LEHRSTUHL 14
SOFTWARE ENGINEERING

Willkommen zur Vorlesung

# *Methodische Grundlagen des Software-Engineering*

im Sommersemester 2011

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

# 20. Sichere Architekturen

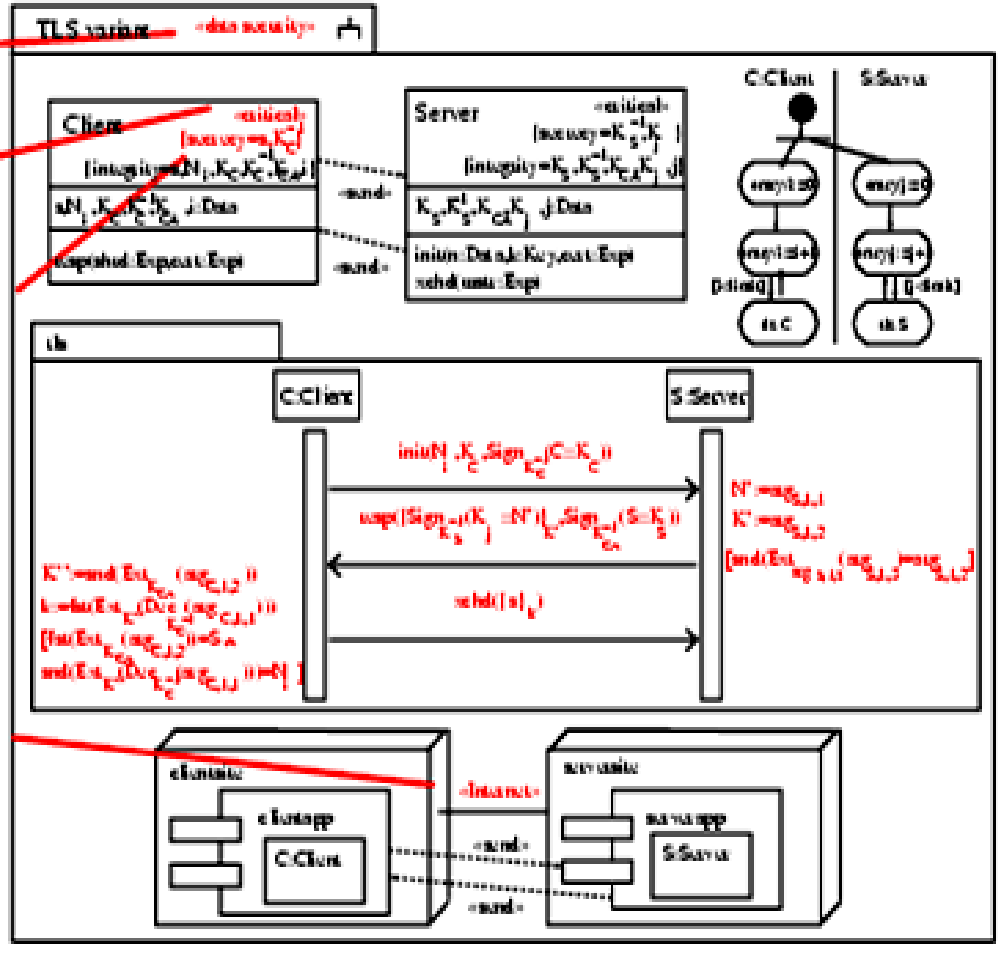
# Einordnung
## Einführung UML / UMLsec

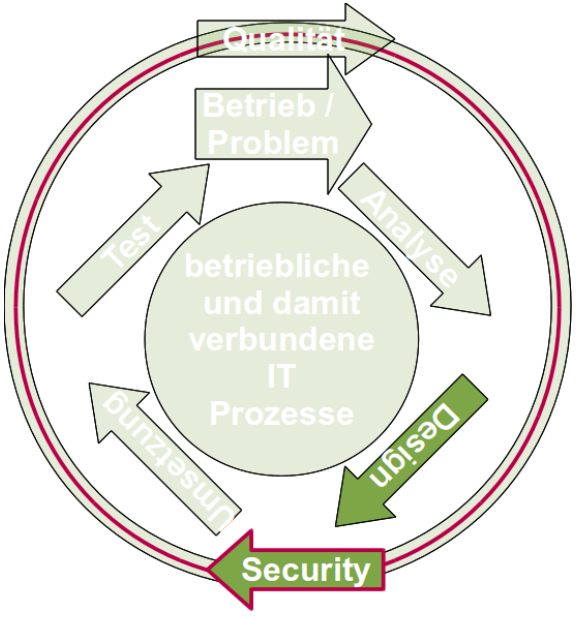
«data security»

«critical»

$\{secrecy = \{s, K_C^{-1}\}\}$

«Internet»



# Einordnung
## Einführung UML / UMLsec

- Business Prozesse
- Qualitätsmanagement
- Testen
- Sicherheit
- Sicheres Software Design
  - Einführung UML / UMLsec
  - Kryptographische Protokolle
  - Biometrische Authentifizierung
  - Elektronische Geldbörse
  - Weitere Anwendungsbeispiele

# Java Sicherheits-Architektur

Ursprünglich (JDK 1.0): Sandkasten-Modell.

Zu simplistisch und restriktiv.

JDK 1.2/1.3: feinere Sicherheitskontrolle (signing, sealing, guarding objects, . . . )

Aber: komplex, also Verwendung fehleranfällig.

# Java Sicherheits-Politiken

Berechtigungseinträge bestehen aus:

- Schutzdomänen (URL's und Signaturschlüssel)
- Ziel-Resourcen (z.B. Dateien auf lokaler Machine)
- zugehörige Berechtigungen (z.B. read, write, execute)

# Signed und Sealed Objects

Brauchen Integritätsschutz für Objekte, die zur Authentisierung verwendet oder zwischen JVMs ausgetauscht werden.

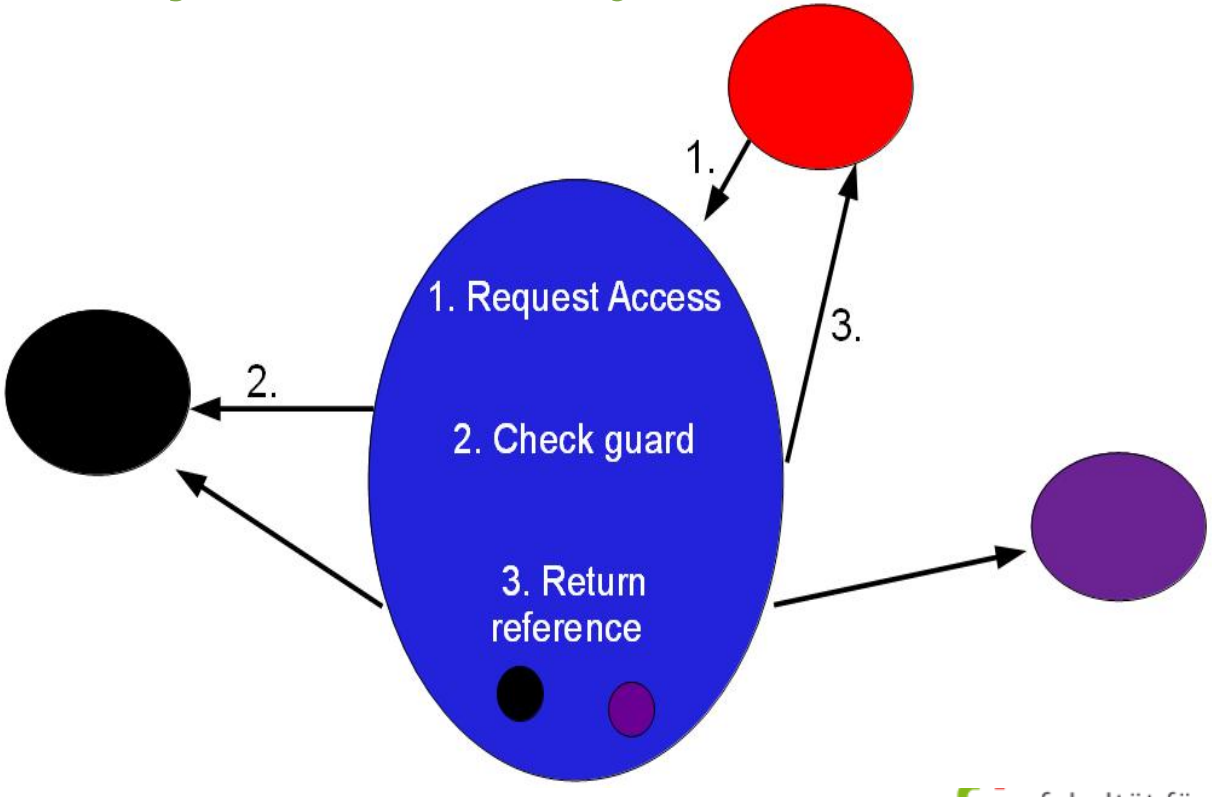
SignedObject enthält Objekt und seine Signatur.

Für Vertraulichkeitsschutz: SealedObject ist ein verschlüsseltes Objekt.

# Guarded Objects

java.security.GuardedObject schützt Zugang zu anderen Objekten.

- Zugang über getObject Methode.
- checkGuard Methode in java.security. Guard kontrolliert Zugang.
- Gibt Referenz oder SecurityException.



