3.2 Static Model Creation

Input

- Use case model
- Description of the problem
- Expert knowledge from the are of the problem
- General knowledge

Goal: Creation of a class diagram (first without operations)

Strategy (according to OMT)

1. Identify classes
2. Identify associations
3. Identify attributes
4. Introduce inheritance
5. Restructure the model

3.2.1 Identification of classes

Candidates are

- Persons or roles (e.g., student, employee, client, ...)
- Organisations (e.g., firm, section, university, ...)
- Items (e.g., article, car, house, ...)
- Concepts (e.g., order, lecture, contract, ...)

1. Step: collect candidates

Approach: search for nouns in the use cases.

2. Step: drop unsuited classes

drop criteria:

- redundant classes
- irrelevant classes
- vague classes
- attributes or attribute values
- operations

3. Step: add missing classes

criteria for missing classes:

- attributes from step 2 which are not covered by classes yet
- expert knowledge from the problem area

Example ATM:

1. Step:

Collect nouns in the primary scenario of the use case “Draw money at ATM”

Client
Card
ATM
PIN
User
Code
Number
Consortium
Bank
Choice
Draw, Deposit, Transaction, Statement
Amount
Bounds
Transaction
Completion
Execution
Request
Balance
Money
Receipt
Credit

2. Step: drop unsuited classes

Client

Card

ATM

PIN *(attribute)*

User *(redundant)*

Code *(attribute)*

Number*(attribute)*

Consortium

Bank

Choice *(attribute)*

Amount *(attribute)*

Bounds *(attribute)*

Transaction

Completion *(activity)*

Execution *(activity)*

Request *(activity)*

Balance *(attribute)*

Money *(irrelevant)*

Receipt *(vague)*

Credit*(redundant)*

Draw, Deposit, Transaction, Statement *(attribute values of choice)*

3. Step: add missing classes

- Account (*is implied by the balance attribute*)
- Teller
- Terminal
- Tellertransaction
- Outtransaction (*instead of choice*)

3.2.2 Identify Associations

Candidates are physical or logical connections with a certain duration like

- conceptional connections (works for, is employee of ...)
- property (has, is contained in, ...)
- several objects work together on the same task

1. Step: collect candidates

Approach: check for

- verbs
- genitive (client's credit card)
- possessive pronouns (his credit card)

Note:

- verbs describe also activities (e.g., the ATM checks the PIN)
- many objects within a sentence can indicate a frequent cooperation which is based on a association (e.g., the ATM delegates the bank code to the consortium)
- associations are in general harder to determine than classes. Expert and general knowledge are important.

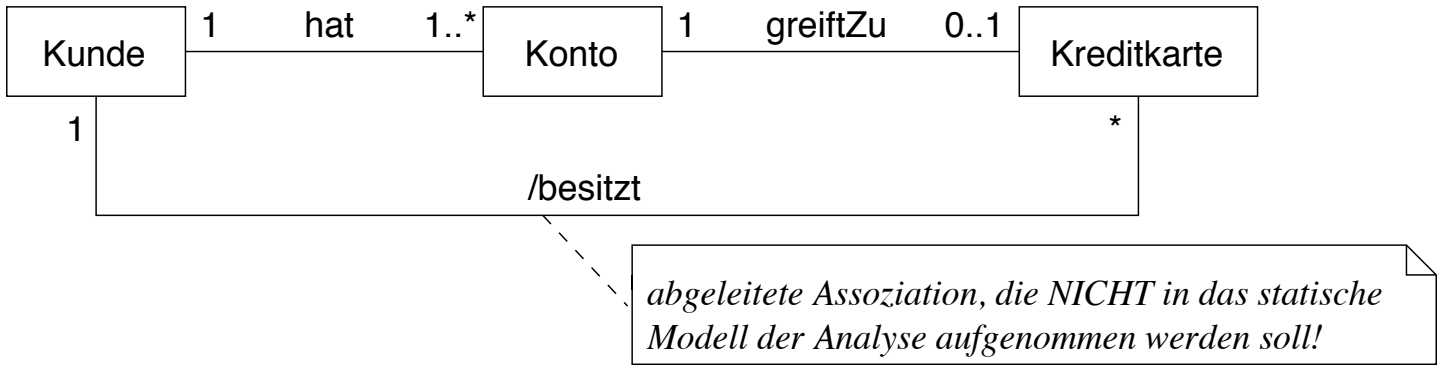
Example ATM:

- The client has a credit card (... his/her credit card ...)
- The consortium has an ATM (... ATM delegates ...)
- The consortium consists of banks (... delegates to the bank ...)
- The bank keeps accounts (Knowledge)
- A client has accounts (Knowledge)

2. Step: drop unsuited associations

Renounce derivated associations if possible.

Example ATM:



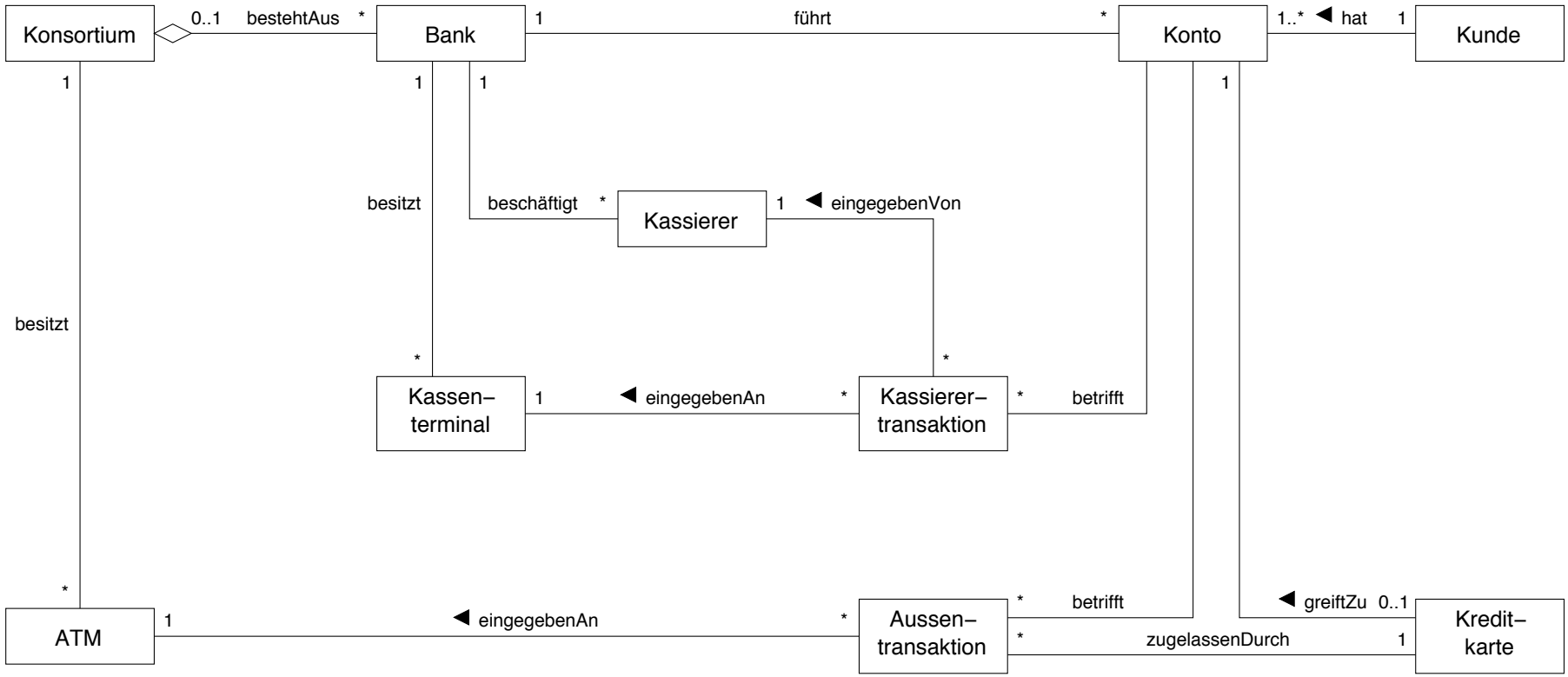
3. Step: add missing associations

4. Step: add perhaps multiplicities and role names

Remark

One can postpone the definition of tricky multiplicities to a later stage.

Example ATM: First class diagram



3.2.3 Identify Attributes

Guideline

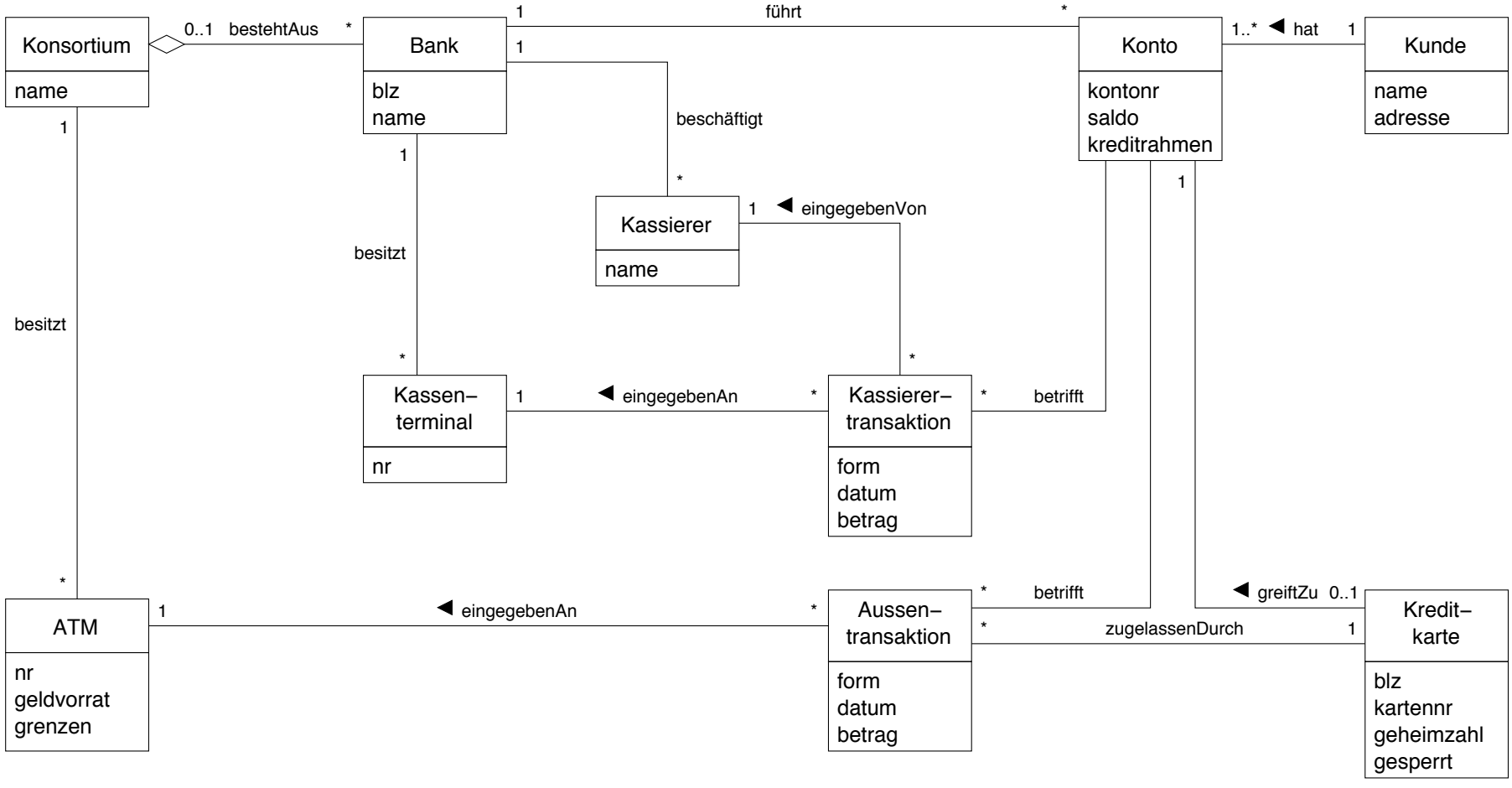
- Only attributes which are relevant for the application (knowledge important)
- No attributes which has objects as values (use associations for this)
- Attribute types can be omitted at this stage

Example ATM:

The primary scenario of the use case "Draw Money at ATM" leads to the following attributes

- blz is an attribute of Bank and Kreditkarte (Card)
- kartennr, geheimzahl are attributes of Kreditkarte
- form, datum and betrag are attributes of Kassierertransaktion (Tellertransaction) and Aussentransaktion (Outtransaction)
- grenzen is an attribute of ATM
- saldo is an attribute of Konto (Account)

