LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2013

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

3.3: UMLsec

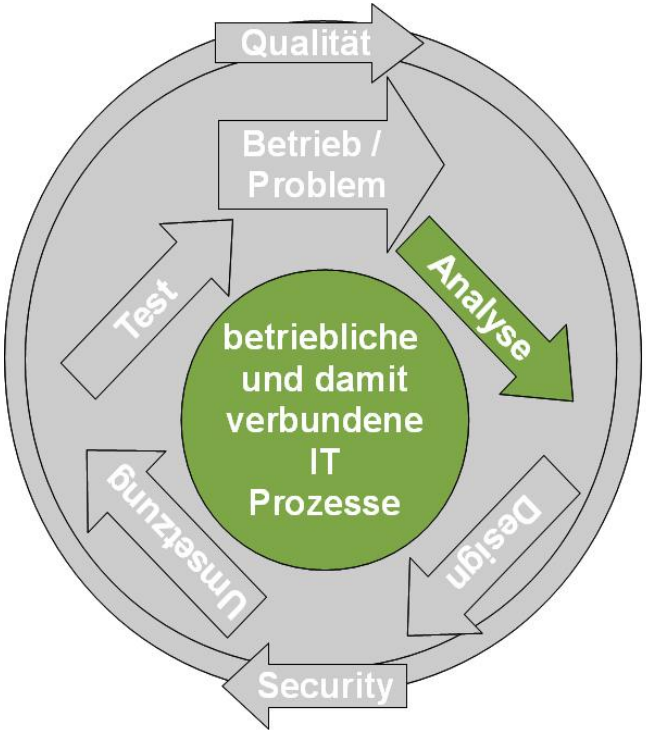
v. 26.06.2013

3.3 UMLsec

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
- Kapitel 4.1

Einordnung
3.3 UMLsec

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Model-Driven Architecture
  - Sicherheitsanforderungen
  - UMLsec
  - UML-Analysis
  - Design Principles
  - Examples
    - TLS Variant
    - CEPS Purchase



Introduction of UMLsec

- UML extension UMLsec
  - allows to express security-related information within diagrams in UML system specification.
  - in form of a UML profile using the standard UML extension mechanisms.
- Stereotypes and tags: used to formulate security requirements and assumptions on the system environment.
- Constraints
  - give criteria that determine whether the requirements are met by the system design, by referring to the execution semantics.
  - can be checked automatically using tool support[1].

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

Outline

- List requirements on a UML extension for secure systems development.
- Discuss how far our extension meets these requirements.
- Explain details of the extension by means of examples.
- Demonstrate the usefulness of the extension
  - enforcing established rules of secure systems design
  - indicate with an example how one could use UMLsec to apply security patterns.

Requirements on a UML Extension for Development of Security-Critical Systems.

- Formulate necessary properties of an UML extension for secure systems development.
  - Like the OMG Requests for Proposals (RFPs): distinguish mandatory and optional requirements.

Requirements on a UML Extension for Development of Security-Critical Systems.

Main **mandatory requirements**:

- Provide basic security requirements such as secrecy, integrity, authenticity.
- Allow considering different threat scenarios depending on adversary strengths.
- Allow including important security concepts (e.g. *tamper-resistant hardware*).
- Allow incorporating security mechanisms (e.g. access control).
- Provide security primitives (e.g. (a)symmetric encryption).
- Allow considering underlying physical security.
- Allow addressing security management (e.g. secure workflow).

The **optional requirement:**

- Include technology-specific security concepts (Java, smart cards, CORBA, …)

Requirements on a UML Extension for Development of Security-Critical Systems.

Note:

- Goal: not to aim for completeness by including all kinds of security properties as primitives.
- Focus on those that have a comparatively intuitive and universally applicable formalization, such as secrecy, integrity, and message authentication.
- Other properties, such as entity authenticity, have meanings that depend more on the context of their specific use.
  - Can be added by more sophisticated users on-the-fly.

