

Willkommen zur Vorlesung  
*Softwarearchitekturen im Finanz- und  
Versicherungsbereich*  
im Sommersemester 2010  
Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

# 5. Secure Architectures

# 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**.

Berechtigungseinträgen bestehen aus:

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

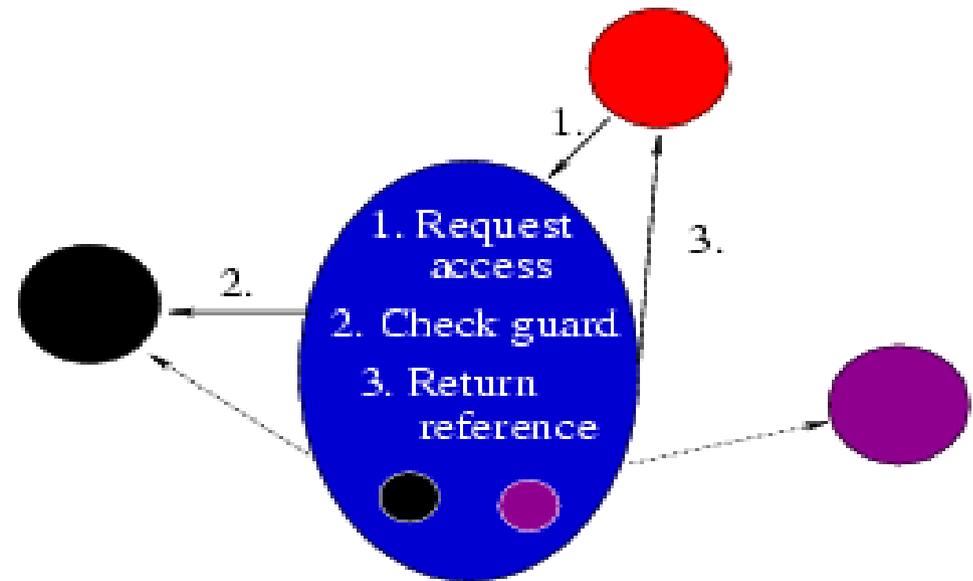
Brauchen **Integritätsschutz** für Objekte, die zur Authentisierung verwendet oder zwischen JVMs ausgetauscht werden.

**SignedObject** enthält Objekt und seine Signatur.

Für **Vertraulichkeitsschutz**: **SealedObject** ist verschlüsseltes Objekt.

`java.security.GuardedObject` schützt Zugang zu anderen Objekten.

