1.x capabilities to show alternate flows of control in a single diagram are limited. Although selection is possible, it is only practical to use it for very simple cases. It is not feasible to represent more complex ways of control (as loops)in a sequence diagram.

Under UML 1.x, our proposal is to use a combination of an *Activity Diagram* and a set of *Sequence Diagrams*. The activity diagram is used to specify the flow of control for the transactions (conditional behavior, loops, parallelism, ...). Figure 1 shows an activity diagram describing the flow of control for the transactions in the use case *Withdraw Funds*.

For every activity in the activity diagram, a sequence diagram is used to model the transaction having place. Each of those diagrams has an object per actor and an additional object representing the whole system as a black box. Figure 2 shows the interactions for activity *Account Holder Identification*.

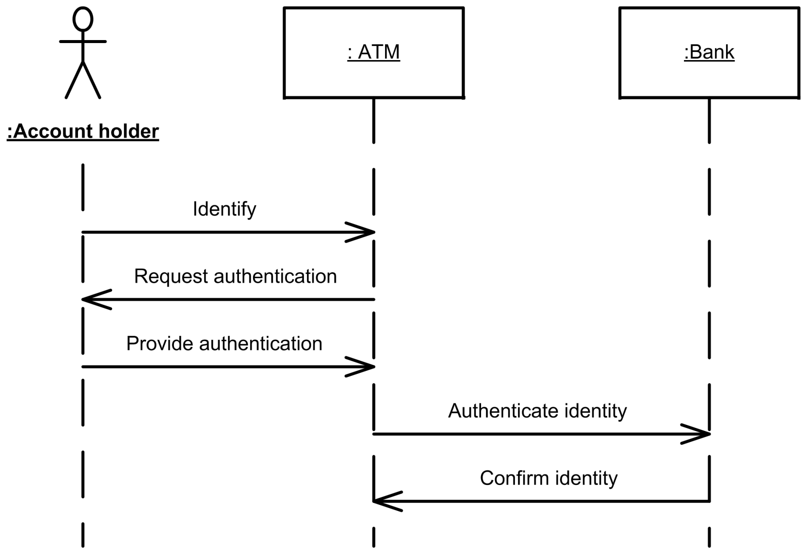

Figure 2: Sequence Diagram showing the transactions for the activity *Account Holder Authentication*.

UML 2.0

UML 2.0 provides a better way to represent the former example through *Interaction Overview Diagrams* [8]. Interaction overview diagrams are a specialization of activity diagrams where every node of a diagram contains either an inline sequence diagram or an interaction occurrence. Figure 3 shows an interaction overview diagram for the use case *Withdraw Funds*.

The use of *Interaction Overview Diagrams* allows the use of complex decision structures and loops (using decision nodes), parallel actions (using fork and join nodes) and timing information (using time constraints).

4 USE CASE STANDARD RELATIONSHIPS

Requirements for a simple system may be expressed as a set of use cases. But having a plain use case model for more than 80-100 use cases makes it very difficult to manage, so some structuring technique is needed in order to reduce its complexity [14].

UML offers two standard relationships between use cases: inclusion and extension. Both relationships may be used to structure use case models. Following subsections show how inclusion and extension relationships may be integrated with the proposed use case behavior modeling. In section 5 we use this relationships for