Example ATM: Class diagram with attributes



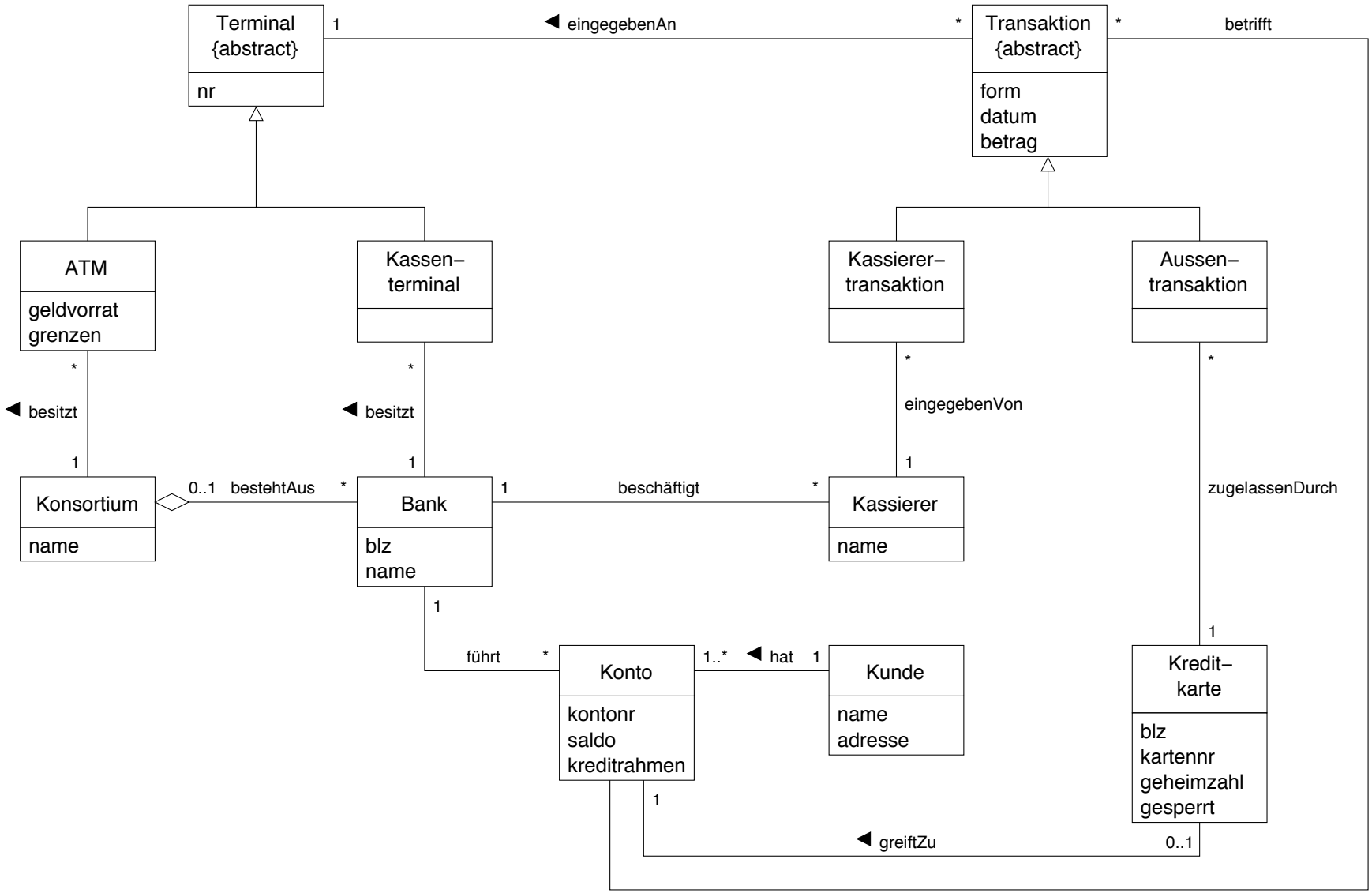
3.2.4 Introducing Inheritance

- Mainly **generalisation**
- Collect common properties of existing classes (attributes, associations) in a superclass.
- Specify which classes are abstract.

Example ATM:

- Aussentransaktion and Kassierertransaktion have the same attributes and a common association to Account.
  $\Rightarrow$ Generalisation to an abstract class Transaktion.
- ATM and Kassenterminal can be generalised to the (abstract) class Terminal.

Example ATM: Class diagram with inheritance



3.2.5 Restructuring the model

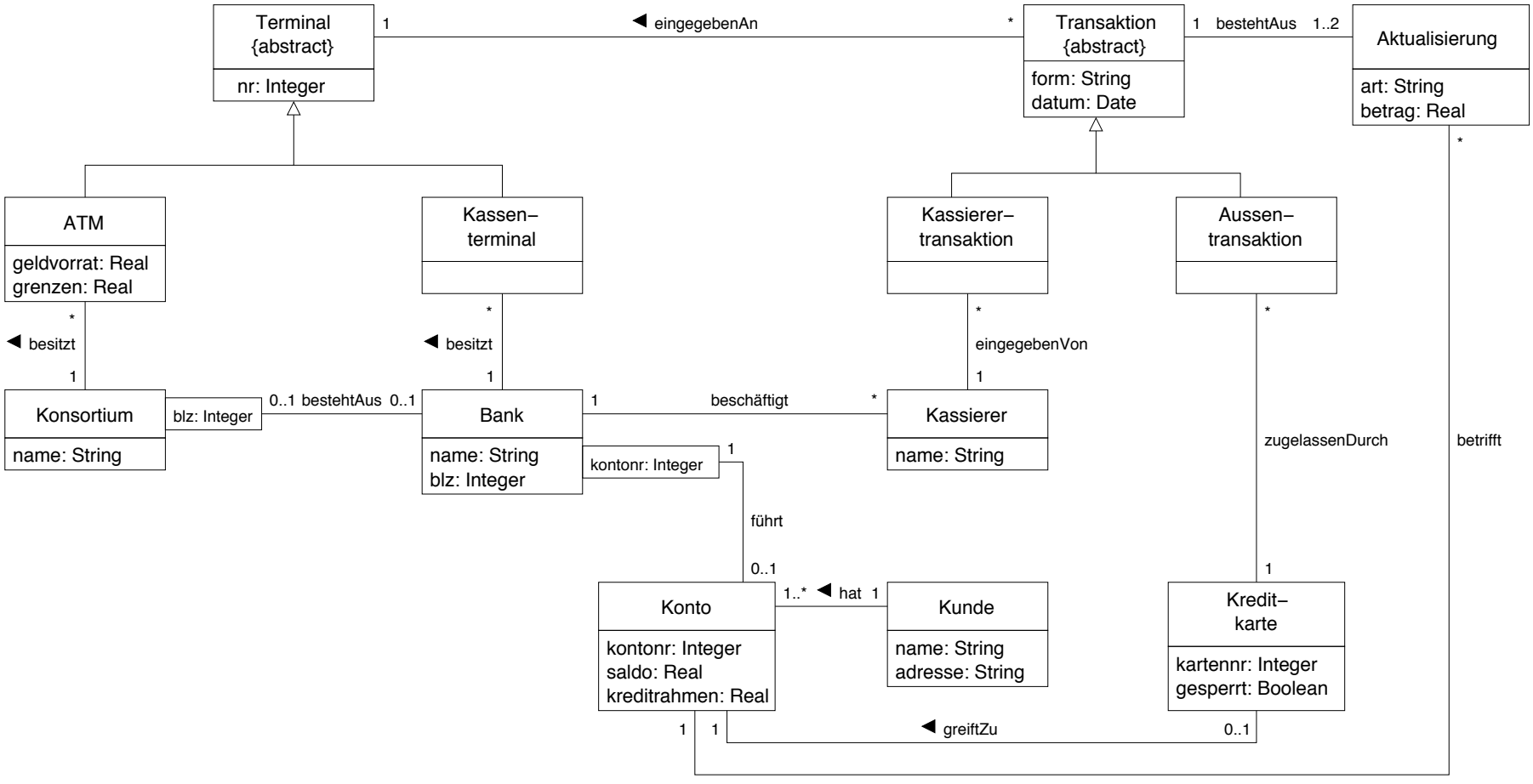
Questions

- Any classes or associations missing?
- Any superfluous classes or associations?
- Any wrong associations or attributes?
- Qualifier for associations?
- Types of the attributes?
- Missing multiplicities?

Example ATM:

- A transactions which affects two accounts
  $\Rightarrow$ introduce the class "Aktualisierung".
- A credit card is a piece of plastic. The corresponding object stores therefore neither the PIN (geheimzahl) nor the bankcode (blz). (redundant)
- Use qualifiers for the associations between Konsortium und Bank and between Bank and Konto.

Example ATM: Restructured class diagramm



Conclusion of Section 3.2

- The static model describe the structural and data orientated aspects of a system.
- The static model is given by class (and object) diagrams.
- The steps in the development of a static model are:
  - Identify classes
  - Identify associations
  - Identify attributes
  - Introduce inheritance
  - Restructure the model

3.3 Modelling of Interactions

Interaction = specific pattern of cooperation and communication between objects for the solution of a task (e.g., use case).

Goal:

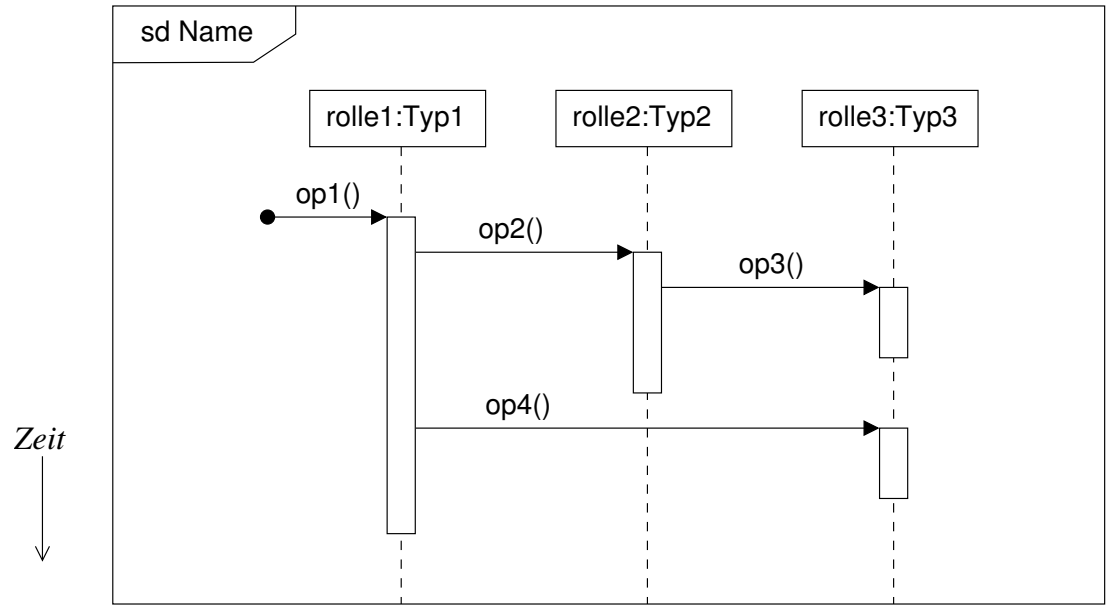
Conception of a set of *interaction diagrams* for each use case.

We distinct between to kinds of interaction diagrams.

- Sequence diagrams
- Communication diagrams

3.3.1 Sequence diagrams

Describe the messaging behaviour of objects over *time* (i.e, send and receive).