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

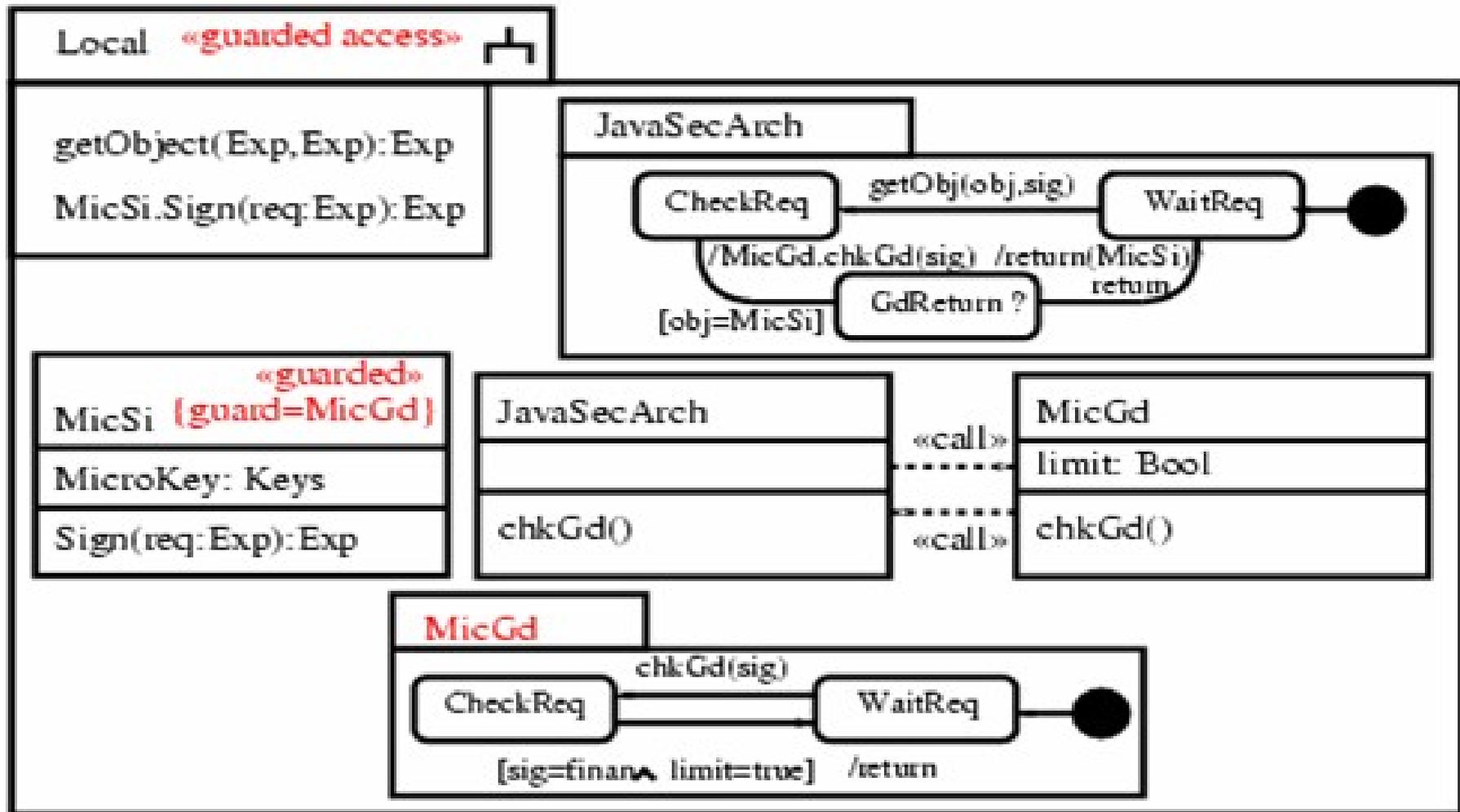


## Aufgabe 5.1

- a) Warum ist dieser Mechanismus nicht völlig sicher (Hinweis: Referenz) ?  
[3 P.]
- b) Wie könnte man ihn erweitern, um dieses Problem zu begrenzen ? [3 P.]

- Rechtevergabe abhängig von **Ausführungskontext**.
  - Zugangskontrollentscheidungen bezüglich verschiedener **Threads**.
  - Können verschiedene **Schutzdomäne** betreffen.
  - Methode **doPrivileged()** unabhängig von Ausführungskontext.
- Gibt Werkzeuge zur Überprüfung.

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



Enforces overall security policy ?

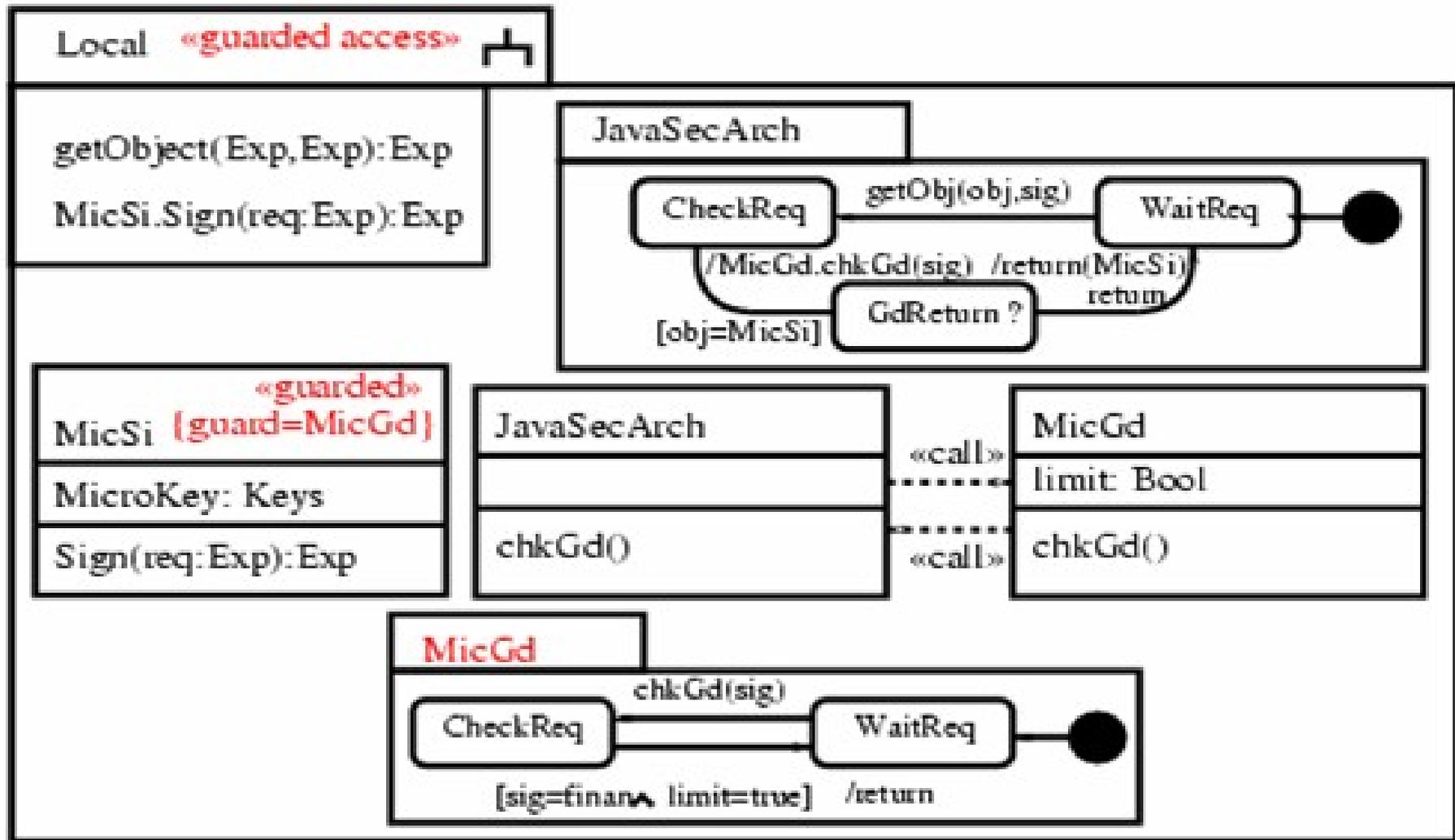
# <<guarded access>>

Ensures that in Java, <<guarded>> classes only accessed through {guard} classes.