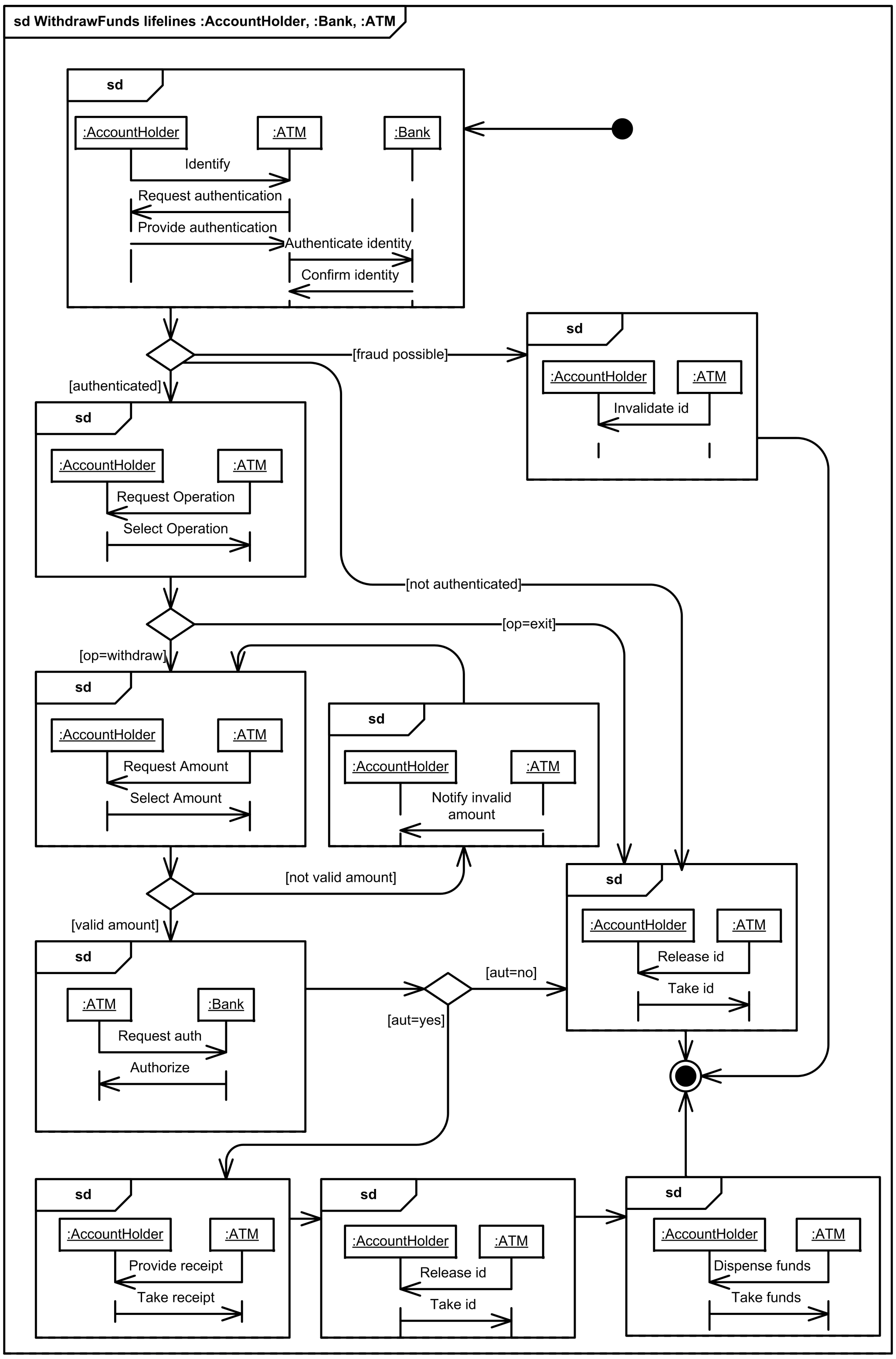

Figure 3: Interaction Overview Diagram showing the interactions for the use case *Withdraw Funds* and the corresponding control flow.

hierarchically structuring use cases. It must be remarked that a clustering mechanism (e.g. grouping by main actor, feature or subsystem) is complementary with our approach.

Inclusion Relationship

An «*include*»relationship between two use cases means that *the behavior defined in the including use case is included in the behavior of the base use case* [8]. That is, a base use case explicitly incorporates the behavior of another use case at a location specified in the base.

The include relationship is intended to be used when there are common parts of the behavior of two or more use cases. These common parts are then extracted to separate use cases, to be included by every base use case sharing these parts. The relationship between *Withdraw Funds* and *Authenticate Account Holder* in Figure 4 is an example of inclusion relationship.