# Problem: Komplexität

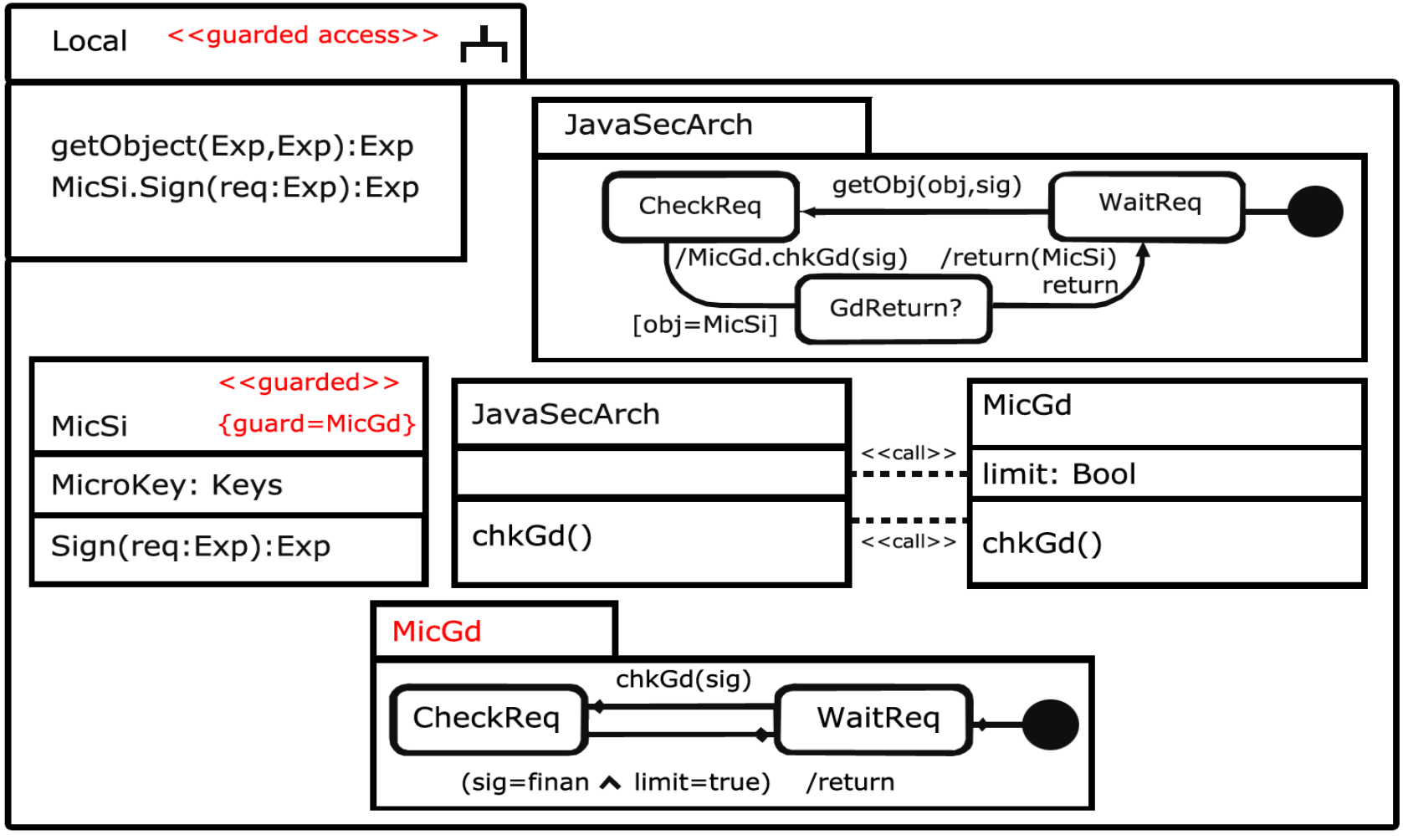
- Rechtevergabe abhängig von Ausführungskontext.
- Zugangskontrollentscheidungen bezüglich verschiedener Threads.
- Können verschiedene Schutzdomäne betreffen.
- Methode doPrivileged() unabhängig von Ausführungskontext.

Gibt Werkzeuge zur Überprüfung.

# Entwurfsprozess

- Zugangskontroll-Anforderungen für sensitive Objekte formulieren.
- Guard objects mit Zugangskontrollen definieren.
- Überprüfen, dass Schutz der guard objects hinreichend ist.
- Überprüfen, dass Zugangskontrolle konsistent mit Funktionalität ist.
- Überprüfen, dass mobile Objekte hinreichend geschützt sind.

# Frage




Guarded objects korrekt eingesetzt ?

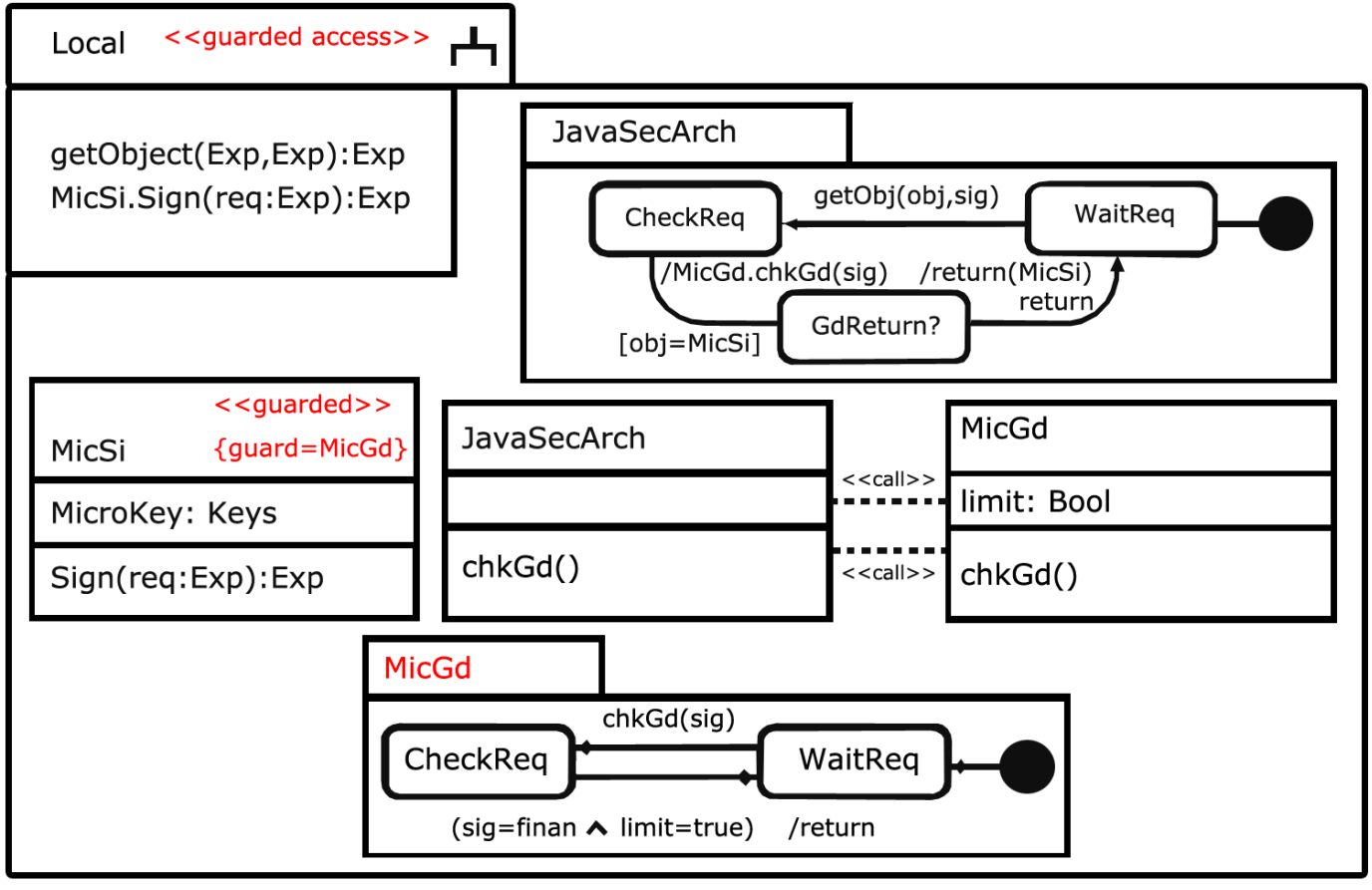
# <<guarded access>>

Stellt sicher, dass in Java <<guarded>> Klassen nur durch {guard} Klassen aufgerufen werden können.

Constraints:

- Referenzen der <<guarded>> Objekte bleiben geheim.
- Jede <<guarded>> Klasse ist durch eine zugehörige {guard} Klasse abgesichert.

# Example <<guarded access>>

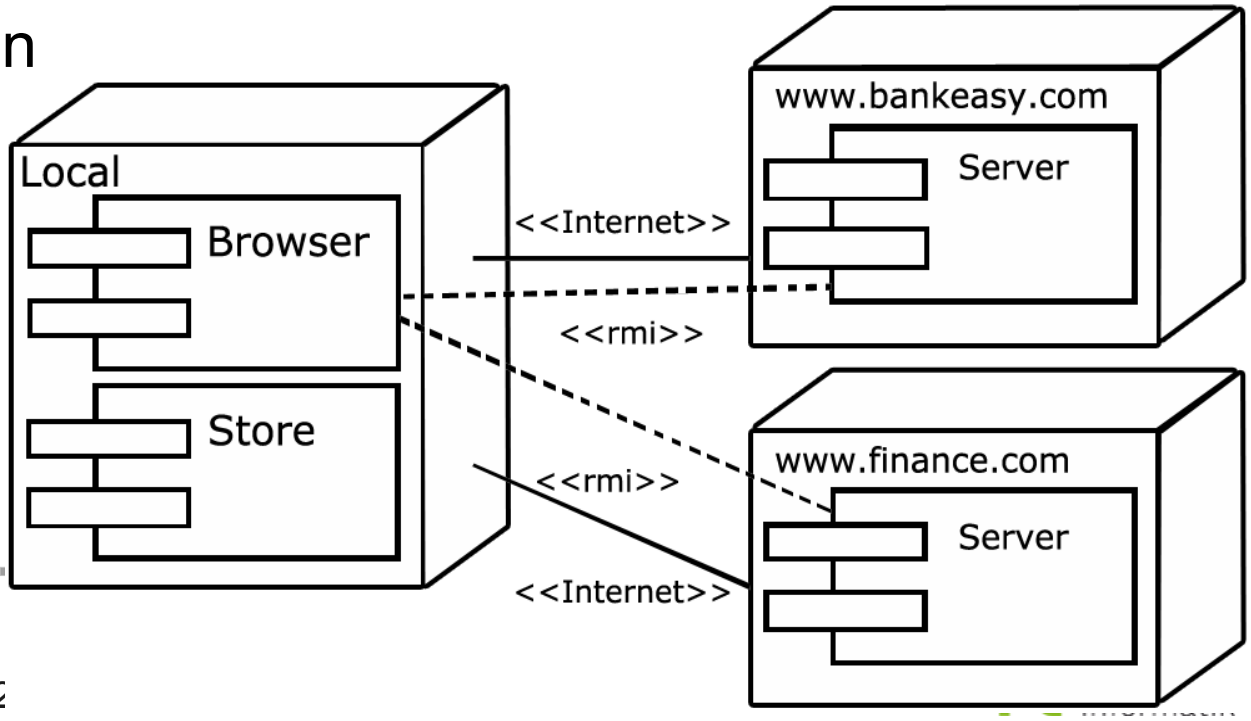



<<guarded access>> überprüft, dass Guard korrekt definiert.

