
*Vorlesung*

# *Methodische Grundlagen des Software-Engineering*

im Sommersemester 2014

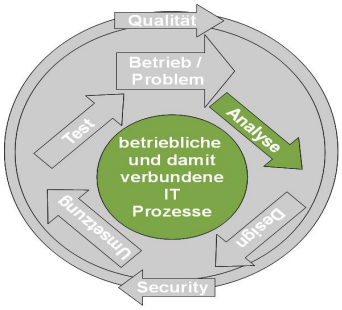
Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 3.4: Sichere Architekturen

v. 07.07.2014

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Einführung: Software Security
  - Hintergrund IT-Sicherheit
  - Wiederholung: Metamodellierung
  - Modellbasierte Sicherheit mit UMLsec
  - Sichere Architekturen
  - Kryptographische Protokolle
  - Protokollanalyse
  - Biometrische Authentisierung
  - Biometrische Authentisierung: Analyse
  - Elektronische Geldbörsen
  - Clouds
  - Elektronische Signatur
  - Bankarchitektur




**Literatur:**
[Jür05] Jan Jürjens: **Secure systems development with UML**, Springer-Verlag 2005.
Unibibliothek (e-Book):
http://www.ub.tu-dortmund.de/katalog/titel/1361890
Papier-Version:
http://www.ub.tu-dortmund.de/katalog/titel/1091324
**Kap. 4**

- **Letzer Abschnitt:** Modellbasierte Sicherheit
  - UMLsec
  - fair exchange, secure links, secure dependency
- **Dieser Abschnitt:** Sichere Architekturen
  - Guard Objects
  - <<no down-flow>>
  - <<data security>>

# 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



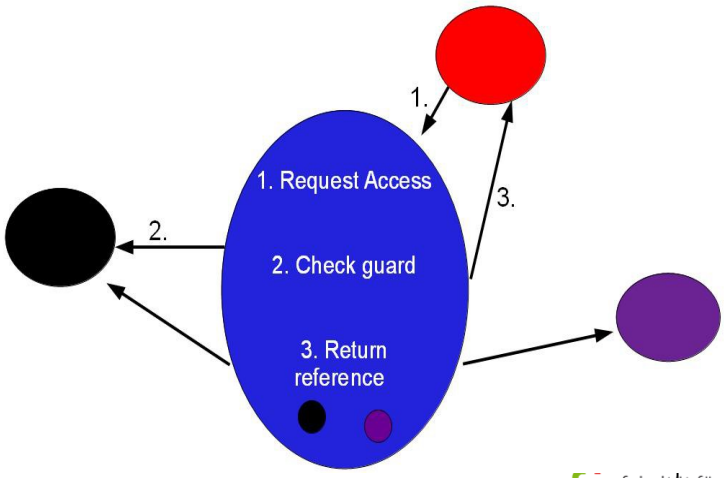

- **Integritätsschutz** für Objekte nötig. → Verwendung zur Authentisierung oder Austausch zwischen JVMs.
- **SignedObject** enthält Objekt und seine Signatur.
- Für **Vertraulichkeitsschutz: SealedObject** = verschlüsseltes Objekt.

# 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



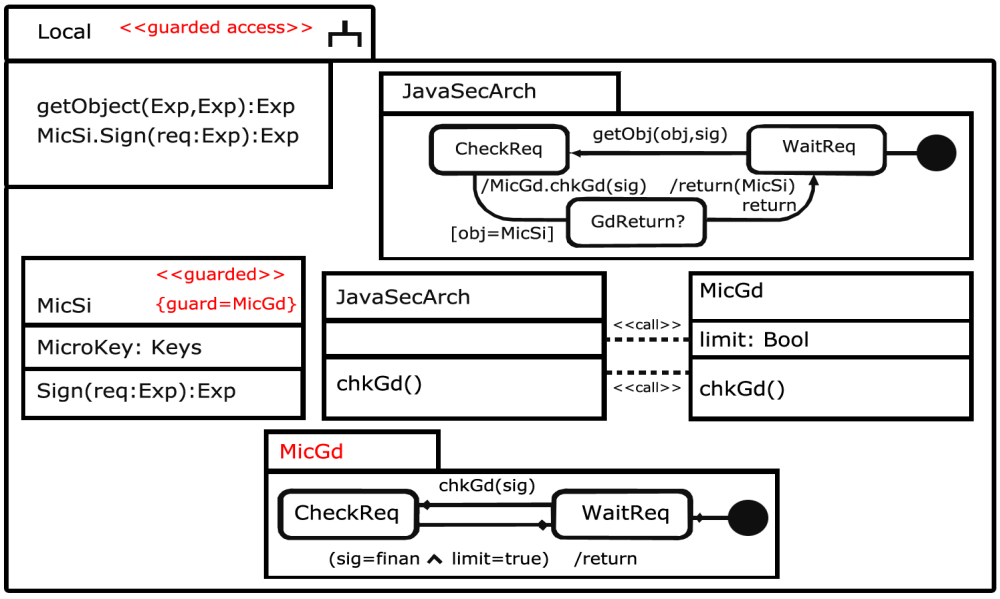
- Rechtevergabe abhängig von **Ausführungskontext**.
- Zugangskontrollentscheidungen bezüglich verschiedener **Threads**.
- Können verschiedene **Schutzdomäne** betreffen.
- Methode **doPrivileged() unabhängig** von Ausführungskontext.

Gibt Werkzeuge zur Überprüfung.

# Entwurfsprozess

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

## Frage




Guarded objects korrekt eingesetzt ?



**Literatur:**

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.3: S. 67 Abb. 4.12

## <<guarded access>>



Stellt sicher: in Java <<guarded>> Klassen nur durch {guard} Klassen aufrufbar.

Constraints:

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



**Literatur:**

Jan Jürjens: Secure systems development with UML

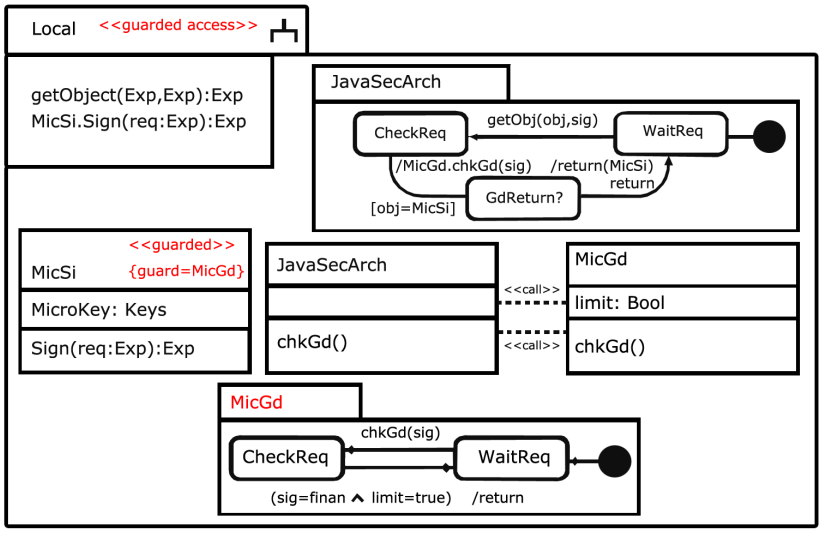
- Kap. 4.1.2: S. 65

# Guarded Objekte
# <<guarded access>>

- Jedes <<guarded>> Objekt im Teilsystem kann nur durch {guard} spezifizierte Objekte an den <<guarded>> Objekt angebracht, zugegriffen werden.
- Formal: Nehme an name $\notin K^{p}_{A}$ für Gegnertyp A unter Berücksichtigung das jeder Name name einer Instanz von einem <<guarded>> Objekt ist, bedeutet, dass eine Referenz öffentlich nicht verfügbar ist.
- Annahme: für jeden <<guarded>> Objekt gibt es eine Statechart Spezifikation eines Objekts dessen Name in {guard} gegeben ist.
  => Um Weitergabe von Referenzen zu modellieren.

# Example <<guarded access>>






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



**Literatur:**

Jan Jürjens: Secure systems development with UML

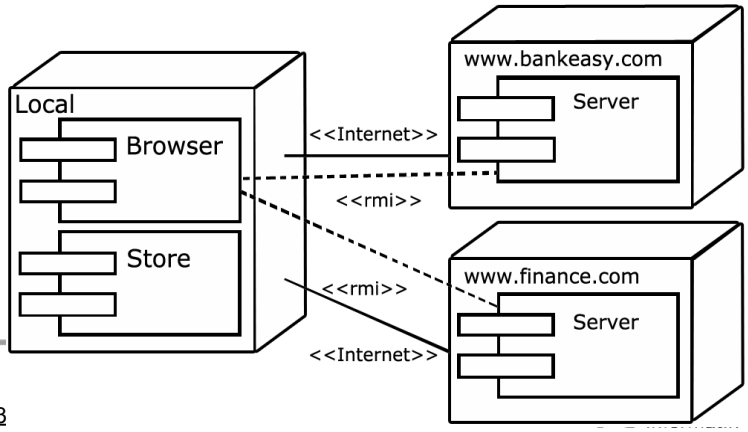
- Kap. 4.1.3: S. 67 Abb. 4.12

# Beispiel: Finanz-Anfrage



Internetbank bankeasy.com und Finanzberater finance.com bieten Dienstleistungen an. Applets benötigen Privilegien:

- Signierte Applets von der Bank: Daten zwi. 13 und 14 Uhr **lesen** und **schreiben** dürfen.
- Signierte Applets vom Finanzdienstleister: Micropayment-Schlüssel 5 mal die Woche nutzen dürfen.






## Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 5.4.3: S. 122-123 Abb. 5.25

# Guarded Objects
## <<guarded access>>

Beispiel:

- Darstellung mit einer webbasierten Finanzanwendung.
- Zwei Organisationen bieten Services für lokale Benutzer über Internet:
  - Internetbank, Bankeasy
  - Finanzberater, Finanz.
- Nutzen folgende Services:
  - Lokale Kunden benötigt um den Applets Privilegien zu gewähren.
- Zugang zu lokalen Finanzdaten realisierbar durch Nutzung von GuardedObjects.

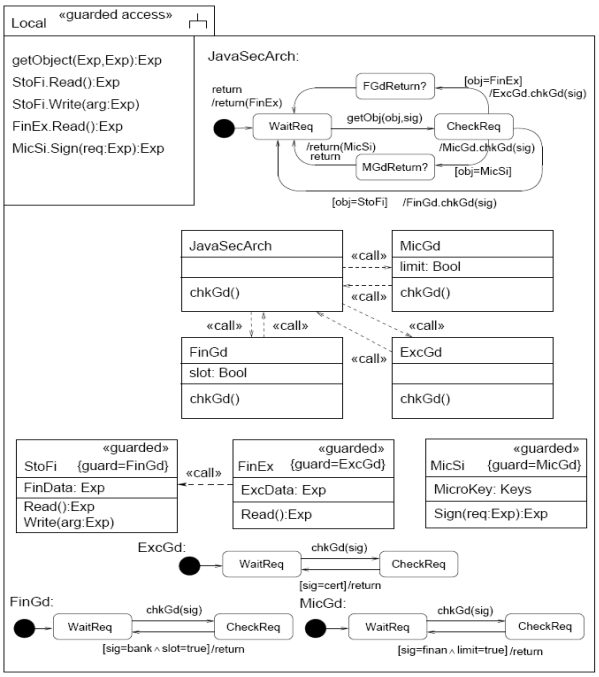
# Guarded
## <<guarded>>

- <<guarded>> benennt Objekte im Umfang von <<guarded access>> die vermutlich geschützt werden sollen.
- Tag:
  - {guard} Name von entsprechendem guard Objekt.
- <<guarded>> Objekte

  StoFi, FinEx, MicSi

  geschützt durch jeweilige {guard} Objekte

  FinGd, ExpGd, MicGd.



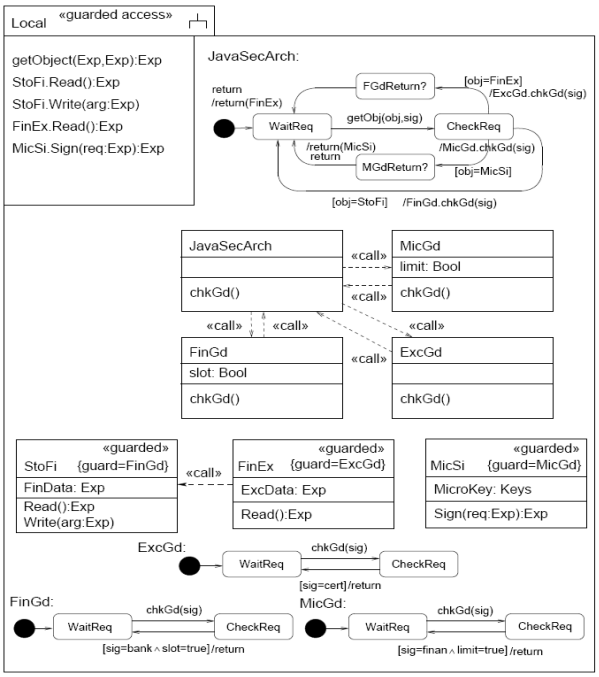



Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 5.4

# Guarded Objekte <<guarded access>>

Vereinfachter relevanter Teil von Java Sicherheitsarchitektur

- Erhält Anfragen für Objektreferenzen.
- Sendet sie zu den guard Objekten der drei guarded Objekte.
- <<guarded>> Objekte StoFi, FinEx, und MicSi können nur über ihren dazugehörigen Guard zugegriffen werden.
  - Teilsysteminstanzen führen die Bedingung mit zugehörigem <<guarded access>> bzgl. Standardgegner.

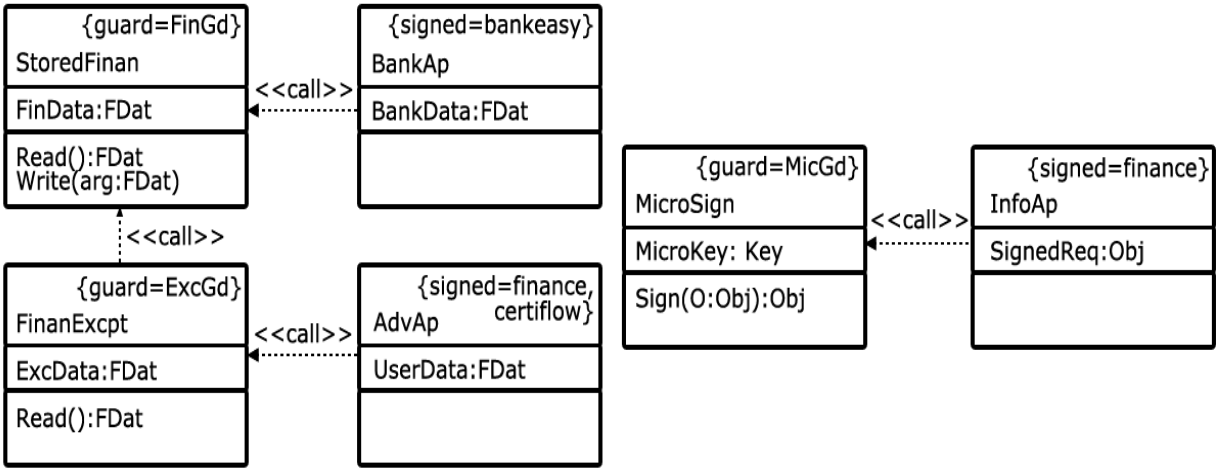


# Finanz-Anfrage: Klassendiagramm



Für Integrität und Vertraulichkeit: Daten als **Signed**- und **Sealed**-Objekte über Internet senden.

GuardedObjects kontrollieren Zugriff.



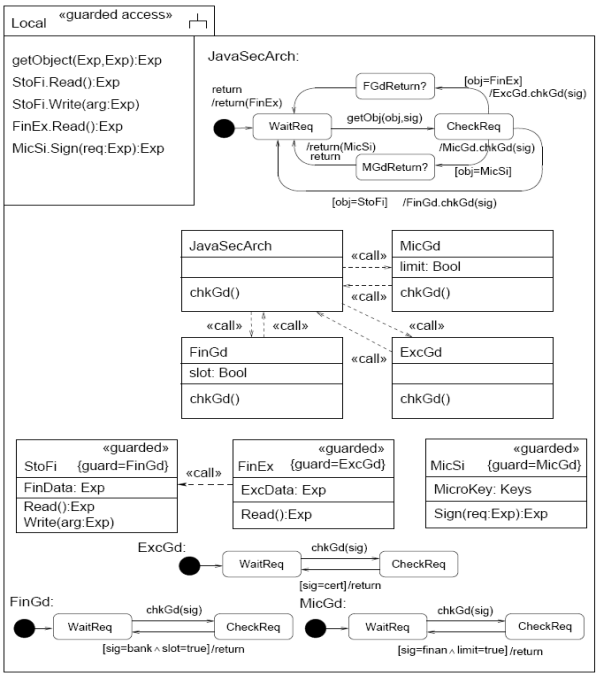
# Guarded Objekte Beispiel

Zugriffskontrolle durch Guard Objekte FinGd, ExpGd, und MicGd durchgeführt.

- Verhalten ist spezifiziert.

Applets signiert durch die Bsank

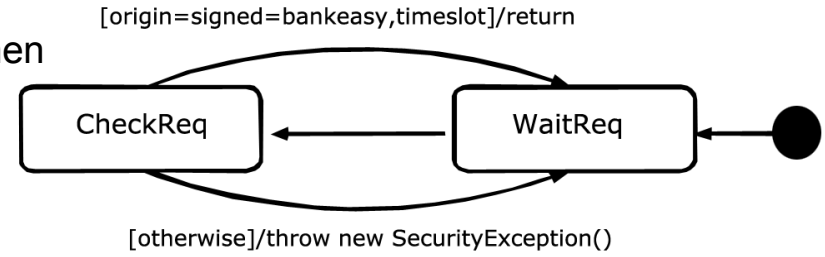
- Lesen und schreiben der im lokalen Datenspeicher gespeicherten Finanzdaten, nur zwi. 13-14 Uhr.
- Durchgesetzt durch das FinGd guard Objekt.
  - Bedingungsslot ist erfüllt wenn und nur wenn zwi. 13-14 Uhr ist.