Note:

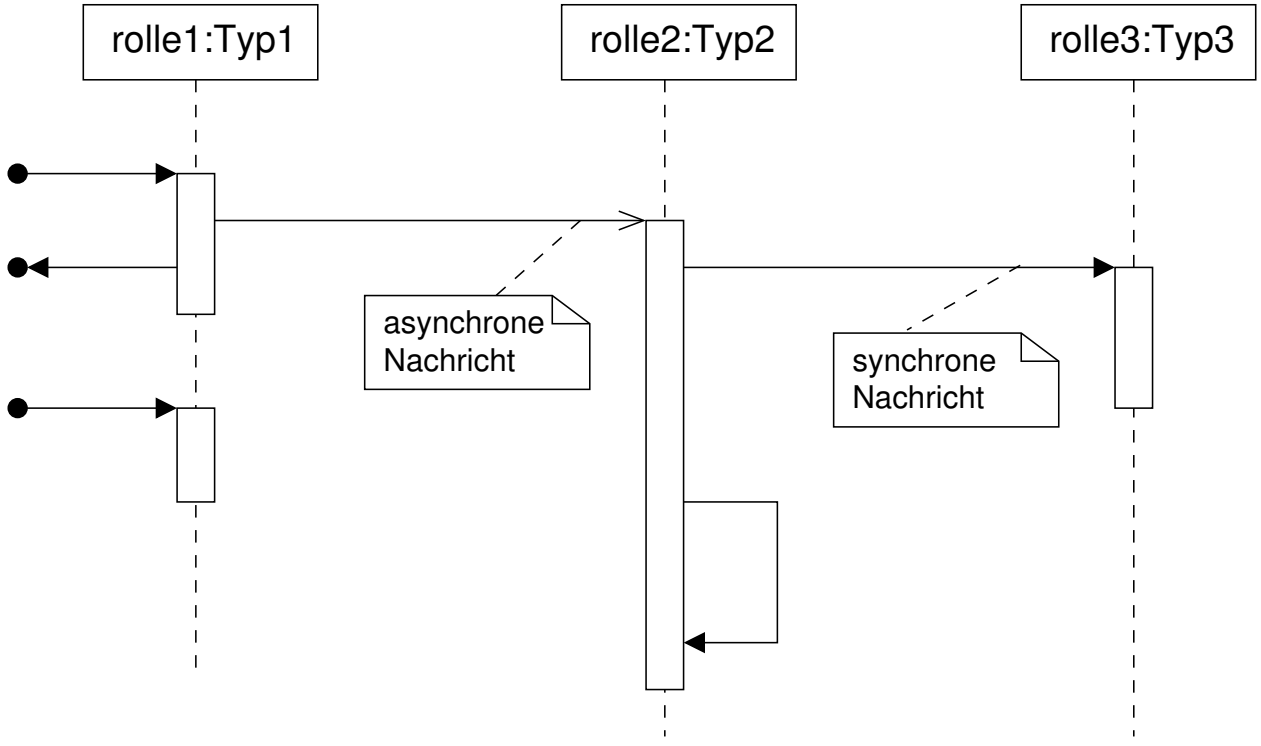
Receiving is an **event**.
Sending is an **action**.

Activation

The time interval in which an object is active because

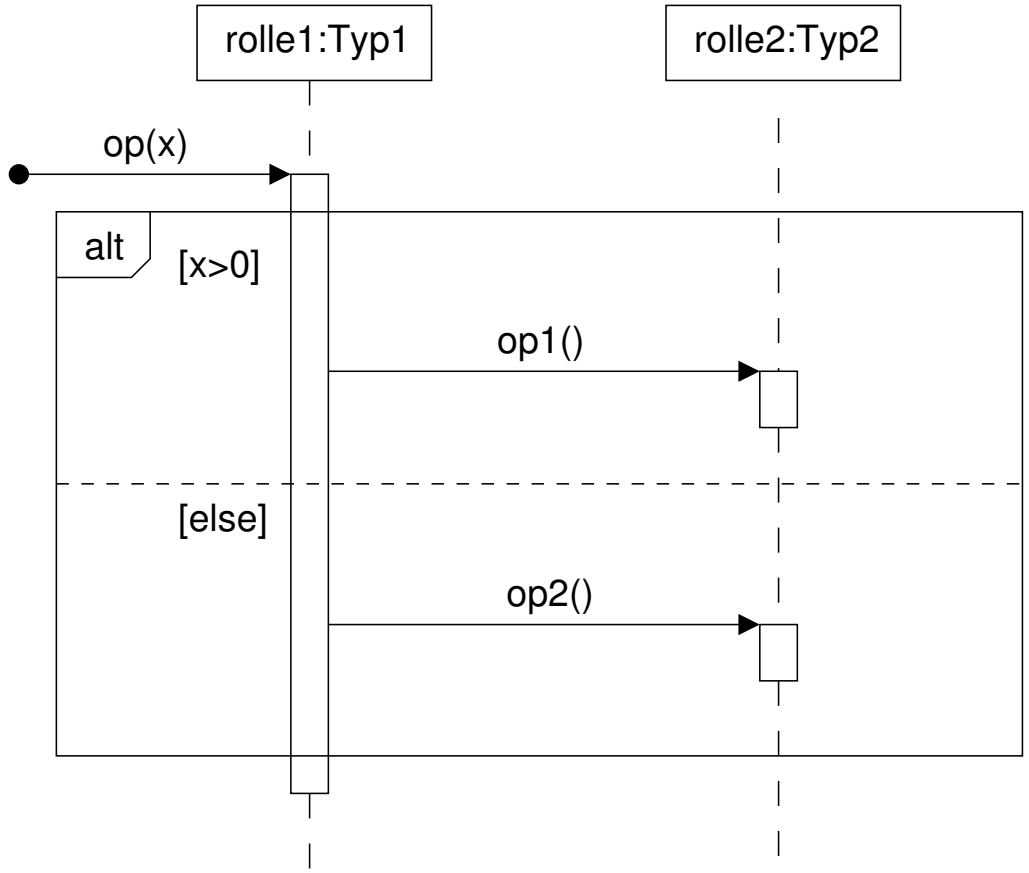
- it performs something;
- it is waiting for the completion of an activity of another object which received a (synchronous) message.

Asynchronous Messages and Parallel Executions



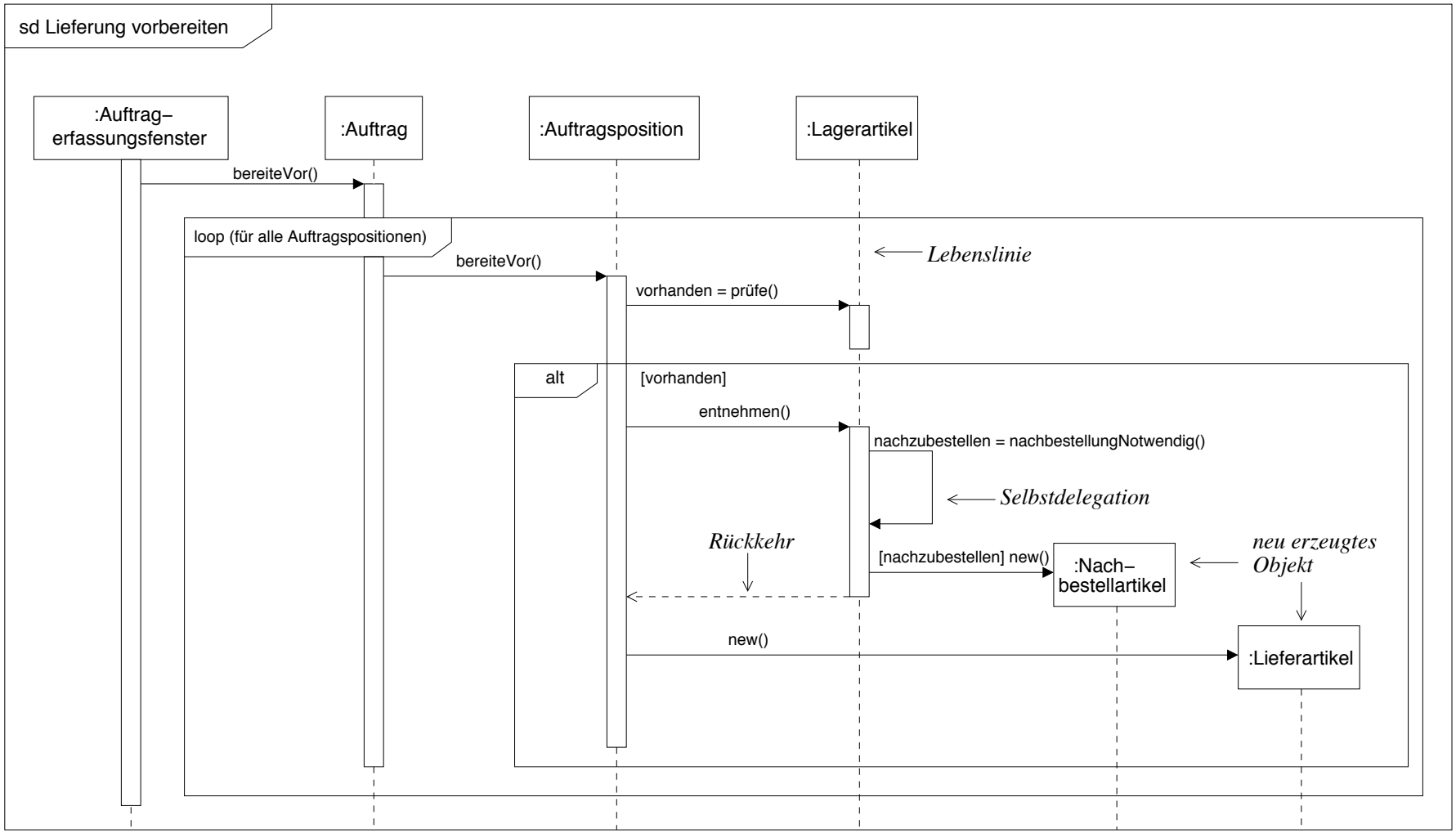
The sending object continues its processing after sending an asynchronous message (i.e., it works in parallel with the receiver).

Alternatives



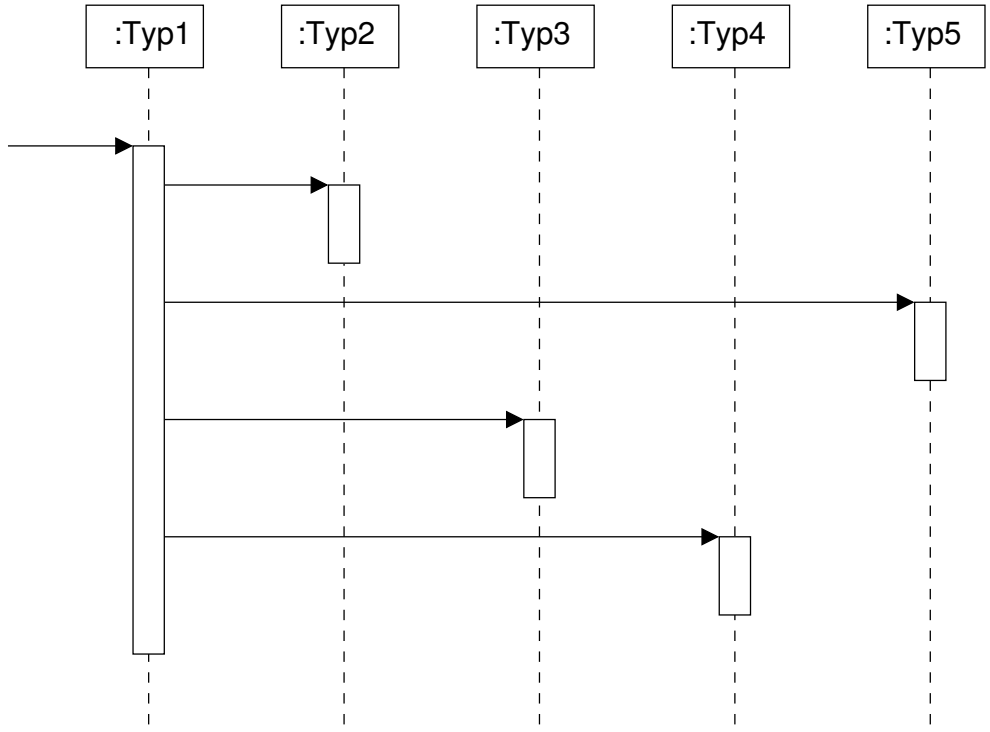
Example: Sequence diagram for "Preparation of Delivery"

UML 2.0



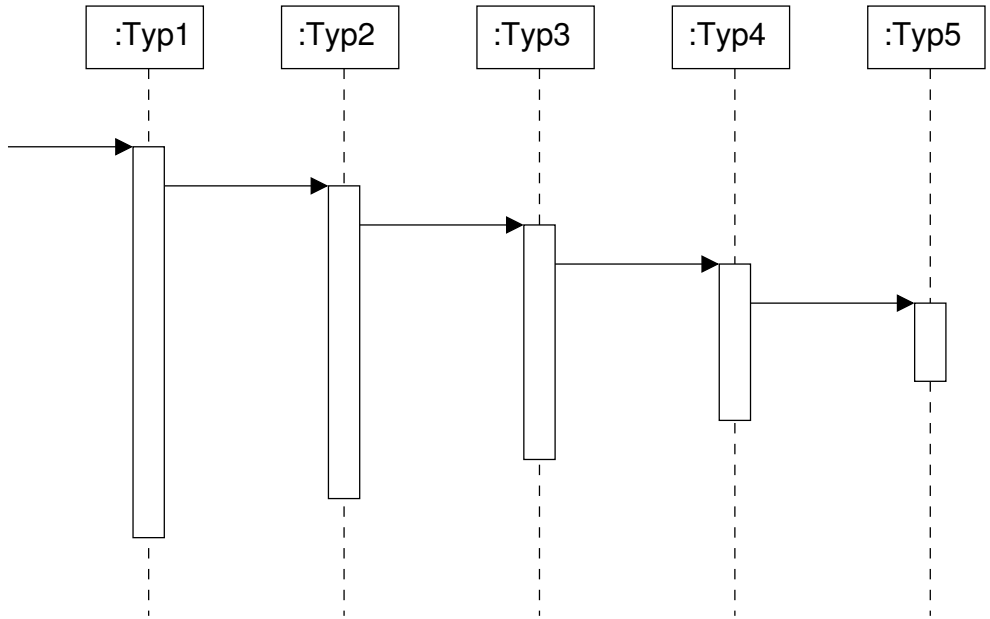
Typical structures of sequence diagrams

Centralised structure ("Fork")



An object controls the other objects (and is responsible for the correct execution).