# Beispiel: Finanz-Anfrage

Die Internetbank bankeasy.com und ein Finanzberater finance.com bieten Dienstleistungen an. Die Applets benötigen dabei bestimmte Privilegien.

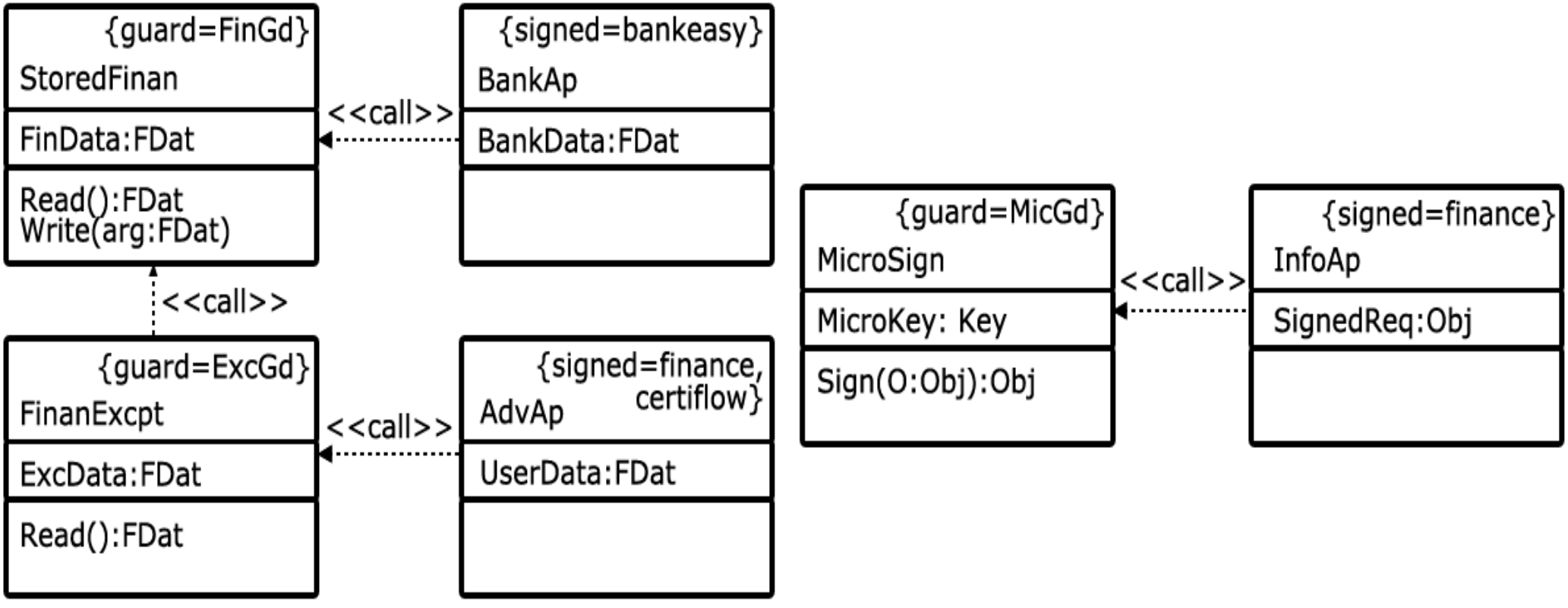
- Applets von der Bank, die signiert sind, sollen Daten zwischen 13 und 14 Uhr lesen und schreiben dürfen.
- Applets von dem Finanzdienstleister, die signiert sind, sollen den Micropayment-Schlüssel 5 mal die Woche nutzen dürfen.



# Finanz-Anfrage: Klassendiagramm

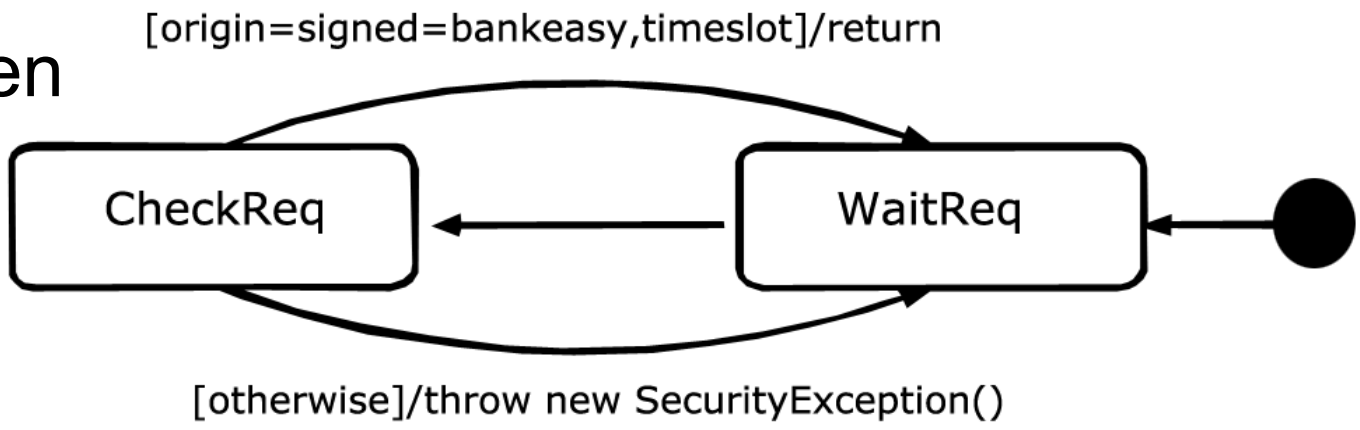
Für Integrität und Vertraulichkeit werden Daten als Signed- und Sealed-Objekte über Internet gesendet.

GuardedObjects kontrollieren Zugriff.

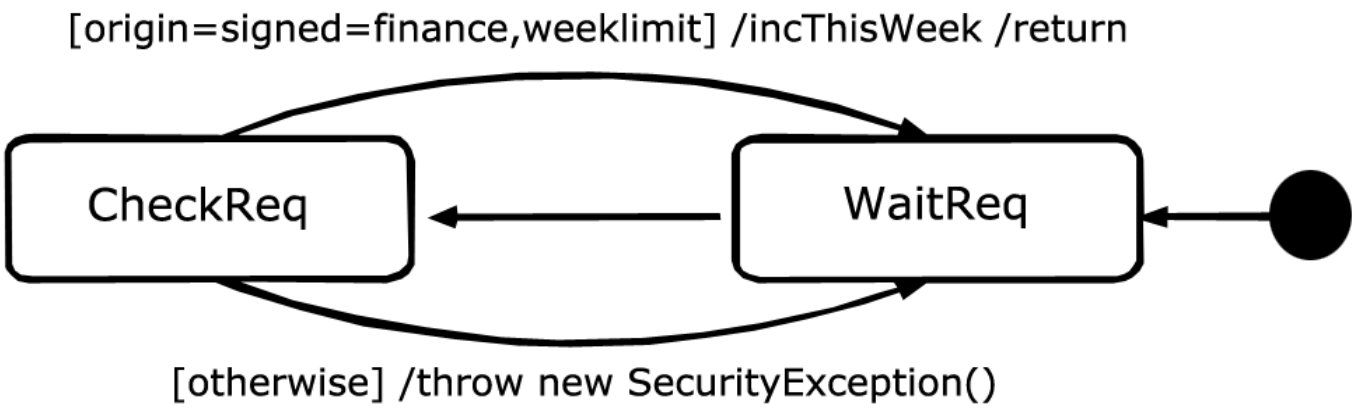


# Finanz-Anfrage: Guard Objects (Schritt 2)

timeslot *true* zwischen 13 und 14 Uhr.




weeklimit *true* bis Zugang fünf mal gewährt wird; incThisWeek erhöht Zähler.



# Finanz-Anfrage: Validierung

Guard Objekt gibt genügend Schutz (Schritt 3):

Analyse-Ergebnis: UML Spezifikation für Guard Objekte erteilt nur Genehmigungen, die durch Zugangsanforderungen impliziert sind.

- z.B.: Unter der Annahme, dass das aktuell ausgeführte Applet von Finance stammt und signiert ist, wird der Zugriff gewährt, wenn ein Micropayment Key genutzt wird, der bisher weniger als 5 mal zum Einsatz kam.

Zugangskontrolle konsistent mit Funktionalität (Schritt 4):

Analyseergebnis: Jedes Objekt, das laut Spezifikation Zugriff auf ein guarded Objekt erhalten soll, erhält diesen auch.