Constraints:

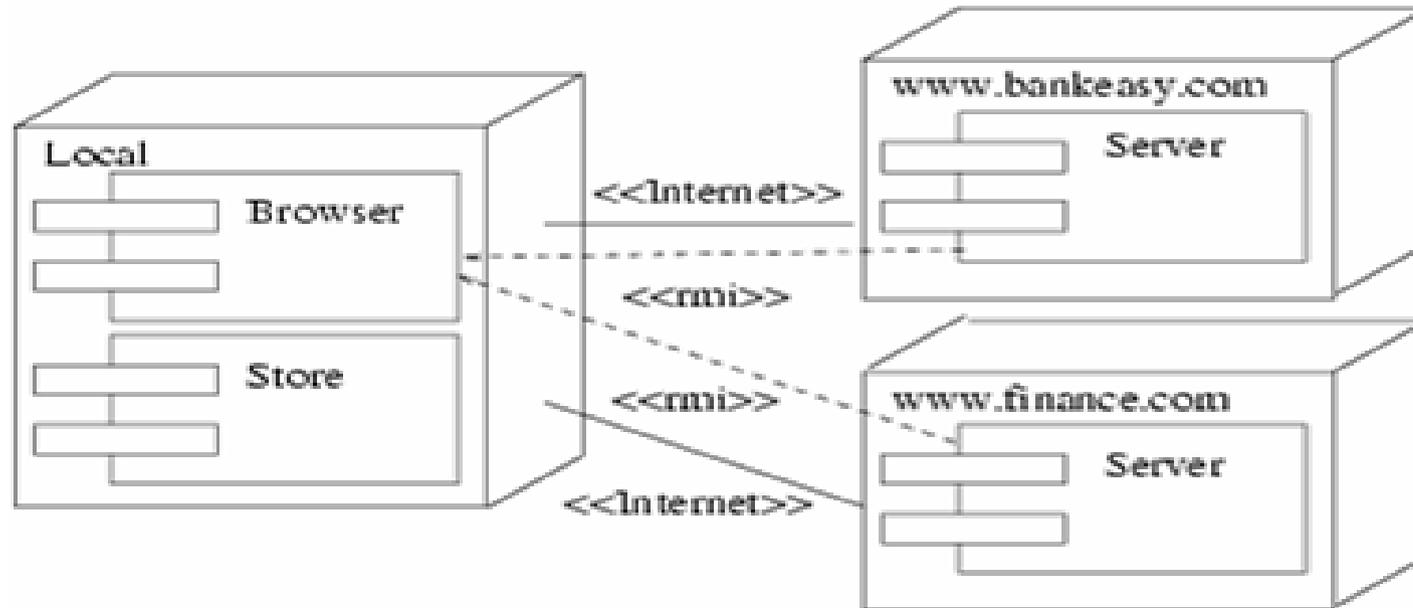
- References of <<guarded>> objects remain secret.
- Each <<guarded>> class has {guard} class enforcing security policy.

# Example <<guarded access>>



<<guarded access>> fulfilled.

# Example: Financial Application



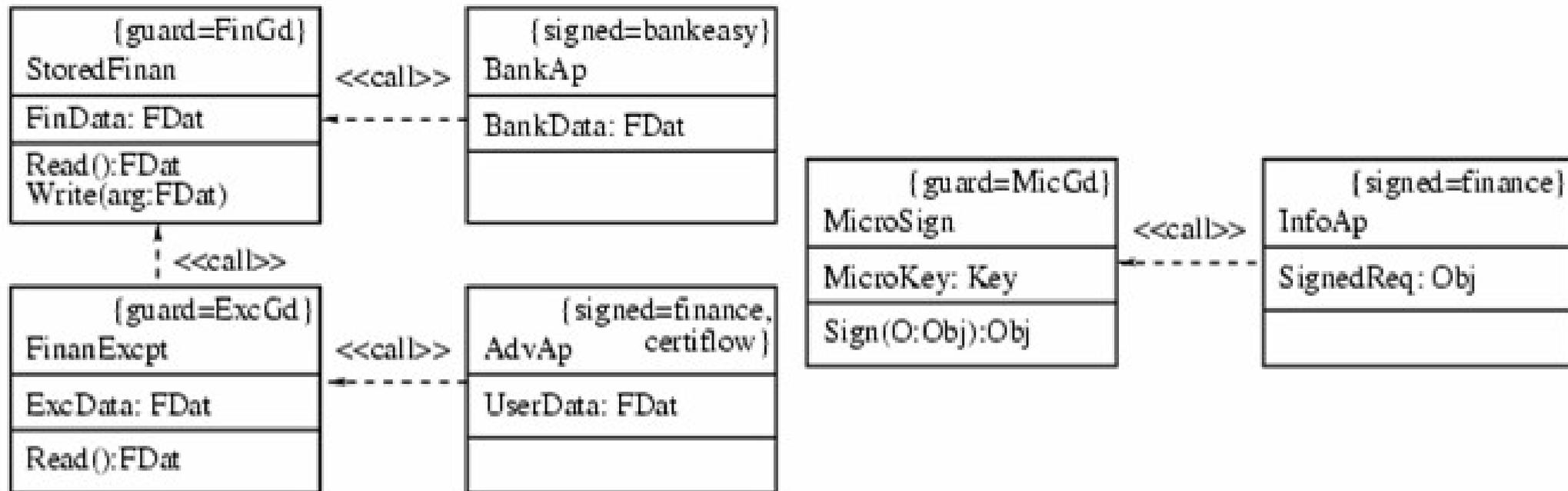
Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain Privileges (step1).

