# Tailoring UML Activities to Use Case Modeling for Web Application Development

Alexander Lorenz, Hans-Werner Six

Department of Mathematics and Computer Science, FernUniversitaet in Hagen
58084 Hagen, Germany
{Alexander.Lorenz, HW.Six}@FernUni-Hagen.de

## Abstract

UML activity models (activities, for short) have become widely accepted for specifying the dynamic behavior of use cases. For an adequate specification of use cases in the context of interactive systems, however, activities must be adapted in several aspects. We present a tailoring of activities to these needs yielding so-called interaction-oriented activities. From such activities we derive two kinds of activities focusing on the development process. The first activity is a user-friendly variant that is devoted to the requirements engineering stage. The second activity is obtained by a smooth transformation of the first one. It is a more detailed variant serving as a software specification guiding the implementation. We demonstrate how the latter activity can systematically be mapped to a specific target platform. As an example platform we choose J2EE with Web tier based on the framework Struts.

## 1 Interaction-Oriented Activities

In recent years, UML activity models [3] have become widely accepted as a means for the specification of use case behavior. However, activities must be refined before they can successfully be applied to the context of interactive systems. To this end, we present *interaction-oriented activities* that modify and enhance their predecessors [1, 2] and make them UML 2.0 compliant.

For a sufficient comprehension of the proper meaning of an activity diagram, the information which tasks are performed by the user and which by the system, should become clear at first glance (see e.g. [6, 7]). Furthermore, the information displayed on the screen during the course of an interaction must also be captured by any useful model specifying the interaction between a user and the system. According to Cockburn [6], a simple, not very detailed notation of "semantic" information – mainly dynamic generated content – is appropriate for that purpose.

We therefore introduce a distinction between "user actions" and "system actions" by defining

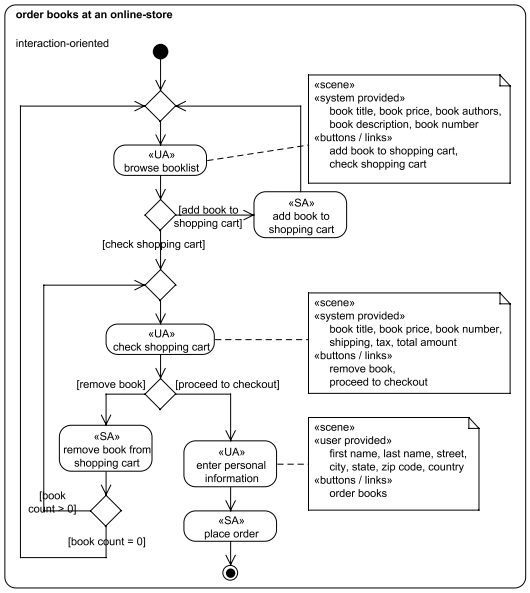


Figure 1: Interaction-oriented activity diagram

two stereotypes *«UA»* and *«SA»* for "*user action*" and "*system action*", respectively. Additionally, we use note symbols, so-called *scenes* that are attached to user actions and consist of semi-formal text describing the screen information needed by the user to perform his/her action. A scene basically depicts information provided by the system and / or entered by the user, and a list of buttons and / or links.

Figure 1 shows an interaction-oriented activity diagram that models the behavior of a use case describing a simplified ordering of books at an online-store.

# 2 Tailoring Activities to the Development Process

With regard to the development process we propose two kinds of interaction-oriented activities. The first one is devoted to the requirements engineering stage, while the second one serves as (part of the) detailed software specification driving the implementation process.

## 2.1 User-Oriented Activities

Activity diagrams suitable for requirements engineering should be of low complexity and as non-technical as possible so that users or at least trained domain experts are able to understand and validate them.

To this end, we introduce so-called *user-oriented activities* that focus on user comprehension. Only for didactical reasons, we explain the modification in terms of mapping an interaction-oriented activity to a user-oriented activity. Actually, we have introduced interaction-oriented activities only for the purpose of providing a basis from which more refined activities can be derived. So we do not use (plain) interaction-oriented activities in real life but always start the development process with user-oriented activities.

The mapping comprises of two steps, each of which reduces the complexity of an interaction-oriented activity diagram. The first step minimizes the number of system actions in the diagram. The motivation stems from our experience that users are more interested in what they have to do rather than in what the system has to do. We therefore omit trivial or non-relevant system actions like e.g. "save user input". Furthermore, any continuous sequence of system actions is merged into a single, more abstract system action.

