# Modeling Cross-Device Systems with Use Case Diagrams

Dennis Wolters[1], Christian Gerth[2], and Gregor Engels[1]

[1] Department of Computer Science, Paderborn University, Germany
{dennis.wolters, engels}@uni-paderborn.de
[2] Faculty of Business Management and Social Sciences,
Osnabrück University of Applied Sciences, Germany
c.gerth@hs-osnabrueck.de

**Abstract.** Information systems often support a variety of different device types like desktop computers, smartphones or tablets. Some can even be used in a cross-device manner, i.e., using multiple devices in parallel or being able to switch from one device to another while interacting with the system. Even though there is increasing support to implement such cross-device systems, existing modeling languages only provide limited means to specify them. In this paper, we present an extended form of UML use case diagrams, which allows to properly take the cross-device features of a system into account and serves as a first step towards a model-based development process for cross-device systems.

**Keywords:** cross-device system, requirements analysis, use case modeling

## 1 Introduction

Due to the availability of multiple devices, users start using information systems in a cross-device manner [2,11], i.e., by performing tasks on different devices, use them in parallel or switch from one to another. If a software system does not support such cross-device interactions, users have to coordinate these interactions themselves which is usually a cumbersome task because application states have to be recreated or synchronized somehow [2], e.g., by manually copying data or using synchronization tools. For seamless interactions as described by Satyanarayanan [12], cross-device interactions must be considered at design time in such a way that the realized system coordinates the interaction and the user has little to none coordination overhead. For this purpose, various approaches [5,9,15] exist that ease the implementation of cross-device systems. However, the support to model such systems in early phases of development, e.g., during requirements analysis, is rather limited.

During requirements analysis, use cases are defined to sketch the functionality provided by the system under development. The relation between use cases, actors and other systems are visualized using UML use case diagrams. If it is not



In: S. España, M. Ivanović, M. Savić (eds.): Proceedings of the CAiSE'16 Forum at the 28th International Conference on Advanced Information Systems Engineering, Ljubljana, Slovenia, 13-17.6.2016, published at http://ceur-ws.org

important which device type is used to interact with the system, e.g., when only a single device type should be supported, the standard use case diagrams suffice. However, when developing cross-device systems it is important to specify, which functionality is offered by which type of device and what kind of interactions are possible. Standard UML use case diagrams do not support modeling such information. In this paper, we present an approach for the high-level modeling of cross-device systems in terms of extended use case diagrams. Our approach provides a clear separation of concerns and can be used during requirements analysis to pragmatically take cross-device interactions into account.

The remainder of this paper is structured as follows: In Section 2, we introduce two examples for cross-device systems and identify the shortcomings of standard UML use case diagrams. Subsequently, we present our extension to use case diagrams in Section 3. Related work is discussed in Section 4. The paper is concluded in Section 5, which also gives an outlook on future work.

## 2 Scenarios and Requirements

This section provides two scenarios, which describe examples for cross-device systems. We explain the limitations of standard UML use case diagrams with regard to modeling cross-device interactions supported by such systems. Afterwards, we derive requirements for an approach to model cross-device systems with extended use case diagrams.

In the use case diagrams in this paper, we assume for associations between an actor and a use case a default multiplicity of 1 on the actor's end and 0..1 on the use case's end. Hence, by default an actor does not have to perform a use case but if a use case is performed, the associated actors are required.

**Scenario 1**: In this scenario a railway company wants to provide a ticket system, which enables users to buy tickets at Ticket Vending Machines (TVMs), on computers, and with smartphones. A smartphone is seen as special kind of computer, which is mobile and allows location-specific services. It shall be possible that a customer starts the booking process on one device and migrates to another while keeping the current progress. This way the customer can start the booking process on his personal computer at home by selecting a train connection and choosing ticket options, pay with his smartphone on the way to the train station, and afterwards switch to a TVM to receive the ticket.