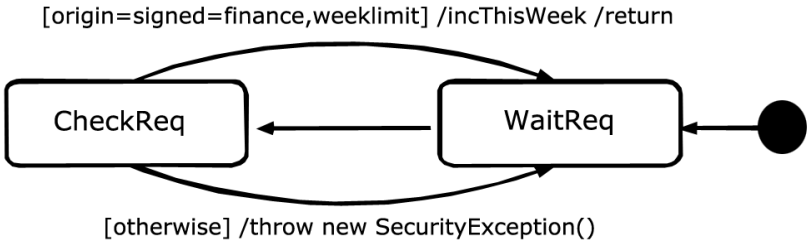
# Finanz-Anfrage: Guard Objects (Schritt 2)



- **timeslot** *true* zwischen 13 und 14 Uhr.




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






## Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 5.4.3: S. 124-125 Abb. 5.26

## Finanz-Anfrage: Validierung



Guard Objekt gibt **genügend Schutz** (Schritt 3):

- **Analyse-Ergebnis**: UML Spezifikation für Guard Objekte erteilt Zugangsanforderungen implizierte Genehmigungen.
  - z.B.: Annahme: aktuell ausgeführtes Applet stammt von Finance und ist signiert.
  - Einsatz genutzter Micropayment Key weniger als 5 mal.

  → Zugriff gewähren.

Zugangskontrolle konsistent mit **Funktionalität** (Schritt 4):

- **Analyseergebnis**: Jedes Objekt, das laut Spezifikation Zugriff auf guarded Objekt erhalten soll, erhält diesen auch.
- **Mobile Objekte** ausreichend geschützt (Schritt 5): über Internet gesendete Objekte sind signiert und versiegelt.



**Literatur:**

Jan Jürjens: Secure systems development with UML

- Kap. 5.4.3: S. 125

# 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.

# Sicherheitsanforderungen: sicherer Informationsfluss

- Traditioneller Weg Sicherheit in Computersystemen zu gewährleisten: **multi-level secure** Systeme (verschiedene Stufen der Sensibilität von Daten).
- Zur Einfachheit, zwei Sicherheitsstufen: **high**, bedeutet starke Sensibilität oder stark gesichert, und **low**, bedeutet weniger Sensibilität oder weniger gesichert.
- Wo gesicherte Teile eines Systems mit unsicheren Teilen interagieren, muss man gewährleisten, dass es keine “indirect leakage” von empfindlichen Informationen von einem sicheren zu einem unsicheren Teil gibt.

# Sicherheitsanforderungen: sicherer Informationsfluss

- Sicheren Informationsfluss gewährleisten → “no down-flow” Regel durchführen: **niedrige** Daten können **hohe** Daten beeinflussen aber nicht vc. vs..
- Gegenteil, “no up-flow“, erzwingt, dass nicht vertrauenswürdige Teile eines Systems indirekt hohe Daten nicht manipulieren kann: **hohe** Daten können **niedrige** Daten beeinflussen aber nicht vc. vs..
- Diese Sicherheitsanforderungen, **sicherer Informationsfluss** oder **non-interference:** strikte Definitionen der Sicherheit und Integrität, welche implizite Informationsflüsse erkennen kann → **covert channels**.

# Sicherer Informationsfluss

- Alternativer Weg von Spezifizierung secrecy-& integrity-like Anforderungen.
- Schutz gegen partieller Informationsfluss.
- Kann schwieriger sein, insb. beim Umgang mit Verschlüsselung.
- Für jeden Teil des Datensystems eine der zwei Sicherheitsstufen:
  - hoch, sehr vertrauenswürdig.
  - niedrig, weniger vertrauenswürdig.

Gegeben Menge von Nachrichten H und Folge m von Multimengen:

- $m^H$ Nachrichtennamen für Folge von Ereignismultimengen beim Löschen aller Ereignisse, die nicht in H sind.
- $m_H$ Nachrichtennamen für Folge von Ereignismultimengen beim Löschen aller Ereignisse, die in H sind.

Definition: Gegeben sei ein Teilsystem S und eine Menge von hohen Nachrichten H:

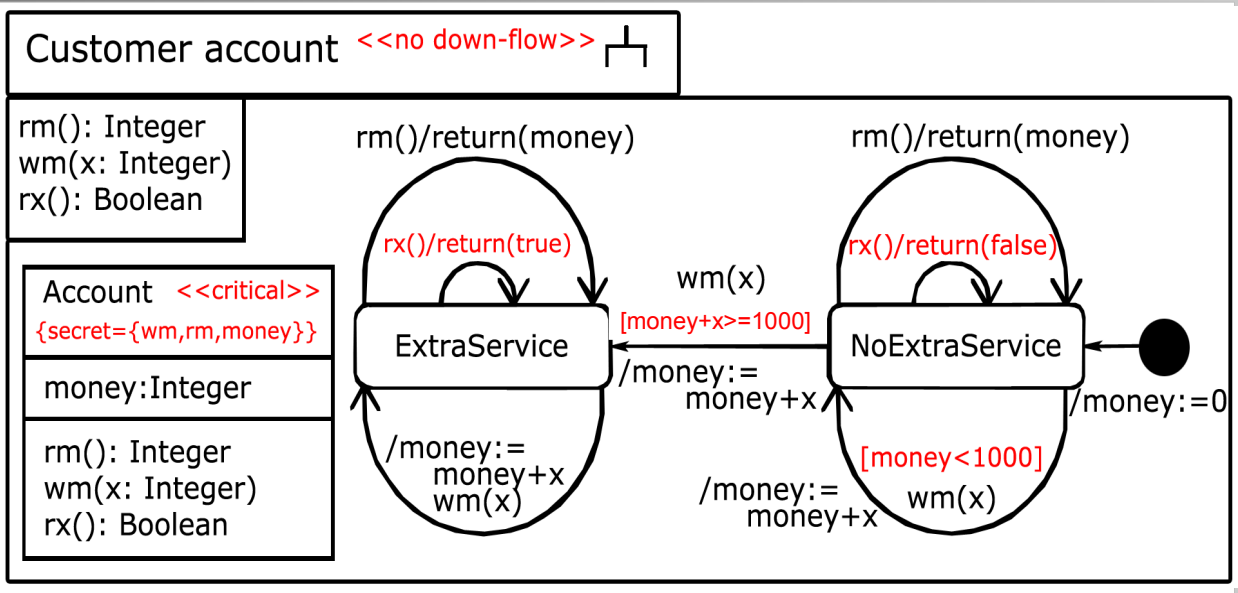
- A verhindert down-flow gegenüber H, wenn für beliebige zwei Sequenzen i; j von Ereignis Multimengen und beliebige 2 Ausgangssequenzen:
  $o \in [[S]]_A(i)$ und $p \in [[S]]A(j)$, $i_H = j_H$ impliziert $o_H = p_H$ und
- A verhindert up-flow gegenüber H, wenn für beliebige zwei Sequenzen i; j von Ereignis Multimengen und beliebige Ausgangssequenzen:
  $o \in [[S]]_A(i)$ und $p \in [[S]]_A(j)$, $i^H = j^H$ impliziert $o^H = p^H$.

Intuitiv:

- Down-flow verhindern: Ausgabe einer nicht-hohen (oder niedrigen) Nachricht hängt nicht von hohen Eingaben ab.
  - Strikte Sicherheitsanforderungen für als hoch markierte Nachrichten.
- Up-flow verhindern: Ausgabe eines hohen Wertes hängt nicht von niedrigen Eingaben ab.
  - Strikte Integritätsanforderungen für als hoch markierte Nachrichten.

→ Verallgemeinerung von „non-interference“ für deterministische Systeme zu Systemmodells, die auf Grund Unterspezifizierung nicht deterministisch sind.

## Frage




Keine partielle Preisgabe von Geheimnissen?

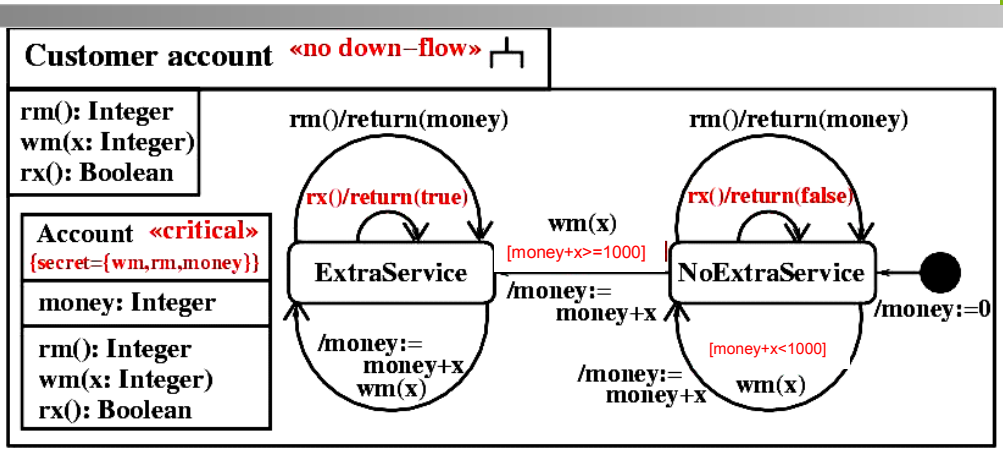


**Literatur:**

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.2: S. 65 Abb. 4.11

# Sicherer Informationsfluss: Beispiel




- Geheimes Attribut money beinhaltet Kontostand eines Kunden.
  - Lesbar durch rm(): Rückgabewert auch geheim.
  - Money durch die Operation wm(x) erhöhen.
- Wenn money 1000 übersteigt, zu State ExtraService übergehen.
- Öffentliche Operation rx(): prüft, ob zusätzl. Dienstleistungen erbracht werden soll.