UMLsec: General Ideas (1)

Add security-relevant information to UML model elements.

Define labels for UML model elements:

- called stereotypes.

Different stereotypes available:

- Security assumptions on the physical level of the system, such as stereotype <<Internet>>.
- Security requirements on the logical structure of the system or on specific data values, such as stereotypes <<secrecy>>, <<critical>>.
- Security policies that system parts are supposed to obey, such as stereotypes <<fair exchange>>, <<secure links>>, <<data security>>, <<no down – flow>>.

UMLsec: General Ideas (2)

- Activity diagram:
  - secure control flow, coordination
- Class diagram:
  - exchange of data preserves security levels
- Sequence diagram:
  - security-critical interaction
- Statechart diagram:
  - security preserved within object
- Deployment diagram:
  - physical security requirements
- Package:
  - holistic view on security

The Extension

Give profile following the structure in [UML03]:

- Applicable Subset: Profile concerns all of UML.
- Stereotypes, Tagged Values, and Constraints:
  - List of stereotypes from UMLsec, their tags and constraints and corresponding tags (all DataTags).
  - The stereotypes do not have parents.
  - Concepts apply both to type and instance level.
  - For simplicity focus on the instance level
  - By "subsystem" we mean, more precisely, "subsystem instance".
- UMLsec requires no prerequisite profiles.

[UML03] Object Management Group. OMG Unified Modeling Language Specification v1.5, March 2003. Version 1.5. OMG Document formal/03-03-01.

UMLsec Profile: Stereotypes

Stereotype	Base Class	Tags	Constraints	Description
fair exchange	subsystem	start, stop, adversary	after start eventually reach stop	enforce fair exchange
provable	subsystem	action, cert, adversary	action is non-deniable	non-repudiation requirement
rbac	subsystem	protected, role, right	only permitted activities executed	enforces role-based access control
Internet	link			Internet connection
encrypted	link			encrypted connection
LAN	link, node			LAN connection
wire	link			wire
smart card	node			smart card node
POS device	node			POS device
issuer node	node			issuer node
secrecy	dependency			assumes secrecy
integrity	dependency			assumes integrity
high	dependency			high sensitivity
critical	object, subsystem	secrecy, integrity, authenticity, high, fresh		critical object
secure links	subsystem	adversary	dependency security matched by links	enforces secure communication links
secure dependency	subsystem		«call», «send» respect data security	structural interaction data security
data security	subsystem	adversary, integ., auth.	provides secrecy, integrity, authenticity, freshness	basic data security requirements
no down-flow	subsystem		prevents down-flow	information flow condition
no up-flow	subsystem		prevents up-flow	information flow condition
guarded access	subsystem		guarded objects accessed through guards	access control using guard objects
guarded	object	guard		guarded object

UMLsec Profile:
Tags

Tag	Stereotype	Type	Multip.	Description
start	fair exchange	state	*	start states
stop	fair exchange	state	*	stop states
adversary	fair exchange	adversary model	1	adversary type
action	provable	state	*	provable action
cert	provable	expression	*	certificate
adversary	provable	adversary model	*	adversary type
protected	rbac	state	*	protected resources
role	rbac	(actor, role)	*	assign role to actor
right	rbac	(role, right)	*	assign right to role
secrecy	critical	data	*	secrecy of data
integrity	critical	(variable, expression)	*	integrity of data
authenticity	critical	(data, origin)	*	authenticity of data
high	critical	message	*	high-level message
fresh	critical	data	*	fresh data
adversary	secure links	adversary model	1	adversary type
adversary	data security	adversary model	1	adversary type
integrity	data security	(variable, expression)	*	integrity of data
authenticity	data security	(data, origin)	*	authenticity of data
guard	guarded	object name	1	guard object

Well-formedness Rules

Stereotypes and tags in more detail.