The second step concerns decision nodes and again is divided into two parts. In many Web applications, for example, a user may first enter data into some fields on the screen and afterwards decide how to continue by choosing a particular button. According to UML, such a scenario must be modeled by two nodes: an action succeeded by a decision node. Users, however, regard the entire procedure as a cohesive task because it is related to a single screen and not completed until a button click has taken place. To improve user comprehension, we merge the action and its following decision node.

In the second part of this step, we eliminate all other decision nodes by replacing them with actions. We stereotype these nodes with «UD» or «SD» for "*user decision*" and "*system decision*", respectively. The emerging actions are regular actions except for the existence of more than one outgoing control flow. The semantics does not comply with UML because UML actually defines the semantics of an action with more than one outgoing control flow as a starting point of concurrent flows [3].

The absence of decision nodes not only improves the comprehension of the user but also reduces the overall complexity of the diagram. For the same reason, we suggest to downsize the notation of merge nodes.

Figure 2 illustrates the user-oriented version

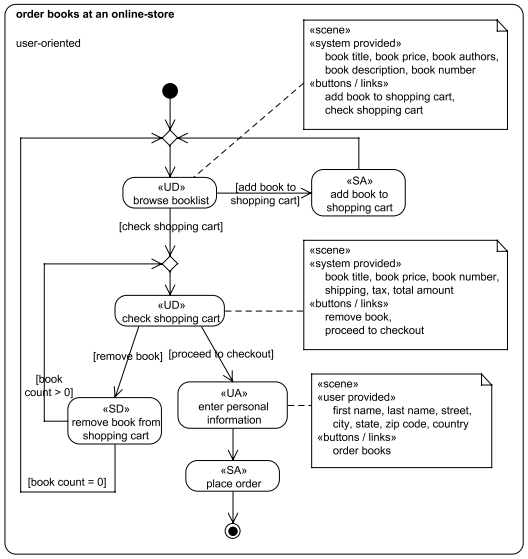


Figure 2: User-oriented activity diagram

of the interaction-oriented activity diagram in figure 1.

## 2.2 Software Specification-Oriented Activities

The *software specification-oriented activity* (*specification-oriented activity*, for short) is a more detailed activity designed to serve as (part of the) software specification driving the implementation process. The transformation of a user-oriented activity to a specification-oriented activity consists of two steps.

The first step expands the user-oriented activity by additional system actions. The expansion step again is divided into two parts. Firstly, if a user node is not directly followed by a system node on some outgoing control flow, an additional system action is inserted as direct successor. The new node serves as placeholder for the system reaction. Secondly, if a user node is not directly preceded by a system node and its related scene contains at least one system-provided attribute, then a system action is inserted as direct predecessor. The additional system action is responsible for pre-filling the scene.

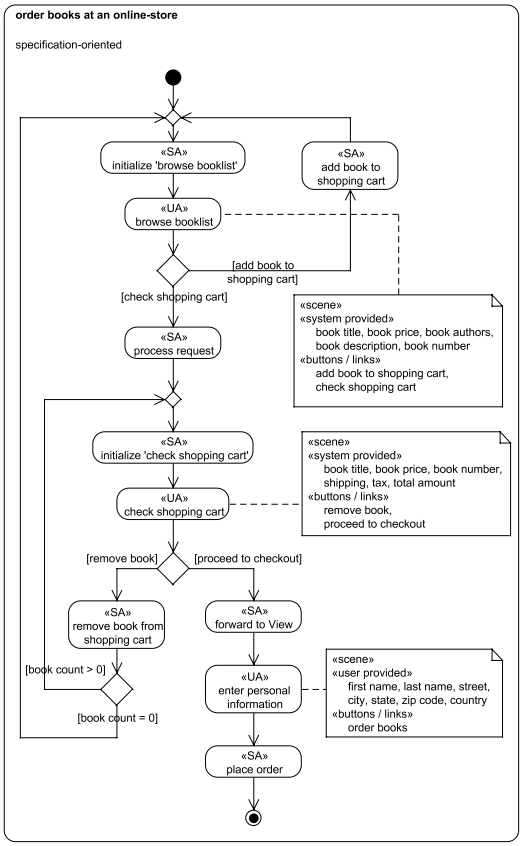


Figure 3: Specification-oriented activity diagram

The determination of a user node where an additional system action must be inserted as direct successor or predecessor can be achieved by a simple syntactical analysis of the activity. Hence, the first step of the transformation can be performed automatically.

The second step restores the UML-compliance of the diagram by reversing the second step of the mapping presented in section 2.1: Each node stereotyped as «UD» or «SD» is simply disassembled into a user action, respectively system action, and a following decision node. Obviously, this step can also be performed automatically. Thus, the entire transformation of a user-oriented activity to a specification-oriented activity can be carried out in an automated way.

