Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2014

Prof. Dr. Jan Jürjens

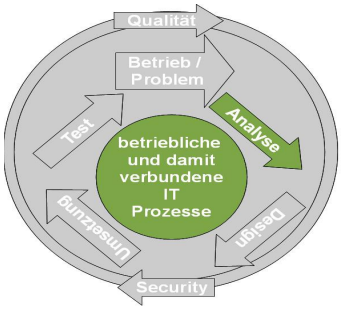
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 3.3: Modellbasierte Sicherheit mit UMLsec

v. 02.07.2014

Einordnung
3.3 Modellbasierte Sicherheit mit UML

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Einführung: Software Security
  - Hintergrund IT-Sicherheit
  - Wiederholung: Metamodellierung
  - Modellbasierte Sicherheit mit UML
  - 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:**
  Wiederholung: Metamodellierung
- **Dieser Abschnitt:** Modellbasierte Sicherheit mit UML
  - UMLsec
  - Profil (1. Teil)

UML-Profil:

- Spezialisierung von Standard UML-Elementen zu konkreten Metatypen.
- Profil zu Modell hinzufügbar und im gesamten Modell verfügbar.
- Verschiedene Profile für verschiedene Anwendungsdomänen.
- Einige Profile vordefiniert und bei OMG verfügbar.

Definition eines Profils:

- Paket von Stereotypen und Tagged Values.
- Klassendiagramm definiert Beziehungen zwischen neuem Stereotyp und dem zu beschreibenden Element.

UML Profil
Erweiterungsmöglichkeiten

- Stereotypen:

  spezialisiert Benutzung von Modellelementen *<<label>>*.
  (kann auch durch Symbol visualisiert werden, z.B  für
  <<subsystem>>).

- Tagged value:

  fügt *{tag=value}* Paare zu stereotypisierten Elementen hinzu.

- Constraint:

  verfeinert Semantik eines stereotypisierten Elements.

- Profil:

  sammelt obige Informationen.

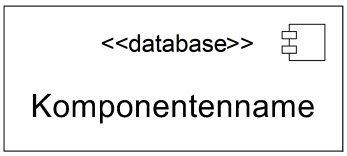
UML Profil
Stereotypen

Definition von Stereotypen:

Definition neuer Stereotypen gemäß **Metamodell**:
(NB: Der Pfeil ist eine Spezialisierung.)

Konkretes Beispiel
einer Definition:

Spätere Verwendung im
Komponentendiagramm:



Tagged Values:

- Werte beschreiben **Eigenschaften eines Stereotypen** (fachliches Konzept).
- Tagged Values: **Name-Wert Paare.**
- **Einsatz** von Tagged Values im Modell:
  - **Kommentarfeld**, das mit Element verbunden ist oder
  - **Geschweifte Klammern** {Name = Wert} direkt in Element geschrieben.

Anforderungen an die UML Extension für Security I

Stellt **Sicherheitsanforderungen** wie **secrecy**, **integrity**, **authenticity** bereit.

Ermöglicht

- Erwägung verschiedener **Bedrohungsszenarios** abhängig von Fähigkeiten des Gegenspielers.
- Einfügen wichtiger **Sicherheitskonzepte** (wie z.B. *tamper-resistant hardware*).
- Aufnehmen von **Sicherheitsmechanismen** (wie z.B. access control).

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 50 1. Abschnitt

Anforderungen an die UML Extension für Security II

Stellt **grundlegende Sicherheit** bereit (z.B. mit (a)symmetrischer Verschlüsselung).

Ermöglicht:

- Begutachtung der **physikalischen Sicherheit**.
- Einbeziehen des **Sicherheitsmanagement** (z.B. secure workflow).

Also: Fügt **domain-spezifisches** Sicherheitswissen hinzu (Java, smart cards, CORBA, ...).

Literatur:

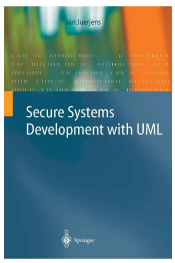
Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 50 1. Abschnitt

Modellbasierte Sicherheit mit UMLsec

Erweiterung der UML für **sichere Systementwicklung**.

- Evaluiert UML Modelle auf ihre Sicherheit.
- Zusammenfassung **bestehender Regeln** sicheren Entwickelns.
- Zugänglich an **nicht** auf sichere Systeme **spezialisierte** Entwicklern.
- Diskutiert Sicherheitsanforderungen aus **frühen** Designphasen im **Systemkontext**.
- In der Zertifizierung verwendbar.

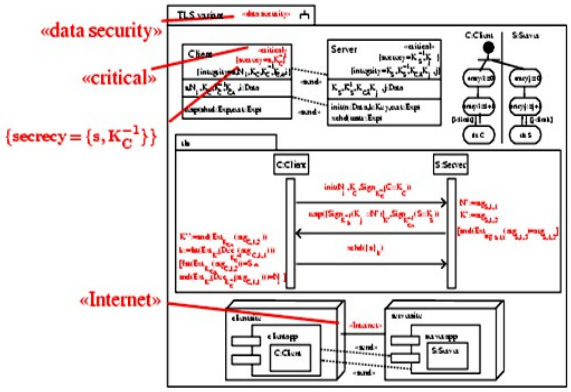


Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 50 1. Abschnitt von UMLsec Profile

UMLsec



- Fügt **wiederkehrende** Sicherheits-anforderungen, **feindliche** Szenarios, und **Sicherheit-mechanismen** als vordefinierte **Marker** hinzu.
- Nutzt verknüpfte logische constraints um die Spezifikation zu **verifizieren**. Einsatz von **Model-Checker** und **ATPs**, die auf formalen Semantiken basieren.
- Stellt sicher das die Modellspezifikation in UML die Sicherheitsanforderungen im Kontext des Dolev-Yao Angreifermodells **durchsetzt**.

Was deckt UMLsec ab ?

- **Sicherheitsanforderungen**: <<secrecy>>,…
- **Bedrohungsszenarien:** Vewende Threatsadv(ster).
- **Sicherheitskonzepte:** Zum Beispiel <<smart card>>.
- **Sicherheitsmechanismen:** Z.B. <<guarded access>>.
- **Grundlegende Sicherheit:** eingebaute Verschlüsselung
- **Physikalische Sicherheit:** Im Verteilungsdiagramm.
- **Sicherheitsmanagement:** Im Aktivitätsdiagramm.
- **Technologie spezifisch:** Java, CORBA Sicherheit.

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S.50 1. Abschnitt

UMLsec: Generelle Ideen

- **Aktivitätsdiagramm**: Sicherer Kontrollfluss, Koordination
- **Klassendiagramm:** Austausch von Daten; hält Sicherheitslevel bereit
- **Sequenzdiagramm:** Sicherheitskritische Interaktion
- **Zustandsdiagramm:** Sicherheit innerhalb eines Objekts
- **Verteilungsdiagramm:** Physikalische Sicherheitsanforderungen
- **Package:** Ganzheitliche Betrachtung der Sicherheit

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 3.2: S. 24

UMLsec Profil (Auszug)

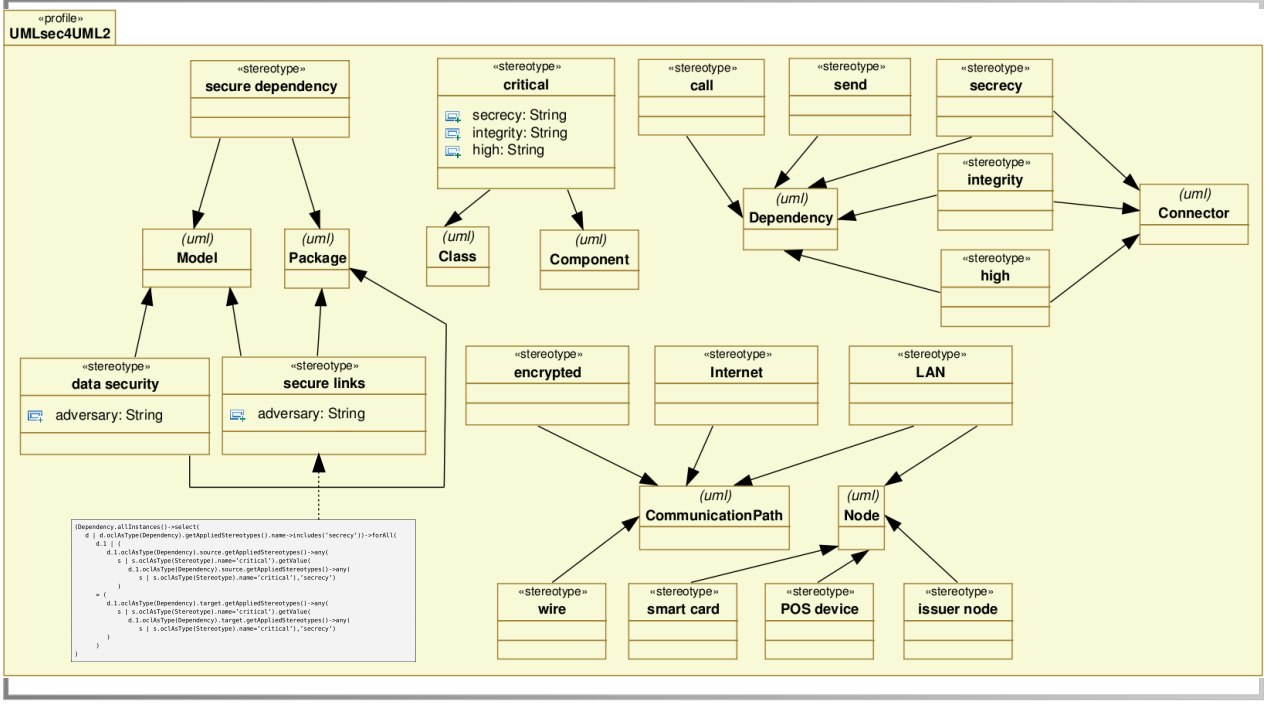
Stereotyp	Basisklasse	Tags	Bedingungen (Constraints)	Erläuterung
Internet	link			Internetverbindung
secure links	subsystem		dependency security matched by links	erzwingt sichere Kommunikationsverbind.
secrecy	dependency			gewährl. Geheimhaltung
secure dependency	subsystem		call, send respect data security	strukturelle Interaktion Datengeheimhaltung
no down-flow	subsystem	high	prevents down-flow	Informationsfluss
data security	subsystem		provides secrecy, integrity	grundlegende Daten-sicherheitsanforderung
fair exchange	package	start, stop	after start eventually reach stop	Erzwingt einen fairen Handel
guarded access	Subsystem		guarded objects acc. through guards.	Zugangskontrolle durch „guard objects“

Literatur:

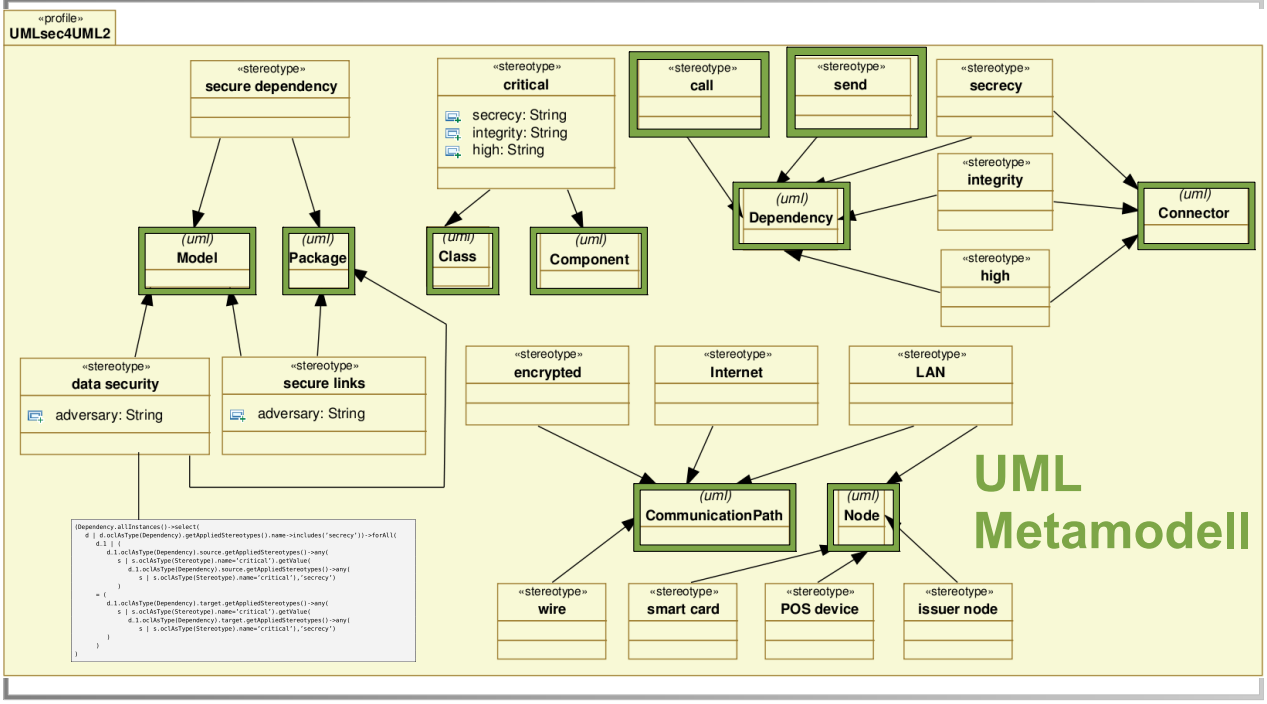
Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 51 Abb. 4.1