- Constraints use security-aware interpretation of UML diagrams.
- <<fair exchange>>, <<provable>>, <<secure links>>, <<data security>>:
  - parameterized over adversary type w.r.t. which the security requirements should hold.
- {adversary}: values of the form (T;C).
  - T: Adversary type, such as T = default for the adversary defined later, which may also be self-defined.
    - If ommitet T = default.
  - C: Logical condition on the previous knowledge K^p_A of the adversary[1].
    - If omitted C ensures that data included in {secrecy} tag of <<critical>> does not appear as subexpressions in K^p_A.
- a* represents an arbitrary multiplicity of a tag.

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

Well-formedness Rules

- Constraints associated with stereotypes:
  - give a range from structural syntactic conditions,
    - such as <<secure links>>,
  - to relatively deep semantic conditions,
    - such as <<no down-flow>>.
  - advantage:
    - first find violations against simpler structural conditions, then analyse the behavioral part of the specification
    - automated mechanical verification is also available[1]
- Seems to be more efficient than trying to establish the overall security all at once.
- Industrial setting: allows a scaling of the necessary costs.

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

Examples for usage of stereotypes

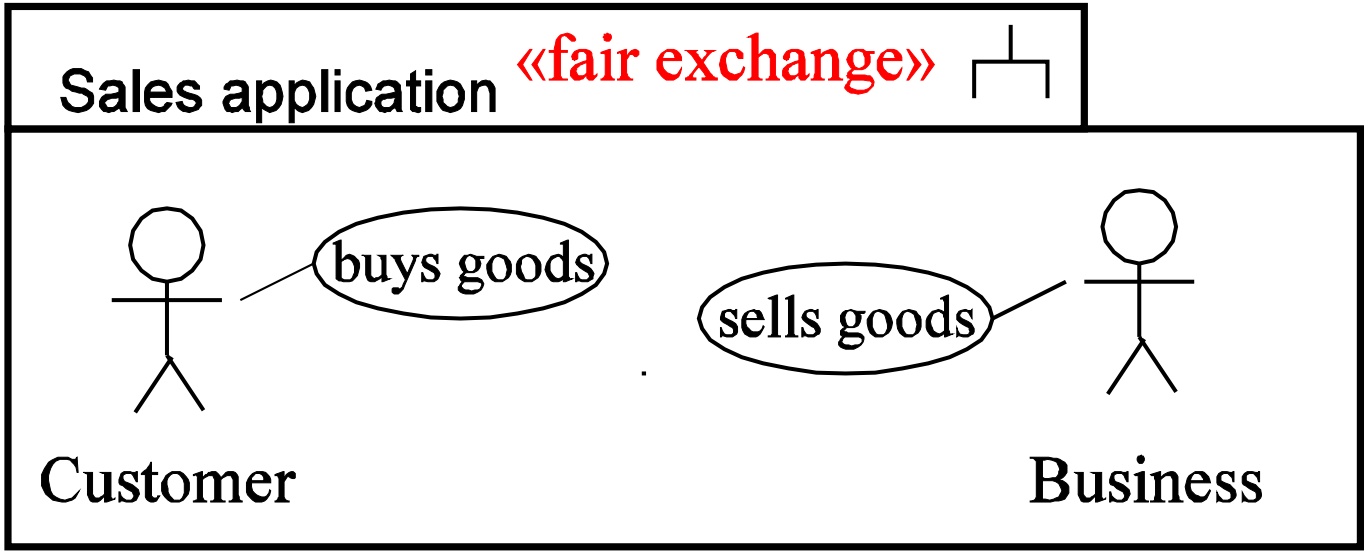
- Examples are just for illustration.
  - No formal proofes for stated properties.
  - Only essential fragments of subsystems of stereotype in question.

- Substantial case-studies for performing security analyses with UMLsec will be discussed in later section.

fair exchange (for use case diagrams)

- Transactions should be performed in a way that prevents both parties from cheating.
- Applicable to subsystems containing a use case diagram.
  - Can be refined by another subsystem only if that is also stereotyped <<fair exchange>>.
- Only informal meaning, as opposed to the stereotypes below.
  - “refinement" is meant here in an informal sense.
- Shows how security requirements (as stereotypes) in other kinds of diagrams below can also conveniently be included in use case diagrams.