- Applets from and signed by bank **read** and **write** financial data between 1 pm and 2 pm.
- Applets from and signed by Finance **use** micropayment key five times a week.

# Financial Application: Class Diagram

**Sign** and **seal** objects sent over Internet for Integrity and confidentiality.

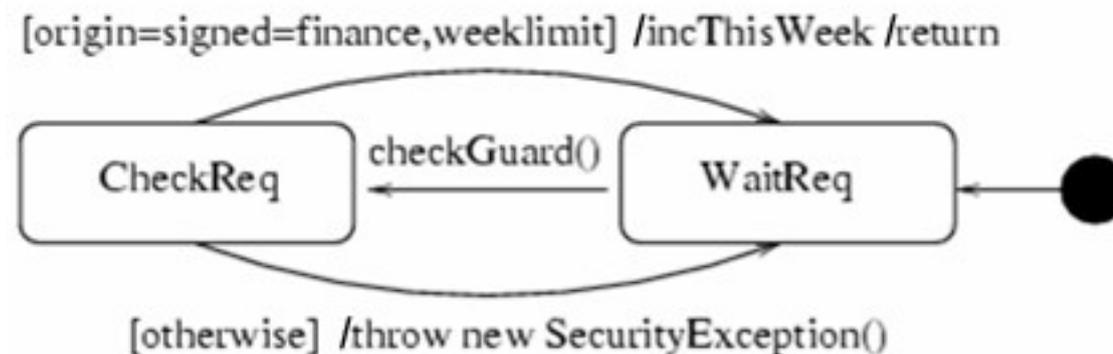
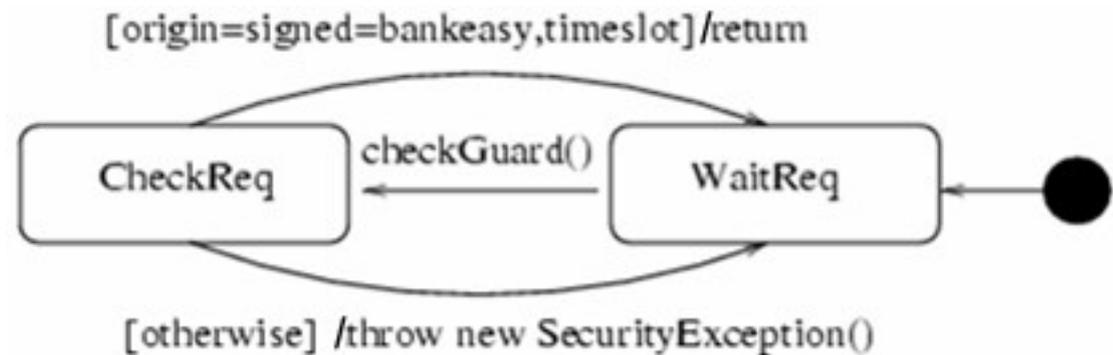
**GuardedObjects** control access.



# Financial Application: Guard Objects (step 2)

**timeslot** true between  
1pm and 2pm.

**weeklimit** true until  
access granted five  
times; **inc ThisWeek**  
increments counter.



Guard objects give **sufficient protection** (step 3).

**Proposition.** UML specification for guard objects only grants permissions implied by access permission requirements.

Access control consistent with **functionality** (step 4). Includes:

**Proposition.** Suppose applet in current execution context originates from and signed by Finance. Use of micropayment key requested (and less than five times before). Then permission granted.