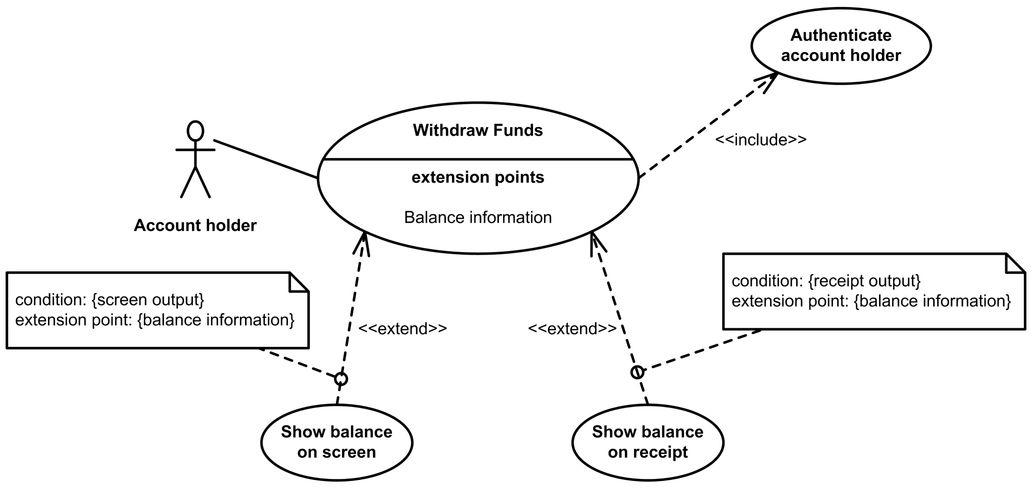
Authenticate account holder
<<include>>
Withdraw Funds
extension points
Balance information
Account holder
condition: {screen output}
extension point: {balance information}
<<extend>>
<<extend>>
condition: {receipt output}
extension point: {balance information}
Show balance on screen
Show balance on receipt

Figure 4: Use Case relationships for inclusion and extension.

Included use cases are not full use cases in the sense that an included use case instance cannot happen by itself, but in the context of a base use case. In this sense, an included use case may be classified as a *subfunction* or *sub use case* [14]. Moreover, an included use case does not provide a result of value to an actor.

In the following subsections we present our proposal for integrating inclusion relationships and use case behavior modeling in both UML 1.x and UML 2.0.

UML 1.x

In UML 1.x, the control flow of the behavior is driven by the activity diagram. A mechanism in order to indicate the inclusion of another use case is needed.

We propose the usage of an stereotyped activity (stereotype «*include*») for the inclusion of use cases. Activities *Authenticate Account Holder*, *Select operation*, *Release identification* and *Invalidate identification* in figure 5 denote the inclusion of use cases.