## <<no down–flow>>

Stelle sicheren **Informationsfluss** sicher.

**Bedingung:**

- Jeder als {secrecy} spezifizierter Datenwert beeinflusst **nur** als {secrecy} spezifizierte Datenwerte.

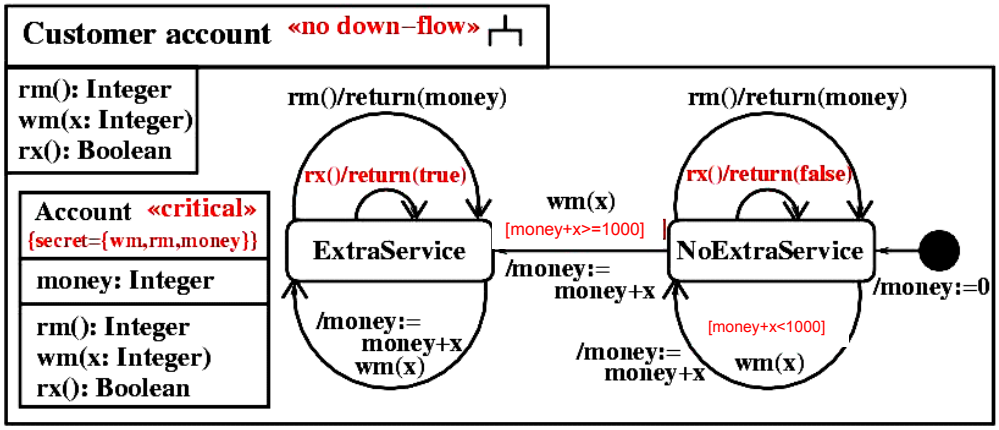
Bedingung mit Bezug auf formale Ausführungsemantik formalisierbar.



**Literatur:**
Jan Jürjens: Secure systems development with UML
- Kap. 4.1.2: S. 64

# Sicherer Informationsfluss

- Verhindert indirekten Verlust aus jeder Teilinformation über hohe Daten durch nicht-hohe Daten in <<no down-flow>>.
  - Durchführung von sicherem Informationsfluss durch Nutzen von {high} assoziiert mit <<critical>>.
- Intuitiv: Wert von beliebigen Daten in {secrecy} kann nur die Werte von Daten in {secrecy} beeinflussen.
- Präzisier: formalisieren anhand formale Verhaltenssemantiken: Einschränkung für <<no down-flow>> (resp. <<no up-flow>>) ist, dass UML machine Exec[[S]] für Teilsystem S down-flow (resp. up-flow) gegenüber in <<high>> spezifizierte Nachrichten und ihre Antwortnachrichten verhindert.
- Beobachtbare Information über Kundenkonto soll keinen Rückschluss über Kontostand zulassen.

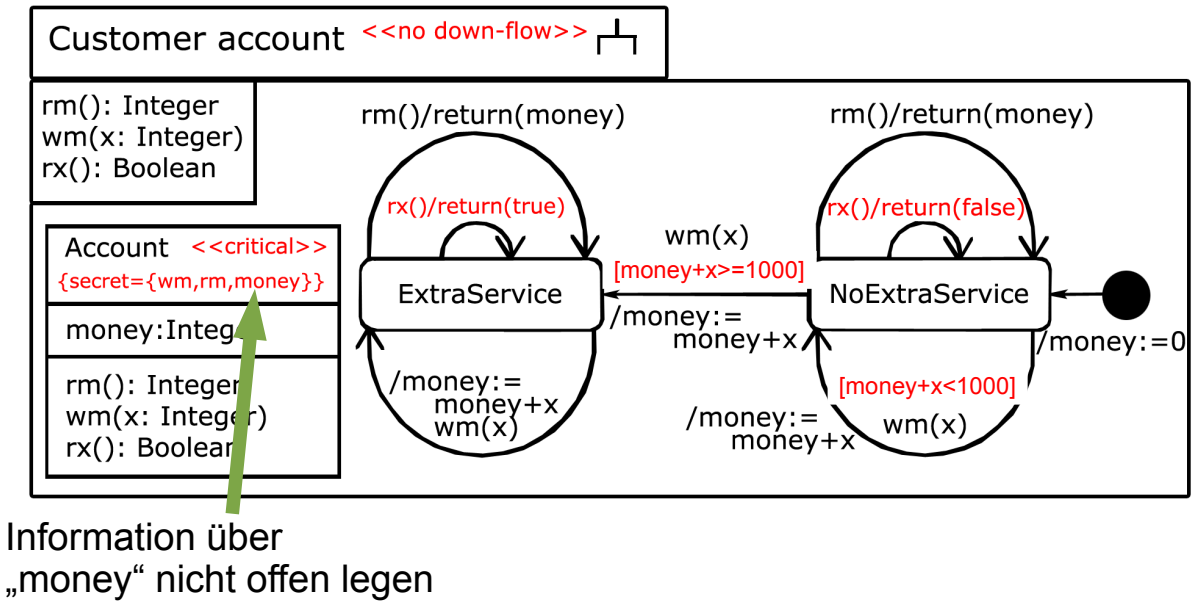



Mit <<no down-flow >> spezifizieren, dass Objekt keine Informationen über sichere Daten (wie Attribut money) herausgeben soll.

Kein partieller Verlust von Geheimnissen ?