Figure 3 depicts the resulting specification-oriented version of the user-oriented activity diagram in figure 2.

# 3 Mapping Specification-Oriented Activities to J2EE with Struts

A software specification-oriented activity is independent of specific platforms like J2EE, Struts, or plain Java. We now demonstrate how such an activity can systematically be mapped to a specific target platform. As an example platform, we choose J2EE [4] with Web tier based on the framework Struts [5]. The mapping comprises of seven steps.

- S1 A class is created that is responsible for the business logic of the activity.
- S2 For each system action with business logic involved, a method is added to the class created in S1.
- S3 For each user action, a `DispatchAction` is created.
- S4 For each system action that follows a user action (with or without a control node in between), a method is added to the corresponding `DispatchAction` created in S3. The method is responsible for processing the user request.

S5 For each system action that does not follow a user action (with or without a control node in between), an `Action` is created (cp. first «SA» and «SA» "initialize check shopping cart" in figure 3). The only method of the `Action` is responsible for implementing the system action.

S6 For each scene, a JavaServer Page (JSP) is created responsible for presenting the information described by the scene.

S7 For each scene that contains user-provided attributes, an `ActionForm` is created accommodating the user input.

Figure 4 depicts the class diagram yielded by the mapping of the activity diagram in figure 3.

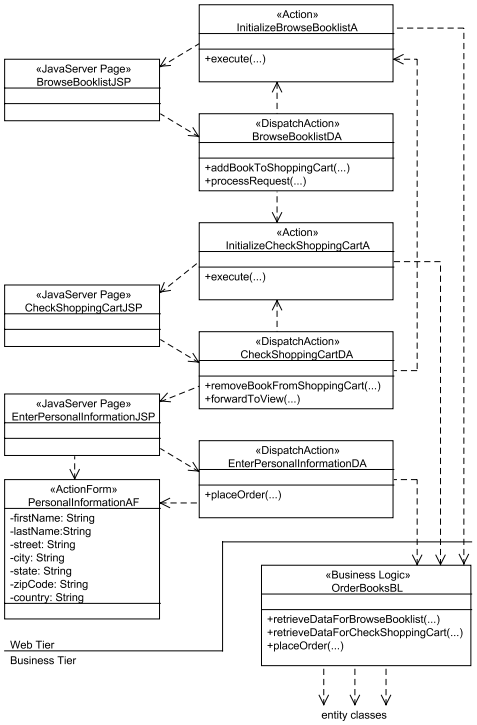


Figure 4: Class diagram resulting from the mapping of the activity diagram in figure 3

# 4 Future Work

Our future work addresses the automated transformation of user-oriented activities to specification-oriented activities using ATL [8]. Currently we are implementing the generation of J2EE-specific code from a specification-oriented activity using the tools MagicDraw [9] and eclipse plug-in openArchitectureWare [10]. Finally we are investigating how user-oriented activities can be validated by an automated execution on a dedicated validation platform.

# About the Authors

Alexander Lorenz is a scientific staff member and PhD candidate at the chair for Software Engineering at the FernUniversitaet in Hagen.

Hans-Werner Six holds the chair for Software Engineering at the FernUniversitaet in Hagen.

# References

[1] Kösters, G., Six, H.-W., Winter, M.: Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specifications, *Requirements Engineering*, Vol. 6, No. 1

[2] Homrighausen, A., Six, H.-W., and Winter, M.: Round-Trip Prototyping Based on Integrated Functional and User Interface Requirements Specification, *Requirements Engineering*, Vol. 7, No. 1

[3] Object Management Group: Unified Modeling Language: Superstructure version 2.0, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML, Aug. 2005

[4] Sun microsystems: Java 2 Platform, Enterprise Edition (J2EE), http://java.sun.com/javaee/index.jsp

[5] The Apache Software Foundation: Struts 1.2, http://struts.apache.org/

[6] Cockburn, A.: Structuring Use Cases with Goals (Part 1), *Journal of Object-Oriented Programming*, Sept.-Oct. 1997, pp. 35-40, http://alistair.cockburn.us/crystal/articles/sucwg/structuringucswithgoals.htm

[7] Lauesen, S.: Task Descriptions as Functional Requirements, *IEEE Software*, March/April 2003, pp. 58-65

[8] Atlas Transformation Language, http://www.sciences.univ-nantes.fr/lina/atl/

[9] No Magic, Inc.: MagicDraw, http://www.magicdraw.com/

[10] openArchitectureWare, http://www.openarchitectureware.org/