**Mobile objects** sufficiently protected (step 5), since objects sent over Internet are signed and sealed.

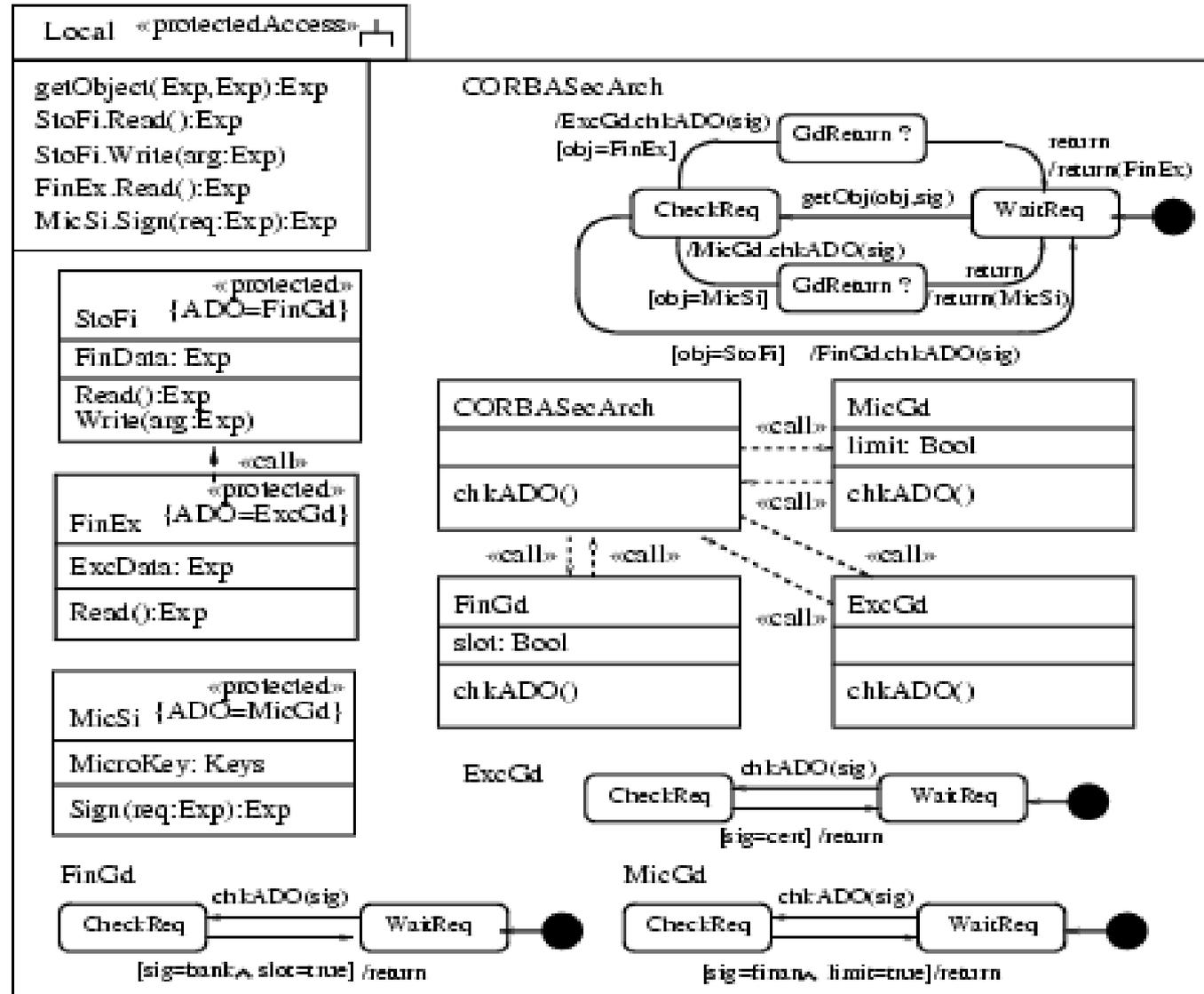
Objektzugangs-Politik kontrolliert **Zugang** von Client zu **Objekt** über gegebene **Methode**.

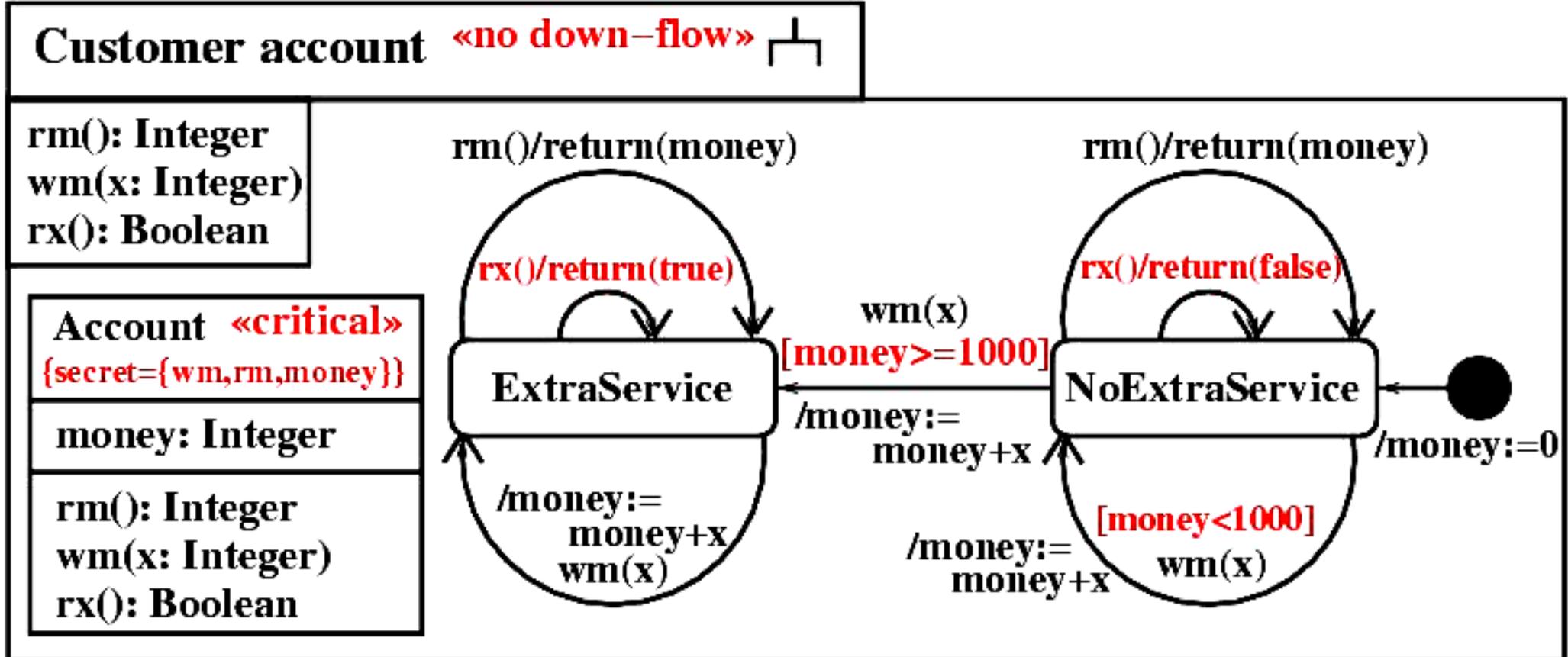
Realisiert durch ORB und Security Service.