Requirements with Use Case Diagrams



Use case diagram describing the following situation:

- a customer buys a good from a business.
- trade should be performed in a way that prevents both parties from cheating.
  - Add requirement by adding <<fair exchange>> to the subsystem containing the use case diagram

Capture security requirements in use case diagrams.

- Constraint: need to appear in corresponding activity diagram.

Requirements with Activity Diagrams

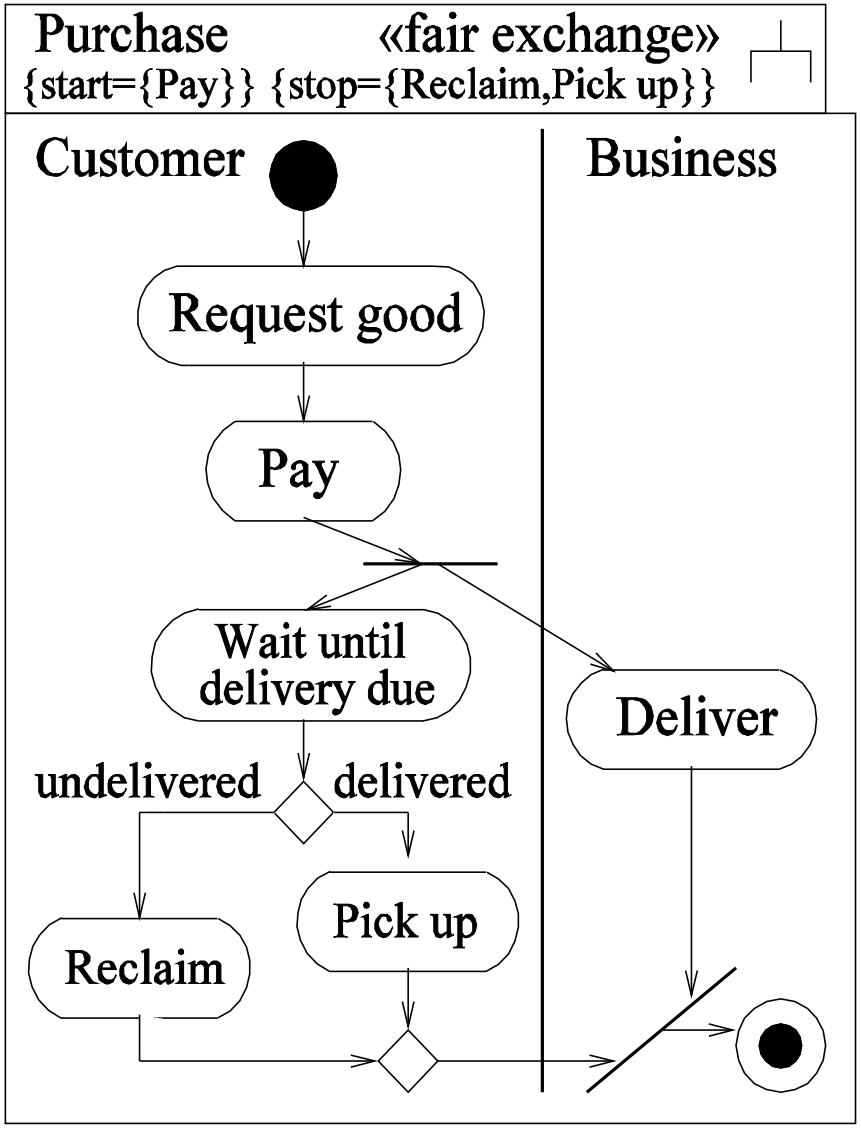
<<fair exchange>> applied to subsystems containing an activity diagram

- associated tags {start}, {stop}, {adversary}.
- {start}, {stop} take pairs (good; state) as values,
  - good is the name of a good to be sold, can be omitted if only one good is to be sold
  - state is the name of a state.
- {adversary} adversary type relative to which the security requirement should hold.
- for every good to be sold, whenever a {start} state in the activity diagram is reached, eventually a {stop} state will be reached, when the system is executed in presence of an adversary of the type A specified in {adversary}.