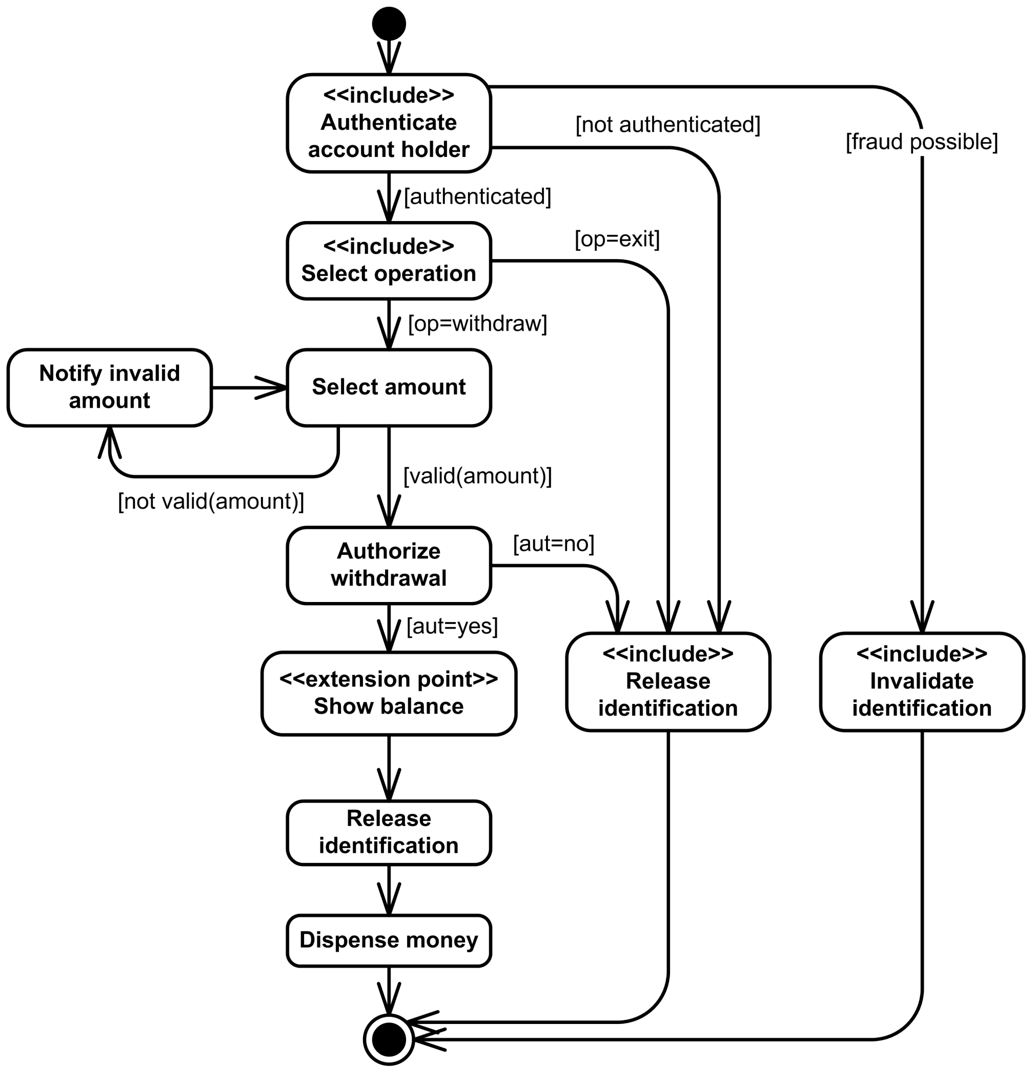

Figure 5: Activity diagram showing the control flow for transactions of the use case *Withdraw Funds* with reference to included use cases and extension points.

UML 2.0

In UML 2.0, *Interaction Overview Diagrams* are used to specify use cases, an «*include*»relationship may be represented by an *Interaction Occurrence*. An interaction occurrence is a reference to another interaction (usually, another interaction diagram). This is a common way to share portions of an interaction between several other interactions [8].

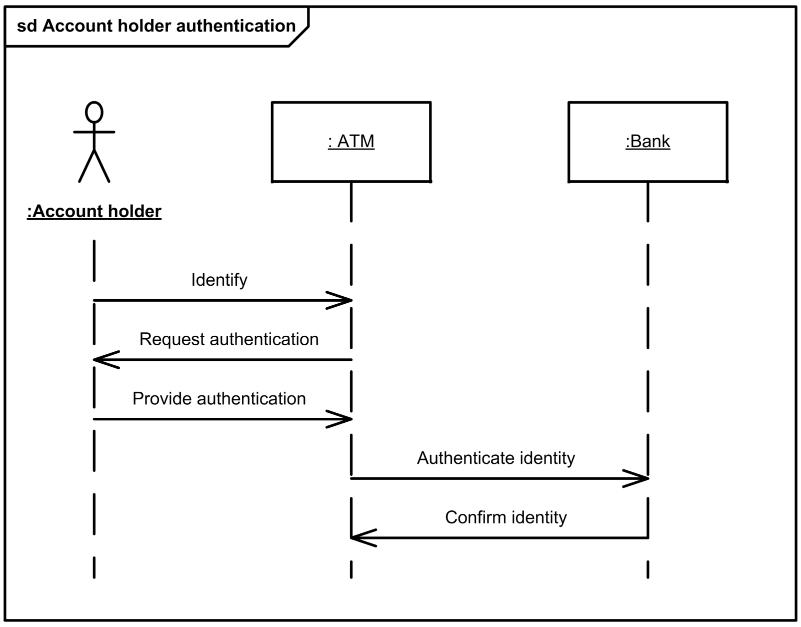

Figure 6: Definition of an interaction for Account Holder Authentication.