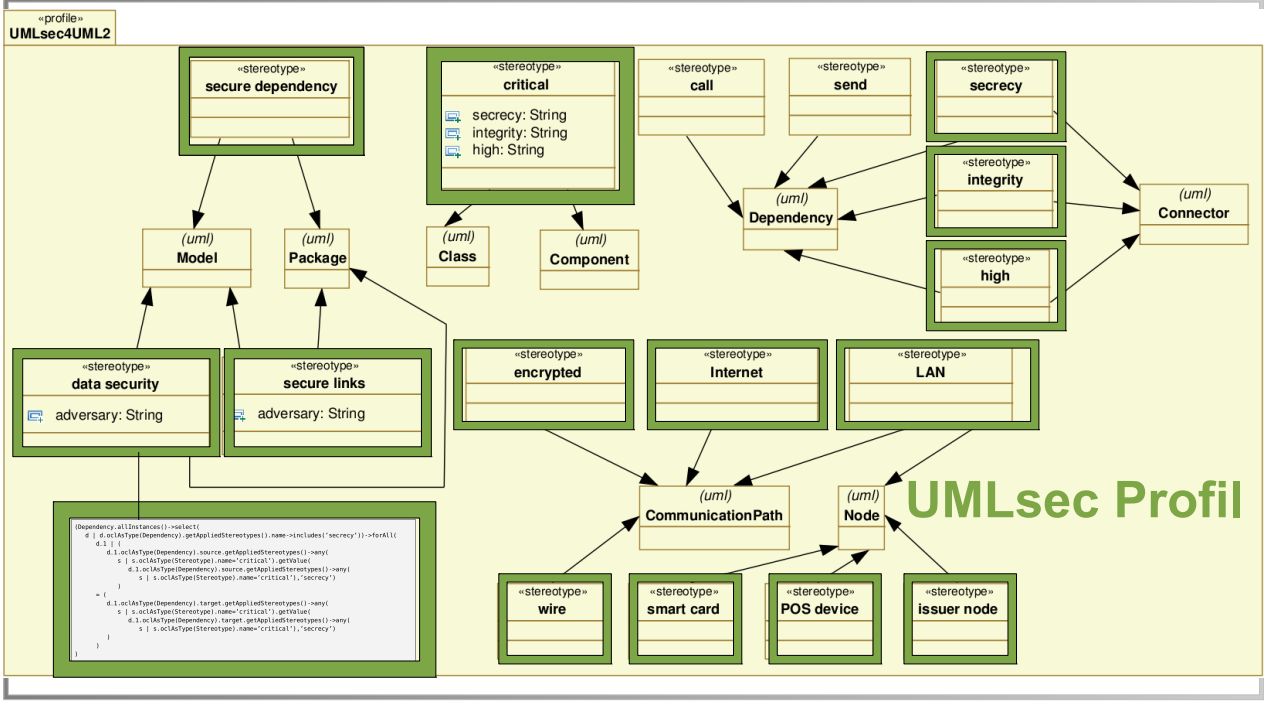
UMLsec-Profil: Metamodell



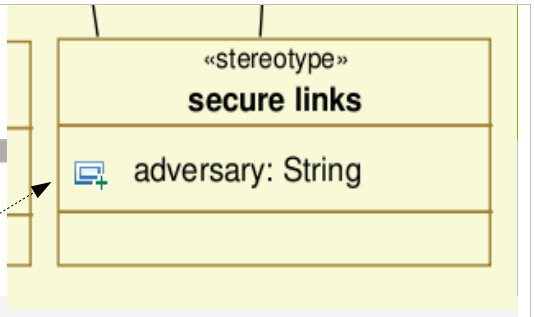
UMLsec-Metamodell: UML-Metamodell



UMLsec-Metamodell: UMLsec-Stereotypen

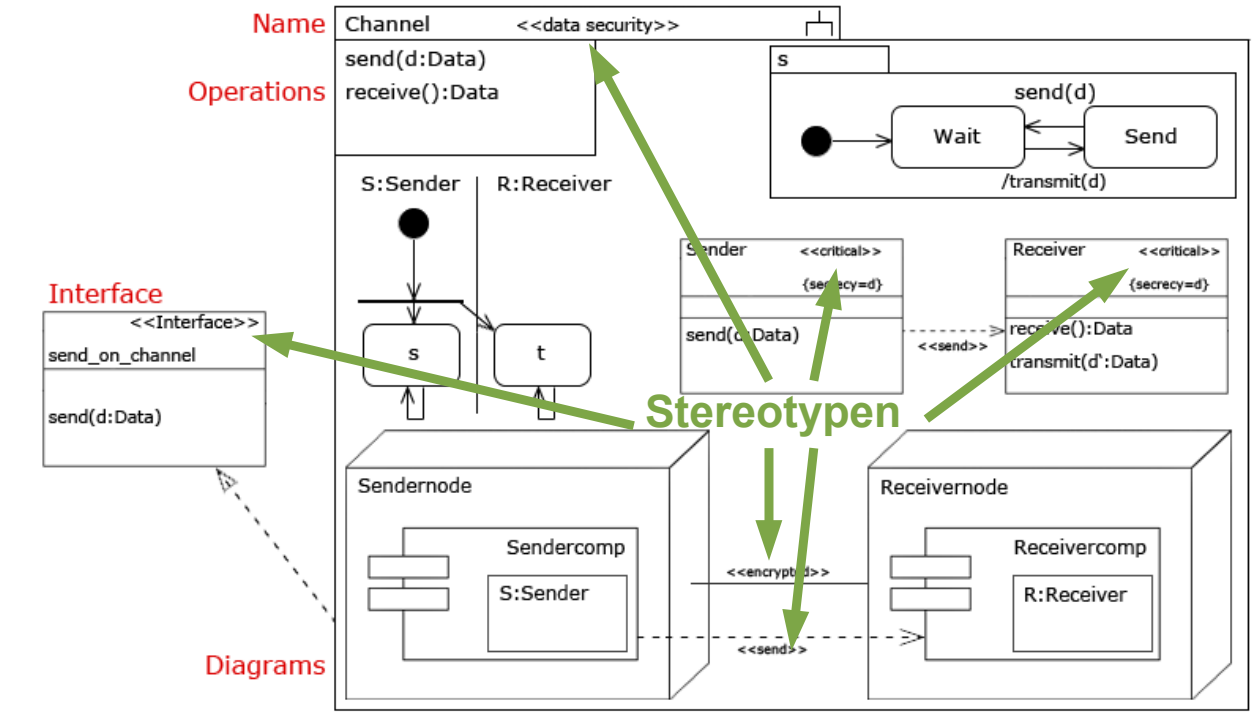


Beispiel: Stereotype <<secure links>> OCL constraint

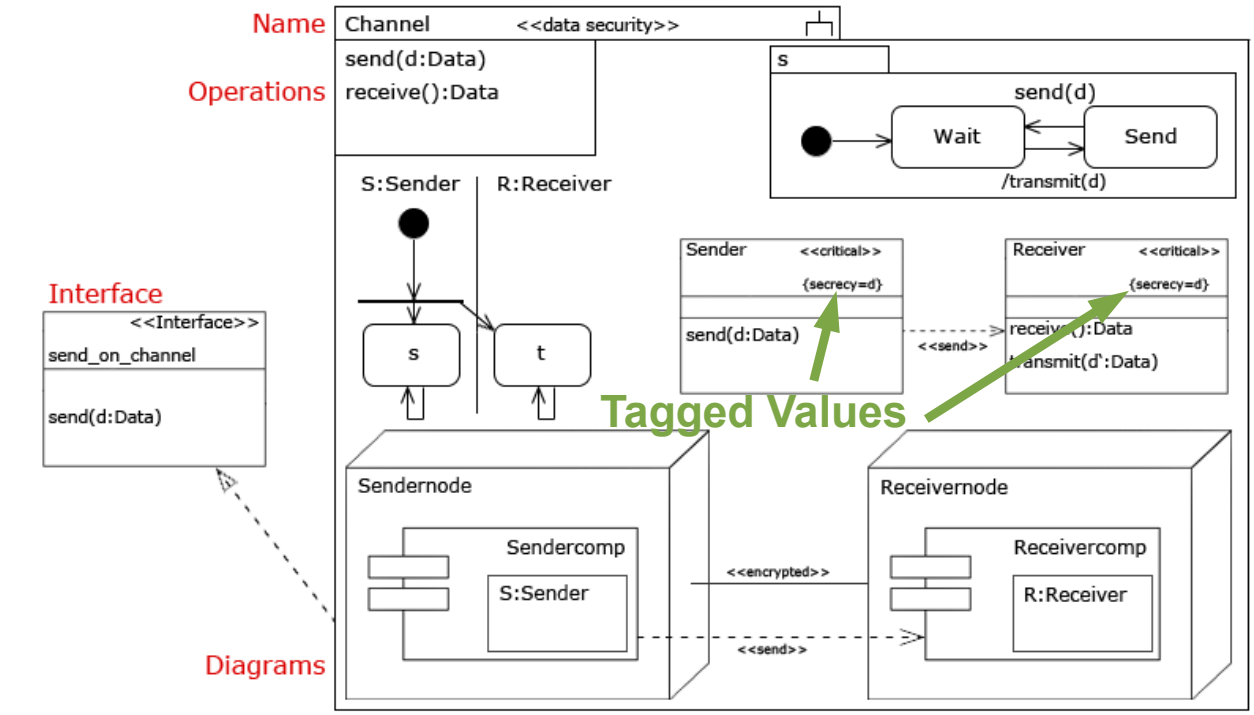



```
(Dependency.allInstances()->select(
   d | d.oclAsType(Dependency).getAppliedStereotypes().name->includes('secrecy'))->forAll(
      d_1 | (
         d_1.oclAsType(Dependency).source.getAppliedStereotypes()->any(
            s | s.oclAsType(Stereotype).name='critical').getValue(
               d_1.oclAsType(Dependency).source.getAppliedStereotypes()->any(
                  s | s.oclAsType(Stereotype).name='critical'),'secrecy')
            )
      = (
         d_1.oclAsType(Dependency).target.getAppliedStereotypes()->any(
            s | s.oclAsType(Stereotype).name='critical').getValue(
               d_1.oclAsType(Dependency).target.getAppliedStereotypes()->any(
                  s | s.oclAsType(Stereotype).name='critical'),'secrecy')
         )
      )
)
```

UMLsec Beispielmodell: Stereotypen



UMLsec Beispielmodell: Tagged Values



UMLsec als integriertes formales Framework

- Basiert auf **Formalisierung** wichtigster **Sicherheitsanforderungen** in einer integrierten Notation.
- Erlaubt Anforderungen **zu ordnen / kombinieren.**
- Ermöglicht **Modularität / Komponierbarkeit**, hierarchische **Zerlegung, Verfeinerung**, … :
- Zum Beispiel:
  - Wenn das System <<secure links>> und
  - das Subsystem <<data security>> genügt,
  - dann erfüllt das System auch <<data security>>.

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 52-53

UMLsec: Definition von Stereotypen und Tags

- Constraints nutzen sicherheitsbewusste Interpretation von UML Diagrammen.
- <<fair exchange>>, <<provable>>, <<secure links>>, <<data security>>:
  - Parametrisiert über Gegnerart bzgl. Anhalten der Sicherheitsbedingungen.
- {adversary}: Werte in Form von (T;C).
  - T: Gegnerart, z.B. T = default für Gegner später definiert, welches selbst definiert werden könnte.
    - Falls ausgelassen T = default.
  - C: Logische Bedingung auf Vorkenntnisse K^p_A des Gegners.
    - Falls ausgelassen, C sichert das die in {secrecy} Tag von <<critical>> einbezogene Werte nicht als Teilausdruck in K^p_A erscheinen.

Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.4

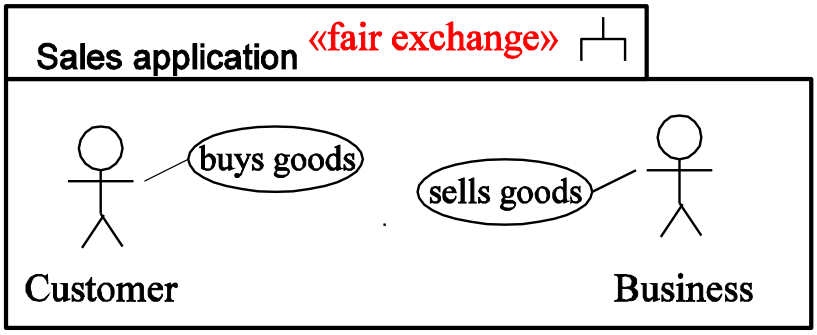
Sicherheitsanforderung: Fair Exchange

- Elektronische Handelsware, Anforderung der **fair exchange** setzt voraus, dass der Handel so durchgeführt wird, sodass beteiligte Parteien vom Betrug verhindert werden.
- Wenn z.B. der Käufer eine Anzahlung machen muss, sollte der Käufer in der Lage sein die Zahlung zu beweisen und das Geld zurückzufordern, falls die Ware danach nicht geliefert wurde.

Fair Exchange in Use Case Diagrammen

- Ausführung der Transaktionen: Beide Parteien vom Betrug verhindern.
- Anwendbar in Teilsystemen durch ein Anwendungsfalldiagramm.
  - Kann durch anderes Teilsystem verfeinert werden, falls es auch mit <<fair exchange>> stereotypiert ist.
- Nur informelle Bedeutung, im Gegensatz zu unteren Stereotypen.
  - “Verfeinerung": hier im informellen Sinn.
- Zeigt, wie die Sicherheitsanforderungen (wie Stereotypen) auch in anderen Arten von Diagrammen wie folgende in Anwendungsfalldiagrammen passend enthalten sein können.

Fair Exchange in Use Case Diagrammen



Use Case Diagramm beschreibt folgende Situation:

- Kunde kauft Ware beim Händler.
- Handel so ausführen, dass beide Parteien vom Betrug ausgeschlossen bleiben.
  - Anforderung hinzufügen durch Hinzufügen von <<fair exchange>> zum Teilsystem, welches das Diagramm beinhaltet.

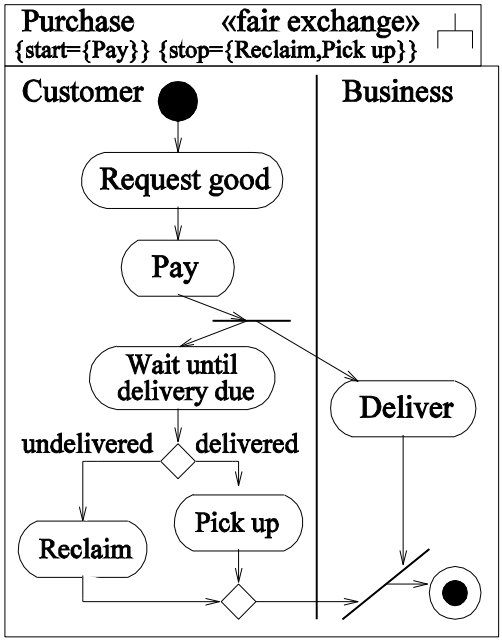
Sicherheitsanforderungen in Use Case Diagrammen erfassen.

- Constraint: müssen im entsprechenden Aktivitätsdiagramm erscheinen.

Fair Exchange in Aktivitätsdiagrammen

Kunde kauft Ware beim Händler.

Wie kann man fairen Handel erzwingen ?
Nach Bezahlung muss Kunde bis zur **Lieferfrist** warten und kann dann die **Zahlung zurückziehen.**



Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 54 Abb. 4.4

<<fair exchange>>

- Sichert generisch Bedingungen für **fairen Handel**.
- Bedingung: Wenn im Aktivitätsdiagramm ein {start} Punkt erreicht wurde, wird immer ein {stop} Punkt erreicht.
- (Das kann nicht für ein System sichergestellt werden, das durch einen Angreifer komplett lahm gelegt werden kann.)

Literatur:
Jan Jürjens: Secure systems development with UML
- Kap. 4.1: S. 54

Fair Exchange in Aktivitätsdiagrammen

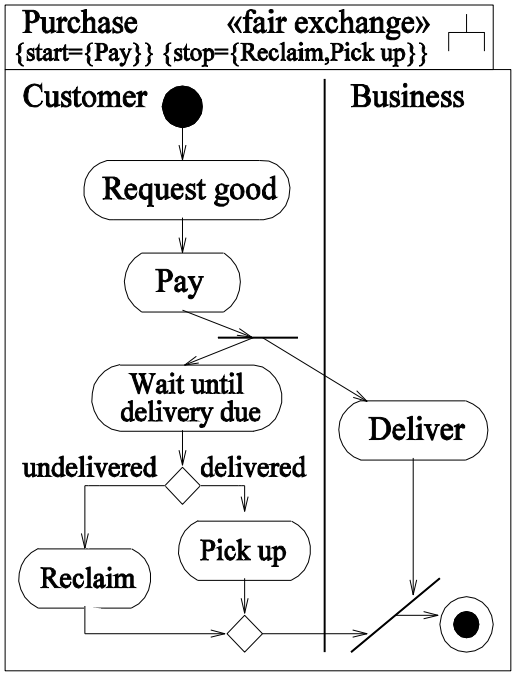
<<fair exchange>> eingesetzt in Teilsystemen beinhaltet ein Aktivitätsdiagramm

- assoziierte Tags {start}, {stop}, {adversary}.
- {start}, {stop} nehmen Paare (good; state) als Werte,
  - good ist der Name einer Ware, die verkauft werden soll; kann entfallen wenn nur eine Ware verkauft werden soll
  - State: Name eines Zustands.
- {adversary} Angreiferart zu der die Sicherheitsanforderung erhalten bleiben soll.
- Wenn das System in Gegenwart von einer in {adversary} spezifizierten Angreiferart A ausgeführt wird, wird für jede zu verkaufende Ware im Aktivitätsdiagramm letztendlich ein {stop} state erreicht und zwar genau dann wenn ein {start} state erreicht worden ist.

Beispiel
<<fair exchange>>

Anwendungsfall im Aktivitätsdiagramm.

- Kunde kauft Ware fürs Geschäft.
- Angreifertyp irrelevant
  - Keine Kommunikationsstruktur ist spezifiziert
- Wie kann fair exchange erzwungen werden?
- Anforderung <<fair exchange>> kann aus den Aktivitäten im Diagramm abgeleitet werden.

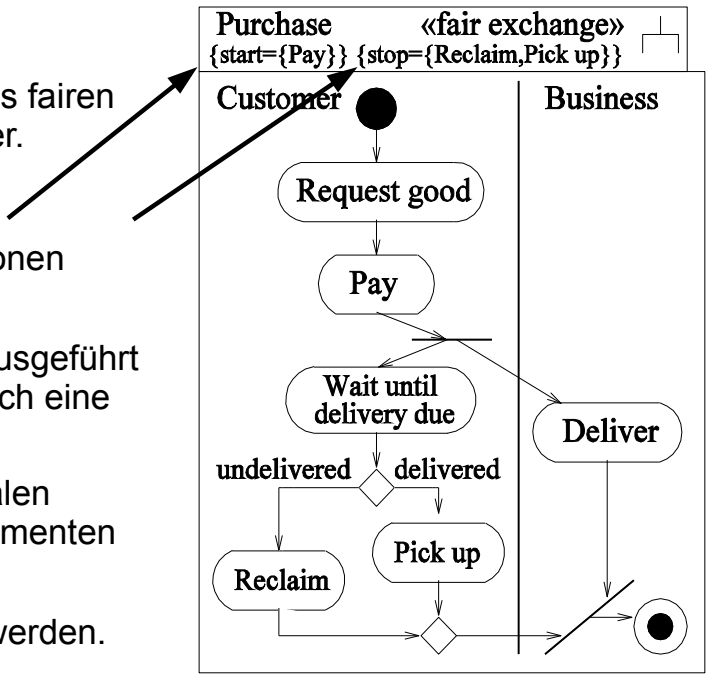


Stereotyp
<<fair exchange>>

Stellt generische Bedingung eines fairen Austauschs (fair exchange) sicher.

Bedingung:

- in {start}, {stop} gelistete Aktionen sollten verbunden sein
  - Wenn eine der früheren ausgeführt wird, dann irgendwann auch eine der späteren.
- Ist formalisiert bzgl. der formalen Semantik von benutzten Fragmenten aus UML.
- Kann automatisch überprüft werden.

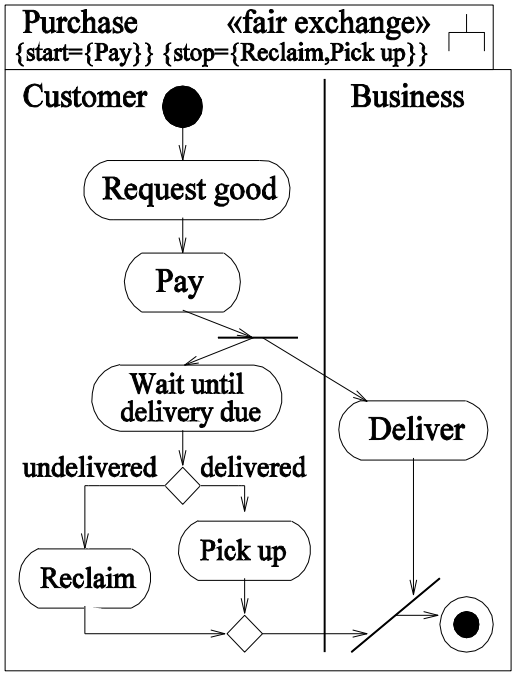


Formal für ein gegebenes Subsystem S:

- S erfüllt die Bedingung von <<fair exchange>> für einen Angreifertyp A wenn für jede Ware, die verkauft wurde, folgende Bedingung eingehalten wird:
  - Für jede Ausführung e von $[[S]]_A$ existiert ein $n \in N$,sodass für jede Sequenz $I_1,.....,I_n$ einer Eingabe von Multimengen eine Ausführung e' existiert, welche eine Erweiterung von e ist und dann die Eingaben von $I_1,.....,I_n$, hinsichtlich der relevanten Ware so verarbeitet, dass mindestens so viele {stop} Zustände in e' sind, wie {start} Zustände in e.

Beispiel
<<fair exchange>>

<<fair exchange>> erfüllt?

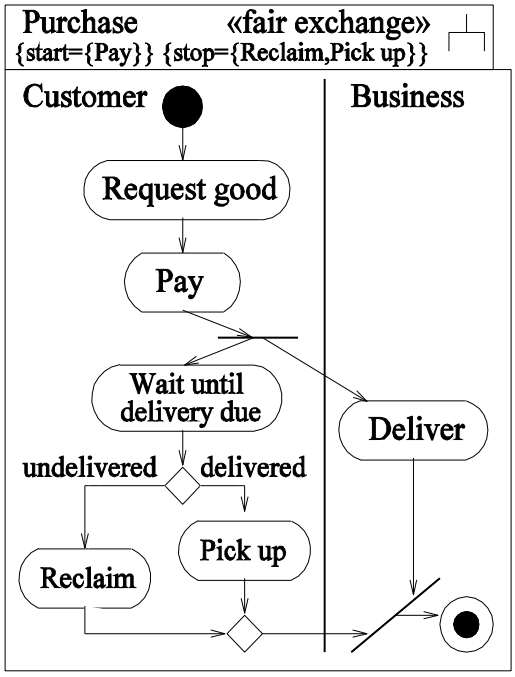


Beispiel
<<fair exchange>>

<<fair exchange>> erfüllt:

- Nach der Zahlung:
  - Kunde kann die Lieferung entgegen nehmen (pick up) oder die Bestellung stornieren (reclaim).

Kann nicht für Systeme sichergestellt werden, die ein Angreifer komplett stoppen kann.



Sicherheitsanforderungen: Non-repudiation

- Ein Weg um <<fair exchange>> zu realisieren, ist es die Sicherheitsanforderung Nichtabstreitbarkeit (**non-repudiation**) für einige Aktionen zu verwenden, was bedeutet, dass diese Aktionen nicht nachträglich verweigert werden können.
- Das heißt, die Aktion ist nachweisbar (**provable**).

Provable

Ein Subsystem S kann mit <<provable>> markiert werden.

Tags: {action},{cert} und {adversary}.

- {cert} enthält einen Ausdruck
  - d.h. einen Nachweis, dass die Aktion in dem Zustand in {action} ausgeführt wurde.
- {adversary} spezifiziert einen Angreifertyp, welchen die Sicherheitsanforderungen standhalten sollten.

S gibt womöglich einen Ausdruck $E \in Exp$ in {cert} aus, aber nur dann wenn der Zustand in {action} erreicht ist und er in Anwesenheit eines in {adversary} spezifizierten Angreifers vom Typ A ausgeführt worden ist.

- Das Zertifikat in {cert} ist eindeutig für jede Subsystem-Instanz.