Example
<<fair exchange>>

Use case in more detail by giving the activity diagram.

- Customer buys goods from a business.
- Adversary type irrelevant
  - no communication structure specified
- How can fair exchange be enforced ?
- Requirement <<fair exchange>> formulated by referring to the activities in the diagram.

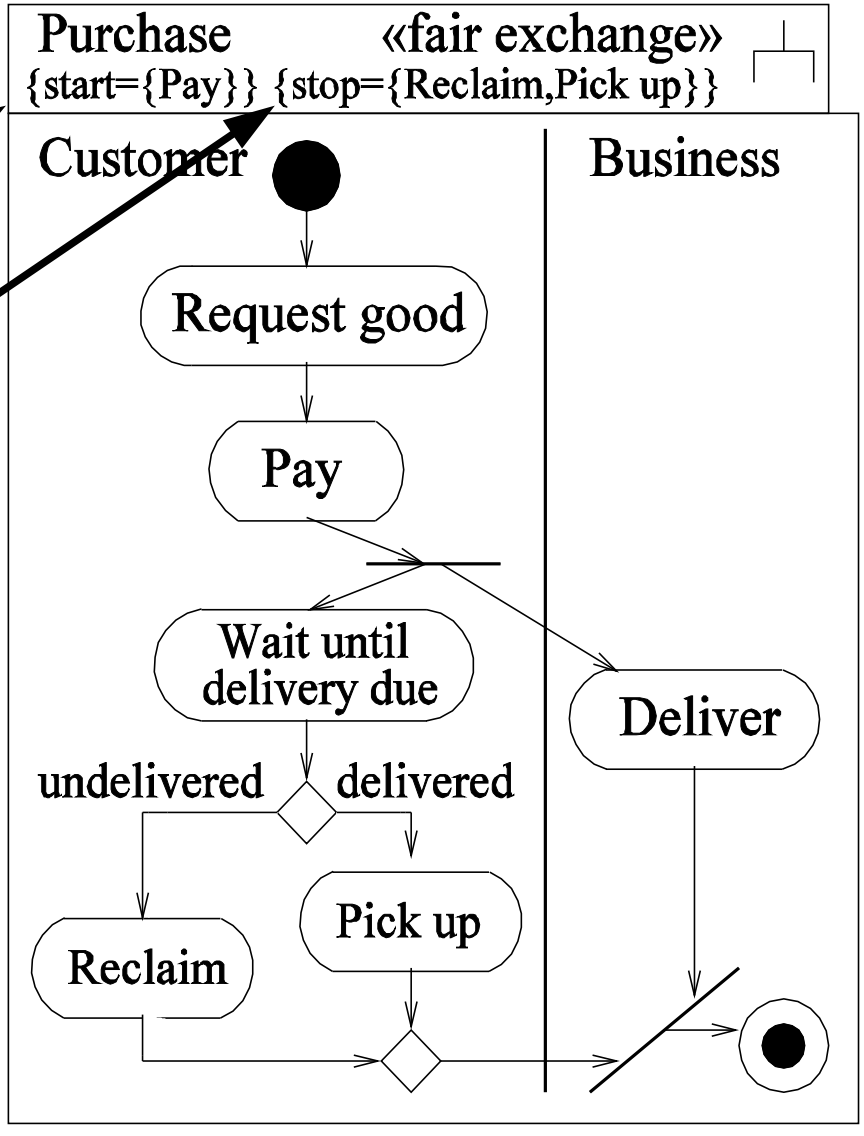


Stereotype <<fair exchange>>

Ensures generic fair exchange condition.