Decentralised structure ("Stair")

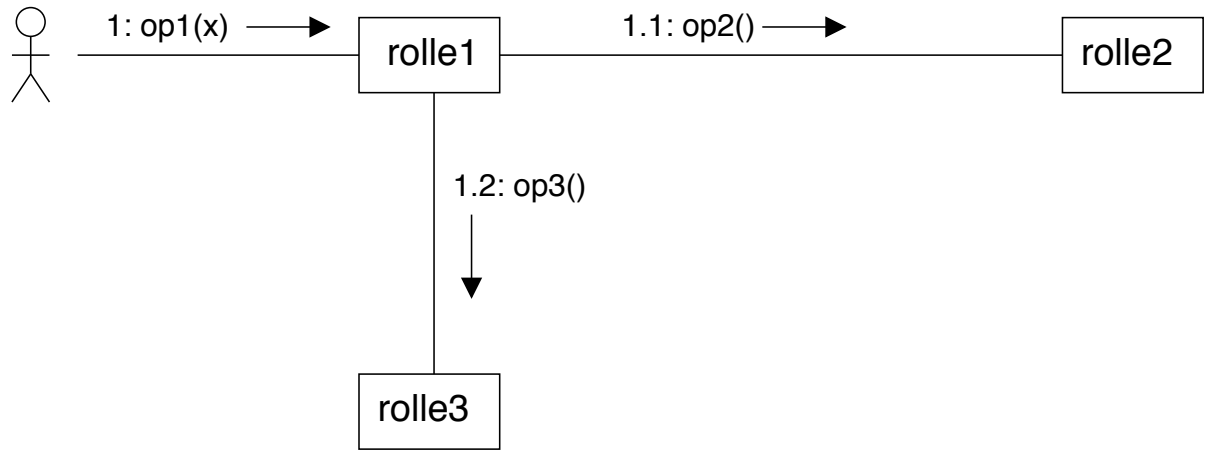


Each object is responsible for the correct execution.

3.3.2 Communication diagrams

Highlight the *structural relations* (temporal and permanent) of all objects which work on an interaction.

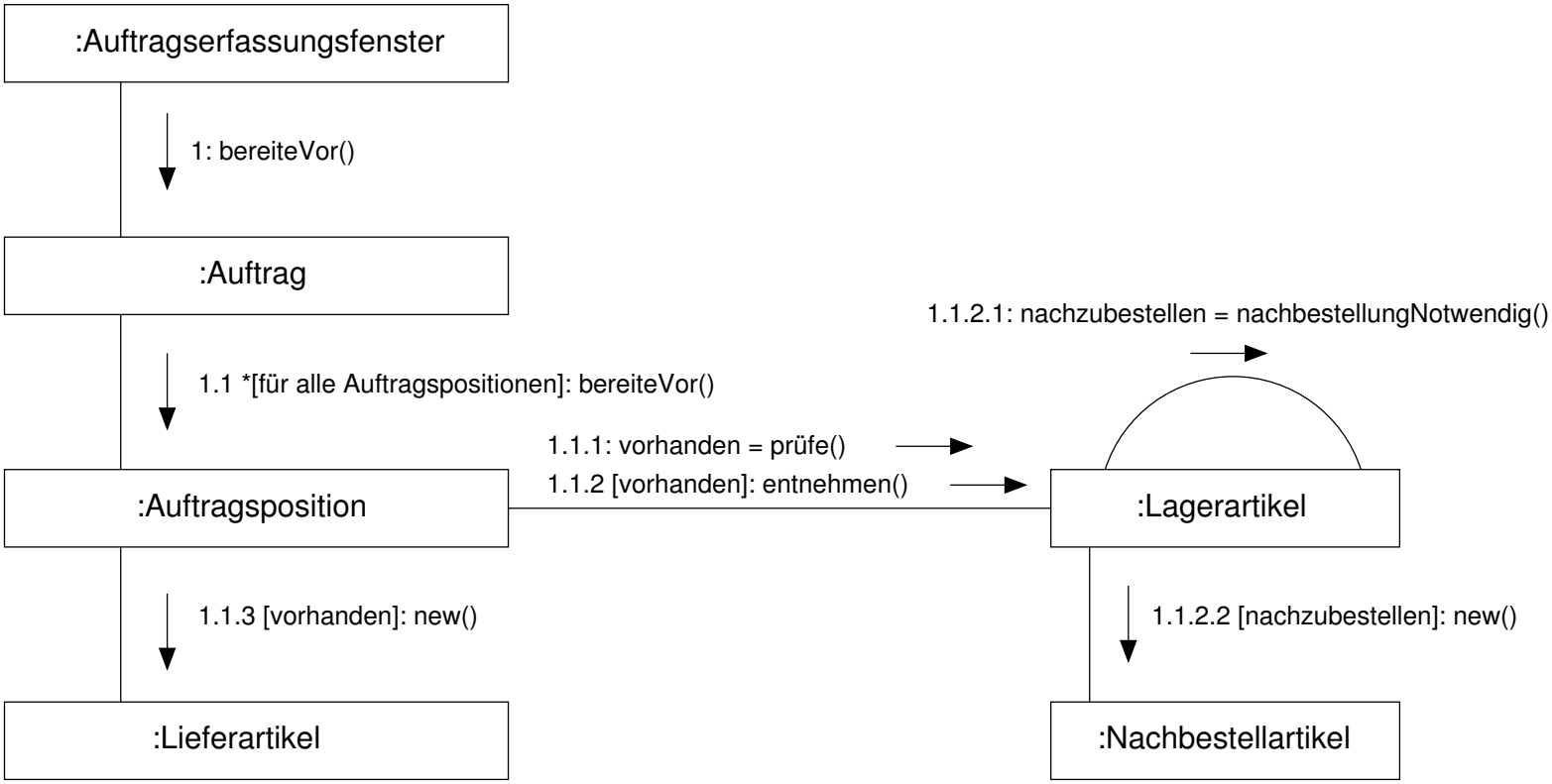
The time is given by an enumeration of the messages (we use 1.1., 1.2. for nested messages).



Remark

The links in the communication diagrams are instances of associations and temporal relations.

Example: Communication diagram for "Preparation of Delivery"



Remark

Sequence and communication diagrams express the same information in a different manner.

3.3.3 Creation of interaction diagrams

Input

- use case descriptions (scenarios)
- static model

Goal

To model cooperation between objects from a use case with interaction diagrams.

Strategy

1. Identify the messages which are exchanged and the objects which send and receive in a use case.
2. Construct interaction diagrams for each use case.

Possible Approaches:

(A) We equip each scenario with its own interaction diagram.

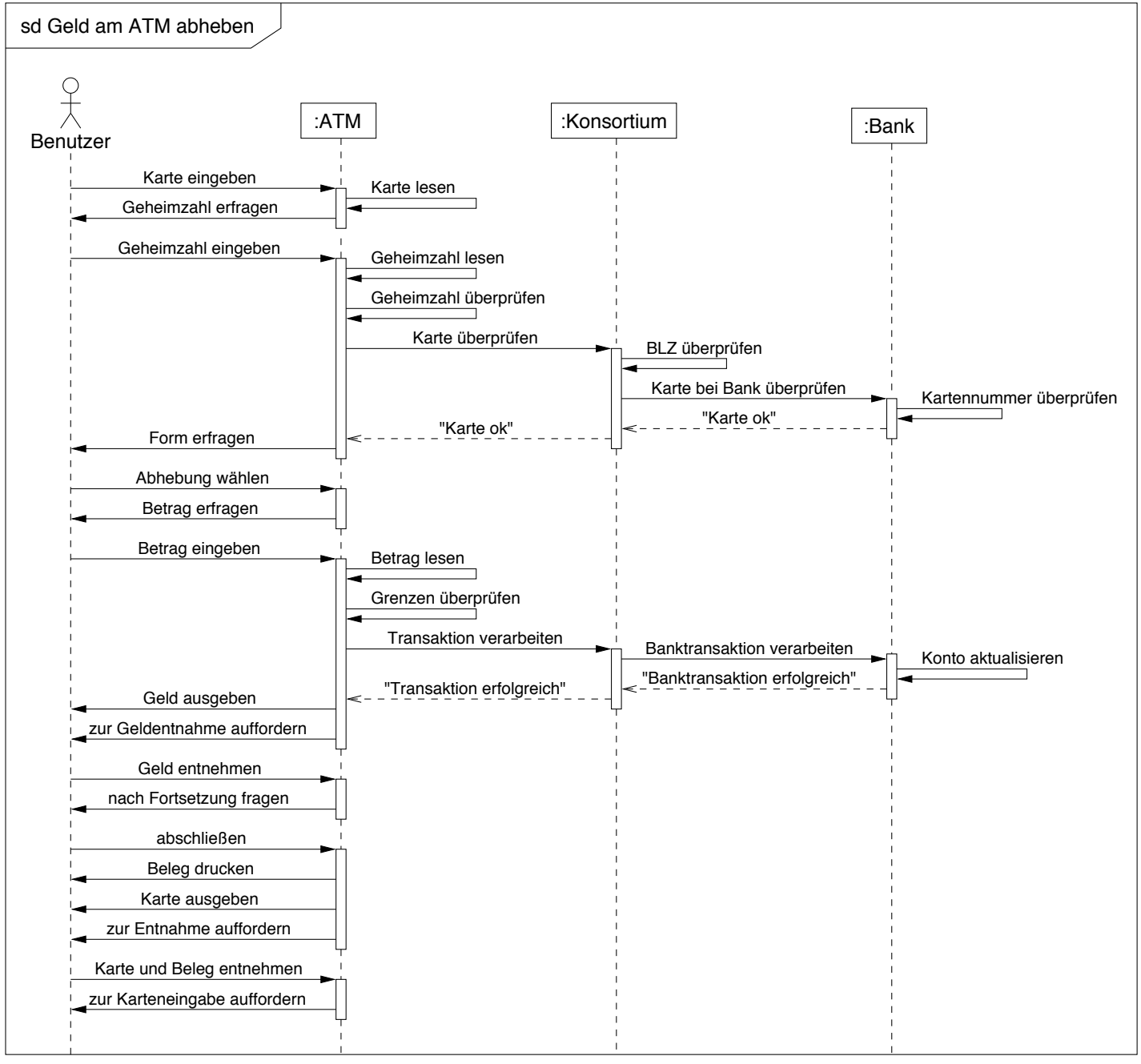
(B) A complex interaction diagram (with alternatives, iterations, ...) which covers all scenarios of a use case
Disadvantage: usually quite complex and large