Figure 1 shows three different ways to model this scenario with UML use case diagrams: (a) Each device type is represented as a system and the use cases are duplicated if offered by multiple device types. (b) The device type information is weaved into different actor roles. Using this approach, we are also able to express that a smartphone is a special kind of computer. (c) For each device type a separate use case is created. In all three diagrams, we face the problem that the device usage information (computer, smartphone, and TVM) is mixed with other concerns. Additionally, it is not specified that it is possible to migrate from one device to another. In (a) and (c), migration possibilities could be specified using «extend» associations between the use cases. However, modeling transitions between different devices with «extend» associations is

not be very intuitive because the use cases are not extended just the device is changed. Moreover, using «extend» relations is not possible for variant (b), since we only have a single use case for all device types.

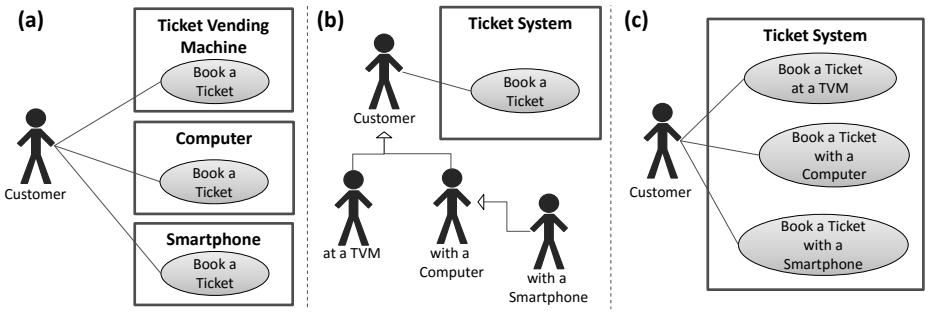


**Fig. 1.** Different ways to model Scenario 1 with standard UML use case diagrams

**Scenario 2**: In the second scenario, a system shall be developed which allows students to collaboratively work on a document, while they are supervised by a teacher. The system shall allow each student to contribute by using their own tablet and/or to collaborate with other students using smart board located in each classroom. So there is a one to one relation between a tablet and a student as well as a many to one relation between students and a smart board.

We omit a standard use case diagram for Scenario 2, since none of the approaches used for Scenario 1 really helps to define the Scenario 2. Modeling the device types as separate systems or duplicating use cases for each device type is not feasible for Scenario 2, because students could be involved with the same instance of a document editing use case on different devices. The approach of specifying the device usage along with the actors does not work either, since it does not allow to specify how many student can use a device. In Section 3, we present an extended use case diagram depicting Scenario 2.

Summarizing, we derive the following requirements for an approach to specify cross-device interactions in extended use case diagrams:

**R1 Explicit device type modeling**: Device types being relevant for the system must be modeled explicitly so that it is clear which device types are in the scope of the system.

**R2 Definition of device usage**: A use case might involve multiple human actors which use certain devices to interact with the system. It must be specifiable which device types have to be used and by whom.

**R3 Variability in the device usage**: Human actors may have the choice between different device types to perform a use case. Such variability in the device usage must be expressible.

**R4 Specification of cross-device interactions**: Cross-device interactions like distributing a use case across different devices or migrating from one device to another must be specifiable.

## 3 Use Case Diagrams for Cross-Device Systems

In this section, we present our extended use case diagrams for modeling cross-device systems. We use the scenarios from previous section to explain our approach and show how we address the Requirements R1 to R4. In the following, we use filled out circles like ❶ to refer to certain parts of Figure 2.b, which shows the extended use case diagram for Scenario 1 and hollow circles like ① to refer to certain parts of Figure 3, which shows the diagram for Scenario 2.

To explicitly define the device types being relevant for the system (see Requirement R1), we define a device type taxonomy. Figure 2.a shows an example of such a taxonomy for Scenario 1. At the top of any device type taxonomy is the class "Device", representing any kind of device. Below more specific device types are declared, in our example TVMs, computers, and smartphones as special kinds of computers. In contrast to Figure 1, the relevant device types are now explicitly listed and their hierarchy is not hidden in the inheritance between actors like in Figure 1.b.