Constraint:

- Actions listed in {start}, {stop} should be linked
  - if one of the former is executed then eventually one of the latter will be.
- Formalized wrt. formal semantics of the used fragment of UML.
- Can be checked automatically.



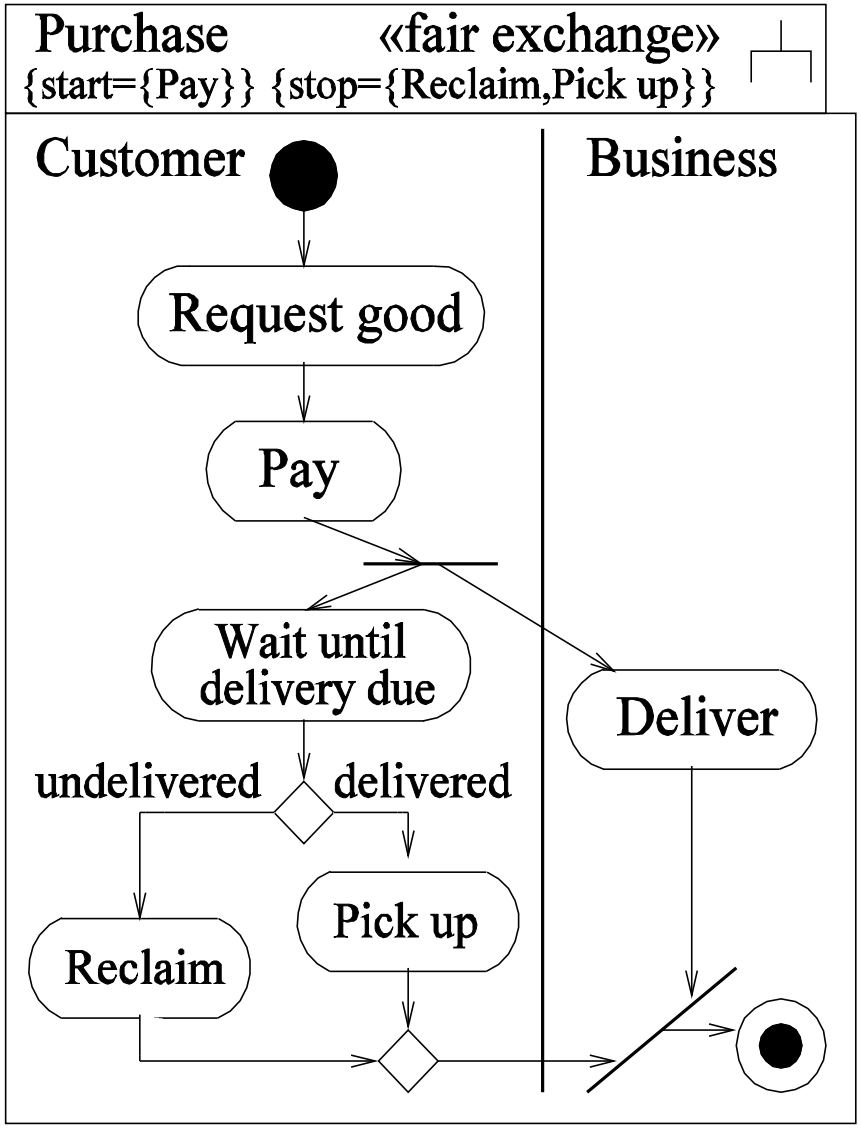
Formalization
<<fair exchange>>

Formalized for a given subsystem S:

- S fulfills the constraint of <<fair exchange>> with respect to adversary type A if for every good to be sold following condition holds:
  - For every execution e of $[[S]]_A$ there exists number $n \in N$ such that for every sequence $I_1,.....,I_n$ of input multi-sets there exists an execution e' which is an extension of e and then processes the inputs in $I_1,.....,I_n$, such that there are at least as many {stop} states in e' as there are {start} states in e, with respect to the relevant good.

Revisit example <<fair exchange>>

<<fair exchange>> fulfilled ?