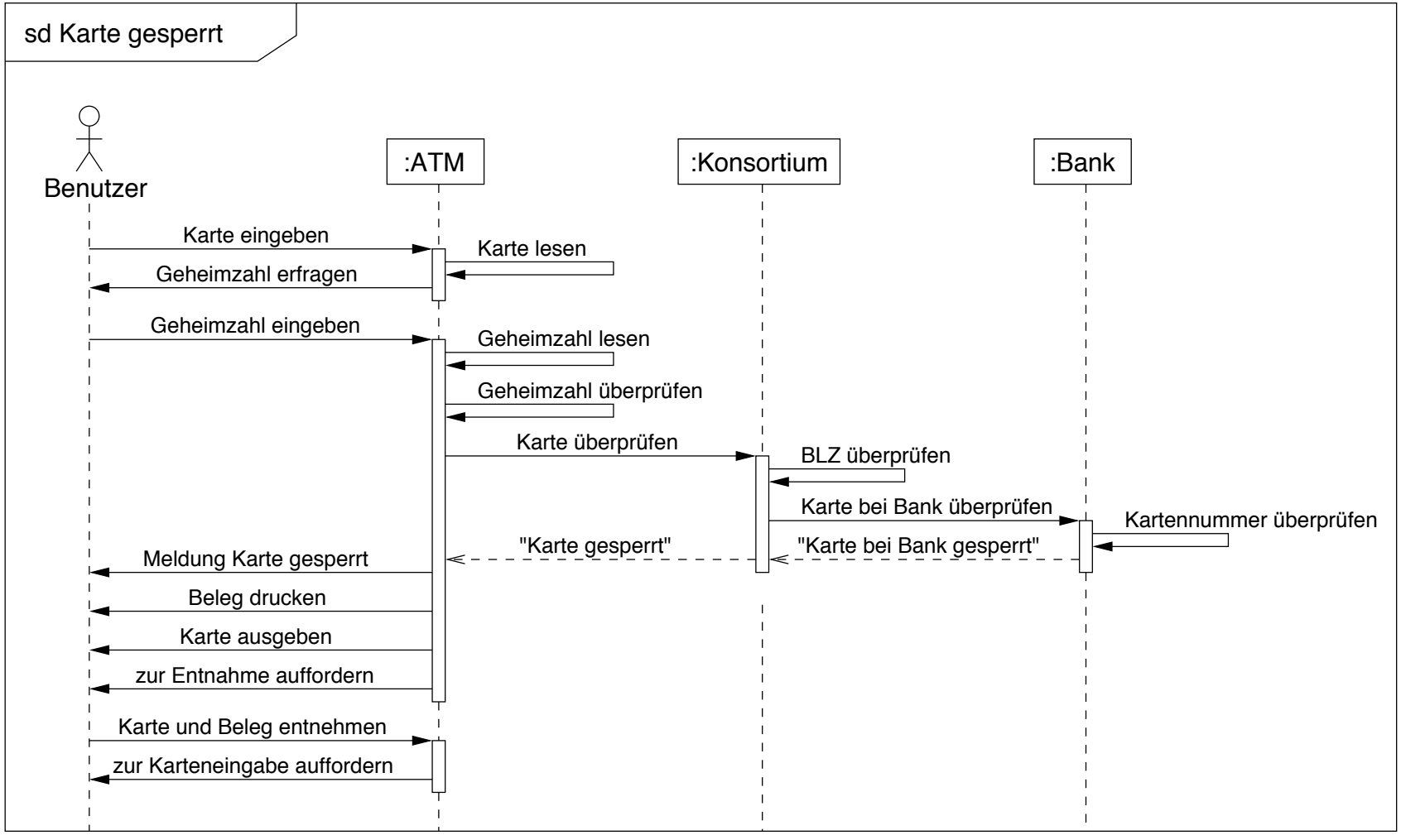
We focus on approach (A) and use sequence diagrams (SDs).

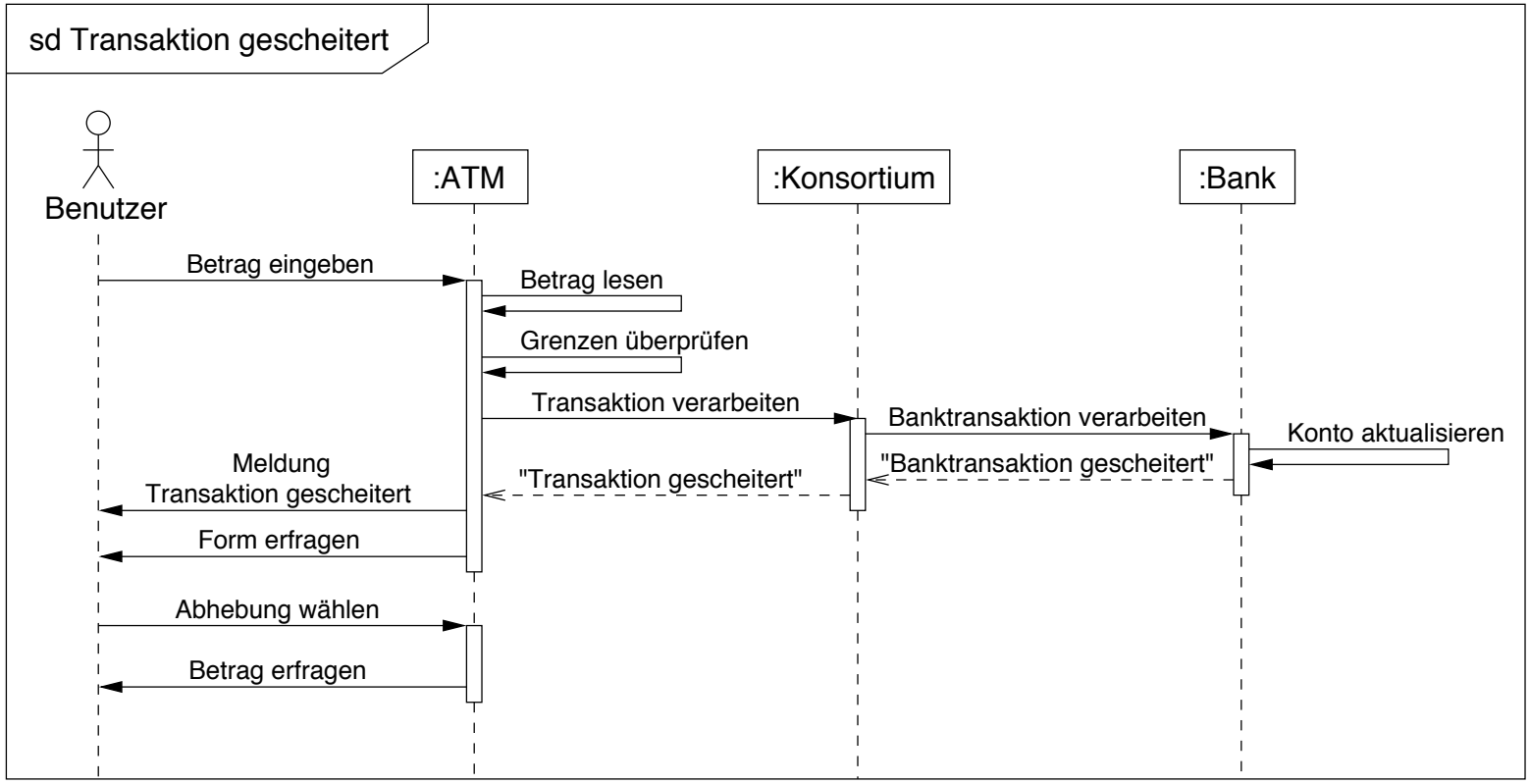
Example: Use Case "Draw Money at ATM"

We construct:

- a SD for the primary scenario
- a SD for the secondary scenario "Credit Card Blocked"
- a SD for the secondary scenario "Transaction Failed"


sd Geld am ATM abheben
Benutzer
:ATM
:Konsortium
:Bank
Karte eingeben
Karte lesen
Geheimzahl erfragen
Geheimzahl eingeben
Geheimzahl lesen
Geheimzahl überprüfen
Karte überprüfen
BLZ überprüfen
Karte bei Bank überprüfen
Kartennummer überprüfen
"Karte ok"
"Karte ok"
Form erfragen
Abhebung wählen
Betrag erfragen
Betrag eingeben
Betrag lesen
Grenzen überprüfen
Transaktion verarbeiten
Banktransaktion verarbeiten
Konto aktualisieren
"Banktransaktion erfolgreich"
"Transaktion erfolgreich"
Geld ausgeben
zur Geldentnahme auffordern
Geld entnehmen
nach Fortsetzung fragen
abschließen
Beleg drucken
Karte ausgeben
zur Entnahme auffordern
Karte und Beleg entnehmen
zur Karteneingabe auffordern


sd Karte gesperrt
Benutzer
:ATM
:Konsortium
:Bank
Karte eingeben
Karte lesen
Geheimzahl erfragen
Geheimzahl eingeben
Geheimzahl lesen
Geheimzahl überprüfen
Karte überprüfen
BLZ überprüfen
Karte bei Bank überprüfen
Kartennummer überprüfen
"Karte gesperrt"
"Karte bei Bank gesperrt"
Meldung Karte gesperrt
Beleg drucken
Karte ausgeben
zur Entnahme auffordern
Karte und Beleg entnehmen
zur Karteneingabe auffordern


sd Transaktion gescheitert
Benutzer
:ATM
:Konsortium
:Bank
Betrag eingeben
Betrag lesen
Grenzen überprüfen
Transaktion verarbeiten
Banktransaktion verarbeiten
Konto aktualisieren
"Banktransaktion gescheitert"
"Transaktion gescheitert"
Meldung
Transaktion gescheitert
Form erfragen
Abhebung wählen
Betrag erfragen

Conclusion of Section 3.3

- Interaction diagrams describe the cooperation and communication between *several* objects.
- We distinct between sequence and communication diagrams (model time respectively structural behaviour).
- A model of interactions is based on the use case description and the static model.
- A use case yields usually several interaction diagrams (each scenario induces one diagram).

3.4 Creation of State and Activity Diagrams

Starting point: a set of SDs from use cases.

Goal

- A state diagram for each class with "interesting behaviour" (i.e., the objects of the class have a non-trivial life cycle)
- Activity diagrams which describe the behaviour of operations

Remark

- State and activity diagrams cover the complete behaviour of *each* object with respect to many scenarios.
- One can omit to introduce an activity diagram if the behaviour of the operation is already given by an interaction diagram.

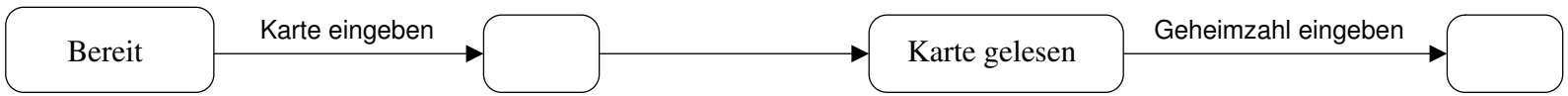
Criteria for "interesting" object behaviour

- There is at least one event which may lead to *different reaction*, dependent on the state of the object.

Example: setting up a digital clock

- At least one event will be ignored in certain states.

Example: ATM



Typical state dependent objects are:

- control objects for use cases and client interfaces
- devices (video recorder, digital clock, ...)
- objects with a limited capacity (full, empty, ...)

State classification

- A state which has at least one activity and can be exited **only if** a completion event occurs is called *activity state*.
- A state which has no activity is called *stable state* (or *inactive state*).

Guidelines for the construction of state diagrams

1. Chose a SD which shows a typical interaction of an object from the class under study (usually the SD of the primary scenario).
2. Project the SD to the lifeline of the object
3. Construct a chain of states and transition from the lifeline of the object such that
   - the intervals in which the object is inactive are modelled by stable states
   - the activations of the object are modelled by activity states (the activity is modelled by local operations)
   - arriving events are modelled by marked transitions from stable to activity states
   - the completion of an activity is modelled by an completion event from an activity to a stable state
4. Express sequences which can be repeated by cycles.

{a certain SD is processed, but without detalisation of the activations yet}

5. Proceed as long as there are SDs of objects from the class under study as follows:
   - choose such a SD and project it to the lifeline of the object
   - find an activity state where the sequence differs from the already inferred behaviour
   - append the new sequence as an alternative to this state
   - find (if possible) a stable state such that one can unify the alternative sequence with the already inferred state diagram

{all SDs processed, but without detalisation of the activations yet}

6. Construct activity diagrams for local operations (those which are called in activity states, cf. step 3)

7. Refine perhaps the existing state diagram by adding conditions to the outgoing transitions of activity states

{all SDs with all activations processed}

8. Integrate (if needed) all secondary scenarios which has no SD yet.

Remark

- The steps 1-5 lead in general to non-deterministic state diagram. Step 7 leads then to a deterministic state diagram.
- A precise formalisation of the above algorithm may be found in

  R. Hennicker, A. Knapp: Activity-Driven Synthesis of State Machines. Konferenzband FASE 2007, Fundamental Approaches to Software Engineering, Springer Lecture Notes in Computer Science 4422, 87-101, 2007.