Provable

Formaler: S erfüllt die Bedingung, wenn folgendes für Angreifertyp A eingehalten wird:

for (Ausführung e von $[[S]]_A$) {

if (Ausdruck in {cert} gibt einen Zustand S in e aus)
then{ Zustand in {action} erscheint als gegenwärtiger Zustand vor S in e.

}

}

Um illegitime Rückzahlforderungen im <<fair exchange>> Beispiel zu vermeiden:

- verwende <<provable>> hinsichtlich Zustand Pay.
- Stelle sicher, dass die Rückzahlaktion überprüft, ob der Kunde die Zahlung belegen kann.

- Verwendbar auf Subsystemen, die Aktivitätsdiagramme enthalten.
- Erzwingt rollenbasierte Zugriffskontrolle im durch das Aktivitätsdiagramm spezifizierten Geschäftsprozess.

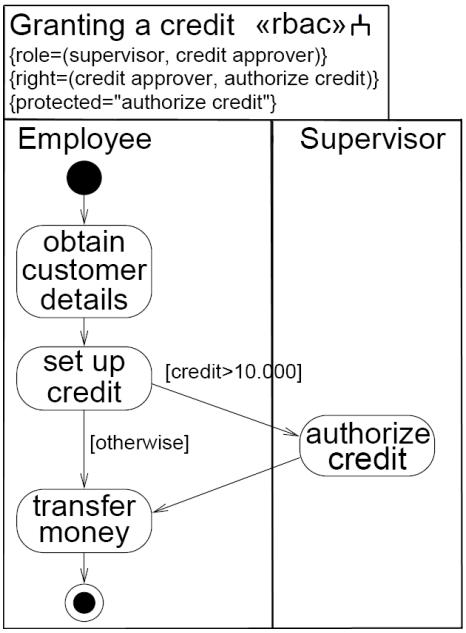
Tags: {protected}, {role}, und {right}.

- {protected} enthält Zustände im Aktivitätsdiagramm, bei denen der Zugriff kontrolliert werden soll.
- {role} Liste von Paaren (actor; role)
  - actor ist ein Akteur im Aktivitätsdiagramm, role ist eine Rolle.
- {right} Liste von Paaren (role; right)
  - role ist eine Rolle.
  - right repräsentiert das Zugriffsrecht zu einer geschützten Ressource.

Setzt voraus, dass Akteure (actors) im Aktivitätsdiagramm nur die Aktivitäten ausführen für die sie die Rechte (rights) haben.

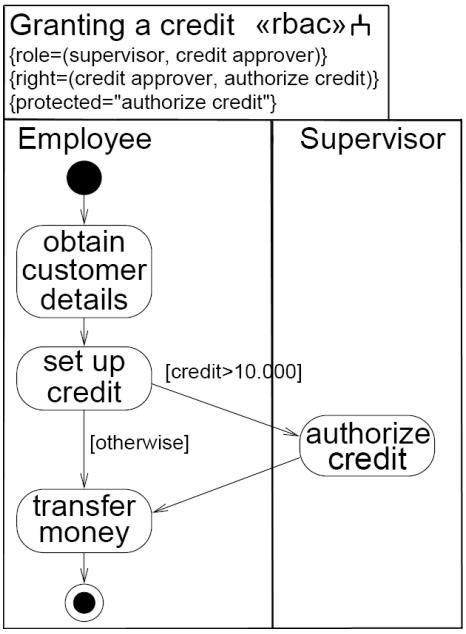
Für ein Subsystem S ist dies wie folgt festgelegt:

- Für jeden Akteur A in S und jede Aktivität a in einer Swimlane von A im Aktivitätsdiagramm in S gibt es eine Rolle R, sodass (A;R) ein Wert von {role} und (R; a) ein Wert von {right} ist.



Beispiel: Rollenbasierte Zugriffskontrolle (<<rbac>>)

- Vereinfachter Teil eines Geschäftsprozesses
  - Ein Kredit wird für einen Kunden einer Bank angelegt
- Die Bankmitarbeiter haben das Recht Kredite anzulegen.
- Für große Kredite e.g. > 10.000, muss der Vorgesetzte den Kredit authorisieren bevor Geld überwiesen wird.
- Geschützte Ressource: authorize credit Aktivität
  - Ein Vorgesetzter (supervisor) in der Rolle eines Kreditbeauftragten (credit approver) hat Sonderrechte
- Das Diagramm wird mit <<rbac>> markiert, wenn die dazugehörigen Bedingungen erfüllt sind.



Beispiel: Rollenbasierte Zugriffskontrolle (<<rbac>>)

- Beispiel: Instanz des Sicherheitsprinzip geteilter Privilegien.
- Stellt sicher, dass einem Mitarbeiter nicht zwei Rollen mit entsprechenden Privilegien zugewiesen werden, die eigentlich getrennt sein sollen.
- Zugriffskontrolle kann man mit Hilfe des Stereotyps <<guarded access>> auf der technischen Sicherheitsarchitekturebene darstellen.

Kommunikationsarchitektur (1)

- Internet, verschlüsselt, LAN, kabelgebunden, Smartcard, POS Geräte
  - Verbindungen (bzw. Knoten) im Verteilungsdiagramm (deployment diagram) markiere die entsprechenden Arten von Kommunikationsverbindungen (bzw. Systemknoten).
- Setzt voraus, dass jede Verbindung oder Knoten mindestens einen von diesen Stereotypen enthält.
- Für jeden Angreifertyp A, gibt es eine Funktion $Threats_A(s)$ mit

  $s \in$ {<<wire>>; <<encrypted>>; <<LAN>>; <<smart card>>; <<POS device>>; << issuer node>>; <<Internet>>}

  und eine Menge von Strings

  $Threats_A(s) \subseteq$ {delete; read; insert; access}:
  - Knotenstereotyp s: $Threats_A(s) \subseteq$ {access}
  - Verbindungsstereotyp s: $Threats_A(s) \subseteq$ {delete; read; insert}.

$Threats_A(s)$ gibt an welche mögliche Aktionen der Angreifer vom Typ A auf Knoten oder Verbindungen vom Stereotyp s hat.

Gegeben UML Subsystem S, Funktion $Threats_A(s)$:

- $threats^A_A(x)$
  - Nimmt einen Knoten oder Verbindung x und einen Angreifertyp A
  - Gibt eine Menge von Strings zurück, $threats^A_A(x) \subseteq$ {delete; read; insert; access}.

Wertet unter Verwendung der Ausführungssemantik und des Sicherheitsframeworks UML Machine Systems UML Subsyteme aus.

Beispiele für Bedrohungen mit häufigen Angreifertypen sind der Standard-Angreifer (default attacker) und der Insider-Angreifer (insider attacker).

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

- Ausführungssemantik: Sect. 3.3.2
- UML Machine Systems: Sect. 3.3.4

Standard-Angreifer: Angreifer mit moderaten Kapazitäten.

Fähigkeiten:

- Auf einer Internetverbindung: read, delete, and insert Nachrichten.
- Auf einer verschlüsselten Internetverbindung, (z.B. ein Virtual Private Network):
  - Löschen von Nachrichten ohne deren verschlüsselten Inhalt zu kennen, in dem er einen Netzwerkserver attackiert.
  - **Nicht** in der Lage den Klartext zu lesen oder Nachrichten mit der richtigen Verschlüsselung einzuschleusen.
- Annahme: Verschlüsselung ist so konzipiert, dass der Angreifer nicht den geheimen Schlüssel gelangen kann.
- Kein direkter Zugriff zum lokalen Netzwerk (LAN) und dadurch nicht in der Lage an lokale Verbindungen oder an sicherheitskritische Geräte zu gelangen.
- Annahme: Smart cards sind fälschungssicher.
  - Womöglich nicht gegen erfahrene Angreifer.
- Kein Zugriff auf POS Geräte oder (Kredit-) Kartenlesesysteme.

- Für Angreiferart A, Stereotyp s, gibt es eine Menge $Threats_A(s) \subseteq$ {delete, read, insert, access} von Aktionen, die der Angreifer ausführen kann.
- Standard-Angreifer: hat die Möglichkeit read, delete, insert und access Nachrichten auf einer Internetverbindung auszuführen.

Standard-Angreifer:

Stereotyp	$Threats_{default}()$
Internet	{delete, read, insert}
encrypted	{delete}
LAN	∅
smart card	∅

- Insider-Angreifer, im Kontext vom elektronische Geldbörsen-System.
- Erhält möglicherweise Zugriff auf die verschlüsselte Internetverbindung.
  - Kennt die entsprechenden Schlüssel und lokalen Systemkomponenten

Stereotype	$\mathsf{Threats}_{insider}()$
Internet	{delete, read, insert}
encrypted	{delete, read, insert}
LAN	{delete, read, insert}
wire	{delete, read, insert}
smart card	$\emptyset$
POS device	$\emptyset$
issuer node	{access}

Elektronisches Geldbörsen-System:
Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 5.3

Abhängigkeiten
<<secrecy>>, <<integrity>>, <<high>>

- Wird bei Abhängigkeiten in statischen Sturkturdiagrammen und Komponentendiagrammen verwendet.
- Markiert Abhängigkeiten, die für Daten bestimmte Sicherheitsanforderungen garantieren, darunter Argumente, Rückgabewerte von Operationen oder Signale.
- Wird in Bedingungen des Stereotyps <<secure links>> verwendet.

Kritische Daten <<critical>>

- Markiert Objekte oder Subsystem-Instanzen, die kritische Daten enthalten
- Tags: {secrecy}, {integrity}, {authenticity}, {fresh}, und {high}, die die entsprechenden Sicherheitsanforderungen repräsentieren.
- {secrecy} markiert Ausdrücke, Attribute oder Argumentvariablen von Objekten deren Geheimhaltung sichergestellt werden soll; markiert Operationen, bei denen sichergestellt werden soll, dass deren Argumente und Rückgabewerte geheim gehalten werden sollen.
- {integrity} hat als Wert Paare (v;E)
  - v: Variable eines Objekts, bei welchem die Integrität geschützt werden soll.
  - E: Menge von in Verbindung zu v stehenden Ausdrücken, die akzeptiert werden

Sicherheitsanforderungen:
Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.1 and 3.3

Kritische Daten

- {authenticity} enthält Paare (a; o) von Attributen des <<critical>> Objekts oder Subsystems
  - a enthält die Daten, bei denen die Authentizität gewährleistet werden soll und
  - o enthält die Herkunft der Daten.
- {fresh} atomare Daten (Elemente der Menge Data ∪ Keys) die neu generiert werden sollten.
- Diese Bedingungen werden von der Bedingung des Stereotyps <<data security>> erzwungen, welche die Subsysteme, die <<critical>> Objekte enthalten, markiert. (s.u.)
- {high} markiert zu schützende Nachrichten, hinsichtlich des sicheren Informationsflusses, erzwungen durch <<no down-flow>> und <<no up-flow>>.
- Synchrone Operationen: Rücknachrichten müssen geschützt werden.

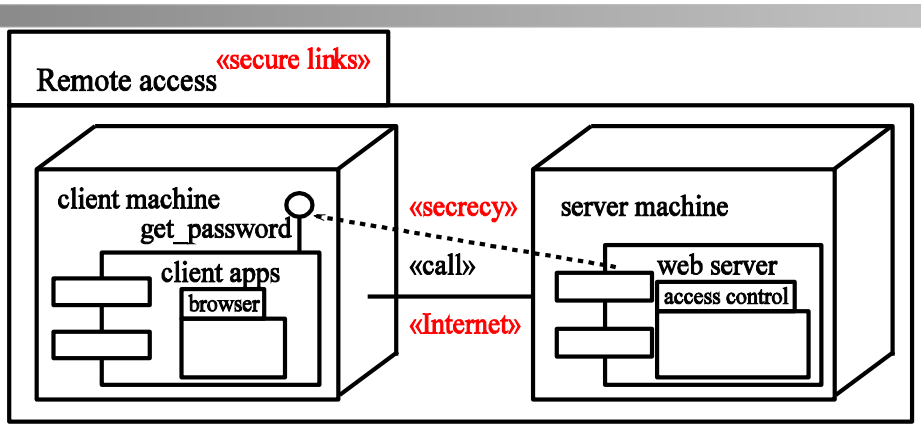
Sichere Kommunikation

- Zusammen mit den Stereotypen <<secrecy>>, <<integrity>>, <<high>>, and <<critical>> kann man verschiedene Bedingungen beschreiben, um mit den folgenden Stereotypen die sichere Übermittlung von Daten sicherzustellen:
  - <<secure links>>
  - <<secure dependencies>>
  - <<data security>>

Sichere Kommunikation

- <<secure links>>
  - Stellt sicher, dass unter Beachtung des Angreifermodells die Sicherheitsanforderungen bei der Kommunikation zwischen den Komponenten von der physikalischen Ebene unterstützt werden.
- <<secure dependencies>>
  - Stellt sicher, dass die Sicherheitsanforderungen in den verschiedenen Teilen des statischen Strukturdiagramms konsistent sind.
- <<data security>>
  - Stellt sicher, dass die Sicherheit auf der Verhaltensebene gegeben ist.
- Man könnte z.B. die Bedingungen von <<secure links>> und <<secure dependencies>> zu einem Stereotyp verschmelzen.