Mobile Objekte ausreichend geschützt (Schritt 5), da Objekte, die über das Internet gesendet werden signiert und versiegelt sind.

# CORBA Zugangskontrolle

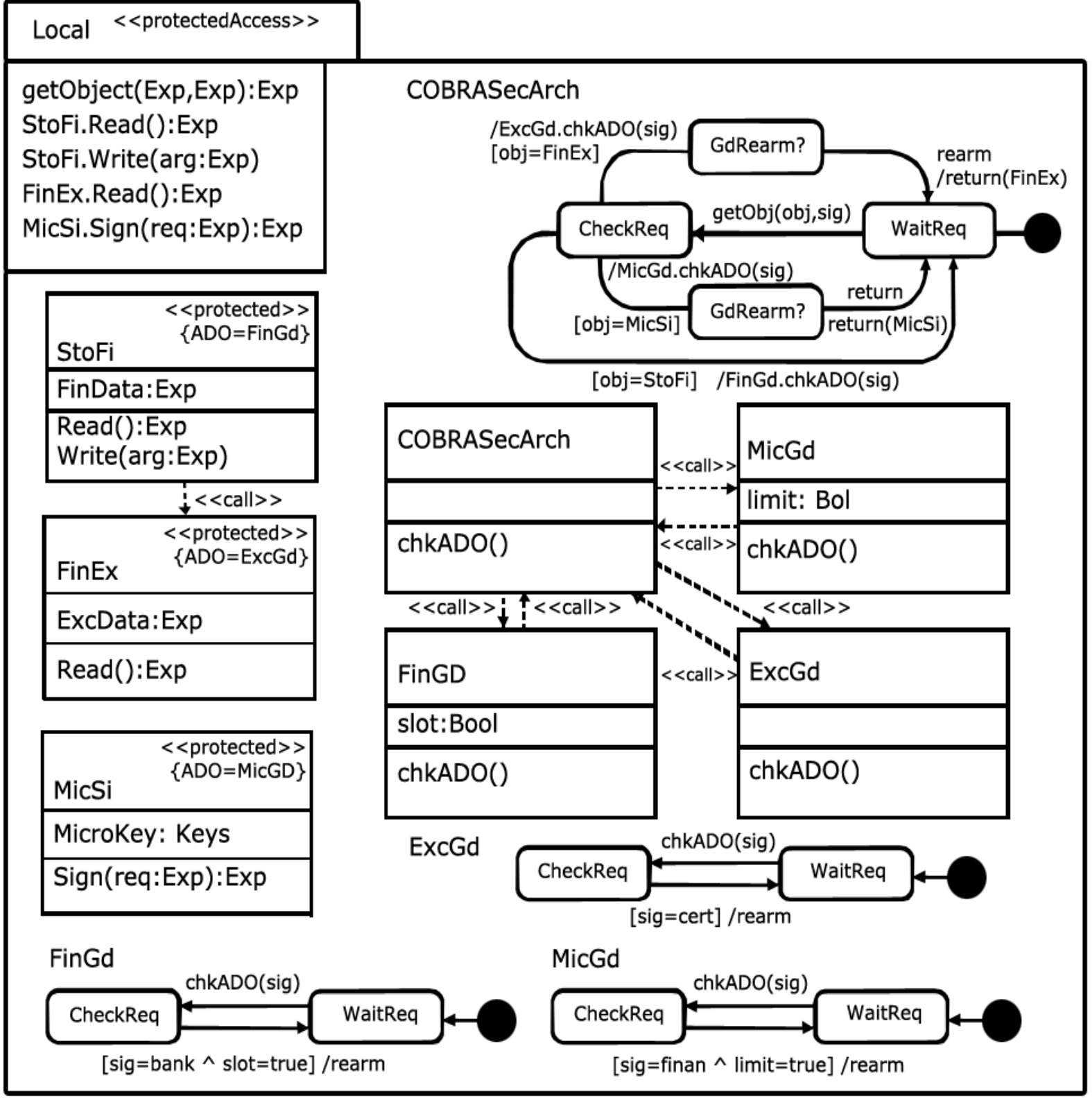
Objekt-Zugangskontrolle kontrolliert Zugang von Client zu Objekt über gegebene Methode.

Realisiert durch ORB und Security Service.

Access Decision Functions entscheiden, ob Zugang erlaubt. Abhängig von:

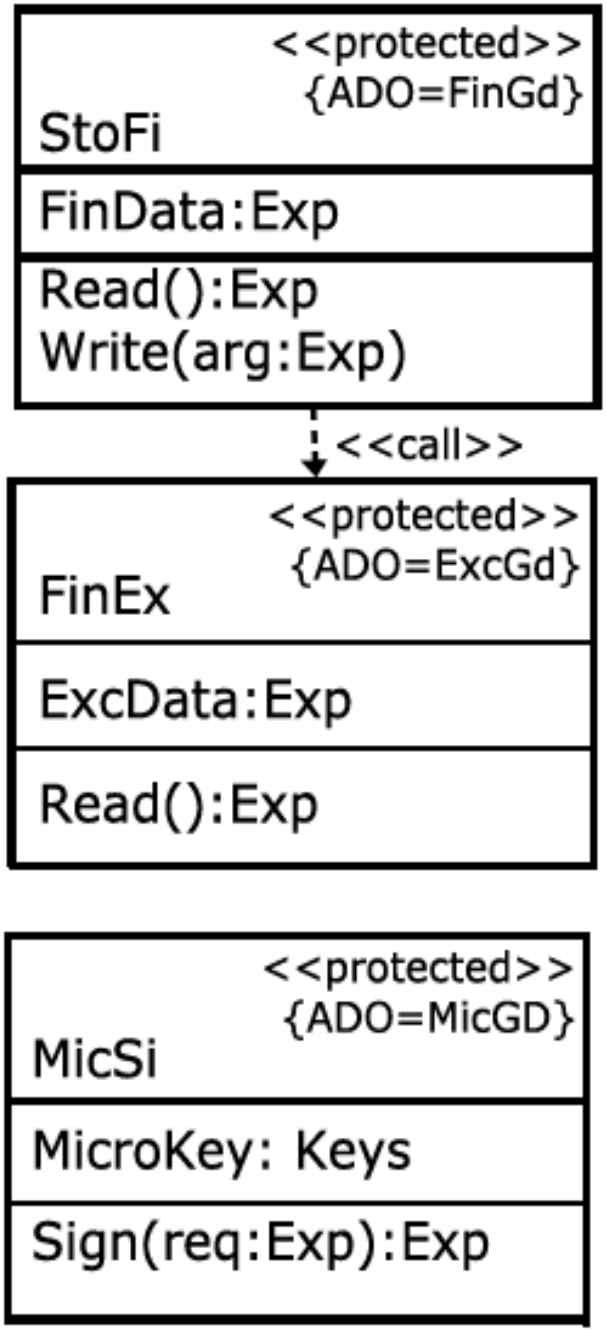
- Aufgerufener Operation,
- Privilegien des Principals, in dessen Vertretung der Client agiert,
- Kontrollattribute des Zielobjektes.

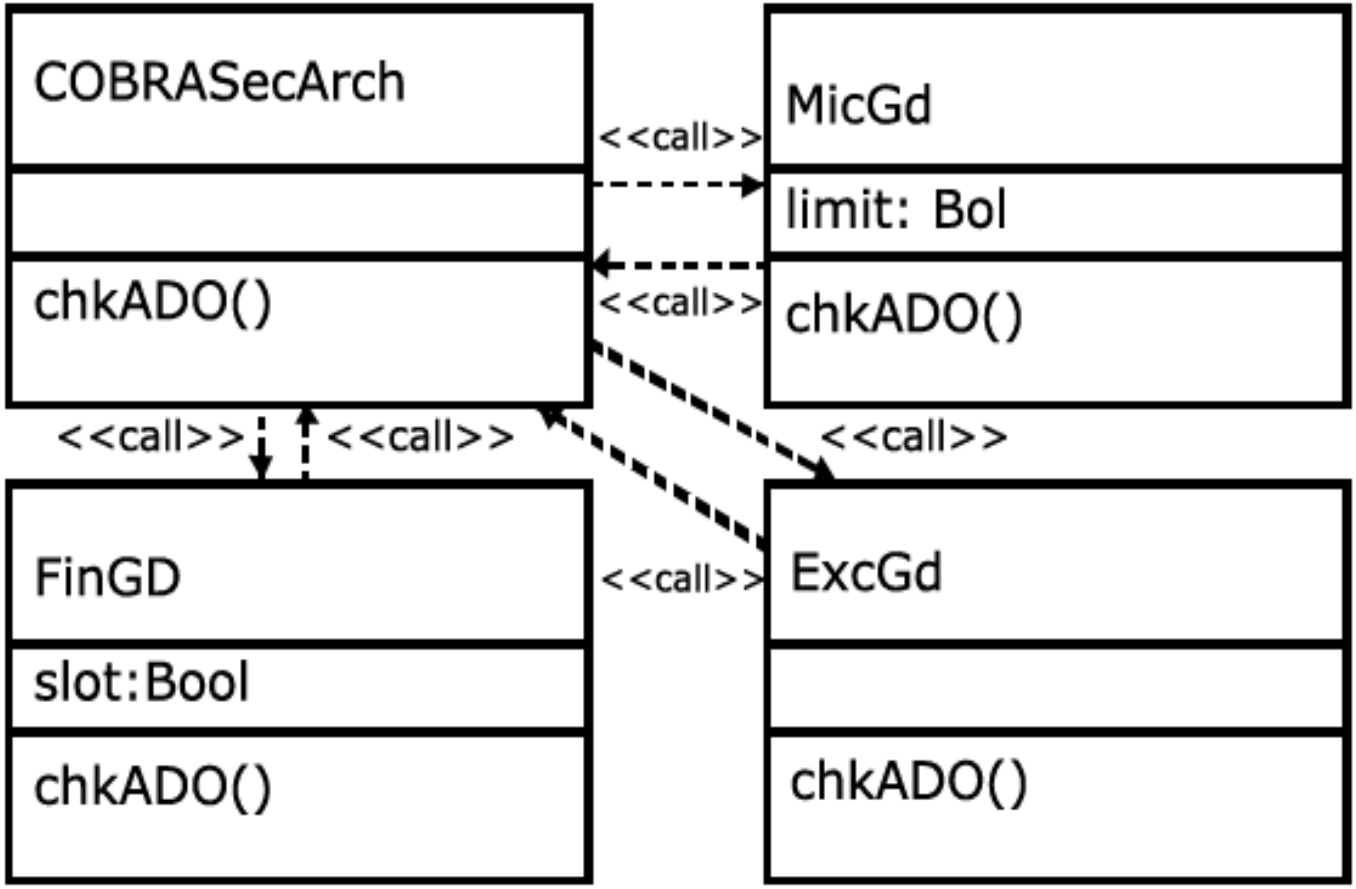
# Example: CORBA Access Control with UMLsec