We propose the usage of interaction occurrences for the inclusion of use cases. Figure 6 shows the definition of an interaction which is then included in the *Interaction Overview Diagram* of figure 7.

Extension Relationship

An «*extends*»relationship specifies that *the behavior of a use case may be extended by the behavior of another (usually supplementary) use case* [8].

The extension takes place at one or more specific extension points defined in the base use case. However, the base use case is defined independently of the derived use case, and it is meaningful independently of it. The base use case defines its extension points, where the behavior of the derived use case may happen optionally. In figure 4 the use case *Withdraw Funds* offers an extension point (*Balance Information*) which is extended by use cases *Show Balance on Receipt* and *Show Balance on Screen*.

An extending use case typically defines a behavior that is not meaningful by itself but in the context of the base use cases that it extends. An extending use case defines a modular behavior increment that augments an execution of the extended use case under specific conditions. When an extension is specified, the extension condition and the extension point must be defined. In figure 4 the extension condition for use case *Show Balance on Receipt* is *paper output* and the extension point is *Balance Information*.

In the following subsections we present our proposal for integrating extension relationship and use case behavior modeling in both UML 1.x and UML 2.0.

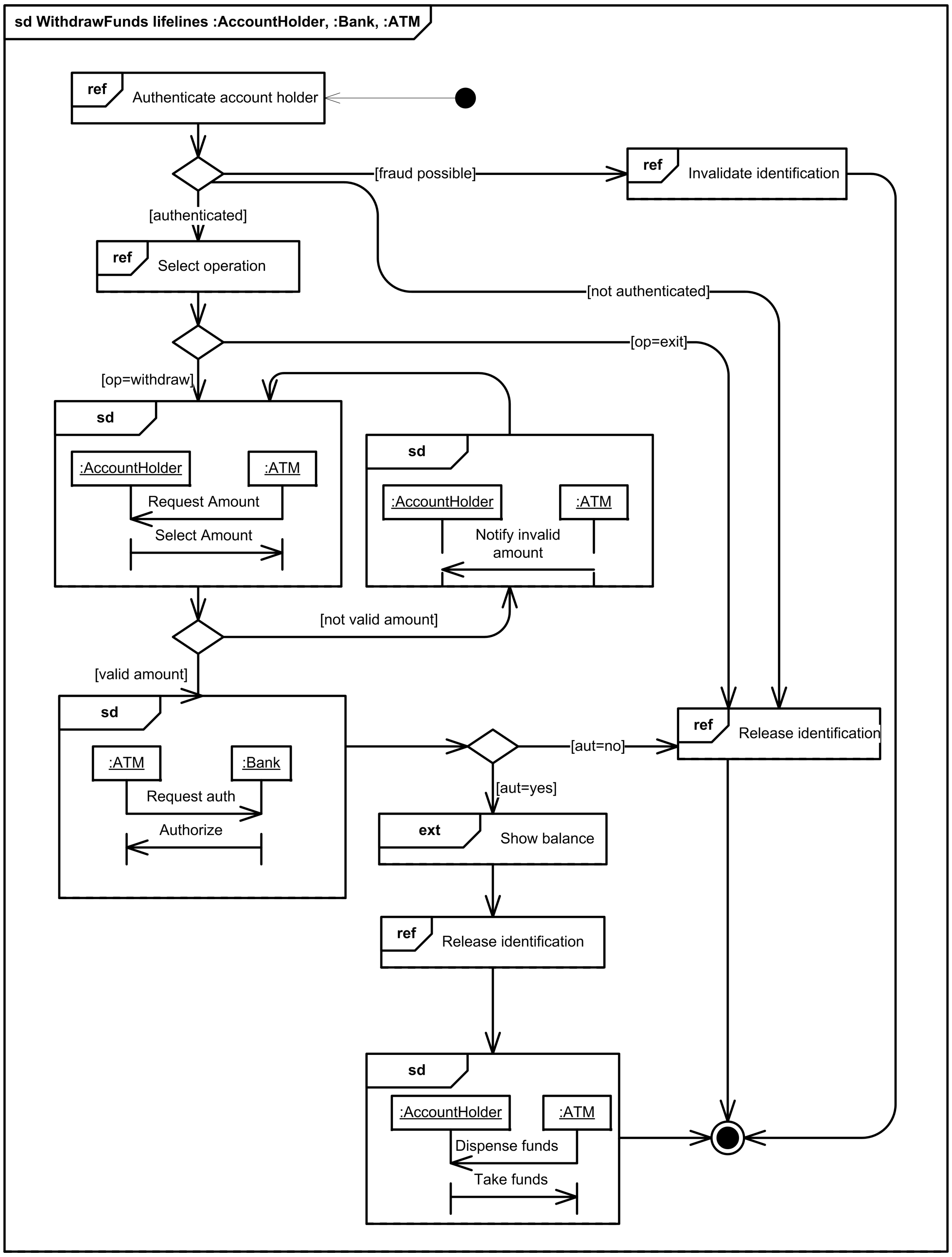

Figure 7: Interaction overview diagram for the use case *Withdraw Funds* with extension point and inclusion of interactions.

UML 1.x

In UML 1.x, the behavior's control flow of the behavior is driven by the activity diagram. A way for indicating a placeholder for the extension point is needed. In the derived use case nothing is needed, as the required information is in the «*extends*»relationship itself.

We propose the usage of a stereotyped activity (stereotype «*extension point*») for the definition of extension points. In figure 5 activity *Show Balance Information* (stereotyped with «*extension point*») denotes an extension point.

UML 2.0

UML 2.0 does not offer a mechanism for specifying extension points in interaction overview diagrams, as their primary intended usage is not the proposed in this paper.