Sicherheit auf der Architektur Ebene: Beispiel



- Geschäftsanwendung: Teil eines E-Commerce Systems
- Soll als Webanwendung realisiert werden.
- Zahlungstransaktionen beinhalten die Übermittlung von geheimen Daten über die Internetverbindung.
- <<secure links>> fordert, dass Sicherheitsanforderungen bei der Kommunikation von der physikalischen Ebene eingehalten werden.
- Ist die Architektur sicher gegen Standard-Angreifer?

Stereotyp <<secure links>>

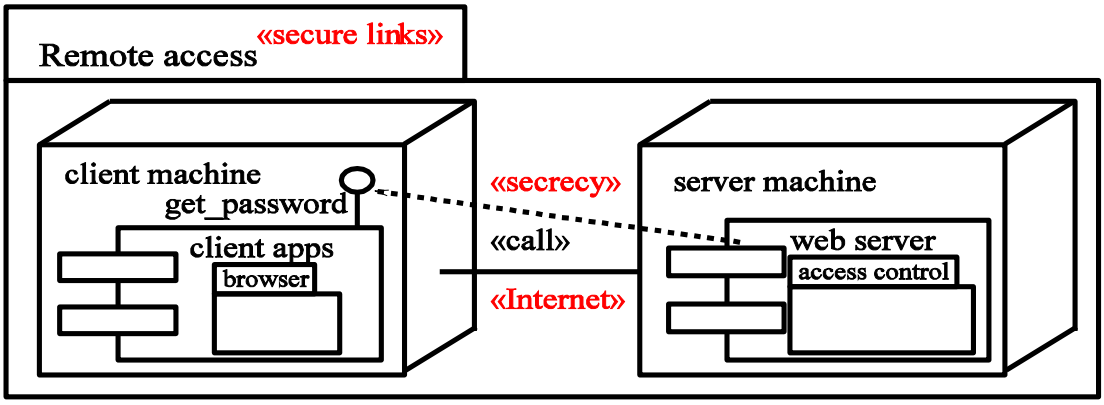
- Zur Erinnerung: $threats_A^S(x) \subseteq \{delete; read; insert; access\}$ mit UML Subsystem S, Knoten oder Verbindung x und Angreifer A.
- Markiert Subsysteme die statische Stukturdiagramme enthalten.
- Stellt sicher, dass die physikalische Ebene Sicherheitsbestimmungen bei der Kommunikation einhält.
- **Bedingung**: Für jede Abhängigkeit *d* mit Stereotyp *s* ∈ {<<secrecy>>, <<integrity>>, <<high>>} zwischen Komponenten auf Knoten *n*≠*m* verläuft Kommunikation über eine Kommunikationsverbindung *l* zwischen *n* und *m* mit Stereotyp *t* sodass:
- Wenn s = <<high>> , dann $threats_A^S(t) = \emptyset$
- Wenn *s* = <<secrecy>>, dann read ∉Threats *(t)*.
- Wenn *s* = <<integrity>>, dann insert ∉Threats *(t)*.

Literatur:

Jan Jürjens: Secure systems development with UML

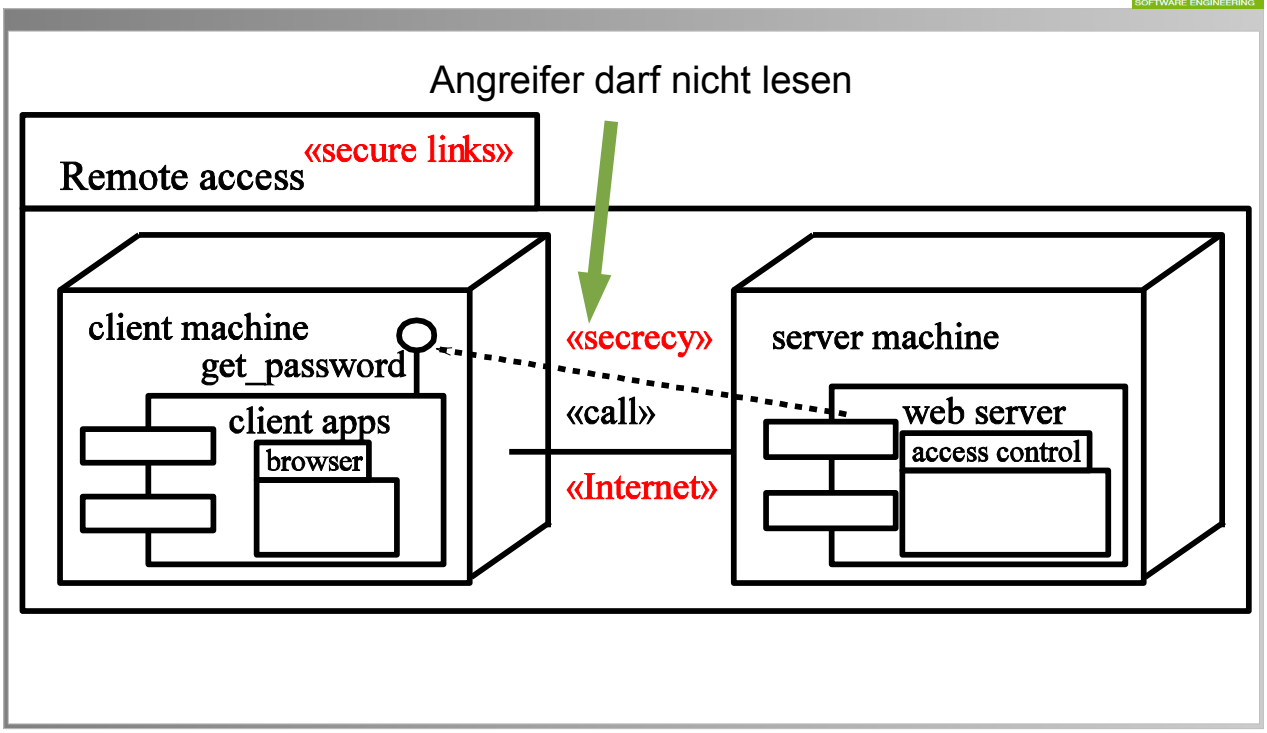
- Kap. 4.1: S. 59

Beispiel
<<secure links>>



Sind die Bedingungen für den Stereotyp <<secure links>> bei einem Standard-Angreifertyp erfüllt?