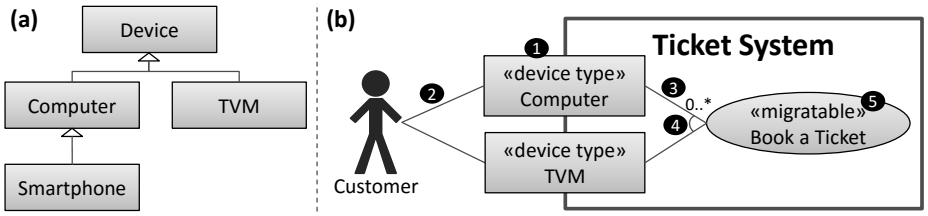


**Fig. 2.** Device type taxonomy (a) and extended use case diagram (b) for Scenario 1

In order to address Requirement R2, we use classes marked with the stereotype «device type» (see ❶) to reference a device type defined in the taxonomy. Consequently, the name of a device type class must match a device type of the corresponding device type taxonomy. In extended use case diagrams, we use device type classes to refine the association between human actors and use cases. This is done by indirectly associating a human actor with a use case, i.e., a human actor is associated with a device type to describe that a human actor uses a certain device type (see ❷). Additionally, a device type is associated with a use case to describe that the interaction of a human actor with this use case is done via this device type (see ❸). Thereby, we can define the device usage explicitly instead interweaving it with other concerns. In Figure 2.b, the human actor *Customer* is associated with use case *Book a Ticket* over the device types *Computer* and *TVM*, meaning either of these device types can be used to book a ticket. This also implies that a smartphone can be used for this use case because we have defined in the device type taxonomy that a smartphone is a special type of computer (see Figure 2.a).

In Section 2, we define that the default multiplicity is 1 on the actor's end and 0..1 on the use case's end. For our language extension, we define it in a

similar manner: The default multiplicity on an association between a human actor and a device type is 1 on the human actor's end and 0..1 on other end. Thus, not every human actor has to use all devices but when a device is used, the associated actors need to be present. Similarly, the default multiplicity on associations between a use case and a device type is 1 on the device type's end and 0..1 on the end of the use case. Thereby, the default multiplicities define that not every device necessarily has to perform all use cases to which it is connected but if a use case is performed, the devices are required.

We are able to define the relation between actors and devices as well as between devices and use cases more precisely by using multiplicities. For instance, we can specify that a TVM can only be part of one ticket booking session at a time, while a computer can be part of multiple sessions, which is defined by the 0..∗ multiplicity (see ❸). On this abstraction level, a session includes a use case instance along with the involved actors and devices. Furthermore, we can express for Scenario 2 that multiple students can use a smart board (see ①) but there should only be one student per tablet.

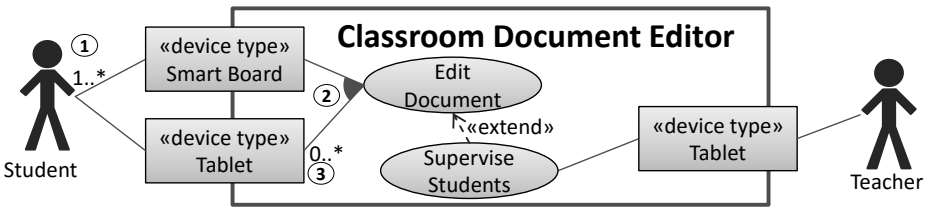


**Fig. 3.** Exemplified extensions to use case diagrams based on Scenario 2

Requirement R3 states the need to express that a variety of different device types might be used to perform a use case. We offer two possibilities to specify this variability: (i) Either we choose a common ancestor of device types which shall support the use case or (ii) by explicitly listing the device types supporting the use case. In Figure 2.b both options are used. By specifying that a computer can be used, it is implied that also a smartphone can be used, since a smartphone is a special type of computer (c.f. Figure 2.a). Whereas the TVM is explicitly listed since it is not defined as a special type of computer. To express the variability in the device usage, we enrich the concrete syntax of UML use case diagrams with OR and XOR operators known from feature diagrams [13] (see ❹ and ②). Basically, these operators are just a new concrete syntax for certain UML constraints (cf. with XOR in [10]). However, we find it very helpful to use this syntax since it is widely known, especially when it comes to the topic of variability, and in contrast to UML constraints like XOR, the feature diagram operators imply a reading direction for the constraint. In Figure 2.b, an exclusive choice operator is used (see ❹) to describe that the booking of a ticket can either be done on a device of type *Computer* or *TVM* but not on both. Whereas in Figure 3, an inclusive choice operator is used to define that a smart board, or

tablets, or both can be used to edit a document (see ②). When an OR or XOR operator is used the lower bound of all multiplicities on the opposite end must be 0. Otherwise the multiplicities might contradict the operator. If no explicit multiplicities on the opposite ends are defined, a default multiplicity of 0..1 is implied. For instance, if we would not imply a lower bound of 0 on the opposite side of the XOR operator in Figure 2.b, the multiplicities would specify that both a computer and a TVM are required, which contradicts the XOR operator (see ❹) that specifies that only one of these device types shall be used.