**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





No partial leakage of secrets ?

<<no down-flow>>

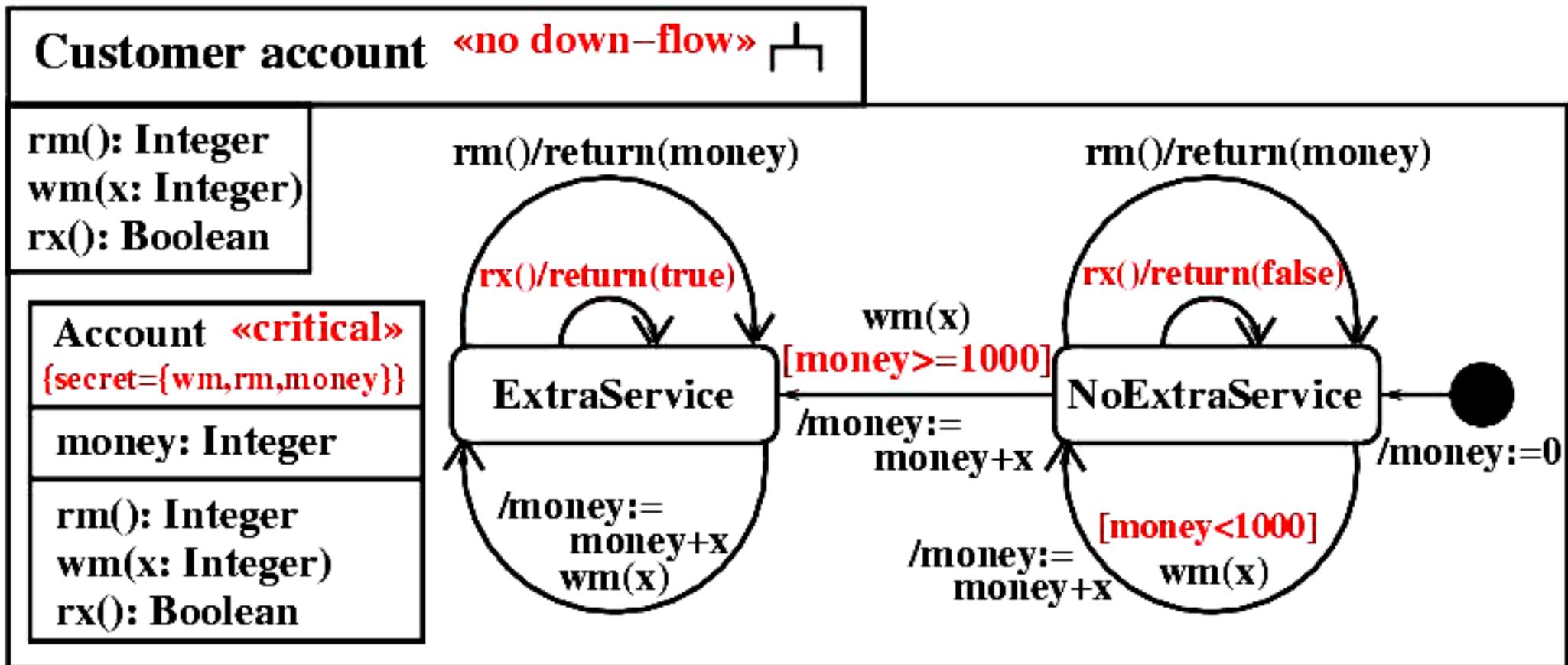
Enforce secure **information flow**.

Constraint:

Value of any data specified in **{secrecy}** may influence **only** the values of data also specified in **{secrecy}**.