We propose to usage a new kind of object node in interaction overview diagrams: *extension point*. In Figure 7, object node *Show Balance* is an example of extension point.

5 USE CASE LEVELS AND STEREOTYPES

Even if UML offers a set of standard stereotypes for use case relationships, the same does no happen for use cases themselves. There are no standard stereotypes for use cases. However, not all the use cases are at the same level of abstraction.

If definitions from Jacobson [1] or OMG [8] are interpreted in a strict way, the abstraction level of a use case must satisfy two conditions:

- A use case must be started by some actor.
- A use case must provide a result of value for some actor.

We name use cases satisfying those conditions as *actor level* or *user level* use cases.

As soon as standard use cases relationships are used, other abstraction levels arise.

The use of the «*include*»relationship implicitly generates a new kind of use cases at a lower level of abstraction. We call this the *subfunction* level (sometimes referred as *sub use case*, as mentioned early). Subfunctions do not need to satisfy the abstraction level restrictions of use cases (i.e.: they do not need to be started by an actor and they do not need to provide a result of value for some actor). Use cases extending a base use case are also subfunctions.

When the number of use cases scales up, some higher level structuring is necessary. One solution is grouping in use cases clusters, using use case packages. Using this solution use cases are grouped attending to some clustering criteria such as primary actor, feature, subject area or development team.

Another solution is to have use cases at a higher level of abstraction. This level is usually mapped to the business process level in information systems development. In systems engineering this level is referred as top-level use cases or system level use cases [7]. It is important to note that the use of this level of abstraction is only necessary when requirements are complex enough.

Thus, our proposal defines three stereotypes for use cases:

- «*system level*»: To be applied to system level use cases.
- «*user level*»: To be applied to user level use cases.
- «*subfunction*»: To be applied to subfunction use cases or auxiliary use cases.

Having a use case model structured by using the above mentioned stereotypes, where each level consists only of use case of the immediate level may offer some advantages. As an example of such advantages, we discuss the usefulness of those stereotypes in the context of criticality analysis.