Requirement R4 demands that cross-device interactions must be specifiable. For this purpose, we introduce the stereotype «migratable», which can be applied on use cases (see ❺) to define that the device being used can be substitute at runtime by another device while keeping the state, as for example described in Scenario 1. The possibility to distribute a use case on multiple devices can be specified by using multiplicities with upper bound greater than 1 or by allowing the usage of multiple device types for a use case. In Scenario 2, a document can be edited on multiple tablets in parallel as defined by the multiplicity 0..∗ (see ③). Moreover, the OR operator allows that both device types can be used at the same time (see ②). Hence, a smart board can be used in addition or as an alternative to multiple tablets.

The different stereotypes introduced by our approach are specified as a UML profile. The usage of OR/XOR operators is optional, since they could be defined, in a less readable form, using UML constraints. Even though it is not commonly used, the UML meta model does not forbid to associate actors and use cases with classes (in our cases classes representing device types). Hence, aside from our UML profile, no additional changes to the UML meta model are necessary.

Our approach provides a clear separation of concerns by modeling device usage as an additional dimension in use case diagrams instead of interweaving it with other concerns like systems, actors or use cases (cf. Figure 1). This allows changing the level of abstraction or the viewpoint whenever needed in the respective project. For instance, cross-device interactions may not be of interest in any case or for all stakeholders. In such a case, we can automatically abstract from device usage by replacing an association from an actor to a device type and from a device type to a use case with an association between the actor and the use case. Similarly, it would also be possible to change the viewpoint by focusing on use cases offered by certain device types, or alternatively, by focusing on the device usage of certain use cases.

In addition to use case diagrams, there usually exist further specifications for each use case. For instance, a textual description as well as an integrated process diagram of the different scenarios of a use case, which visualizes what has to be done in order to reach the goal of the use case. The device usage defined in extended use case diagrams can be refined within such process diagrams using our approach presented in [1]. Thereby, we can define for which tasks a certain device type is needed and when it is possible to migrate to another device. By combining these two approaches we lay the foundation for a model-based development process for cross-device systems.

## 4 Related Work

The development of cross-device systems is supported on various levels. On the implementation level, approaches like Panelrama [15] for web applications or Conductor [5] for Android applications can be used to realize applications supporting cross-device interactions. In [9], test and debug tools for cross-device systems are provided. Instead of considering cross-device interaction at design time approaches like [4] and [6] allow using existing web applications in a cross-device manner. However, these approaches do not work for every web application and they are limited to web technologies. All of these approaches consider cross-device usage at later stage of the development process, some even after development, and they focus on certain platforms, mostly web technologies. In contrast, our approach can be used at the beginning of the development process during requirements analysis to document the need for cross-device interactions independent of a concrete platform.

We model variability in the device usage in use case diagrams. The lack of means to model variability in use case diagrams is also identified in [8]. However, their focus is on modeling variability between use cases by defining alternatives for including other use cases or by specifying that the inclusion of another use case is optional. The PLUSS approach [3] allows modeling variability in textual use case descriptions instead of diagrams. Their approach would also allow to link use cases to certain device types. However, as they do not explicitly focus on device usage the relation between devices and human actors cannot be expressed neither can cross-device interactions be specified.

The Systems Modeling Process (SYSMOD) [14] uses use case diagrams in a similar fashion as we do. They introduce the notion of a user system through which the user can interact with the system. The relation between a human actor and a user system is modeled with information flows. In comparison, their approach whether allows to express variability with regard to the used user system nor does it allow to cover cross-device interactions supported by a system.