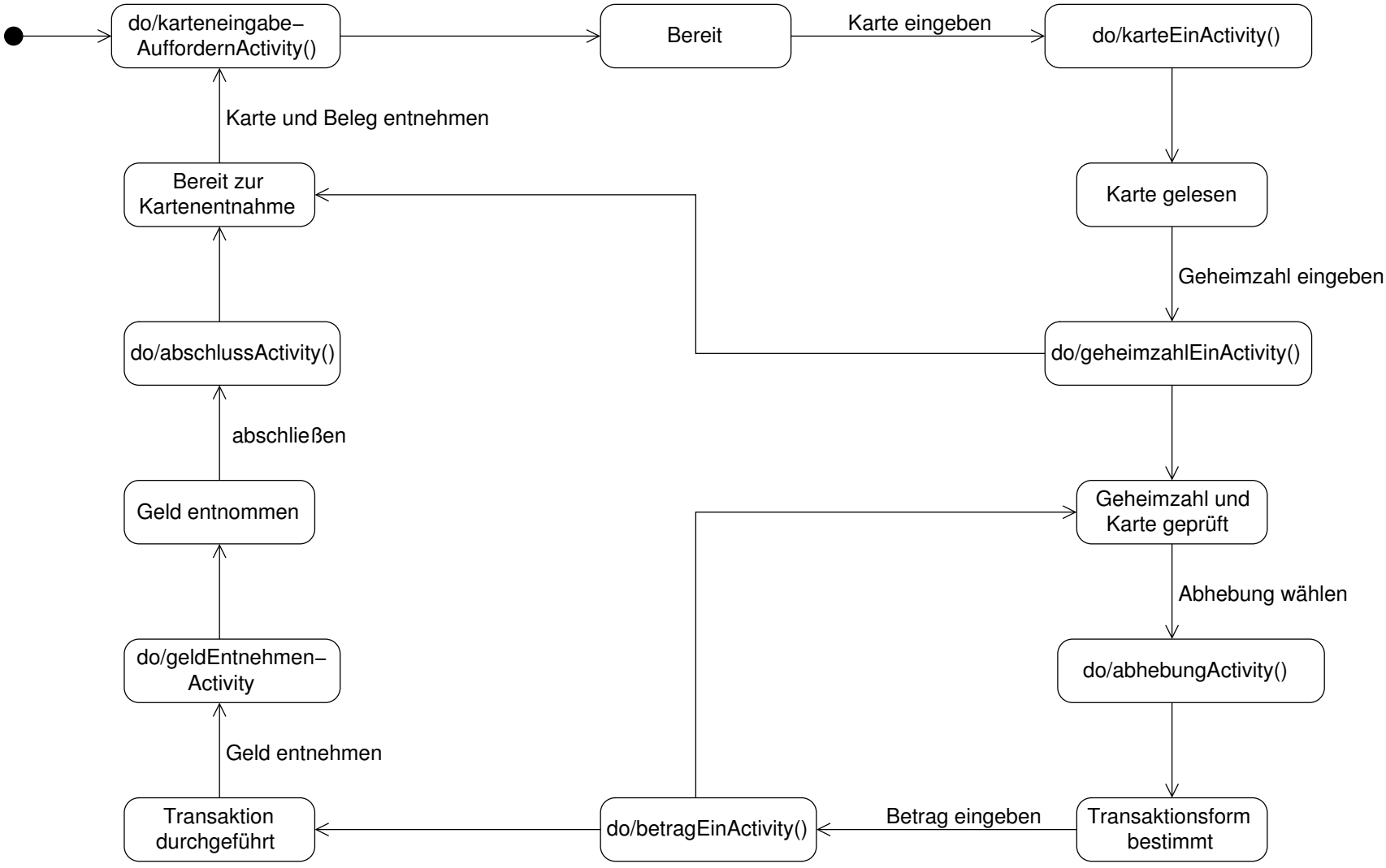
Example ATM:

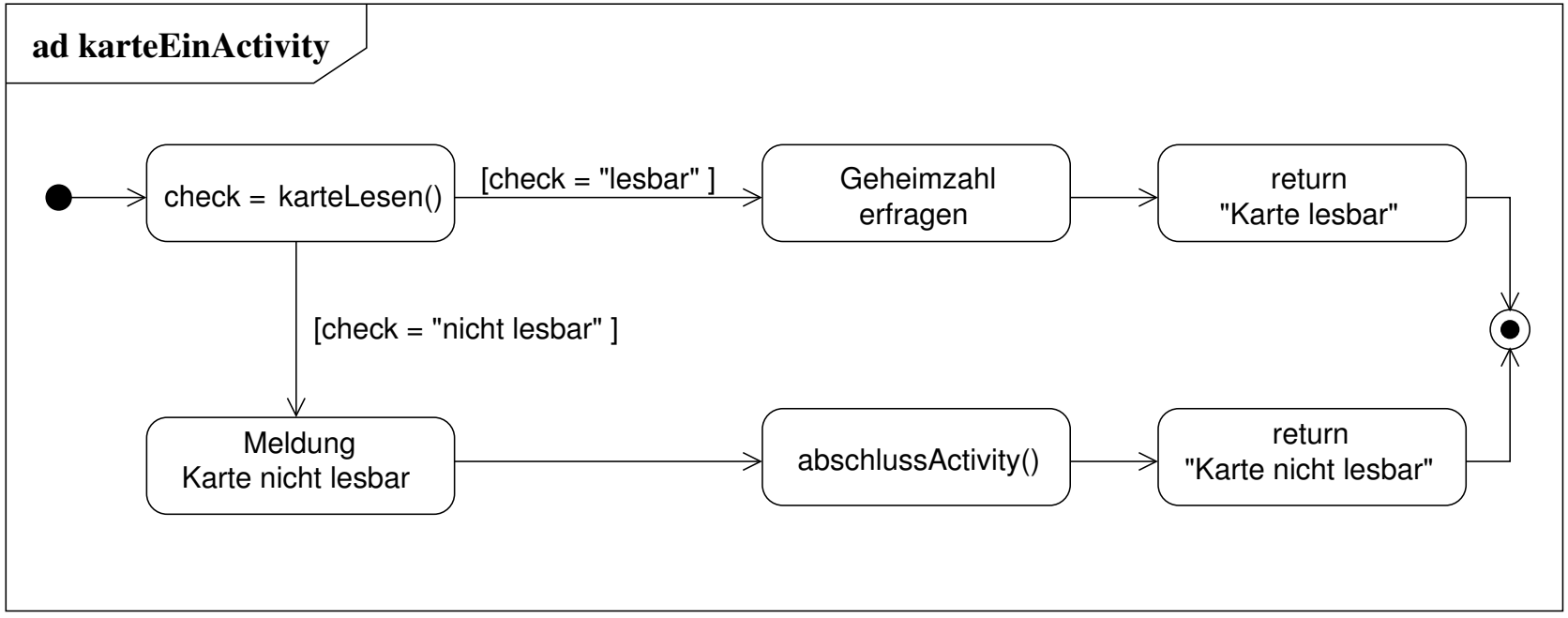
- "Interesting behaviour": ATM (*state diagram*)
- "Operation": (*activity diagram*)
  - ATM:
    - local operations
  - Consortium:
    - check credit card
    - process a transaction
  - Bank:
    - check credit card at the bank
    - process a bank transaction

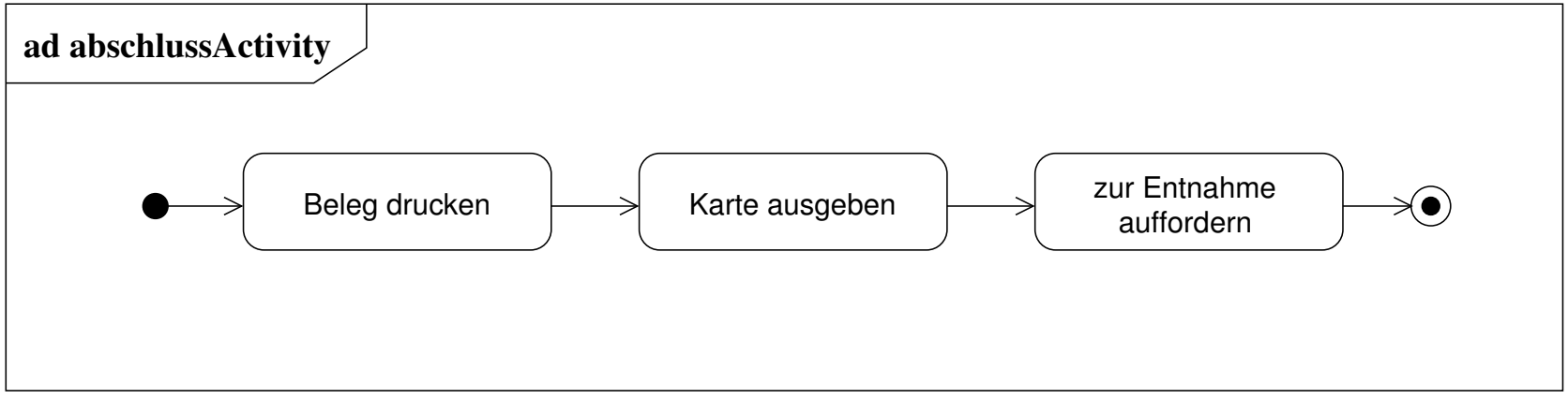
Construction of the ATM state diagram

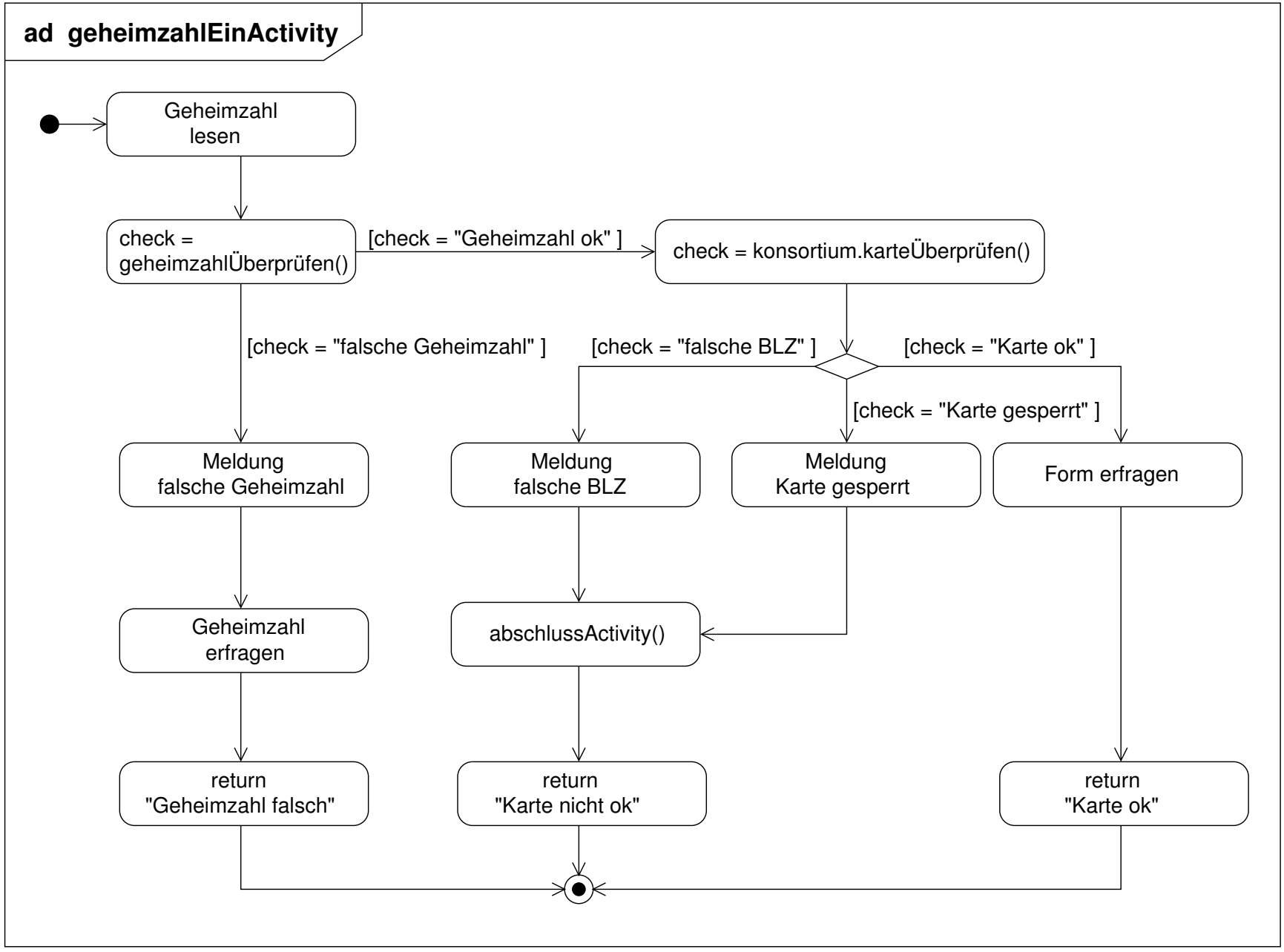
1. Choose SD for the primary scenario of “Draw Money at ATM”
2. Consider the lifeline of “:ATM”
3. Construct a chain of states and transactions which has an activity state after each client interaction
4. Loop at the state ”Bereit”(ready)
5. Integrate SD for ”Karte gesperrt”(card blocked)
   Integrate SD for ”Transaktion gescheitert”(transaction failed)
6. Construct activity diagrams for the local operations which are called in the activity states
7. Introduce conditions for the outgoing transitions of the above activity states
8. Refine the state diagram
9. Integrate secondary scenarios which are not covered yet
   (”Abbruch”, etc.)