Use Case Levels and Criticality Analysis

Criticality analysis is a technique to find which are the more critical elements of a system in terms of a set of factors. Criticality analysis may be applied to functional requirements to find out which are the most critical functional requirements for the mission of a system.

Criticality analysis of a complex use case model is not cost effective as the effort needed to perform the analysis starts to be very high when every use case needs to be analyzed. The objective must be to find the most critical use cases without evaluating criticality for every use case of the system. In such cases, having a structured use case model may help by allowing the application of the criticality inheritance concept [23].

Criticality is computed firstly for the «*system level*»use cases, and then inherited in a top-down manner. In that way, every use case starts having a criticality estimated value which is an upper bound for its actual criticality value.

Criticality analysis at the «*user level*»is only performed for those «*system level*»use cases with criticality above a defined threshold. This process is again applied from «*user level*»use cases to «*subfunction*»use cases.

If criticality analysis needs to be cost effective, a hierarchical structuring mechanism is needed. The approach proposed in this paper provides such a hierarchical structuring mechanism. For more information about criticality analysis of use cases see [24].

6 CONCLUSION

In this paper we have presented a way of formally specifying use case behavior both in UML 1.x and UML 2.0.

For UML 1.x the behavior of a use case is modeled by an activity diagram driving the control flow for the interactions. For every activity in the activity diagram a sequence diagram is used to model the transactions having place. Each of those diagrams has an object for each actor and one more object representing the whole system as a black box. When a use case includes another use case, it has to be specified in the behavior model. The proposed activity stereotype «*include*»is used for this purpose. When a use case has an extension point, the behavior model needs to specify the place where the extension point takes place. The proposed activity stereotype «*extension point*»is used for this purpose.

In UML 2.0 the behavior of a use case is modeled through an *Interaction Overview Diagram* by having a lifeline for each actor plus one more object representing the whole system as a black box. When a use case includes another use case, an interaction occurrence is used. The interaction occurrence references the interaction model of the included use case. When a use case has an extension point, the new object node *extension point* is used as a placeholder for extensions.

Finally, we have presented a set of three stereotypes for use cases allowing an hierarchical structuring mechanism for complex use case models.

Future work in going on to develop a UML metamodel for use case specifications so that it is possible to have a UML profile for use case specifications. Work is also going on to automatically generate textual specifications based on use case behavior models.

REFERENCES

[1] Ivar Jacobson. *Software Engineering - A Use Case Driven Approach.* Addison-Wesley, 1992.

[2] Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components and Frameworks with UML. The Catalysis Approach.* The Object Technology Series. Addison-Wesley, 1998.

[3] Donald G. Firesmith, Brian Henderson-Sellers, and Ian Graham. *OPEN Modeling Language (OML) Reference Manual.* SIGS Reference Library. Cambridge University Press, 1998.

[4] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. The Object Technology Series. Addison-Wesley, 1999.

[5] Ian Alexander and Thomas Zink. Introduction to systems engineering with use cases. *Computing and Control Engineering Journal*, 13(6):289–297, December 2002.

[6] Haig F. Krikorian. Introduction to object-oriented systems engineering, part 1. *IT Professional*, 5(2):38–42, March/April 2003.

[7] Haig F. Krikorian. Introduction to object-oriented systems engineering, part 2. *IT Professional*, 5(3):49–55, May/June 2003.

[8] Object Management Group. *UML 2.0 Superstructure Specification*, 2003. Final Adopted Specification.

[9] Charles F. Bowman and Donald G. Firesmith, editors. *Wisdom of the Gurus: A Sigs Developer's Guide*, chapter Use Cases: The Pros and Cons, pages 171–180. Sigs Reference Library. Cambridge University Press, 1996.

[10] Alistair Cockburn. Goals and use cases. *Journal on Object Oriented Programming*, 10(5):35–40, September 1997. http://members.aol.com/acockburn/papers/usecases.htm.

[11] Alistair Cockburn. Using goal-based use cases. *Journal on Object Oriented Programming*, 10(7):56–62, November/December 1997. http://members.aol.com/acockburn/papers/usecases.htm.