Adapt Cases [7] is an approach based on use cases which is able to specify the adaption of a system. Migrating from one device to another or distributing parts of a system could be described as an adaption of the deployment. Even though Adapt Cases could be used to express this, the description of the adaption logic is on a lower level of abstraction. Adapt Cases could be used as a refinement of our approach, e.g., to describe automated migration or distribution rules.

## 5 Conclusion and Future Work

In this paper, we have presented extended use case diagrams which can be used during requirements analysis for the high-level modeling of cross-device systems. In contrast to standard UML use case diagrams, our extension enables the modeling of device usage as a separate concern and allows the specification of cross-device interactions by refining the association between actor and use cases. Thereby, we can pragmatically specify for use cases which device types can be used and by whom they are used. Due to the clear separation of concerns, it is

possible to change the viewpoint if needed, e.g., by focusing on a certain device type or use case. We have illustrated the benefits of extended use case diagrams with two examples of cross-device systems. We have defined our extension as a UML profile with an optional addition to the concrete syntax of the UML.

We are working on supporting further steps of the model-based development process for cross-device systems, i.e., a method to refine the device type taxonomy to a device type model which allows managing concrete devices at runtime. In addition, further types of models will be considered in the future in order to cover other aspects like specifying the architecture of cross-device systems.

## References

1. Bokermann, D., Gerth, C., Engels, G.: Use Your Best Device! Enabling Device Changes at Runtime. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014, pp. 357–365. No. 8659 in LNCS, Springer (2014)
2. Dearman, D., Pierce, J.S.: "It's on my other computer!": Computing with Multiple Devices. In: CHI 2008. pp. 767–776. ACM (2008)
3. Eriksson, M., Börstler, J., Borg, K.: The PLUSS Approach – Domain Modeling with Features, Use Cases and Use Case Realizations. In: Obbink, H., Pohl, K. (eds.) Software Product Lines 2005, pp. 33–44. No. 3714 in LNCS, Springer (2005)
4. Ghiani, G., Paternò, F., Santoro, C.: Push and Pull of Web User Interfaces in Multi-device Environments. In: AVI 2012. pp. 10–17. ACM (2012)
5. Hamilton, P., Wigdor, D.: Conductor: Enabling and Understanding Cross-device Interaction. In: CHI 2014. pp. 2773–2782. ACM (2014)
6. Husmann, M., Nebeling, M., Pongelli, S., Norrie, M.C.: MultiMasher: Providing Architectural Support and Visual Tools for Multi-device Mashups. In: Benatallah, B., Bestavros, A., Manolopoulos, Y., Vakali, A., Zhang, Y. (eds.) WISE 2014, pp. 199–214. No. 8787 in LNCS, Springer (2014)
7. Luckey, M., Nagel, B., Gerth, C., Engels, G.: Adapt Cases: Extending Use Cases for Adaptive Systems. In: SEAMS 2011. pp. 30–39. ACM (2011)
8. von der Maßen, T., Lichter, H.: Modeling variability by UML use case diagrams. In: REPL@RE 2002. pp. 19–25 (2002)
9. Nebeling, M., Husmann, M., Zimmerli, C., Valente, G., Norrie, M.C.: XDSession: Integrated Development and Testing of Cross-device Applications. In: EICS 2015. pp. 22–27. ACM (2015)
10. Object Management Group: Unified Modeling Language (2015), `http://www.omg.org/spec/UML/2.5/`
11. Santosa, S., Wigdor, D.: A Field Study of Multi-device Workflows in Distributed Workspaces. In: UbiComp 2013. pp. 63–72. ACM (2013)
12. Satyanarayanan, M.: Pervasive computing: Vision and challenges. IEEE Personal Communications 8(4), 10–17 (2001)
13. Schobbens, P., Heymans, P., Trigaux, J.C.: Feature Diagrams: A Survey and a Formal Semantics. In: RE 2006. pp. 139–148 (2006)
14. Weilkiens, T.: Systems Engineering with SysML/UML: Modeling, Analysis, Design. Morgan Kaufmann (2011)
15. Yang, J., Wigdor, D.: Panelrama: Enabling Easy Specification of Cross-device Web Applications. In: CHI 2014. pp. 2783–2792. ACM (2014)