# Sicherer Informationsfluss: Beispiel






Information über „money“ nicht offen legen



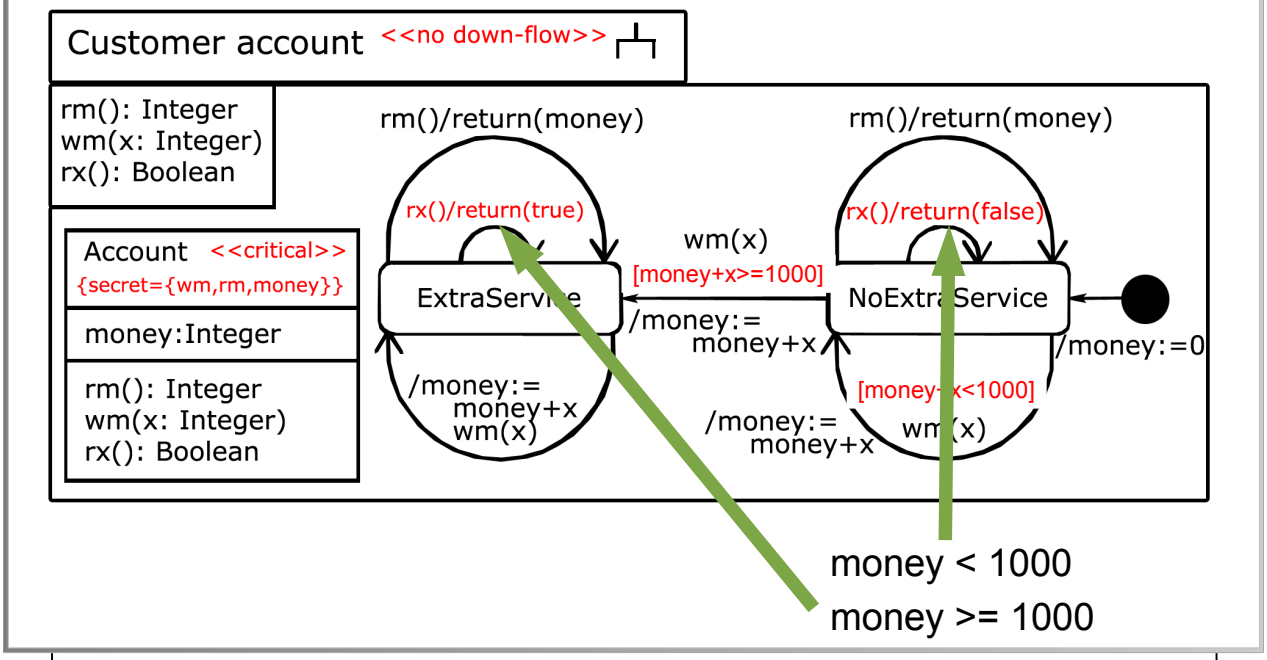
**Literatur:**

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.2: S. 65 Abb. 4.11

# Sicherer Informationsfluss: Beispiel







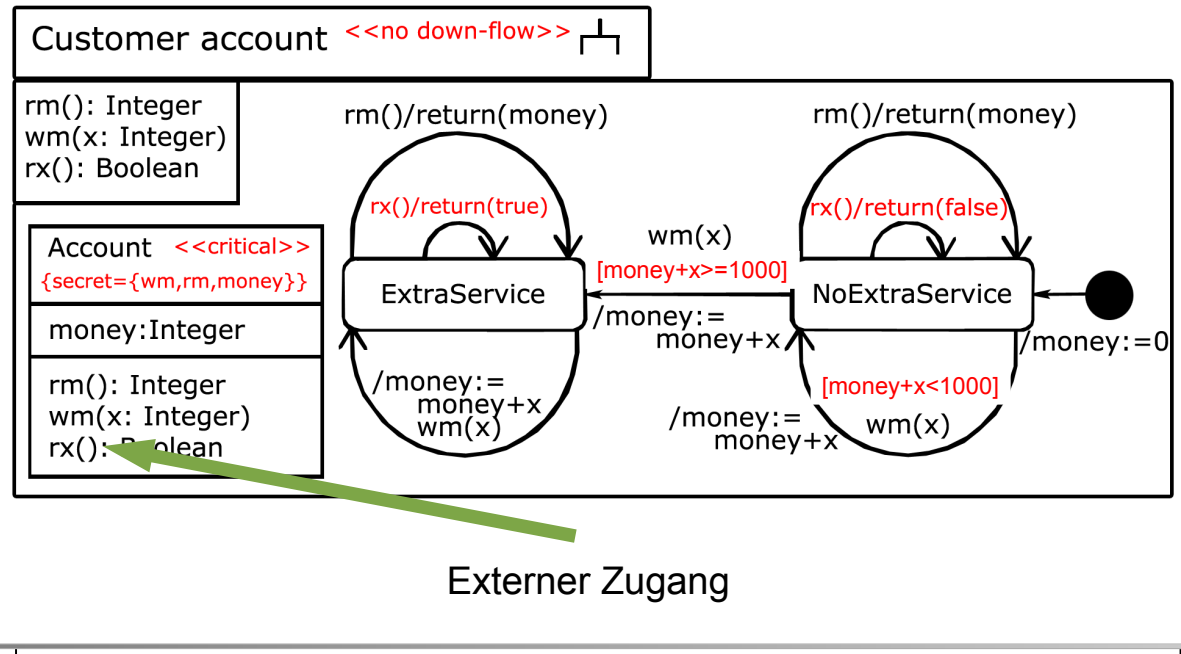

**Literatur:**

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.2: S. 65 Abb. 4.11

# Sicherer Informationsfluss: Beispiel






Externer Zugang



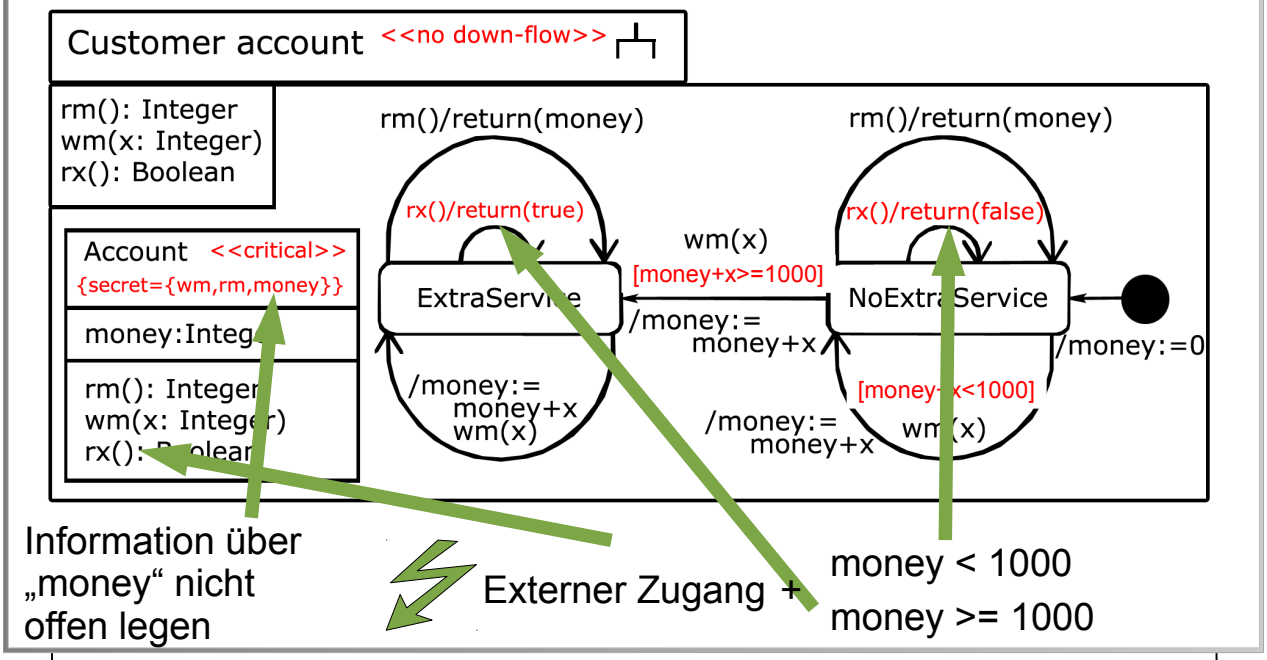
**Literatur:**

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.2: S. 65 Abb. 4.11

# Sicherer Informationsfluss: Beispiel







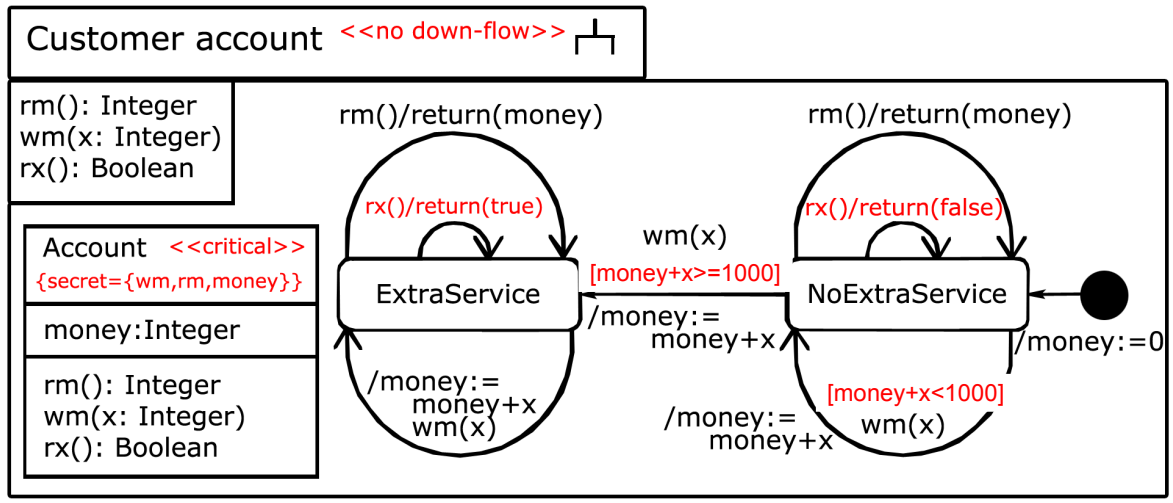

**Literatur:**

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.2: S. 65 Abb. 4.11

# Sicherer Informationsfluss: Beispiel






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



## Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.2: S. 65 Abb. 4.11

# Sicherer Informationsfluss: Beispiel

Wie das unterliegende Formalismus den Sicherheitsfluss unter Benutzung der vorherigen Definition erfasst:

- Sequenzen i; j von Eingabe Multimengen
- Sequenzen $o \in [[A]](i)$ und $p \in [[A]](j)$ von Ausgabe Multimengen der UML Machine A gegeben durch das Verhalten der betrachteten Statechart
- mit $i_H = j_H$ und $o_H \neq p_H$, wobei H Folge von hohen Nachrichten ist.
- Betrachte die Sequenzen
  - i := ({{wm(0)}} ; {{rx()}} )
  - j := ({{wm(1000)}} ; {{rx()}}).

# Sicherer Informationsfluss: Beispiel

Gegeben $i_H$ = ({{ }} , {{ rx() }} ) = $j_H$.

Definition von Verhaltenssemantiken der Statecharts, gibt Ausgangs-Multimengen:

- o := ({{ }}, {{return(false) }} ) ∈ [[A]](i).
- p := ({{ }}, {{return(true) }} ) ∈ [[A]](j).

=>

- $o_H$ = ({{ }}, {{return(false) }} ) ≠ ({{ }}, {{return(true) }} ) = $p_H$

Bedeutet, dass die Bedingung <<no down-flow>> verletzt ist.

Automatisch erkennbar mit Werkzeugunterstützung.



Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.2

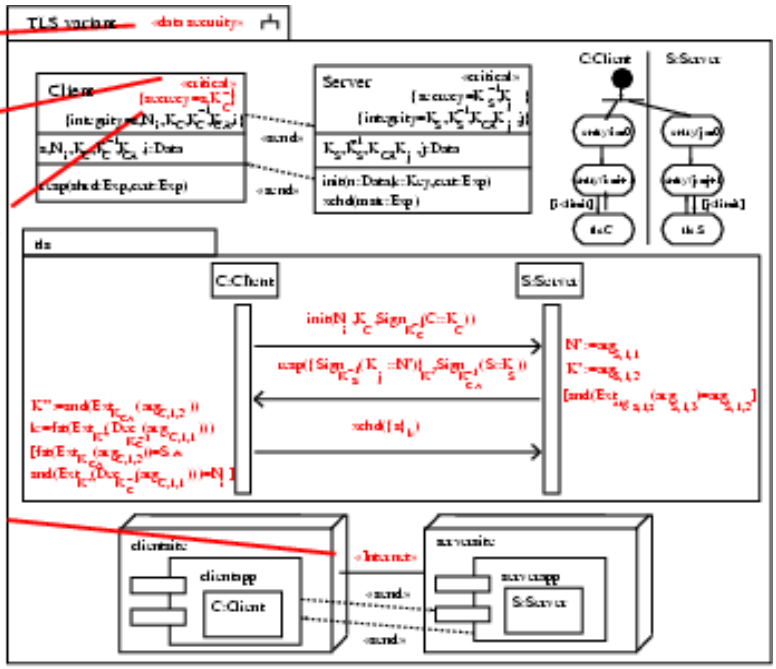
«data security»

«critical»

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

Variante von TLS (INFOCOM`99).
Kryptoprotokoll sicher gegen **Standard-**Angreifer?