Beispiel
<<secure links>>

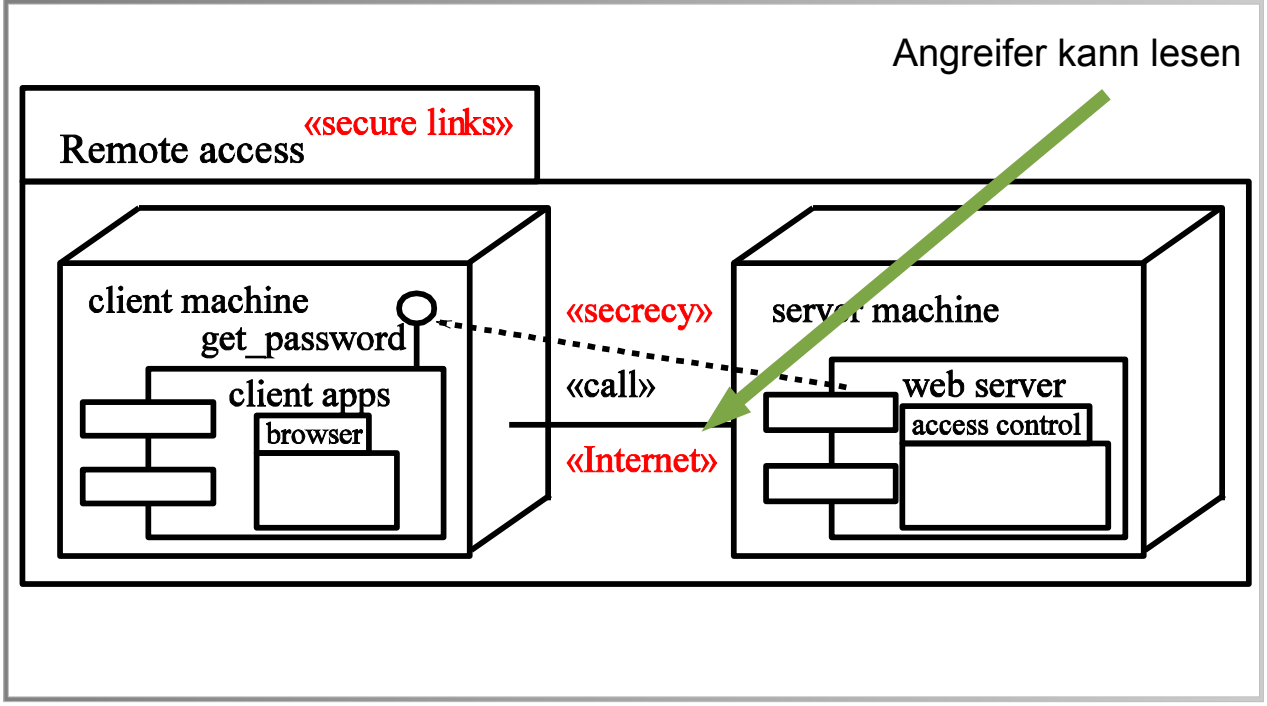


Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 60 Abb. 4.8

Beispiel
<<secure links>>

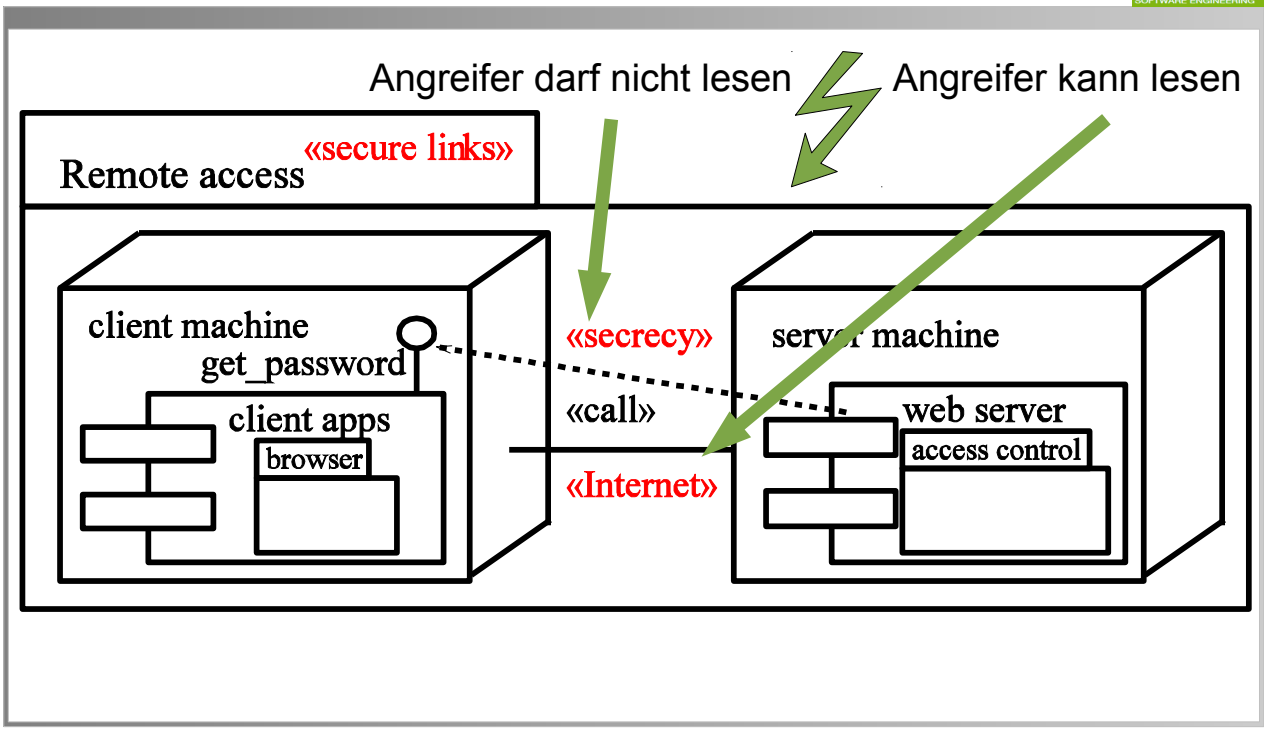


Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 60 Abb. 4.8

Beispiel
<<secure links>>

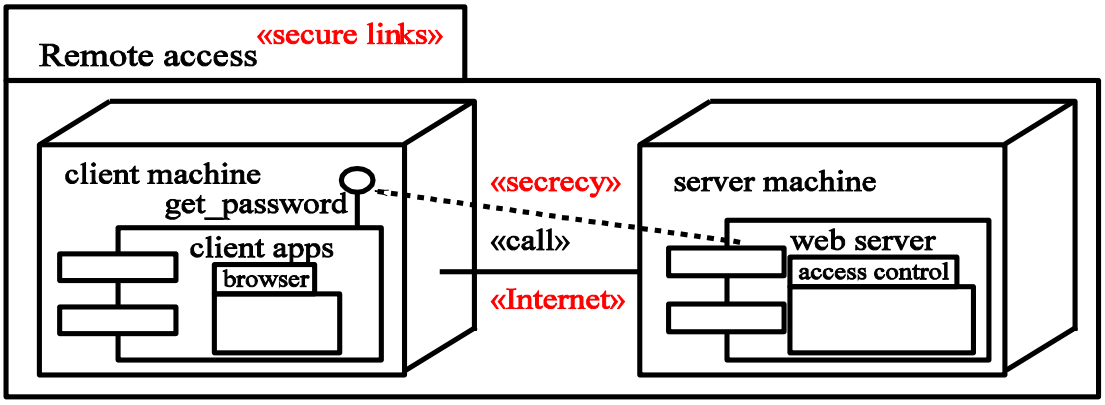


Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 60 Abb. 4.8

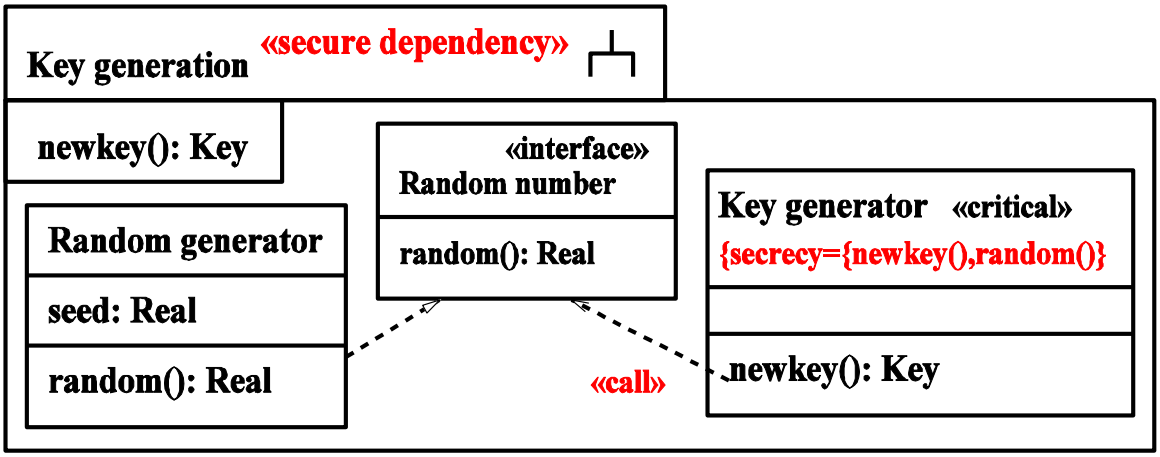
Beispiel
<<secure links>>



Sind die Bedingungen für den Stereotyp <<secure links>> bei einem Standard-Angreifertyp erfüllt?

- Intuitiv: Internet Verbindungen stellen keine Geheimhaltung gegenüber Standard-Angreifern zur Verfügung.
- Technisch: Bedingung ist verletzt, weil die Abhängigkeit mit dem Stereotyp <<secrecy>> markiert ist, aber für <<Internet>> der entsprechenden Verbindung ist read $\in$ $threats_{default}$(Internet)!

Sicherheit auf Ebene der Klassenstruktur: Beispiel



Sind die Sicherheitsmarkierungen im Klassendiagramm konsistent?

Stereotyp <<secure dependency>>

- Kennzeichnet Subsystem welches statische Strukturdiagramme beinhaltet
- Stellt sicher, dass <<call>>- und <<send>>-Abhängigkeiten zwischen Komponenten Sicherheitsanforderungen für versendete Daten **respektieren**.

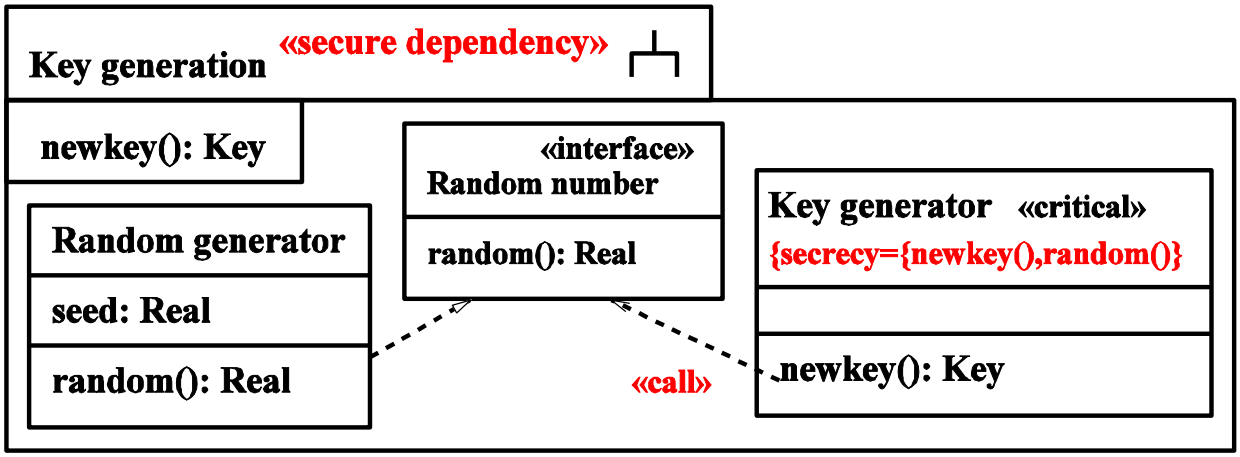
  → Mit Tags {secrecy}, {integrity} vom Stereotyp <<critical>>.
- Genauer: für <<call>>- oder <<send>>-Abhängigkeiten zwischen einem Objekt oder Subsystem *C* und einem Interface *I* eines Objekts oder Subystems *D* müssen folgende Bedingungen erfüllt sein:
  - Nachricht in *I* ist mit {secrecy} (bzw. mit {integrity} oder {high}) in *C* markiert gdw. sie auch in *D* markiert ist.
  - Wenn Nachricht in I als {secrecy} (bzw. als {integrity} oder {high}) in *C* markiert ist, dann ist die Abhängigkeit mit <<secrecy>> (bzw. mit <<integrity>> oder <<high>>) markiert.

Literatur:

Jan Jürjens: Secure systems development with UML

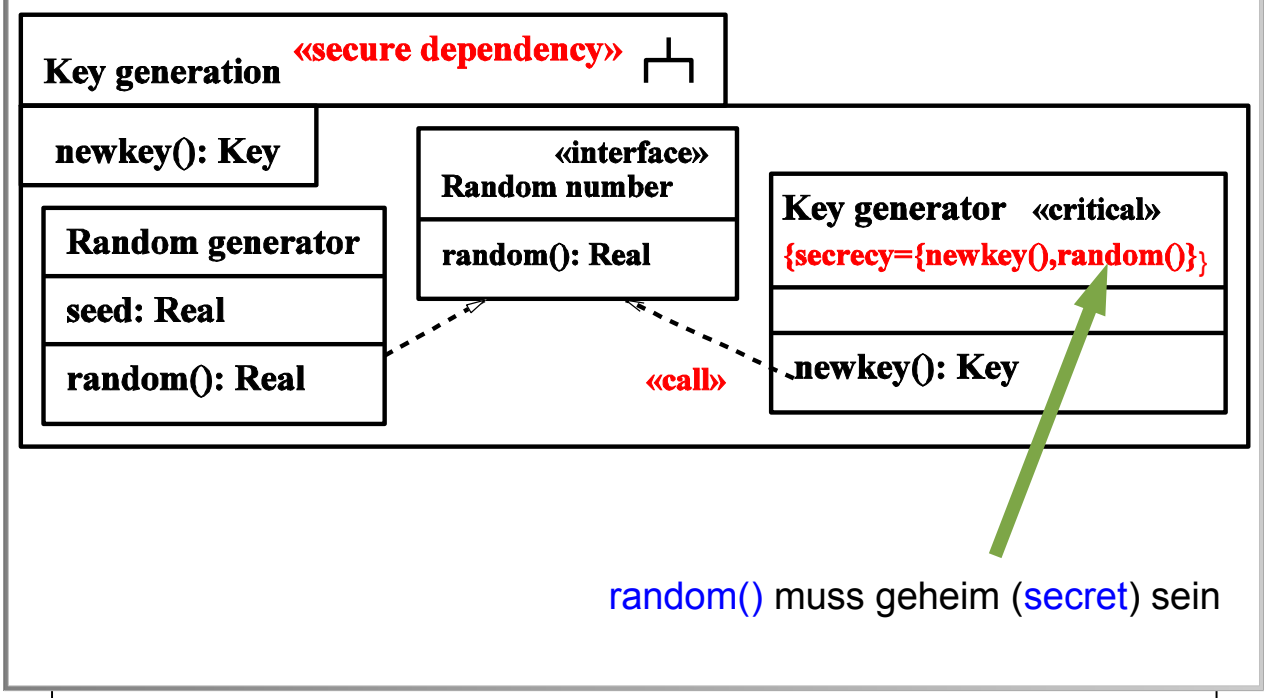
- Kap. 4.1: S. 59-60

Beispiel <<secure dependency>>



<<secure dependency>> erfüllt oder nicht?