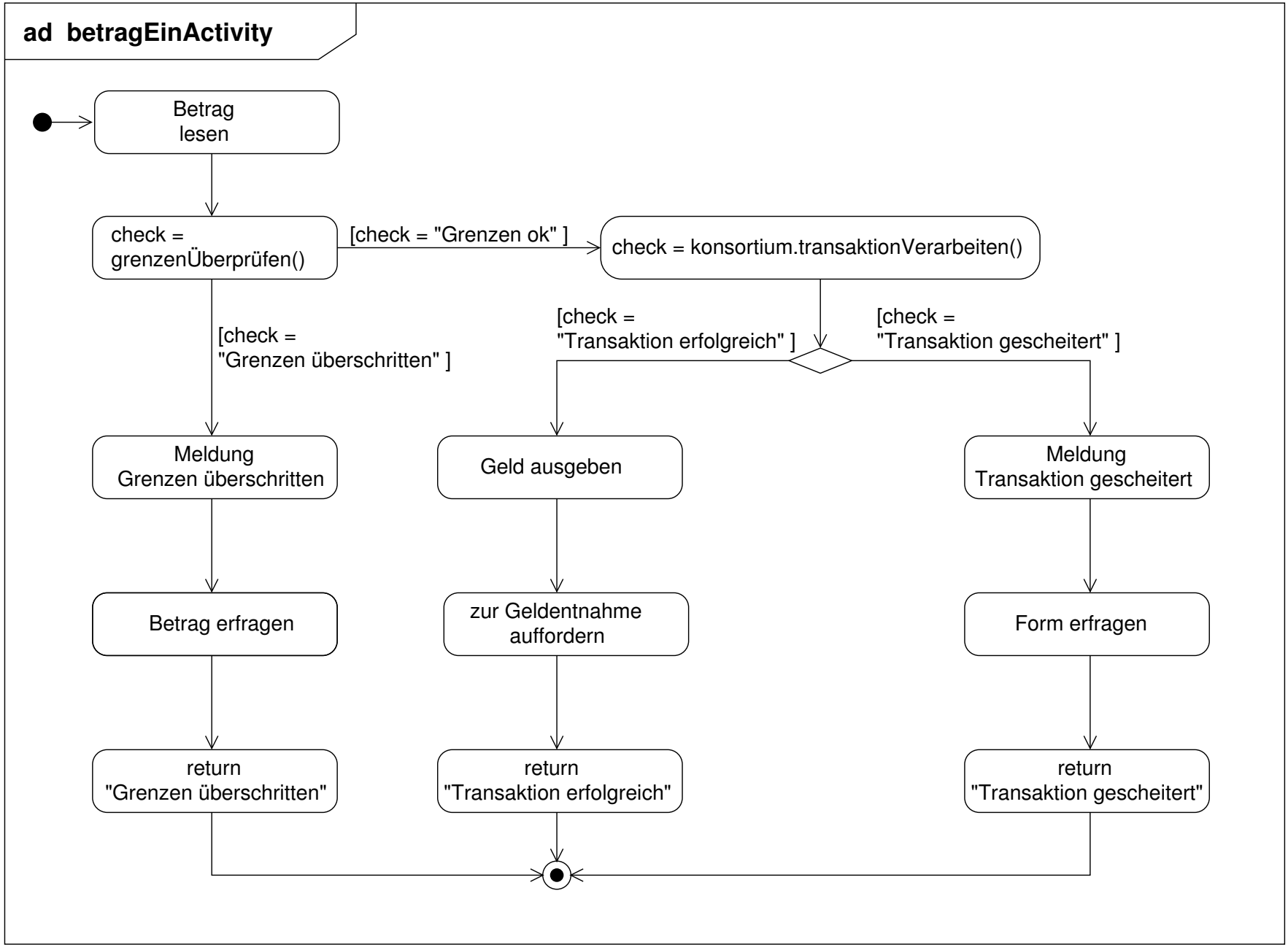
State diagram of the ATM (after the steps 1-5)

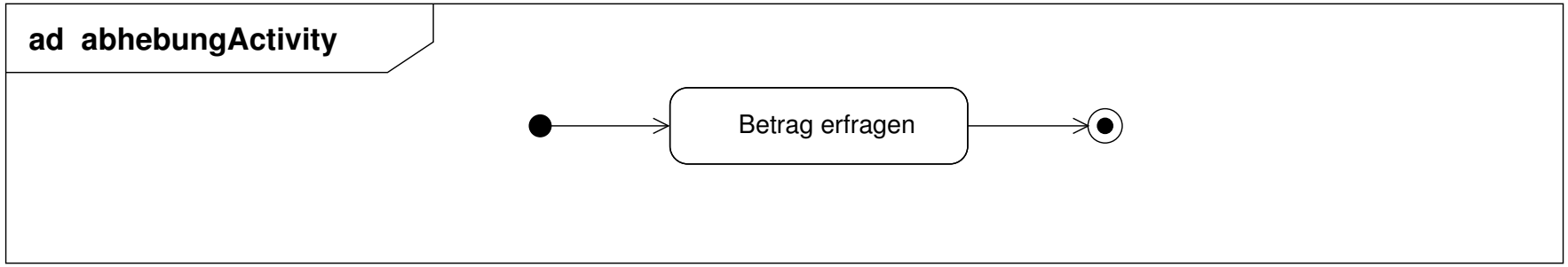


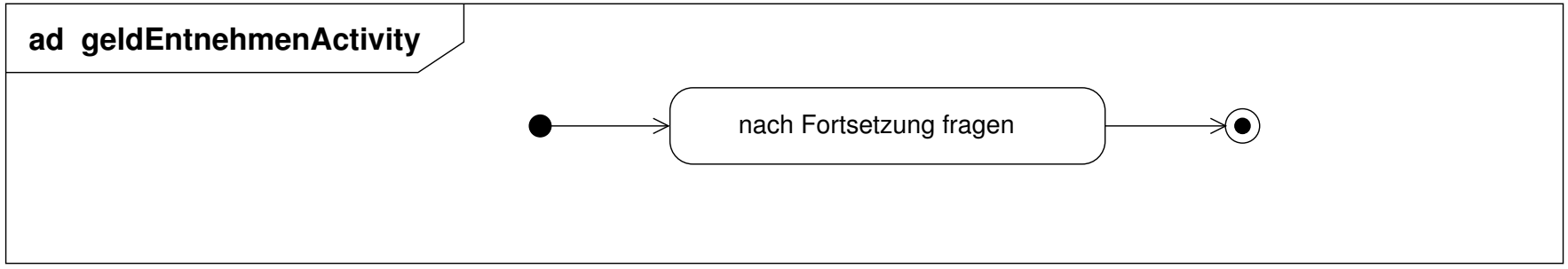

ad karteEinActivity
check = karteLesen()
[check = "lesbar"]
Geheimzahl erfragen
return "Karte lesbar"
[check = "nicht lesbar"]
Meldung Karte nicht lesbar
abschlussActivity()
return "Karte nicht lesbar"

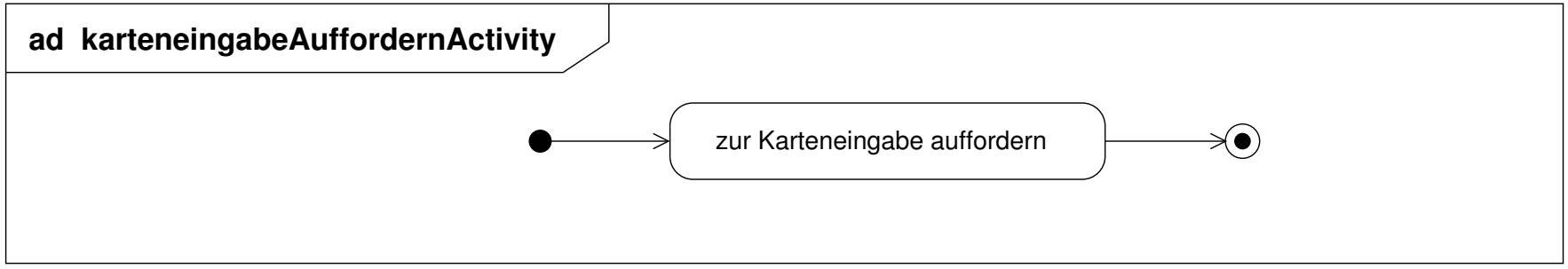

ad abschlussActivity
Beleg drucken
Karte ausgeben
zur Entnahme auffordern


ad geheimzahlEinActivity
Geheimzahl lesen
check = geheimzahlÜberprüfen()
[check = "Geheimzahl ok"]
check = konsortium.karteÜberprüfen()
[check = "falsche Geheimzahl"]
[check = "falsche BLZ"]
[check = "Karte ok"]
[check = "Karte gesperrt"]
Meldung falsche Geheimzahl
Meldung falsche BLZ
Meldung Karte gesperrt
Form erfragen
Geheimzahl erfragen
abschlussActivity()
return "Geheimzahl falsch"
return "Karte nicht ok"
return "Karte ok"