«Internet»






**Literatur:**

Jan Jürjens: Secure systems development with UML

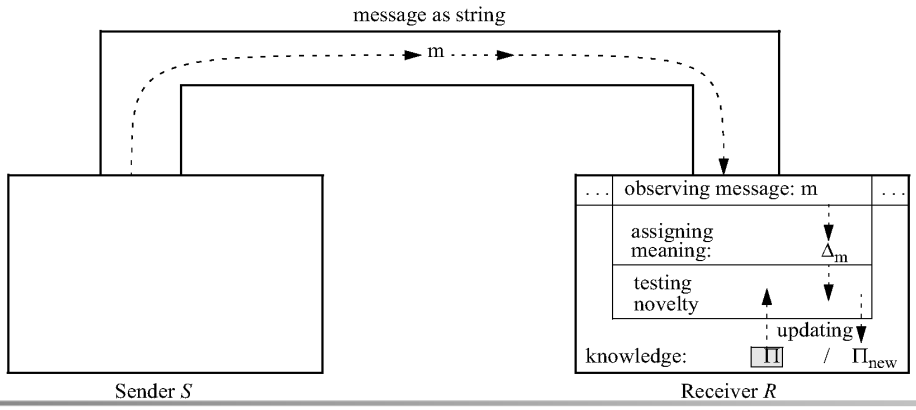
- Kap. 4.1.2: S. 63 Abb. 4.10

# Informationsfluss

- Eine gesendete Nachricht ist nicht unbedingt *sinnvoll* bezüglich des Inhalts für ein Empfänger oder andere *Beobachter*
- Es kann passieren und sogar sinnvoll sein, dass ein beobachteter String zufällig und ohne Information auftaucht:
  aus der Sicht des Beobachters hat die Nachrichtenübertragung kein Informationsfluss verursacht
- In anderen Fällen gelingt es dem Beobachter dem beobachteten String eine Bedeutung zuzuweisen, ungefähr wie folgend:
  - ermittelt eine Behauptung, die die Wahrheit einiger Aspekte seiner Überlegungen ausdrückt;
  - wenn zusätzlich Beobachter diese Wahrheit neu erlernt, dann hat die Nachrichtenübertragung aus der Sicht des Beobachters einen Informationsfluss verursacht

# Informationsfluss basiert auf Nachrichternübertragung

1. Eine Nachricht beobachten: Betrachte ein String *m*
2. Bedeutung zurodnen: Bestimme einen Urteil $\Delta_m$
3. Wissen ausdrücken: Voraussetzung $\Pi$ bilden als Sammlung von Urteilen
   Neuheit testen: Folgern ob $\Pi$ $\Delta_m$ impliziert
4. Wissen updaten: Falls neu (nicht impliziert), füge $\Delta_m$ zu $\Pi$ hinzu und umstellen, Ergebnis in $\Pi_{new.}$



# Informationsfluss und Nachrichtenübertragung

- Eine Nachrichtenübertragung muss nicht unbedingt zu einem Informationsfluss für jeden Beobachter führen.
- Manchmal muss Beobachter Schlussfolgerungen folgern um eine Nachrichtenübertragung als ein Informationsfluss aus seiner Sicht aus zu erscheinen.
- Für eine solche Schlussfolgerung kann der Beobachter a priori Wissen, wie einem vorher erhobenen Schlüssel, nutzen
- Für tatsächlichen Inferenz,
- Beobachter benötigt entsprechende Rechenmittel

## <<data security>>

Stellt sicher:

- **Sicherheitsanforderungen** an als <<critical>> markierte Daten.
  - → Bedrohungsszenario aus zugehörigem Verteilungsdiagramm (Deployment-Diagramm) zugrunde legen.

**Constraints:**

- Als {secrecy}, {integrity}, {authenticity}, {fresh} markierte Daten,
- Erfüllen auf Basis der Ausführungssemantik der UML Diagramme formalisierten Sicherheitsanforderungen. (Einzelheiten s. nächster Vorlesungsabschnitt.)



**Literatur:**
Jan Jürjens: Secure systems development with UML
- Kap. 4.1.2: S. 60-61

# <<data security>>

- Sicherheitsanforderungen der als <<critical>> markierten Daten gegen Bedrohungsszenarien von Verteilungsdiagramm durchgesetzt.
- Einschränkungen: Als {secrecy}, {integrity}, {authenticity}, {fresh} markierte Daten erfüllen jeweilige formalisierte Sicherheitsanforderungen.
- Einschränkung mit <<data security>> erfordert, dass diese Anforderungen bzgl. Angegebenes Gegnermodell erfüllt sind.
- Formalisierung dieser Abschnitt wird im nächsten Abschnitt diskutiert.

# <<data security>>

- Teilsystem S der <<data security>> berücksichtigt Datensicherheitsanforderungen durch <<critical>> und die im Teilsystem enthaltenen zugehörige Tags bzgl. der Bedrohungsszenario aus der Verteilungsdiagramm und des Gegenertyps A in {adversary}

Präziser: Einschränkungen gegeben durch vier Bedingungen, die die Konzepte der secrecy, integrity, authenticity, und freshness nutzen.

- secrecy: Teilsystem bewahrt Geheimhaltung der Daten bezeichnet durch {secrecy} gegen Gegnertyp A.
- authenticity: Für jede (a,o) von {authenticity}, bewahrt S die Authentizität von Attribut a bzgl. seines Ursprungs o gegen Gegnertyp A.

# <<data security>>

- integrity: {integrity} von <<critical>> mit einem Wert (v,E), Subsystem bewahrt die Integrität der Variable v gegen Gegnertyp A, bzgl. E der zulässigen Ausdrücke.
  - Falls E ausgelassen: Integrität von v bewahren bzgl. der Folge von Ausdrücken, die aus denen in Spezifikation von S konstruierbar sind.
  - Gegner darf nicht erreichen, dass die Variable v einen von ihm vorher bekannten Wert annimmt.
- freshness: Innerhalb S <<data security>>, beliebiger Wert data ∈ Data ∪ Keys markiert als {fresh} im jeweiligen Teilsystem Instanz oder Objekt D <<critical>> in S sollte neu in D sein.

# <<data security>>

Vorheriges Wissen des Gegners darf die Datenwerte gemäß den Tags von <<critical>> nicht enthalten, sollte secrecy, integrity oder authenticity gewährleistet werden:

- Nicht erreichbar, falls Gegner diese Daten ursprünglich kennt.
- Weitere Annahmen über Grundkenntnisse des Gegners können angegeben werden.
- Wenn zulässige Ausdrücke oder angestrebter Ursprung der Daten in {integrity} und {authenticity} sich auf Ausdrücke beziehen, die nicht lokal am <<critical>> Objekt bekannt sind und wo diese Tags eingesetzt werden, kann man sie zu <<data security>> zuordnen.
- Annahme: Standardgegner nicht in der Lage Verschlüsselung zu brechen, aber kann Designfehler z.B. in einem Kryptoprotokoll ausnutzen, z.B. durch den Versuch von „man-in-the-middle“ Attacken.

# <<data security>>

Beachte:

- Genug Daten erforderlich, um mit Sicherheitsanforderungen in einem der Objekte oder Teilsysteme die Bedingungen zu erfüllen.
- Mehrere verschachtelte Subsysteme bringen womöglich <<data security>> mit sich.
  - Die Bedingungen sind für jedes Subsystem nötig.
    Das ist wichtig zu beachten, wenn ein Subsystem in ein anderes Subsytem eingefügt werden soll.

Variante des Internet Security Protokoll TLS vorgeschlagen in [APS99]

Ziel:

- Sicherer Kanal zwischen Client und Server über eine unsichere Kommunikationsverbindung.
  - stelle Geheimhaltung und Server Authentizität, wie in {secrecy} und {authenticity} spezifiziert, sicher.
- Um dies zu erreichen müssen einige lokale Attribute dem Stereotyp {integrity} genügen.
  - Der Angreifer sollte nicht in der Lage sein die Werte dieser Attribute zu erlangen.



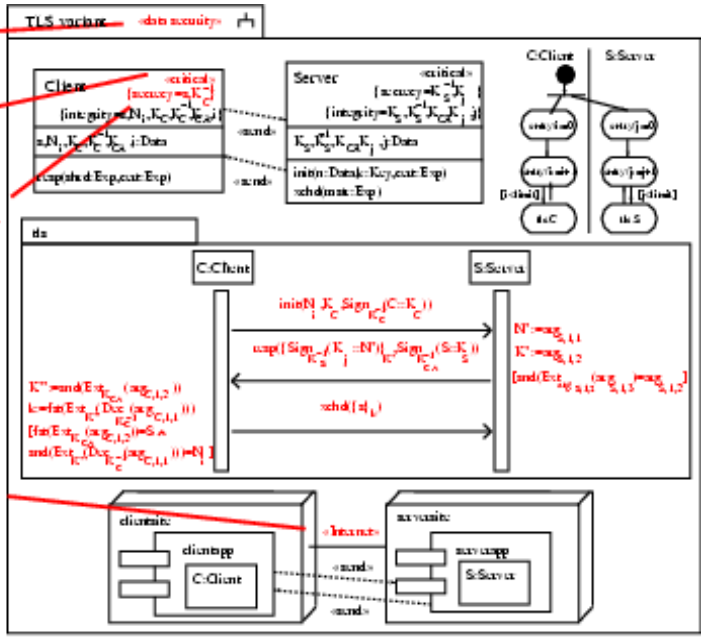
[APS99] V. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost? In Conference on Computer Communications (IEEE Infocom), pages 717-725. IEEE Computer Society, New York, March 1999.