Beispiel <<secure dependency>>



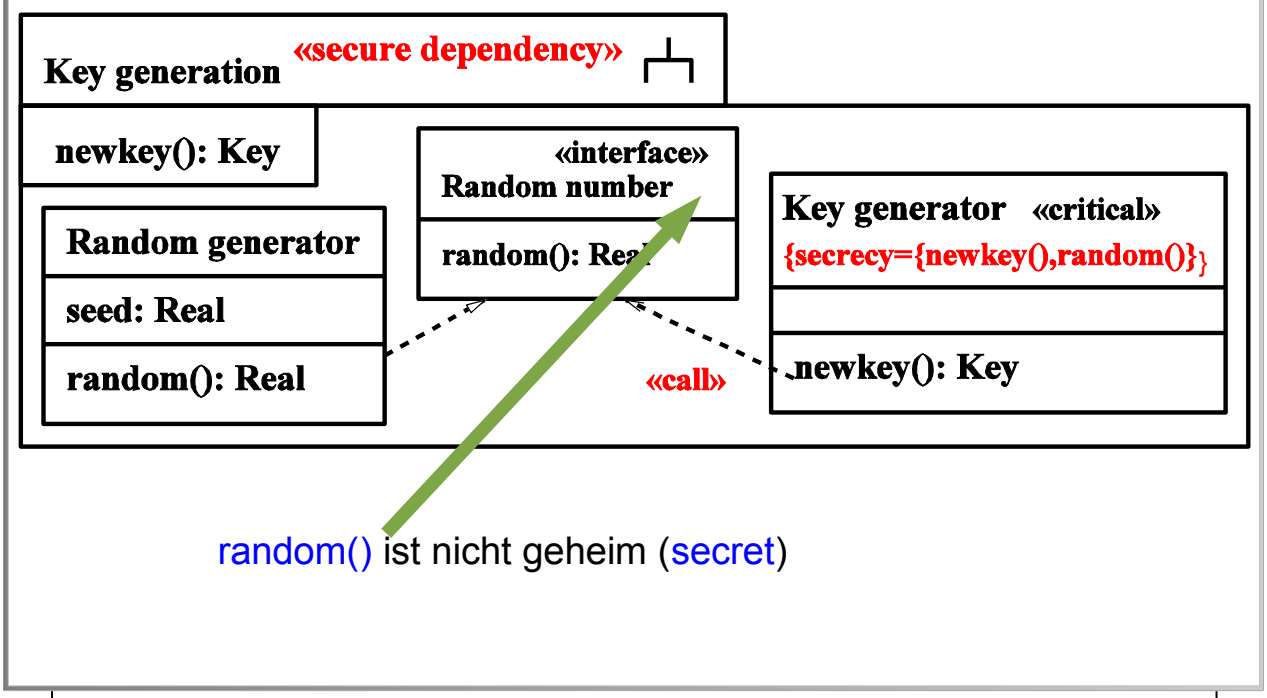
random() muss geheim (secret) sein

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 61 Abb. 4.9

Beispiel <<secure dependency>>



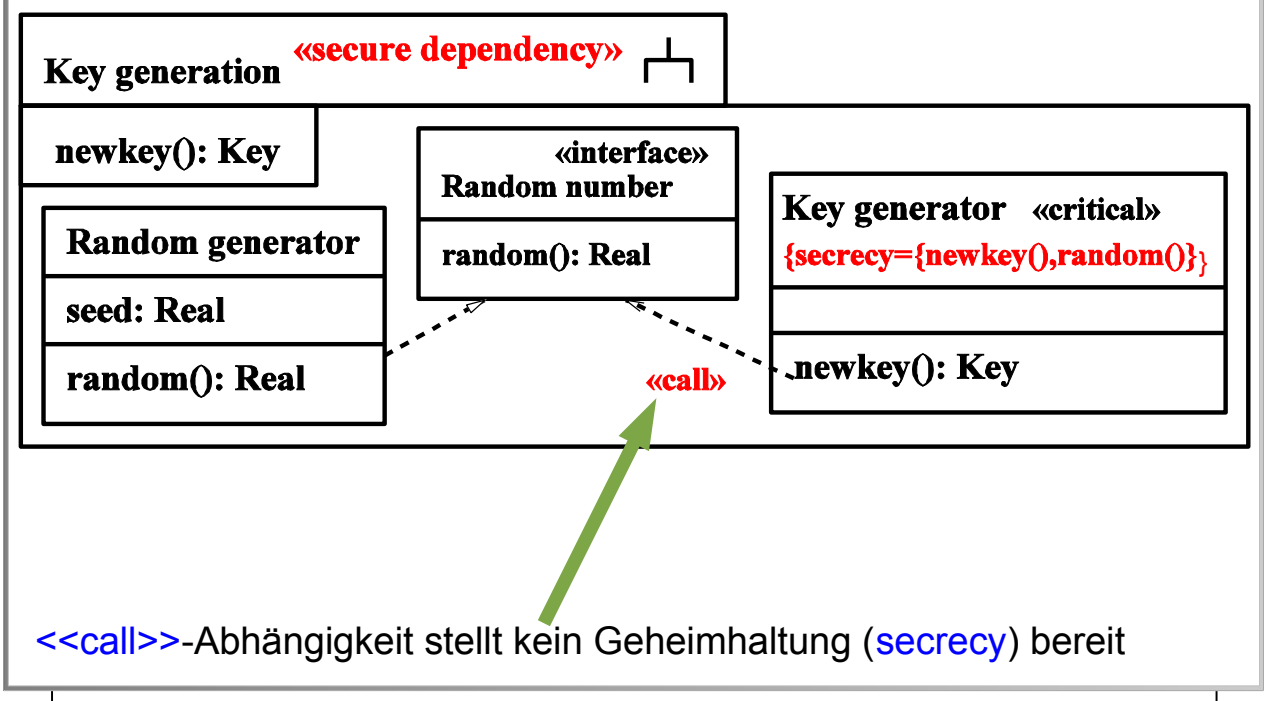
random() ist nicht geheim (secret)

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 61 Abb. 4.9

Beispiel <<secure dependency>>



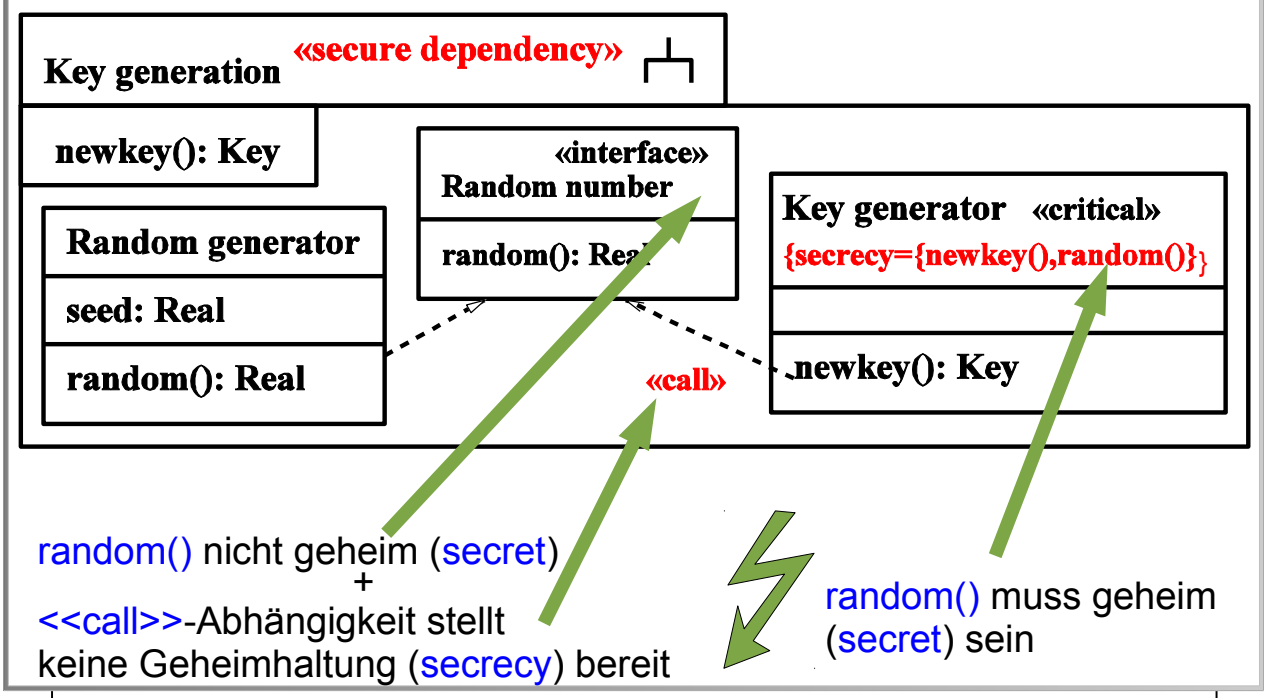
<<call>>-Abhängigkeit stellt kein Geheimhaltung (secrecy) bereit

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 61 Abb. 4.9

Beispiel <<secure dependency>>

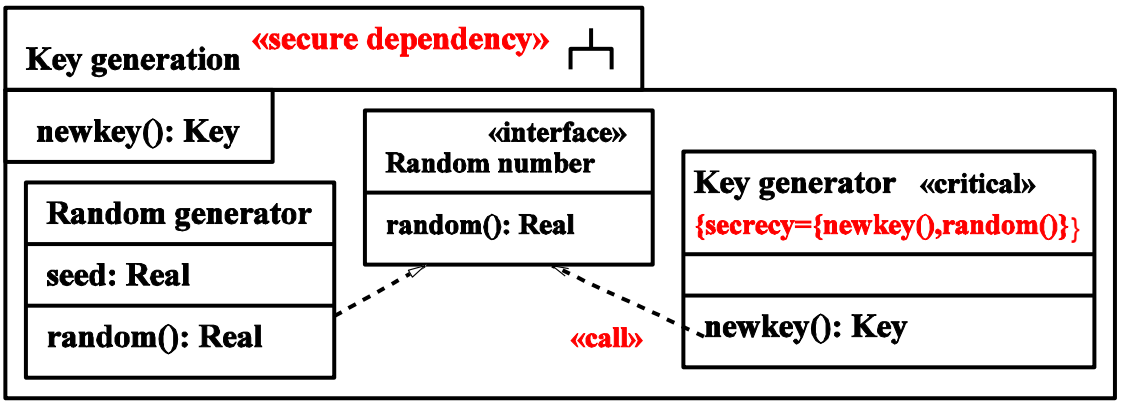


Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 61 Abb. 4.9

Beispiel <<secure dependency>>



Verletzt <<secure dependency>>:

- Random generator und <<call>>-Abhängigkeit bieten keine Sicherheitseigenschaft {secrecy} für random(), die vom Key generator benötigt wird.

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 61 Abb. 4.9; S. 60 vorletzter Abschnitt (Example[...])

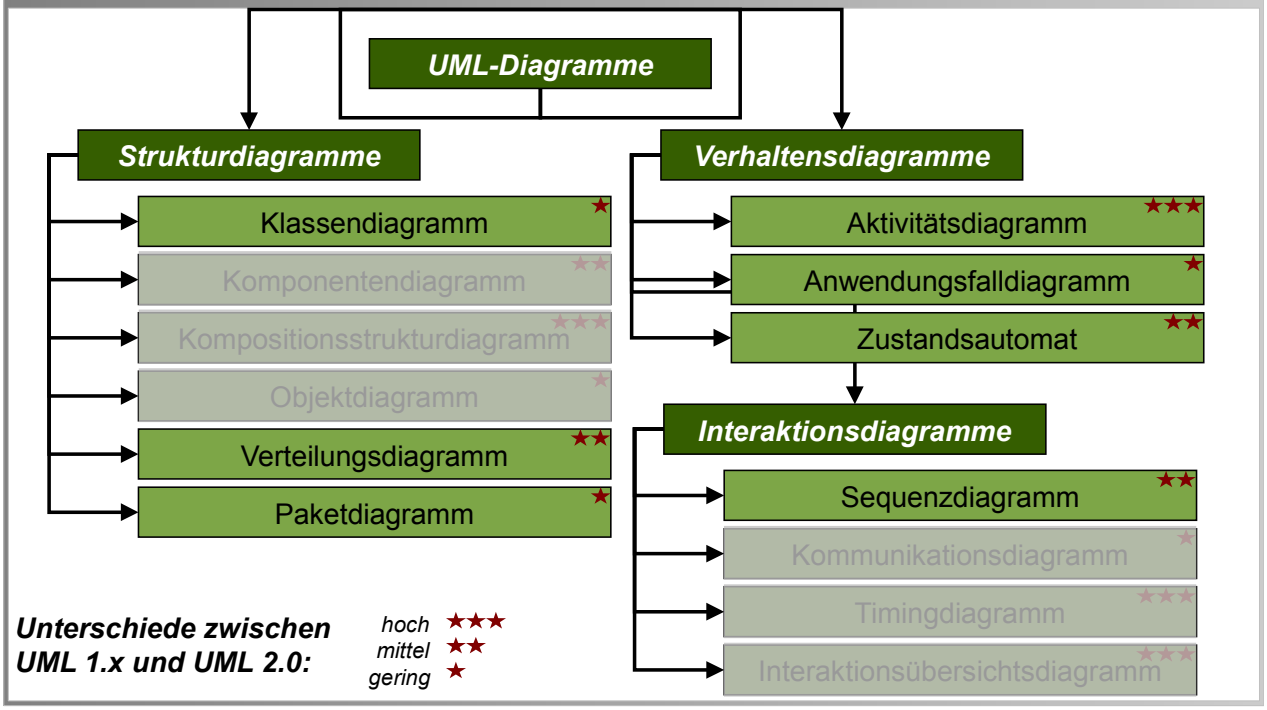
Zusammenfassung

- Idee hinter UMLsec
- Übersicht UMLsec Stereotypen
- Fair exchange
- <<secure links>>
- <<secure dependency>>

Zum selbstständigen Wiederholen, vgl.
http://www-secse.cs.tu-dortmund.de/secse/pages/teaching/ws13-14/swk/index_de.shtml

- Teil 1.0: Modellbasierte Softwareentwicklung: Einführung
- Teil 1.2: Modellbasierte Softwareentwicklung

UMLsec-relevante Diagrammtypen



UMLsec-relevante Diagrammtypen

- **Anwendungsfalldiagramm**: Enthält Anforderungen an das System.
- **Klassendiagramm:** Datenstruktur des Systems.
- **Zustandsdiagramm:** Dynamisches Verhalten der Komponenten.
- **Aktivitätsdiagramm:** Steuerung des Ablaufs zwi. Komponenten.
- **Sequenzdiagramm**: Interaktion durch Nachrichtenaustausch.
- **Verteilungsdiagramm:** Physikalische Umgebung.
- **Paket/Subsystem:** Fasst Diagramme eines Systemteils zusammen.

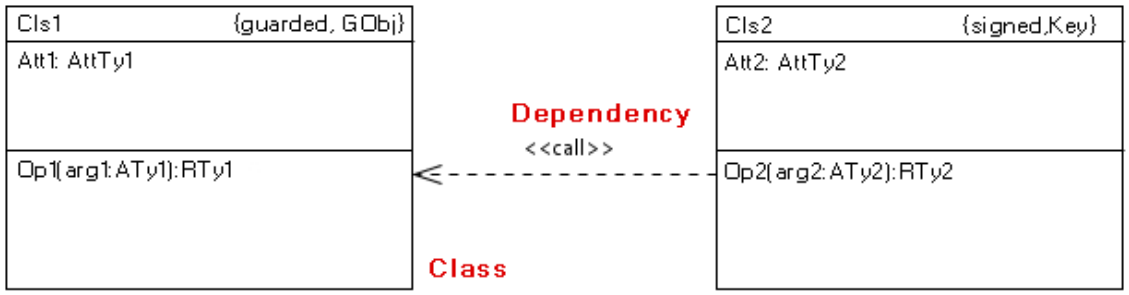
UMLsec Anmerkungen

- **UML2** Vorstellung auf UMLsec relevante Diagramme beschränkt.
- **UMLsec** auf UML 1.5 definiert:
  - Beispiele von UML 2 abweichbar.
- UMLsec zum Großteil auf UML 2 angehoben (Siehe UMLSec4UML2 Profil)
- Um Verwirrung vorzubeugen alle Beispiele in UML 1.5

UMLsec Beispiel
Klassendiagramm

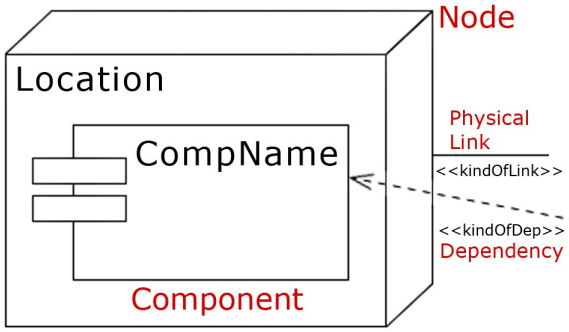
Klassenstruktur des Systems.