# Example: CORBA Access Control with UMLsec

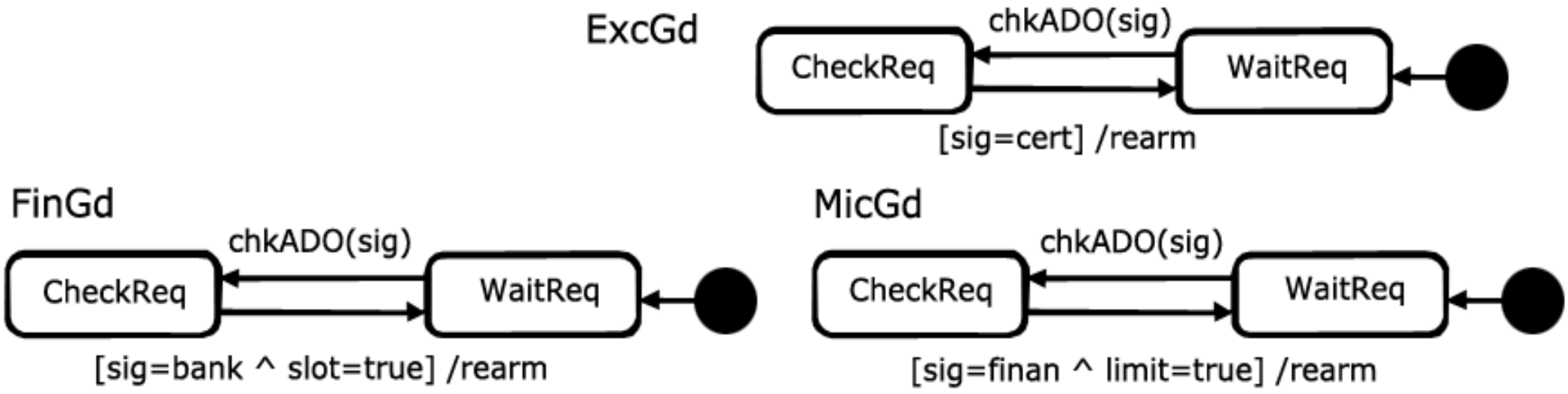




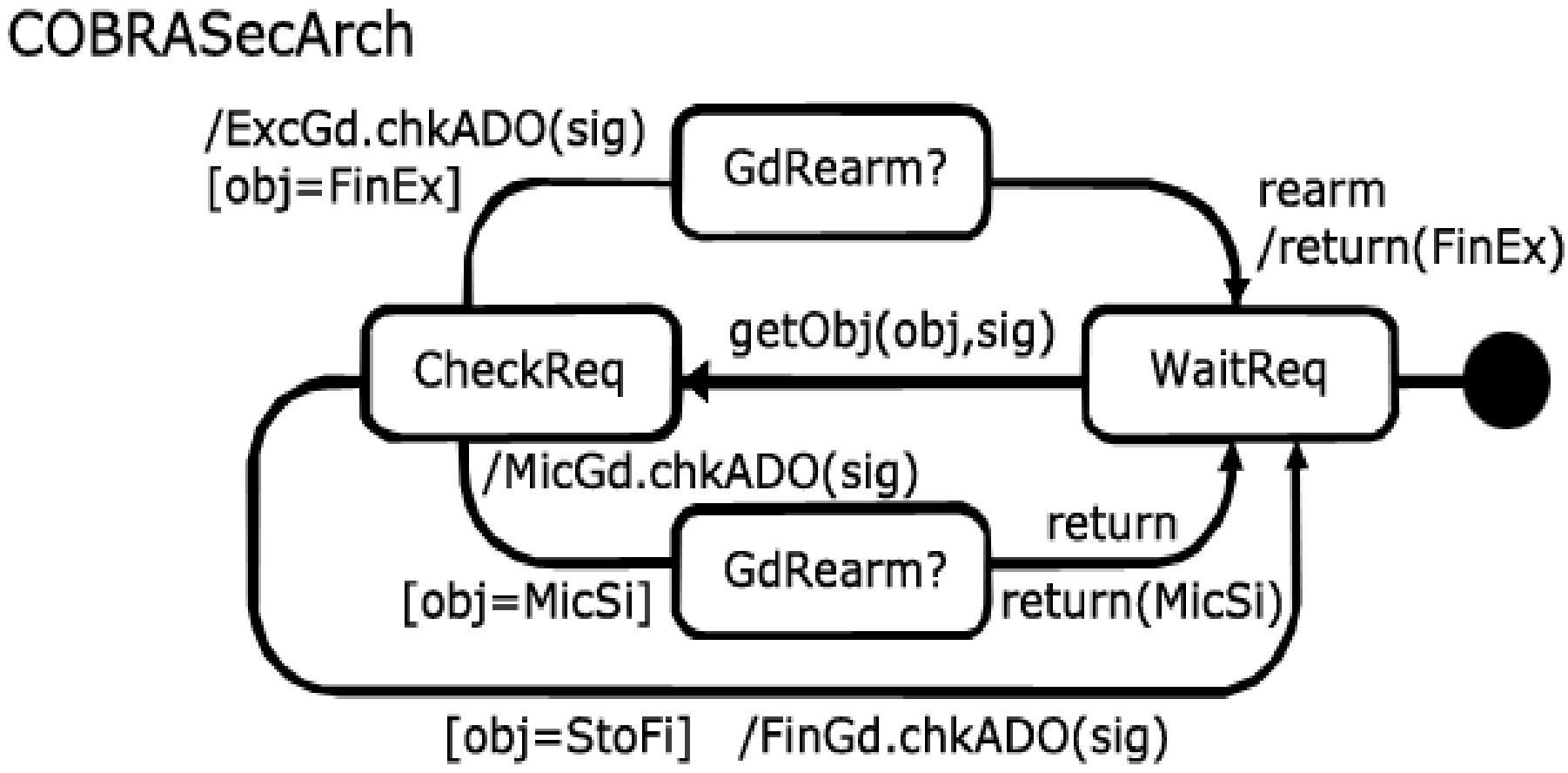
# Example: CORBA Access Control with UMLsec



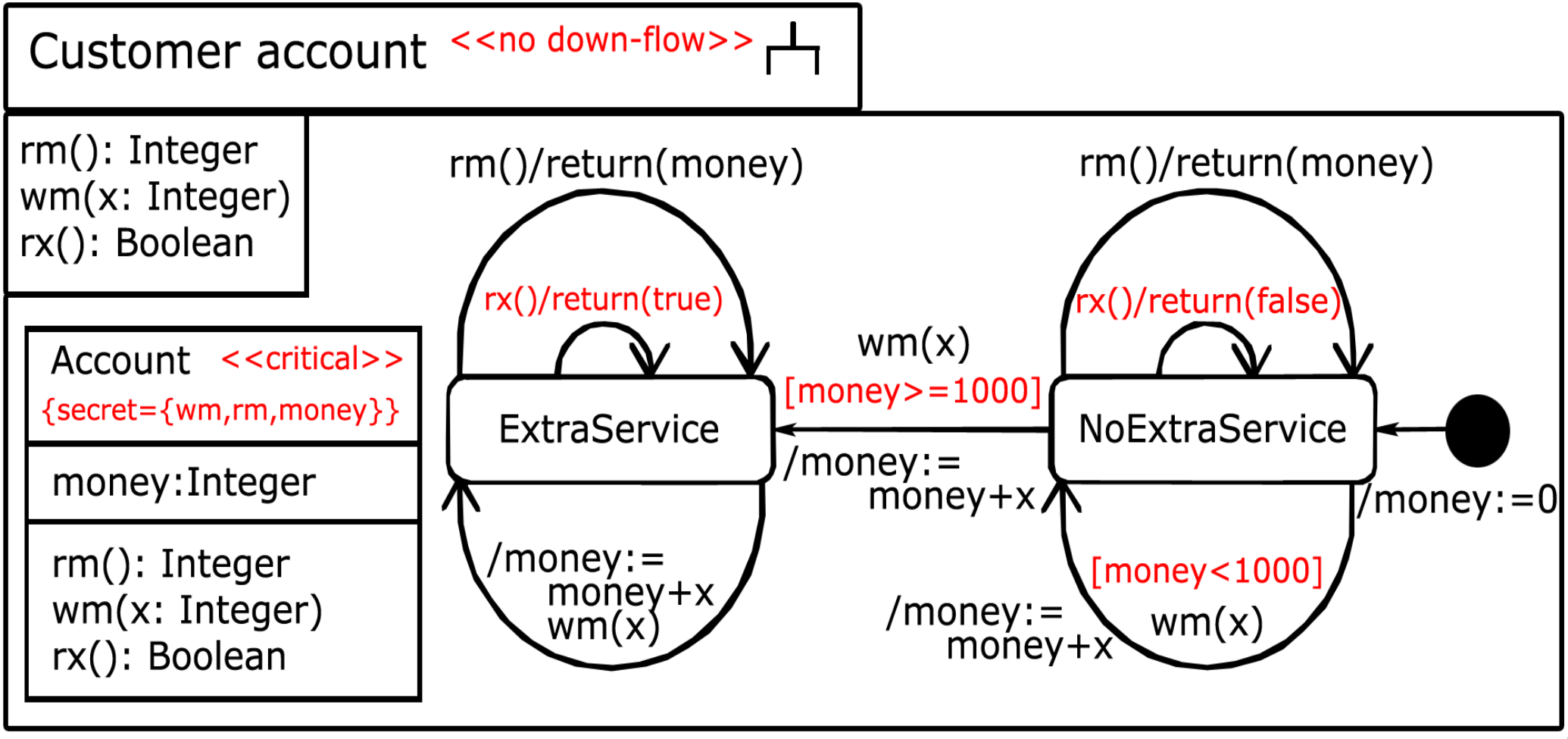


# Example: CORBA Access Control with UMLsec





# Frage




Kein partielle Preisgabe von Geheimnissen?