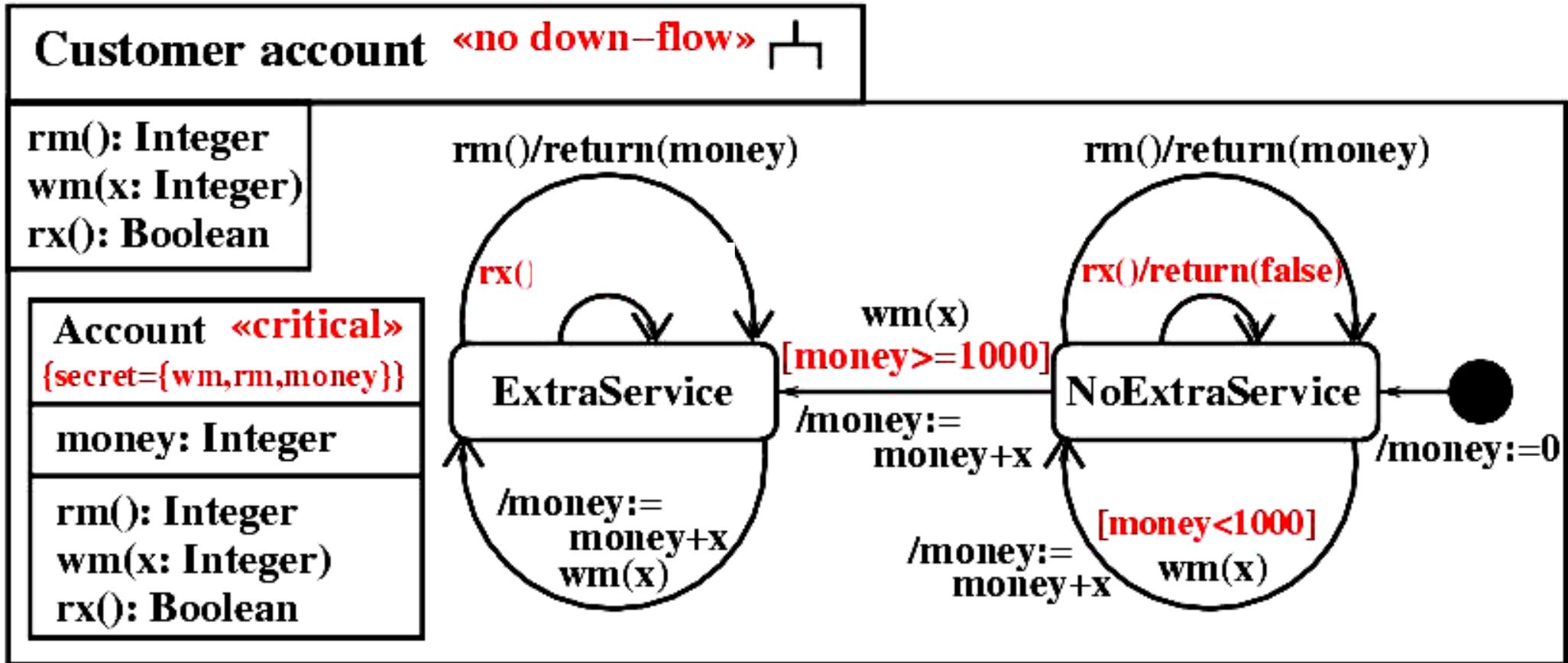
Formalize by referring to formal behavioural semantics.

# Example <<no down-flow>>



<<no down-flow>> **violated**: partial information on input of secret `wm()` returned by non-secret `rx()`.

# Aufgabe 5.2



Modifikation: Im Zustand ExtraService wird auf rx() kein Rückgabewert zurückgegeben. Ist «no down-flow» nun erfüllt (mit Begründung) ? [4 P.]

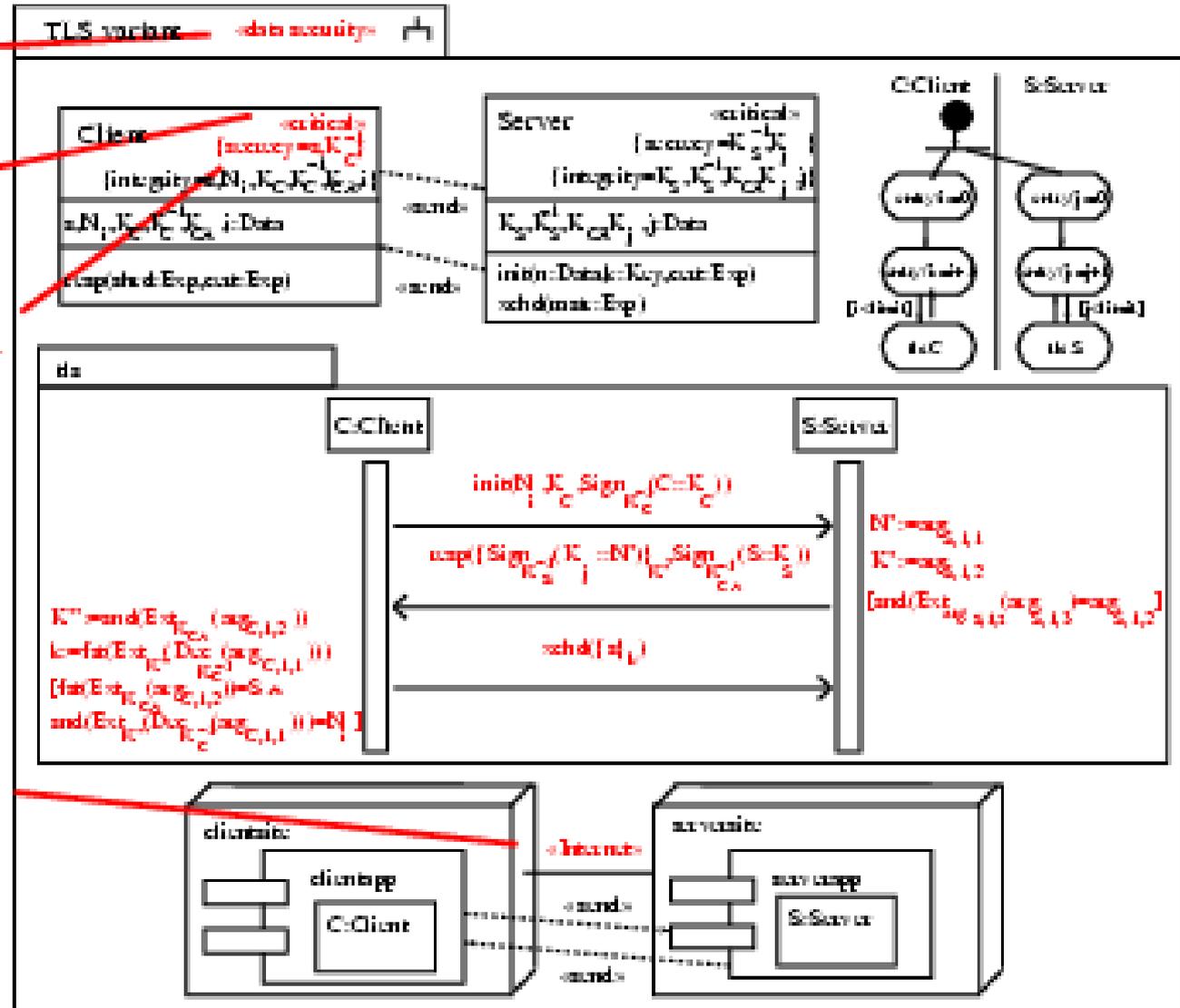
# Secure Use of Cryptography



«data security»

«critical»

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



Variant of TLS  
(INFOCOM`99).  
Cryptoprotocol  
secure against  
default  
adversary ?