Klasse mit Attributen und Operation / Signalen, Beziehungen zwischen Klassen.



- **Problem**:
  - **Verteilung** der Software des Systems auf die Hardware beschreiben.
- Zentrale Frage des **Diagramms**:
  - Wie sieht das **Einsatzumfeld** (Hardware, Server, Datenbanken etc.) des Systems aus?
  - Wie werden **Komponenten** zur Laufzeit wohin verteilt?

UMLsec Beispiel Verteilungsdiagramm

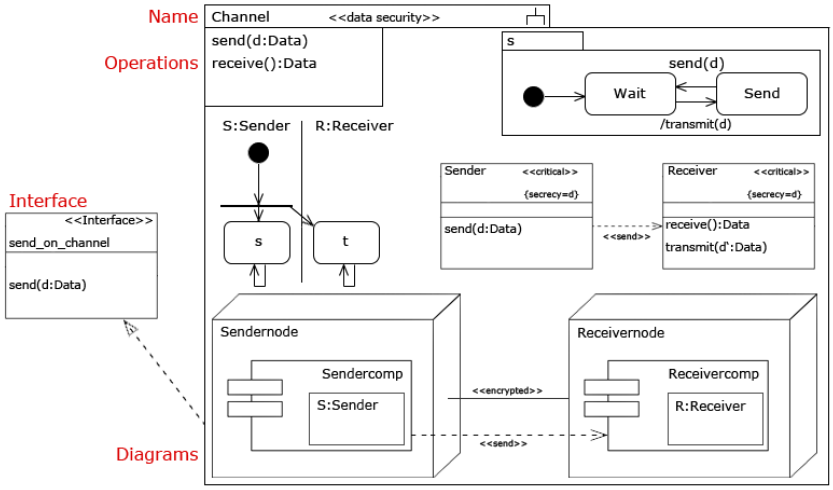


Erklärt **physikalische Ebene**, auf der das System implementiert wird.

UML im Schnelldurchlauf
Paketdiagramm

- **Problem**:
  - Große Softwaresysteme mit mehreren 100 Klassen nicht von einer Person überblickbar.
- Diese zentrale Frage beantwortet das **Diagramm**:
  - Wie kann ich mein Modell so schneiden, dass ich den Überblick bewahre?
- **Stärken** des Diagramms:
  - organisiert Systemmodell in größeren Einheiten durch logische Zusammenfassung von Modellelementen.
  - Modellierung von Abhängigkeiten und Inklusionen möglich.

UMLsec Beispiel
Paketdiagramm



Verwendbar, um **UML Element** zu einer **Gruppe** zusammen zu fügen.

UML im Schnelldurchlauf
Aktivitätsdiagramm

- **Problem**:
  - Abläufe, z.B. Geschäftsprozesse, modellieren.
  - Im Vordergrund: Aufgabe in Einzelschritte zerlegen.
  - Details eines Anwendungsfalles festgelegen.
- Diese zentrale Frage beantwortet das **Diagramm**:
  - Wie realisiert mein System ein bestimmtes Verhalten?

Änderungen gegenüber früheren UML-Versionen

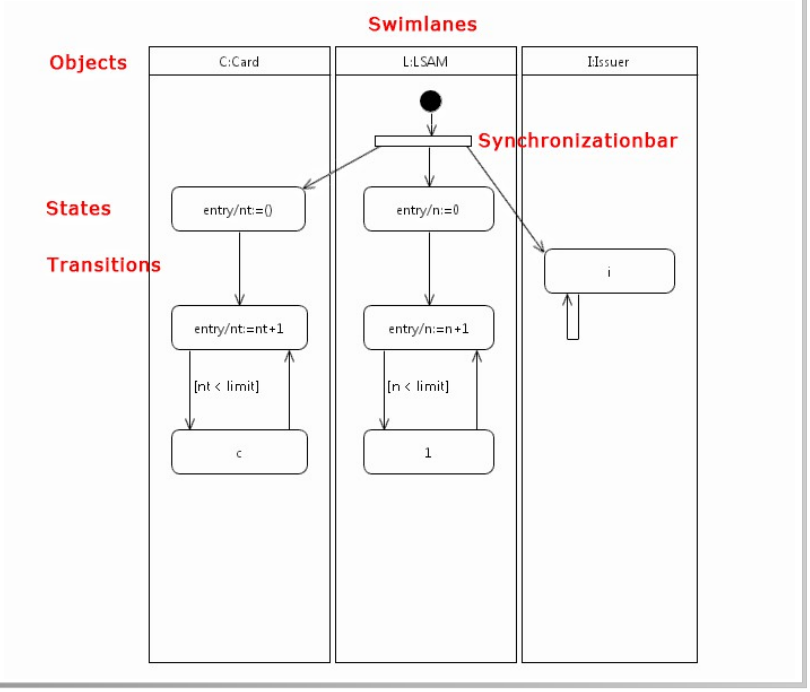
- Keine Sonderform der Zustandsdiagramme, basieren auf **erweiterten Petri-Netzen.**
- Viele Einschränkungen beseitigt:
  - Verbesserte Testbarkeit.
  - Fast automatisch erkennbare **Verklemmungsfreiheit.**
  - Bessere Unterstützung **paralleler Flüsse.**
  - **Ausführbarkeit** fast vollständig möglich.
  - Mehr **Flexibilität** Modellierung durch **Tokenkonzept.**

Tokenkonzept

- Ein Token (auch: Marke) zeigt an, an welchem Punkt sich der Ablauf befindet.
- Es können beliebig viele Token unterwegs sein (parallele Abläufe).
- Token: Keine graphische Darstellung, dienen der Erklärung der Abläufe.

UMLsec Beispiel Aktivitätsdiagramm

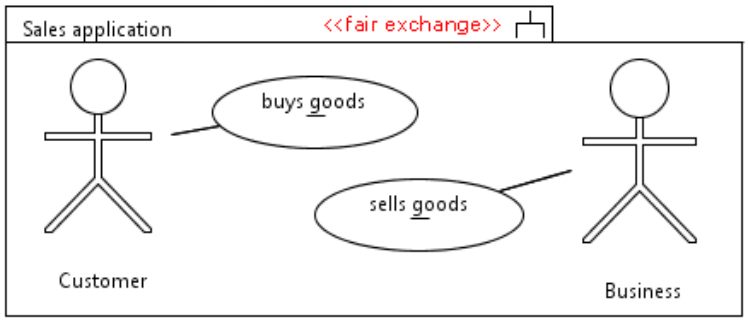
Spezifiziert **Kontrollfluss** zwi. Komponenten desselben Systems. Ist auf einer höheren Abstraktionsebene als **Zustands**- und **Sequenzdiagramme**.



UML im Schnelldurchlauf
Anwendungsfalldiagramm

- **Problem**:
  - Externes Verhalten des Systems soll aus Nutzersicht darstellen.
- Diese zentrale Frage beantwortet das **Diagramm**:
  - Was leistet mein System für seine Umwelt (Nachbarsysteme, Stakeholder)?
- **Stärken** des Diagramms:
  - Präsentiert Außensicht auf das System.
  - Geeignet zur Kontextabgrenzung.
  - Hohes Abstraktionsniveau, einfache Notationsmittel.

UMLsec Beipspiel
Anwendungsfalldiagramm



Spezifiziert einen **Anwendungsfall** des Systems:

- **Szenario** einer Funktionalität, die einem Benutzer oder einem anderen System angeboten wird.
- **Akteure:** mit einer Aktivität verknüpft.

Beschreibung des Verhaltens von Anwendungsfällen

- Vorteile formaler Modellierung von Anwendungsfällen:
  - **Eindeutig** und weniger interpretierbar.
  - **Testfälle** ableitbar.

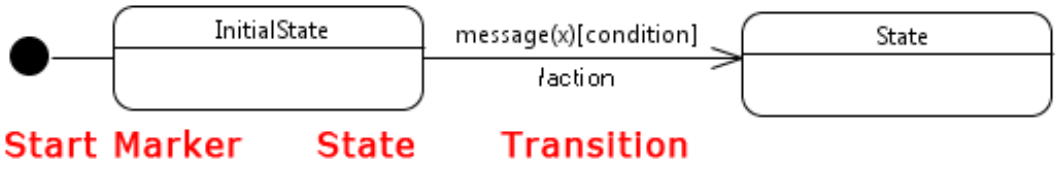
Beschreibung des Verhaltens von Klassen

- Dem Attribut einer Klasse wird ein **Datentyp** zugeordnet.
- Besitzt der Datentyp endliche Menge von gültigen Werten:
  → Verschiedene Zustände der Klasse ergibt sich aus allen möglichen **Kombinationen** dieser **Zustandsvariablen**.

UMLsec Beispiel
Zustandsautomat

Dynamisches Verhalten einzelner Komponenten.

Eingabe verursacht einen Zustandsübergang und möglicherweise eine Ausgabe.



Start Marker State Transition

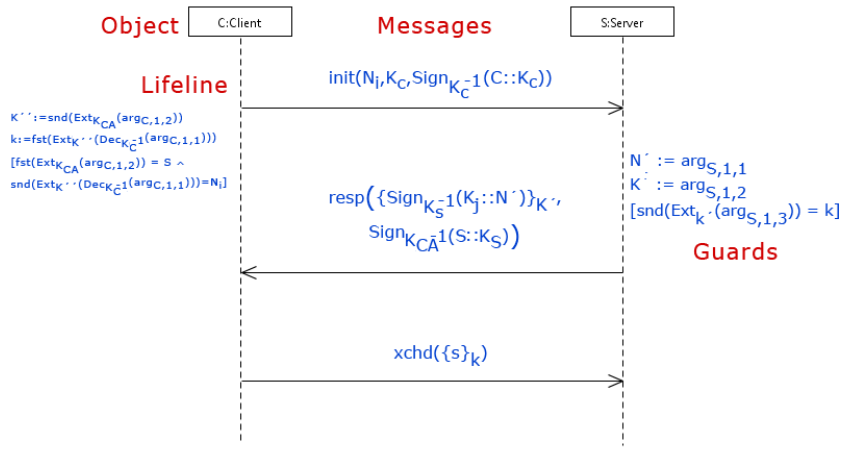
UML im Schnelldurchlauf
Sequenzdiagramm

- Diese zentrale Frage beantwortet das **Diagramm**:
  - Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus?
- **Stärken** des Diagramms:
  - stellt detailliert den Informationsaustausch zwi. Kommunikationspartnern dar
  - präzise Darstellung zeitlicher Abfolge auch mit Nebenläufigkeiten.
  - Schachtelung und Flusssteuerung (Bedingungen, Schleifen, Verzweigungen) möglich.

Häufige Anwendungsfälle

- Abfolge der Nachrichten wichtig.
- Durch Nachrichten verursachte Zustandsübergänge kaum relevant.
- Interaktionen kompliziert.
- Strukturelle Verbindung zwi. Kommunikationspartner nicht relevant.
- Ablaufdetails im Vordergrund.

UMLsec Beispiele
Sequenzdiagramm



Verdeutlicht **Interaktion** zwischen Objekten und Komponenten per **Nachrichtenaustausch**.

Alte Folien

Well-formedness Rules

- Constraints mit Stereotypen assoziiert:
  - gibt eine Auswahl von strukturell syntaktischen Bedingungen,
    - wie <<secure links>>,
  - zu relativ tiefen semantischen Bedingungen,
    - wie <<no down-flow>>.
  - Vorteile:
    - Erst Verletzung gegen einfacher strukturierte Bedingungen finden, dann Verhaltensteil der Spezifikation analysieren.
    - Automatisierte mechanische Überprüfung möglich[1]
- Effizienter als allgemeine Sicherheit auf einmal aufzubauen.
- Industrieller Bereich: erlaubt Skalierung der nötigen Kosten.

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Chap. 6.

Erfasst **Sicherheitsanforderungen** in Anwendungsfalldiagrammen.

Relevantes Stereotyp muss im dazugehörigen Aktivitätsdiagramm auftauchen.

Literatur:

Jan Jürjens: Secure systems development with UML

- Kap. 4.1: S. 53