# Sicherer Einsatz von Kryptographie Beispiel

Variante von TLS (INFOCOM`99).

Ziel ist es mit Hilfe von den öffentlichen Schlüsseln $K_C$ and $K_S$ den geheimen Sitzungsschlüssel K auszutauschen, welcher dann genutzt wird um die geheimen Daten s vor der Übertragung zu verschlüsseln.

Kryptoprotokoll sicher gegen Standard-Angreifer?

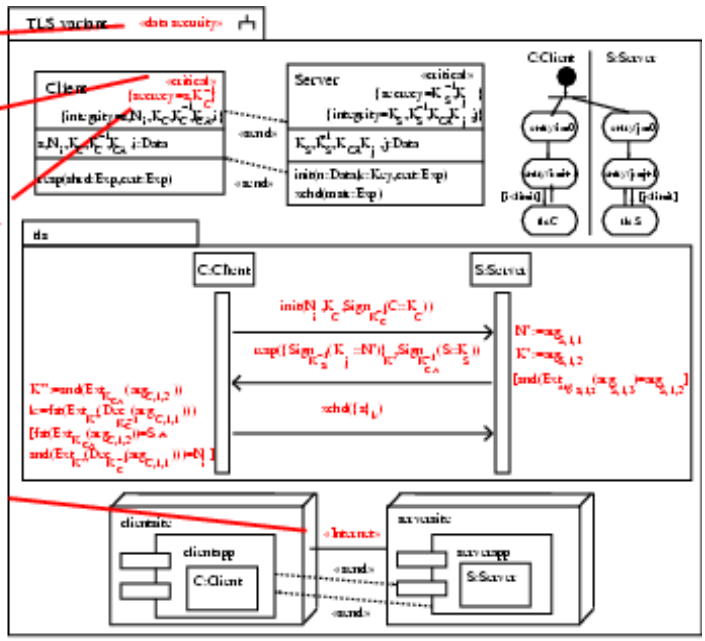


# Beispiel (Variante von TLS)
# <<data security>>

«data security»

«critical»

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




Verletzt {secrecy} von s gegenüber eines Standard-Angreifers.

Mehr Einzelheiten: «Internet» nächster Abschnitt.

# Sicherer Einsatz von Kryptographie: mögliche Erweiterungen

- Eigenschaften von secrecy, integrity, und authenticity werden vom jeweiligen Angreifertyp abhängig gemacht.
- Standard-Angreifer sind im Prinzip Externe für das System;
- Angreifer als Teil vom System sind möglich, wenn diese Zugriff auf relevante Systemkomponenten haben.
  - In dem man z.B. $Threats_A(s)$ definiert um den relevanten Stereotyp s Zugriff zu geben.
- z.B.: E-Commerce Protokoll das Kunde, Händler und Bank beinhaltet
  - Bestellte Ware ist ein Geheimnis, das nur Kunde und Händler bekannt ist, und nicht der Bank.

# Sicherer Einsatz von Kryptographie: mögliche Erweiterungen

- Kann formuliert werden in dem man relevante Daten mit „secret“ markiert und Sicherheitsanalysen vom Angreifermodell „bank“ erstellt.
  - Angreifer wird Zugriff auf Bank-Komponenten gegeben in dem man eine Threats() Funktion definiert.
- Beachte: Angreifer hat nicht unbedingt Zugriff auf das System.
- Kann sinnvoll sein, z.B. to apply {secrecy} to a value received by the system from the outside.
- Bedingungen von <<data security>> stellen nur sicher, dass stereotypisierte Komponenten die vom Environment Secret erhaltenen Werte behalten.
- Stelle sicher, dass die Systemumgebung nicht diese Werte dem Angreifer verfügbar macht.

# UMLsec Intro: Übersicht




- **Sicherheitsanforderungen**: <<secrecy>>,…
- **Angriffszenarien:** Benutze $\text{Threats}_{adv}(ster)$.
- **Sicherheitskonzepte:** Beispiel <<smart card>>.
- **Sicherheitsmechanismen:** Beispiel <guarded access>>.
- **Sicherheitsalgorithmen:** Verschlüsselung eingebaut.
- **Physikalische Sicherheit:** Gegeben in Verteilungsdiagramm.
- **Sicherheitsmanagement:** Benutze Aktivitätsdiagramme.
- **Technologie-spezifisch:** Java-, CORBA-Sicherheit.

# Erfüllt UMLsec die Anforderungen?

- Sicherheitsanforderungen: Formalisierung von grundlegenden Sicherheitsanforderungen, die durch Stereotypen wie <<secrecy>>, etc. bereitgestellt werden
- Bedrohungsszenarien: durch formale Semantiken und der darunterliegenden modellierten physikalischen Schicht stehen dem Angreifer vom Typ *adv* die Menge $Threats_{adv}(ster)$ von möglichen Aktionen zur Verfügung.
- Sicherheitskonzepte: Beispiel <<smart card>>.
- Darunterliegende physikalische Sicherheit:
  - Adressiert von <<secure links>> in Deployment Diagrammen.
- Sicherheitsprimitiven:
  - Entweder eingebaut, wie z.B. Verschlüsselung oder
  - Kann behandelt werden, wie z.B. Sicherheitsprotokolle.
- Sicherheitsmanagement : → Aktivitätsdiagramme

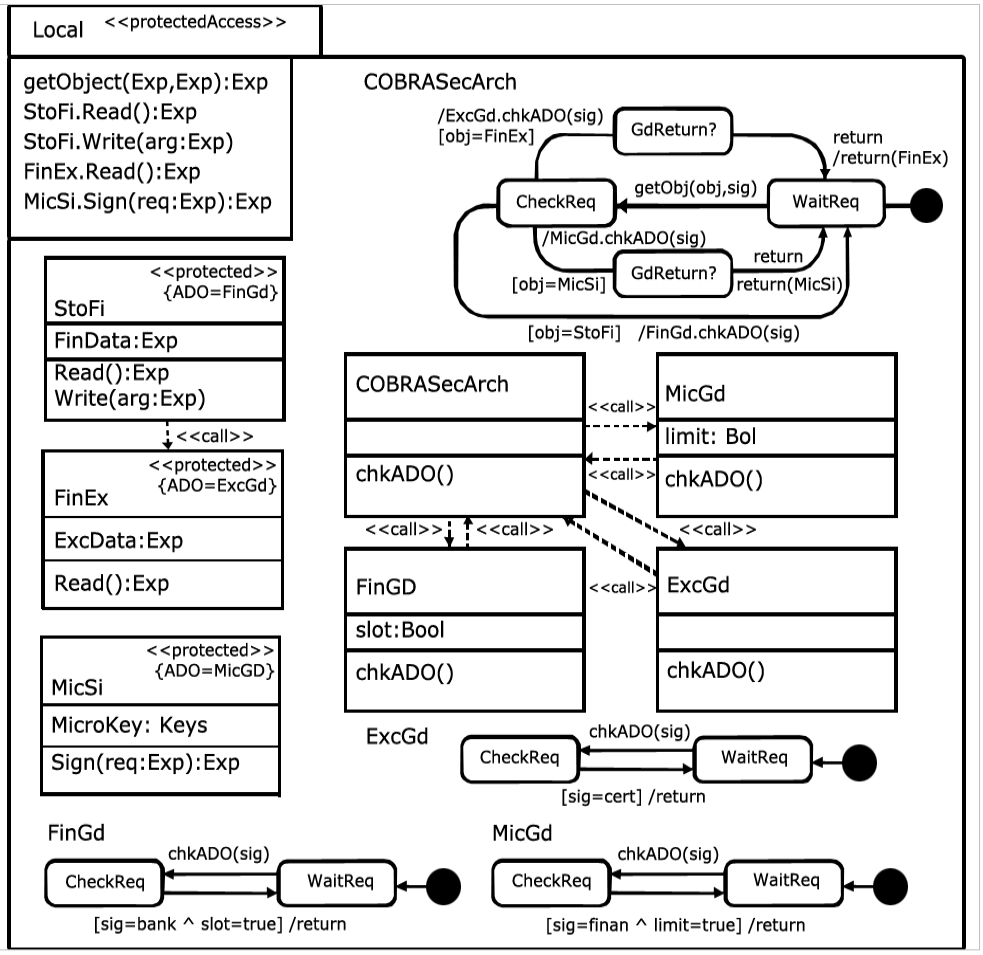
# Zusammenfassung




- Signed und Sealed Objects
- Guard Objects
- <<no down-flow>>
- <<data security>>