# <<no down–flow>>

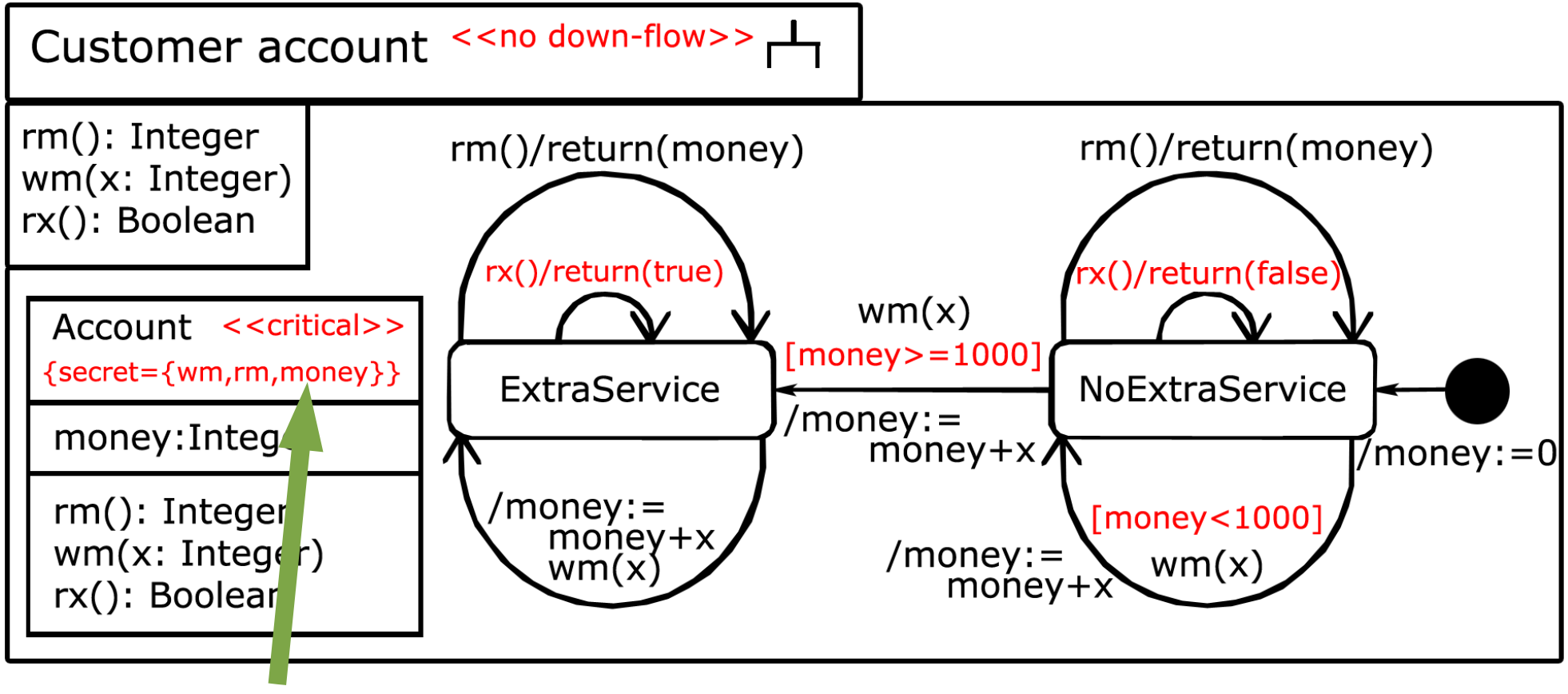
Stelle sicheren Informationsfluss sicher.

Bedingung:

Jeder Datenwert, der als {secrecy} spezifiziert ist, darf nur die Datenwerte beeinflussen, die auch als {secrecy} spezifiert sind.

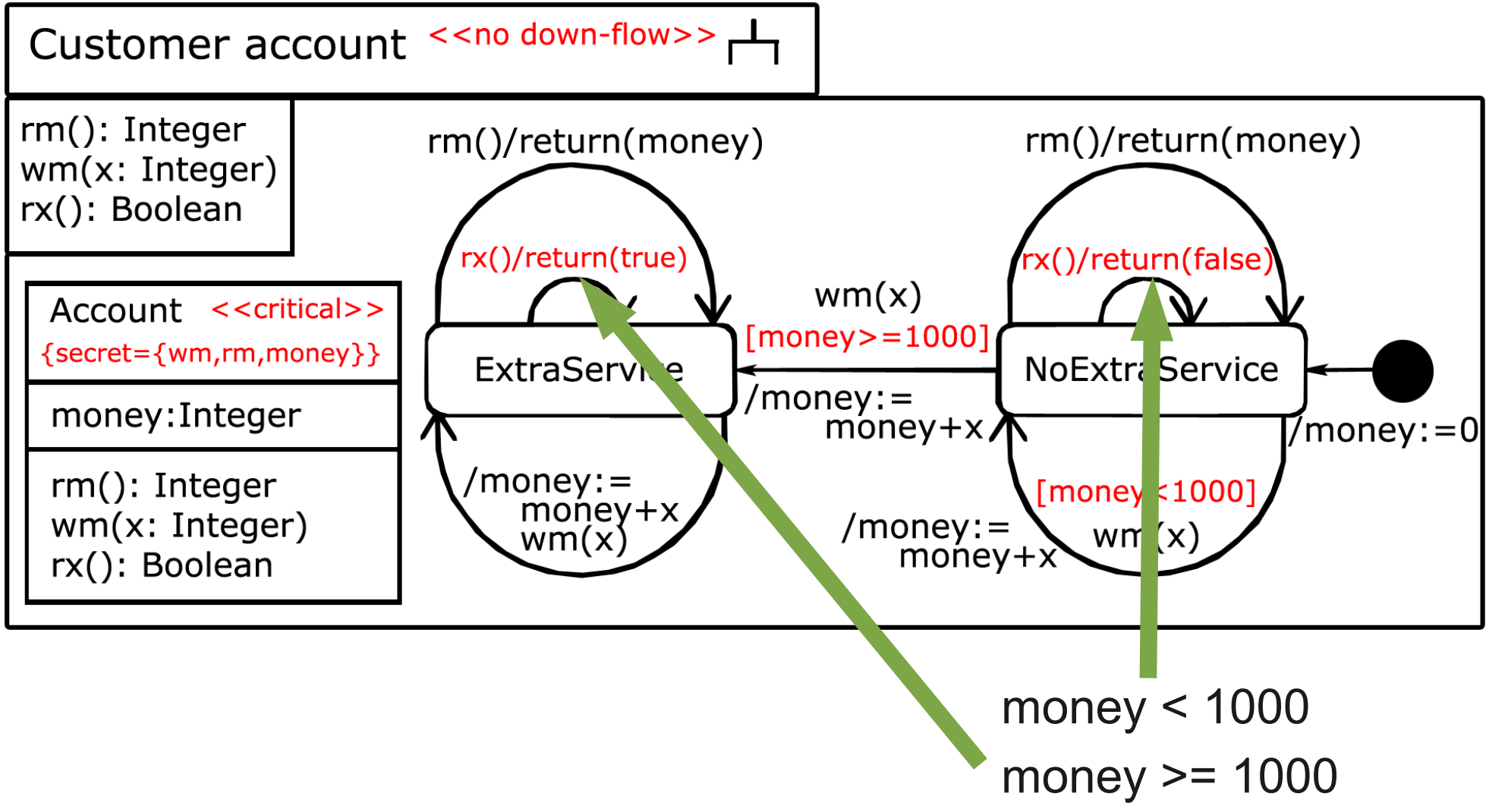
(Kann die Bedingung mit Bezug auf formale Ausführungsemantik formalisieren).