[12] Jonathan Lee and Nien-Lin Xue. Analyzing user requirements by use cases: A goal-driven approach. *IEEE Software*, 16(4):92–101, July/August 1999.

[13] J. Mylopoulos, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.

[14] Alistair Cockburn. *Writing Effective Use Cases*. The Agile Software Development Series. Addison-Wesley, 2001.

[15] Jonathan Lee, Nien-Lin Xue, and Jong-Yih Kuo. Structuring requirement specifications with goals. *Information and Software Technology*, 43(2):121–135, February 2001.

[16] Eman Nasr, John mcDermid, and Guillem Bernat. A technique for managing complexity of use cases for large complex embedded systems. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2002.

[17] Donald G. Firesmith. Generating complete, unambiguous, and verifiable requirements from stories, scenarios, and use cases. *Journal of Object Technology*, 3(10):27–39, November-December 2004. http://www.jot.fm/issues/issue_2004_11/column3.

[18] Kim Soon-Kyeong and David Carrington. Integrating use-case analysis and task analysis for interactive systems. In *Proceedings Asia-Pacific Software Engineering Conference. ASPEC 2002*, pages 12–21. IEEE Computer Society, December 2002.

[19] Girish Keshav Palshikar. The message is the medium. *Embedded Systems Programming*, 15(11):28–36, November 2002.

[20] L.M.G. Feijs. Natural language and message sequence chart representation of use cases. *Information and Software Technology*, 42(9):633–647, June 2000.

[21] Object Management Group. *OMG Unified Modeling Language Specification*, September 2001. Version 1.4.

[22] Object Management Group. *OMG Unified Modeling Language Specification*, March 2003. Version 1.5.

[23] Jesús Carretero, José M. Pérez, Félix García-Carballeira, Alejandro Calderón, Javier Fernández, José Daniel García, Antonio Lozano, Luis Cardona, Norberto Cotaina, and Pierre Prete. Applying RCM in large scale systems: a case study with railway networks. *Reliability Engineering and System Safety*, 82(3):257–273, December 2003.

[24] José Daniel García, José María Pérez, Jesús Carretero, and Félix García-Carballeira. A model for use case priorization using criticality analysis. In *Computational Science and its Applications - ICCSA 2004. Lecture Notes in Computer Science*, volume 3045, pages 495–504. Springer-Verlag, May 2004.

ABOUT THE AUTHORS

José Daniel García received the MS degree in computer science in 2001 from Universidad Politecnica de Madrid. He has been assistant professor since 2002 at the Universidad Carlos III de Madrid. Previously he worked as software engineer and consultant for several worldwide companies like Siemens, DMR Consulting, Telefonica, British Telecom, ING Bank and EADS. He can be reached at josedaniel.garcia@uc3m.es

Jesús Carretero received the MS degree in computer science in 1989 and the PhD degree in 1995 from Universidad Politecnica de Madrid. Since 1989, he has been teaching operating systems and computer architecture at several universities. From 1997 and 1998, he held a visiting scholar position at Nortwestern University in Chicago. He has been a full professor at the Universidad Carlos III de Madrid, Spain, since 2001. He can be reached at jesus.carretero@uc3m.es

José María Pérez received the MS degree in computer in 2001 from the Universidad Politecnica de Madrid. He has been an assistant professor since 2001 at the Universidad Carlos III de Madrid. He can be reached at josemaria.perez@uc3m.es

Félix García received the MS degree in computer science in 1993 from the Universidad Politecnica de Madrid and the PhD degree in computer science 1996 from the same univeristy. From 1996 to 2000, he was an associate professor in the Department of Computer Architecture at the Universidad Politecnica de Madrid. He is currently an associate professor in the Computer Science Department at the Universidad Carlos III de Madrid. He can be reached at felix.garcia@uc3m.es

Rosa Filgueira received the MS degree in computer science from the Universidad Deusto de Bilbao in 2003 and the Master in e-commerce from the Universidad Carlos III de Madrid in 2004. She has been an assistant professor since 2004 at the Universidad Carlos III de Madrid. She can be reached at rosa.filgueira@uc3m.es