«Internet»

<<data security>>

Security requirements of data marked  
<<critical>> enforced against threat  
scenario from deployment diagram.

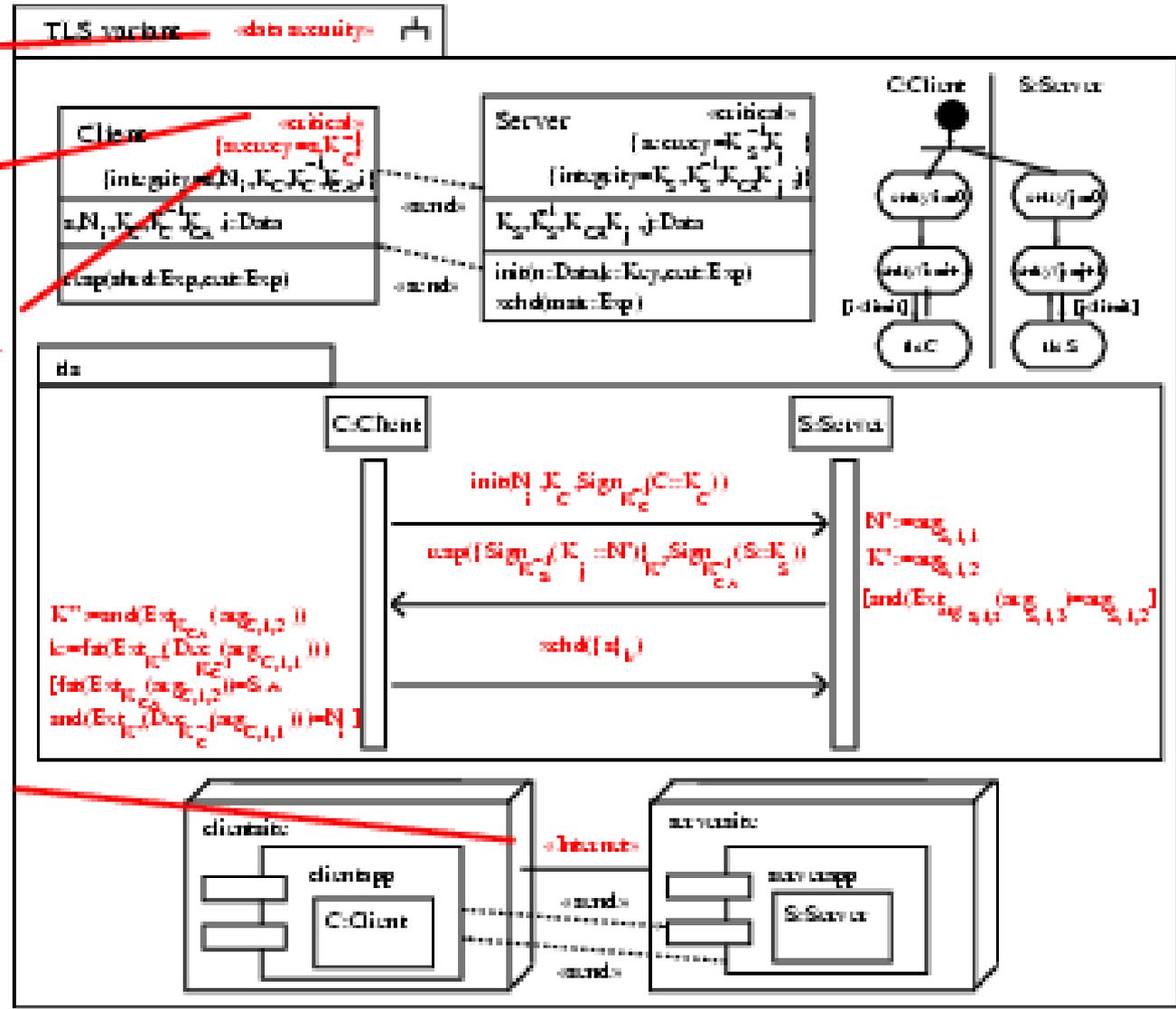
Constraints: Data marked {secrecy}, {integrity},  
{authenticity}, {fresh} fulfills respective  
formalized security requirements.

# Example <<data security>>

«data security»

«critical»

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



Variant of TLS  
(INFOCOM`99).  
Violates {secrecy}  
of s  
against default  
adversary.

«Internet»

**Security requirements:** <<secrecy>>, ...

**Threat scenarios:** Use `Threatsadv(ster)`.

**Security concepts:** For example <<smart card>>.

**Security mechanisms:** E.g. <<guarded access>>.

**Security primitives:** Encryption built in.

**Physical security:** Given in deployment diagrams.

**Security management:** Use activity diagrams.

**Technology specific:** Java, CORBA security.