# Beispiel <<no down-flow>>

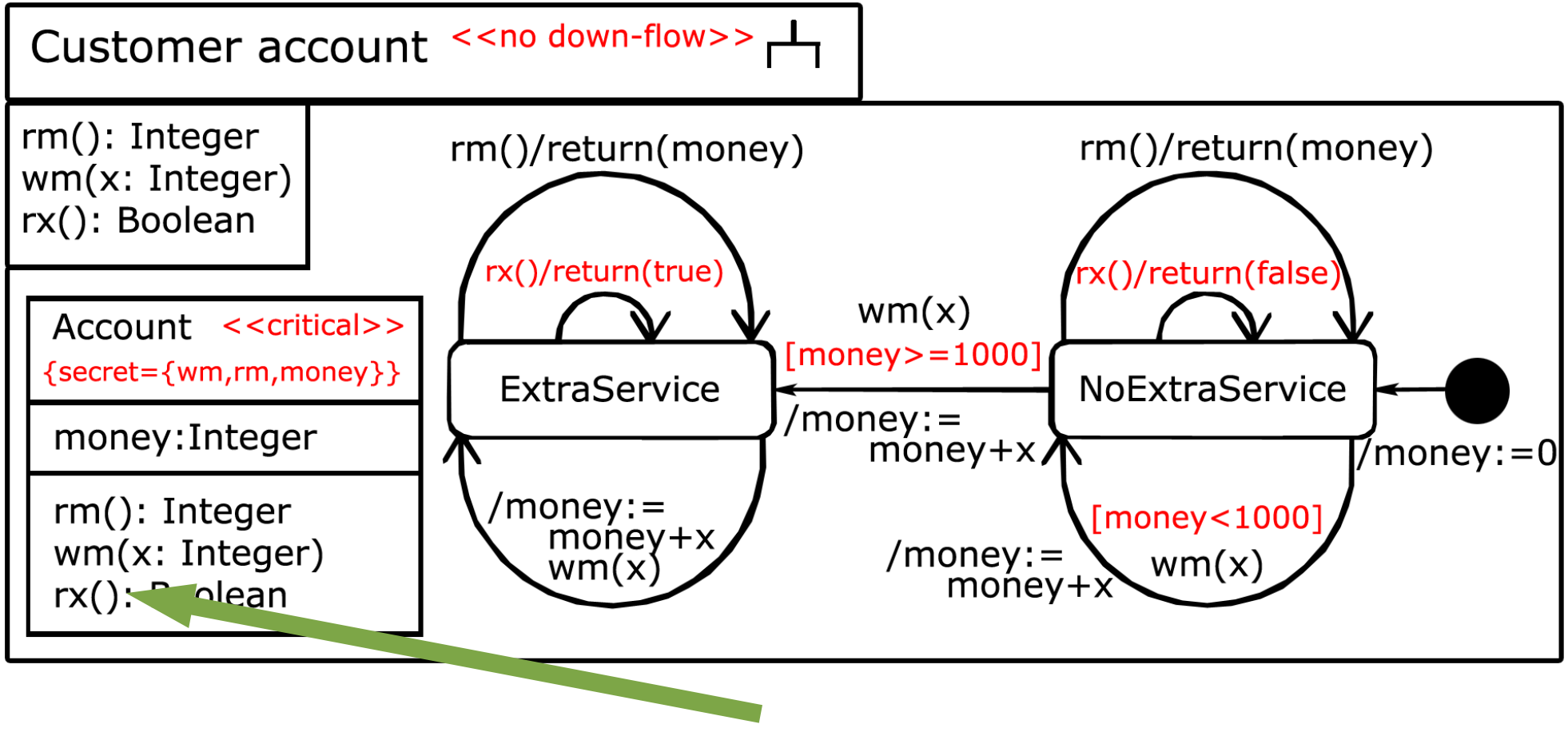



Information über
„money“ darf nicht offen gelegt werden

# Beispiel <<no down-flow>>

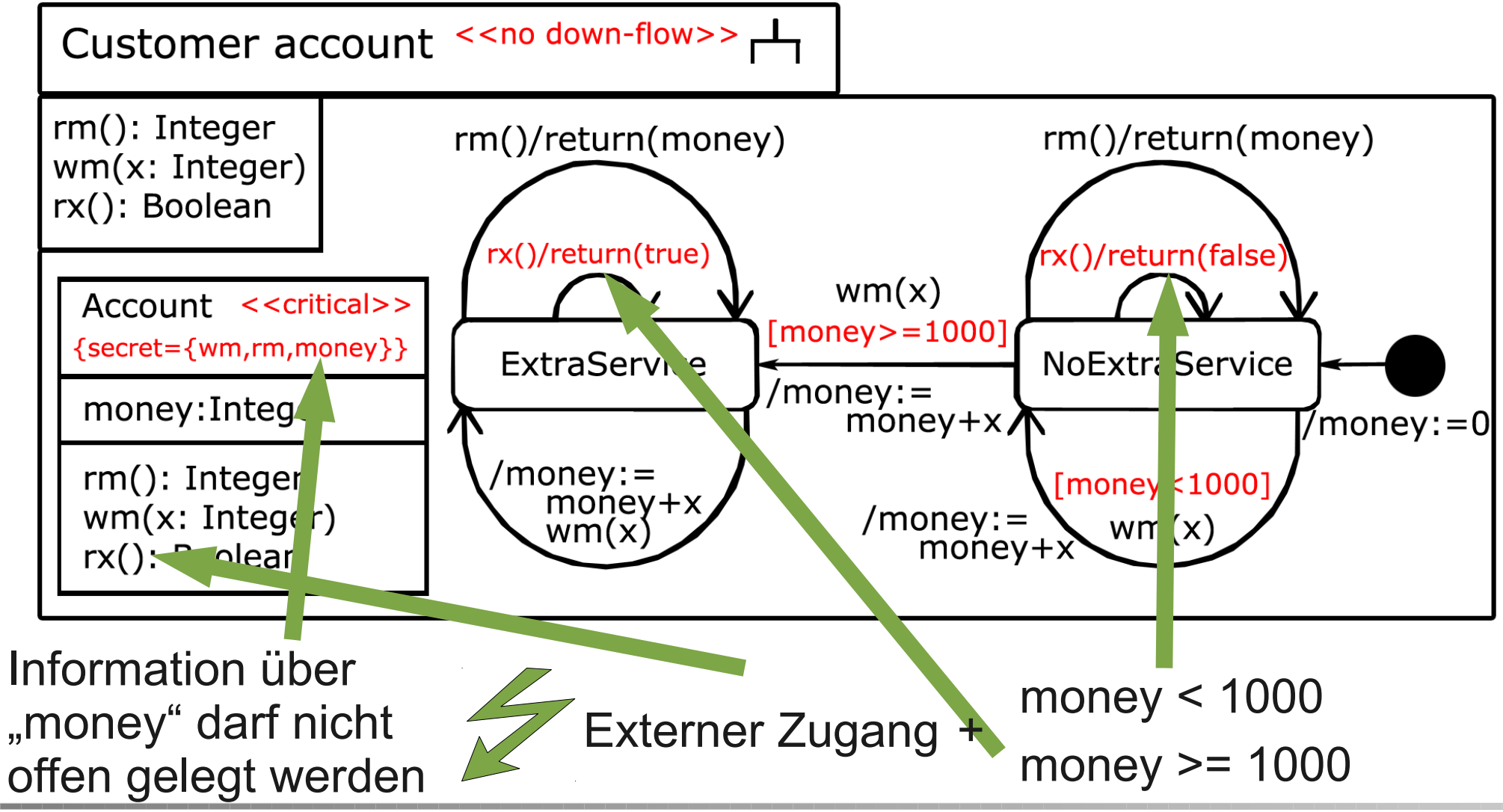


# Beispiel <<no down-flow>>




Externer Zugang

# Beispiel <<no down-flow>>



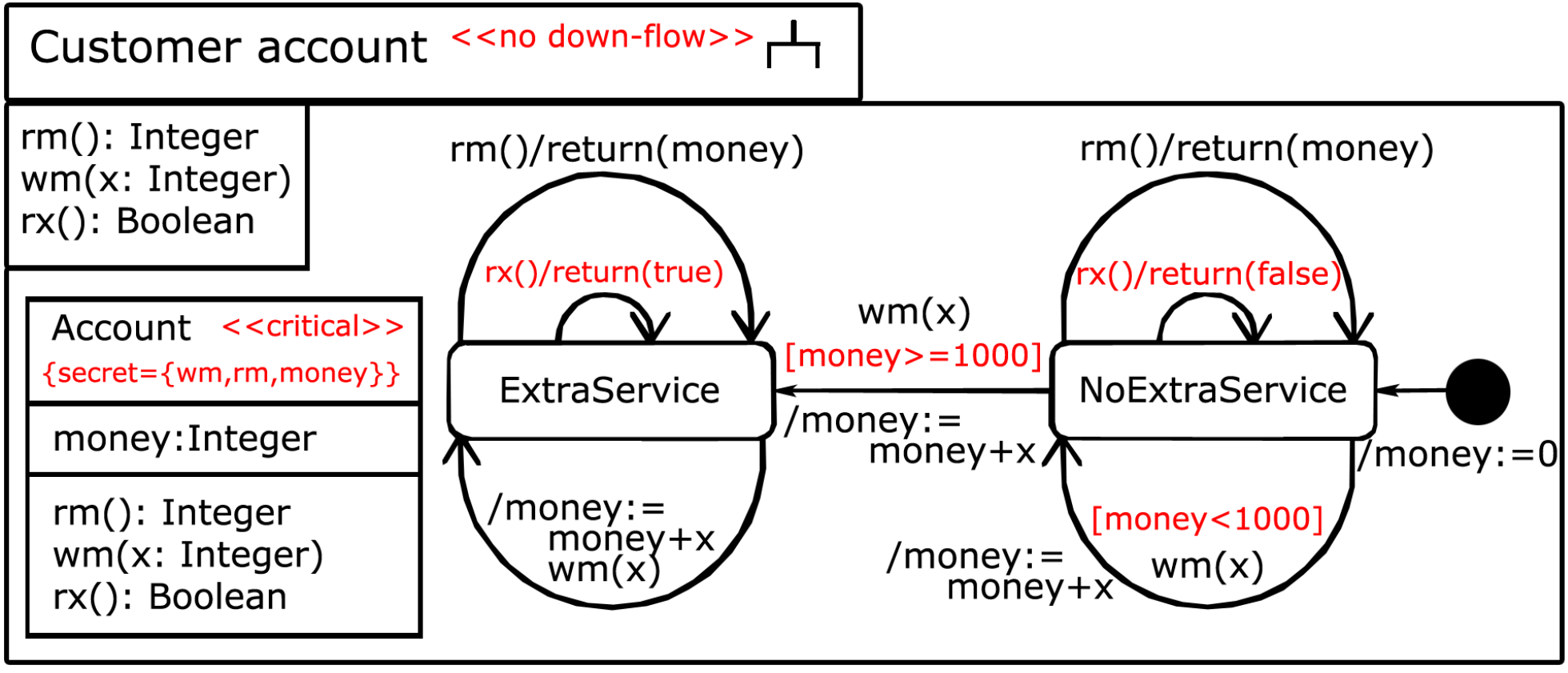

Information über „money“ darf nicht offen gelegt werden

Externer Zugang +

money < 1000

money >= 1000

# Beispiel <<no down-flow>>




<<no down–flow>> verletzt: Partielle Information über das Geheimnis von wm() werden vom nicht geheimen rx() zurückgegeben.