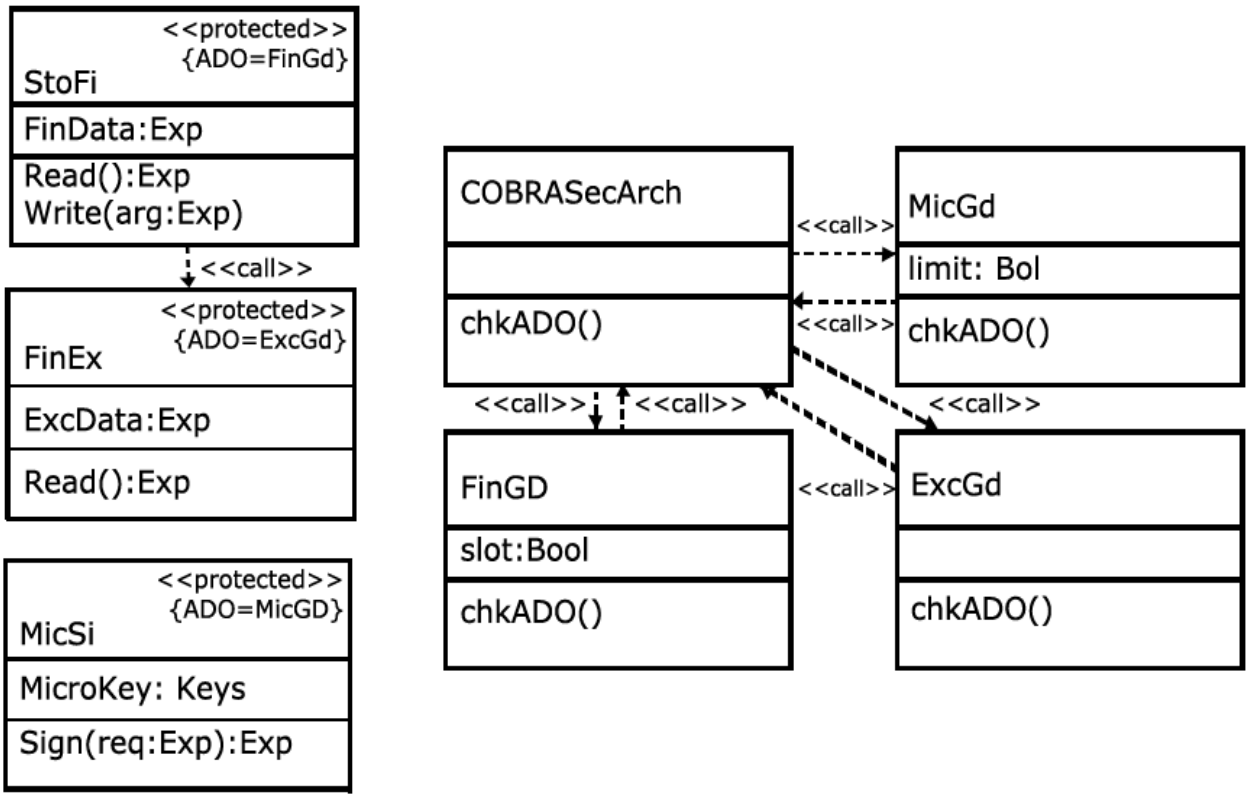
# Alte Folien

# Example: CORBA Access Control with UMLsec



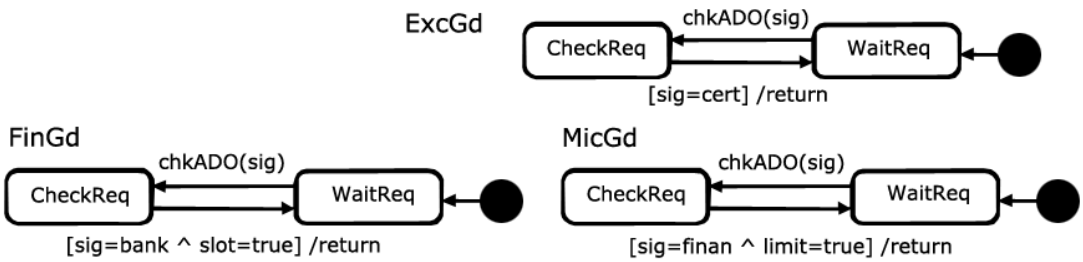
# Example: CORBA Access Control with UMLsec





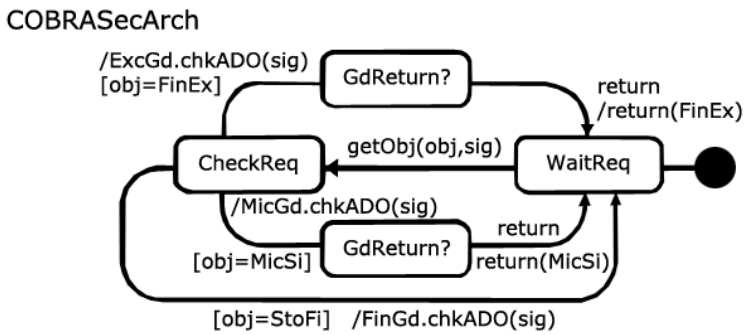
# Example: CORBA Access Control with UMLsec



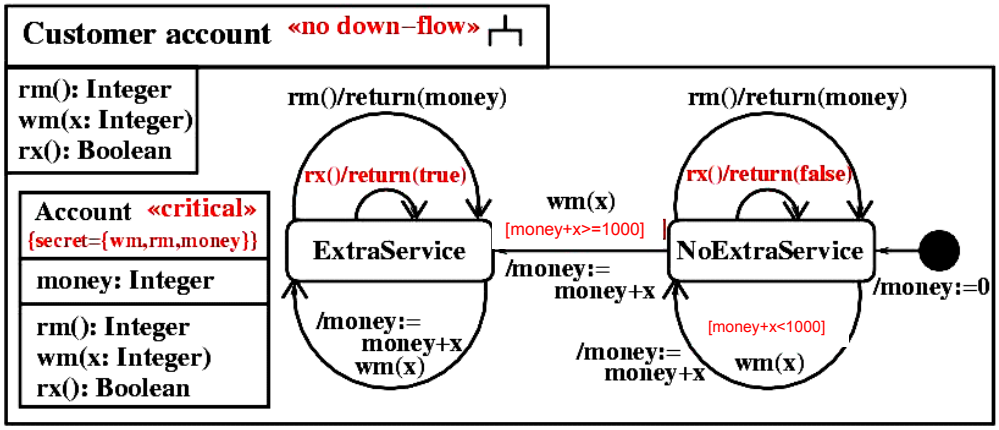


# Example: CORBA Access Control with UMLsec





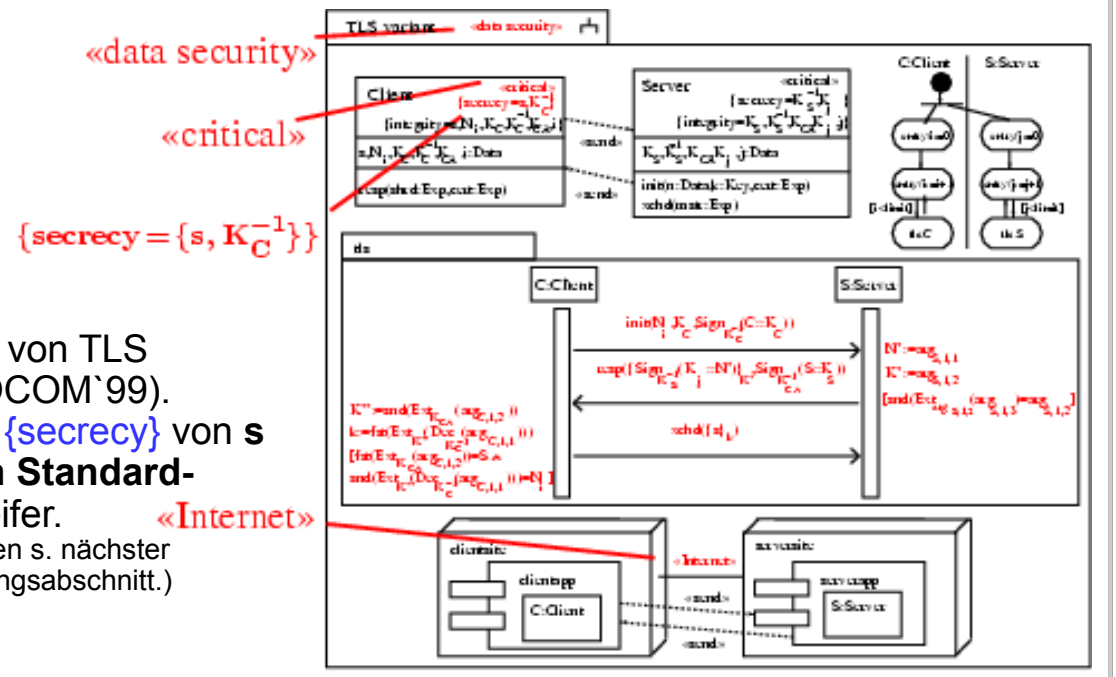
# Sicherer Informationsfluss: Beispiel (3)




- <<no down-fow>> zeigt, dass das Objekt keine Informationen über sichere Daten, wie das Attribut money, verlieren soll.
- Verletzung der Bedingung von <<no down-flow>>:
  - Partielle Information durch das Einput der hohen Methode wm() über den Rückgabewert der nicht-hohen Operation rx() verloren.

# Beispiel <<data security>>






Variante von TLS (INFOCOM`99).

**Verletzt** {secrecy} von **s** gegen **Standard-**Angreifer.

(Einzelheiten s. nächster Vorlesungsabschnitt.)



## Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1.2: S. 63 Abb. 4.10