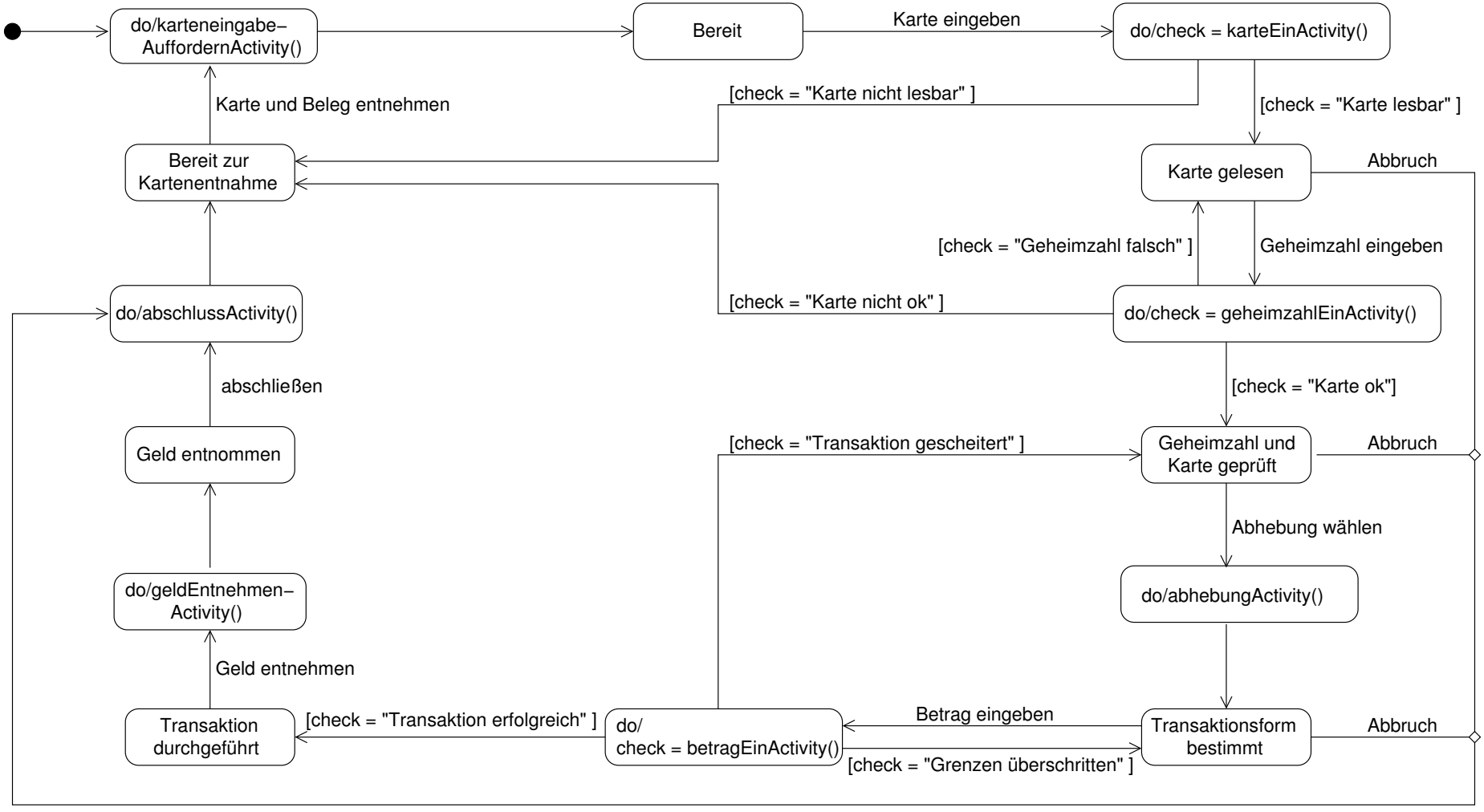

ad betragEinActivity
Betrag lesen
check = grenzenÜberprüfen()
[check = "Grenzen ok"]
check = konsortium.transaktionVerarbeiten()
[check = "Grenzen überschritten"]
[check = "Transaktion erfolgreich"]
[check = "Transaktion gescheitert"]
Meldung Grenzen überschritten
Geld ausgeben
Meldung Transaktion gescheitert
Betrag erfragen
zur Geldentnahme auffordern
Form erfragen
return "Grenzen überschritten"
return "Transaktion erfolgreich"
return "Transaktion gescheitert"


ad abhebungActivity
Betrag erfragen

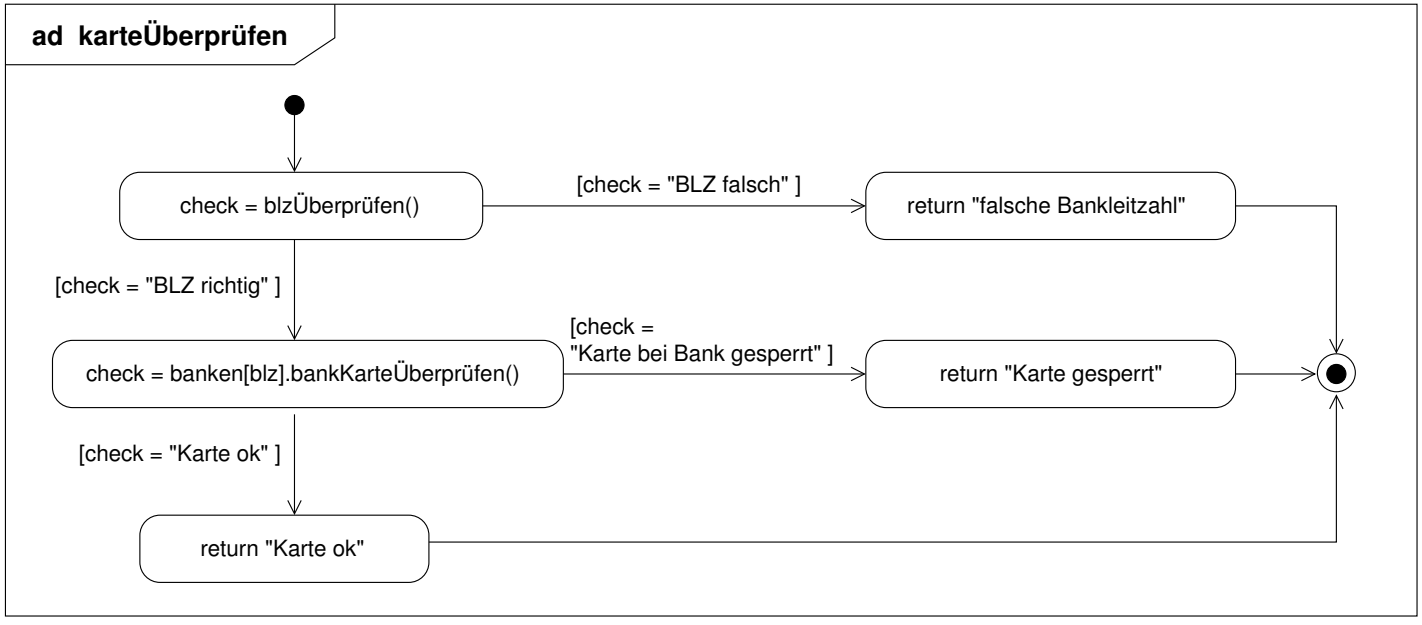

ad geldEntnehmenActivity
nach Fortsetzung fragen

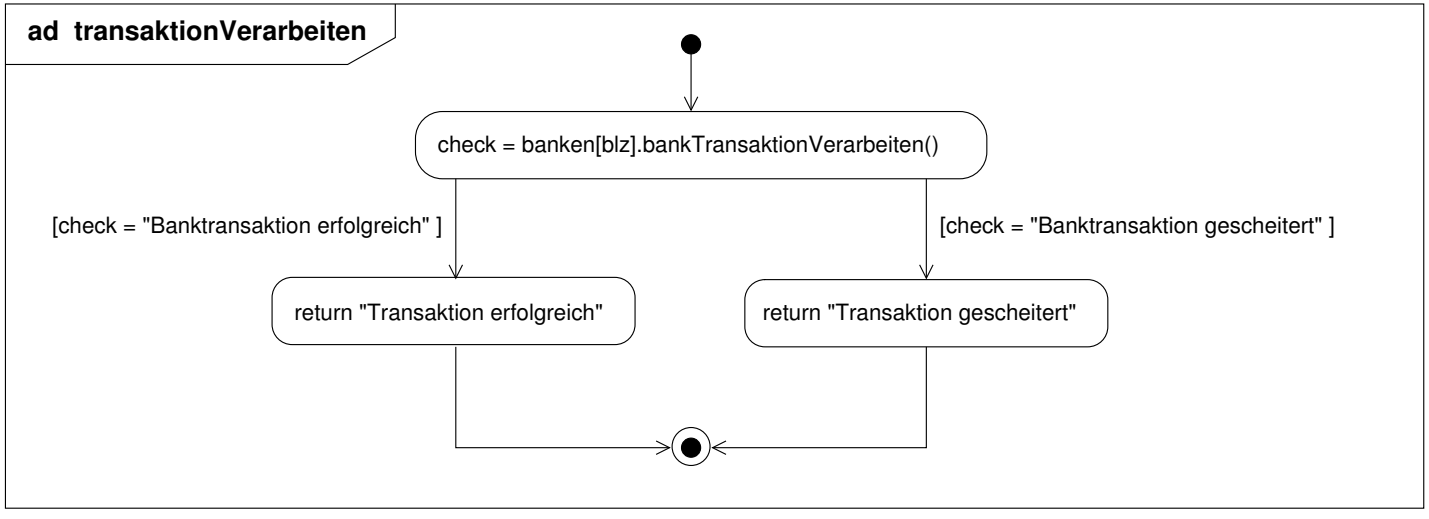

ad karteneingabeAuffordernActivity
zur Karteneingabe auffordern

State diagram of the ATM (after steps 6-8)

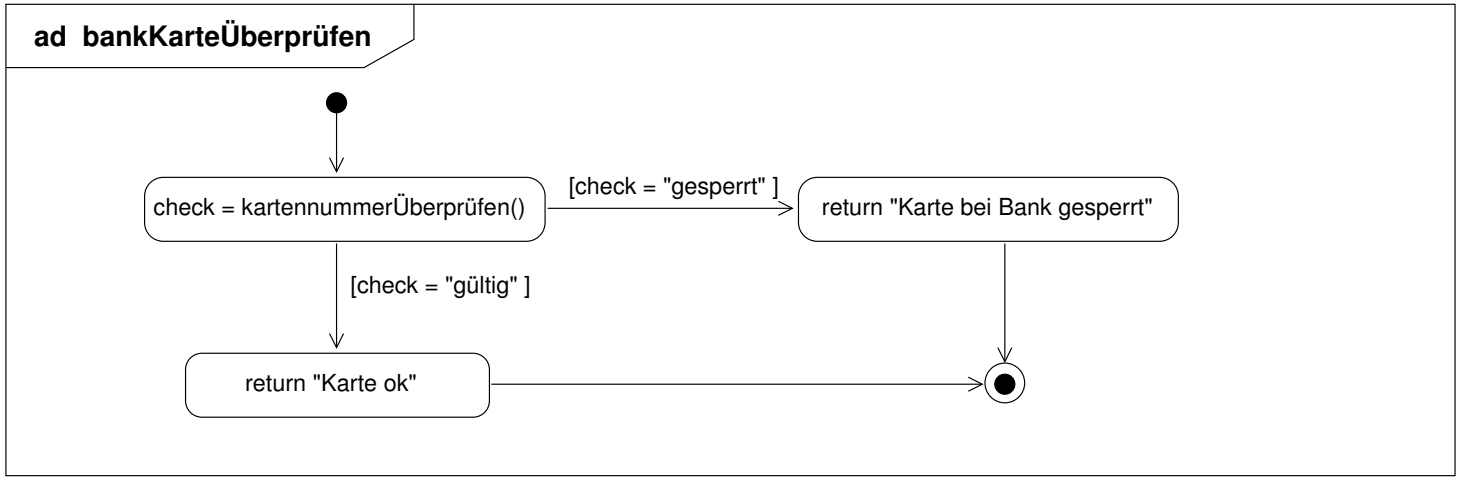


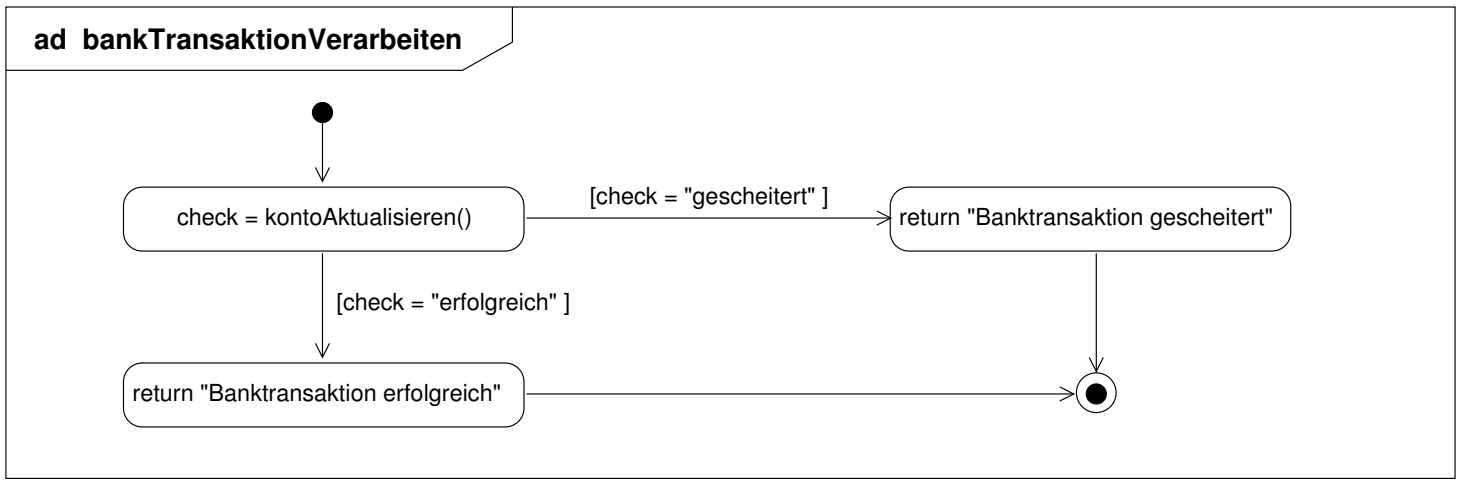
Activity diagrams for the operations of the consortium class





Activity diagrams for the operations of the consortium bank





Remark

The consistency of the diagrams with respect to each other can be checked easily.

Conclusion of Section 3.4

- State and activity diagrams can be constructed systematically from the interaction model
- State diagrams are developed for classes which objects have a non-trivial life cycle
- We distinct between stable and activity states during the construction of state diagrams
- Activity diagrams are used to describe the behaviour of operations