# Sicherer Einsatz von Kryptographie

«data security»

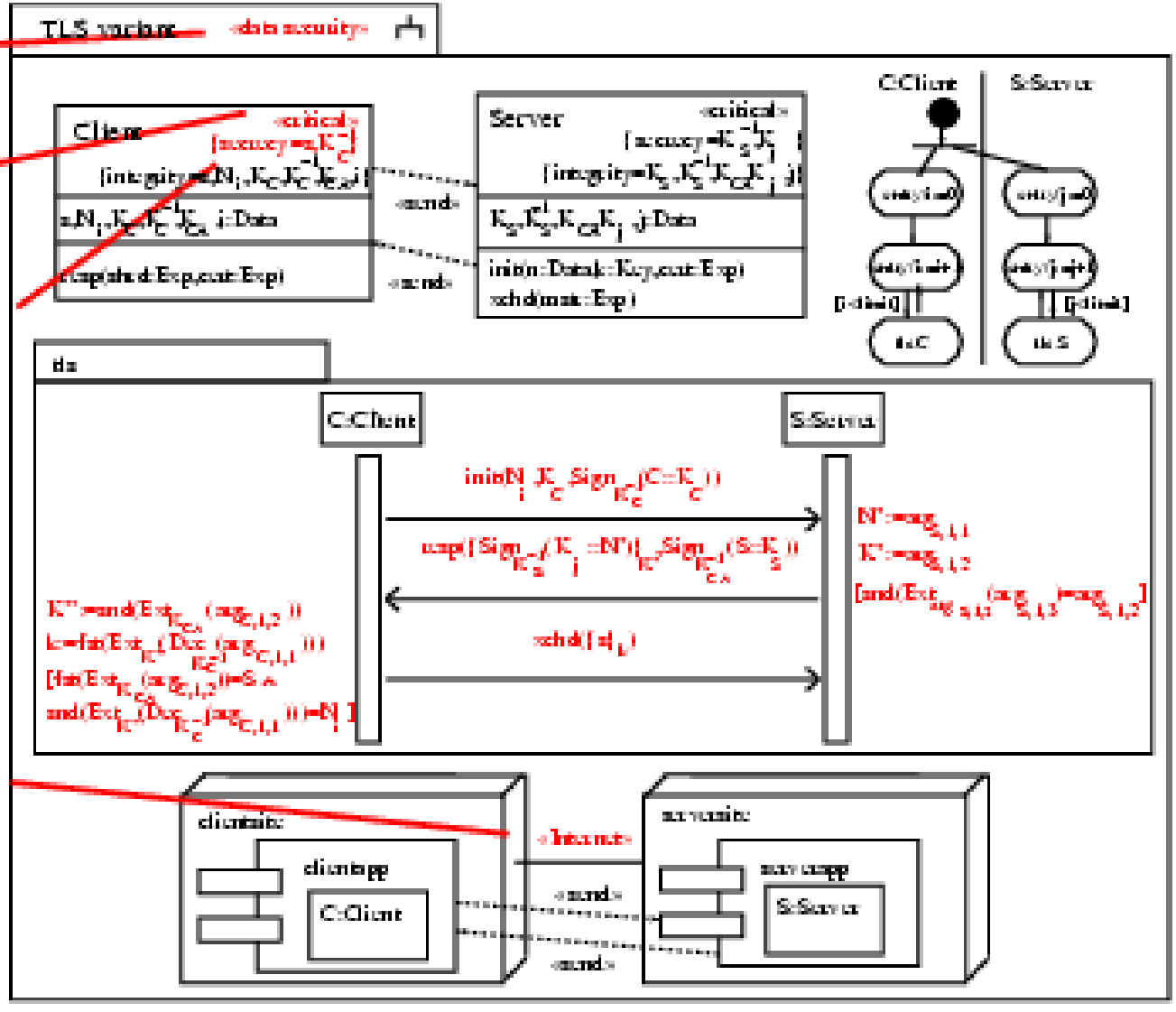
«critical»

$\{secrecy = \{s, K_C^{-1}\}\}$

Variante von TLS (INFOCOM`99).

Kryptoprotokoll sicher gegen Standard-Angreifer?

«Internet»



# <<data security>>

Stellt Sicherheitsanforderungen an Daten, die als <<critical>> markiert sind, sicher. Dabei wird das Bedrohungsszenario aus dem zugehörigen Verteilungsdiagramm (Deployment-Diagramm) zugrundegelegt.

Constraints: Daten, die als {secrecy}, {integrity}, {authenticity}, {fresh} markiert sind, erfüllen die auf Basis der Ausführungssemantik der UML Diagramme formalisierten Sicherheitsanforderungen.

# Beispiel <<data security>>

«data security»

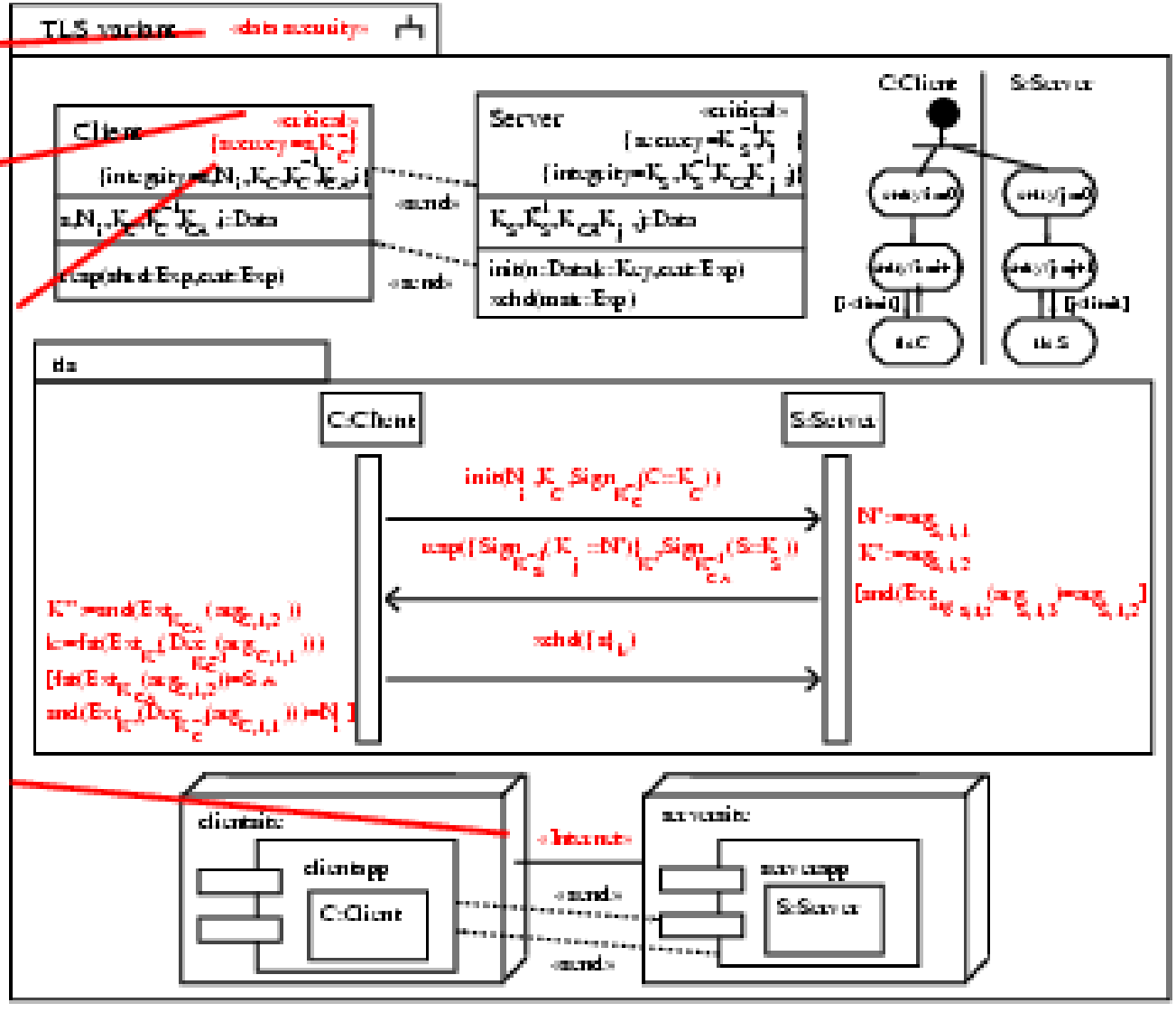
«critical»

$\{\mathbf{secrecy} = \{s, K_C^{-1}\}\}$

«Internet»

Variante von TLS (INFOCOM`99).

Verletzt {secrecy} von s gegen Standard Angreifer.



# UMLsec Intro: Übersicht

**Sicherheitsanforderungen**: <<secrecy>>,…

**Angriffszenarien:** Benutze Threats$_{adv}$ (ster).

**Sicherheitskonzepte:** Beispiel <<smart card>>.

**Sicherheitsmechanismen:** Beispiel <guarded access>>.

**Sicherheitsalgorithmen:** Verschlüsselung eingebaut.

**Physikalische Sicherheit:** Gegeben in Verteilungsdiagramm.

**Sicherheitsmanagement:** Benutze Aktivitätsdiagramme.

**Technologie-spezifisch:** Java-, CORBA-Sicherheit.