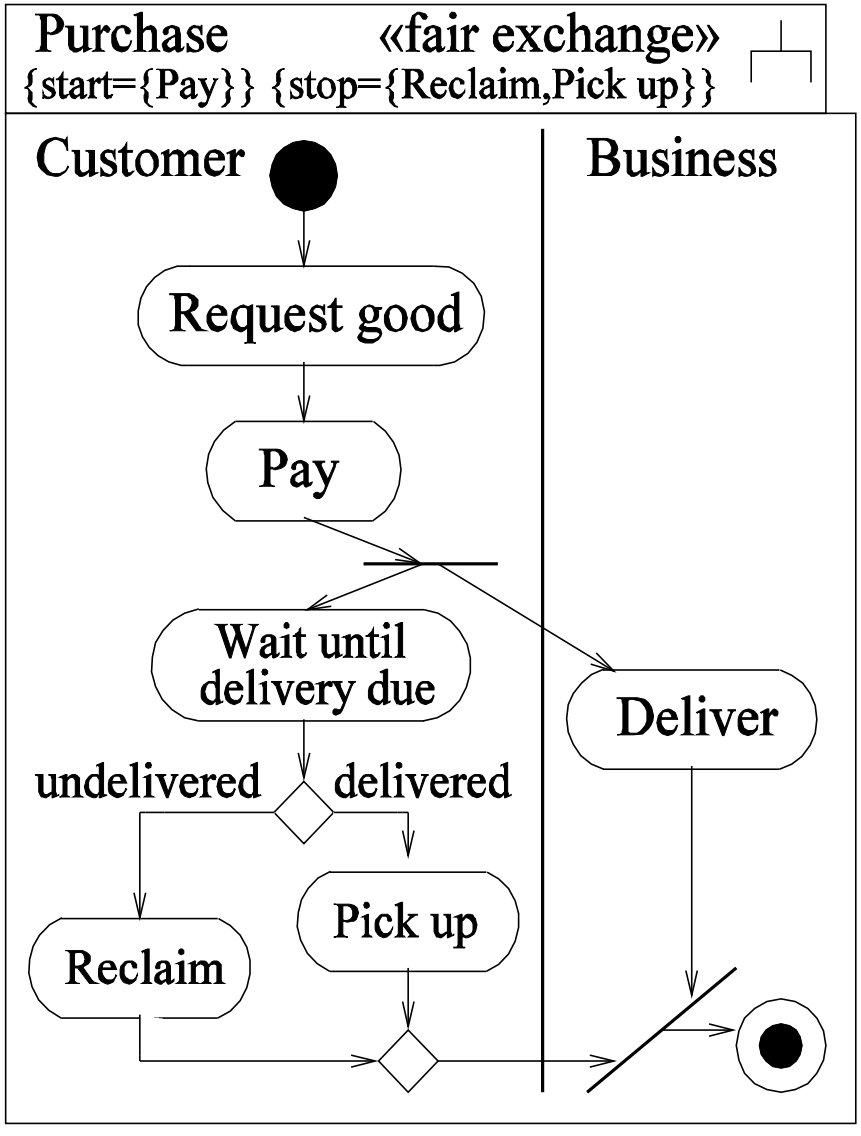


Revisit example <<fair exchange>>

<<fair exchange>> fulfilled:

- After payment:
  - customer is able to either pick up the delivery or reclaim the payment.

Can't be ensured for systems which an attacker can stop completely.



Provable

A subsystem S may be labeled <<provable>> .

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

- {cert} contains an expression
  - proof that the action at the state in {action} was performed.
- {adversary} specifies an adversary type relative to which the security requirement should hold.

S may output expression E ∈ Exp in {cert} only after the state in {action} is reached, when executed in presence of an adversary of the type A specified in {adversary}.

- Here certificate in {cert} is unique for each subsystem instance.

Provable

More formally: S fulfills the constraint if the following holds for adversary type A:

for (execution e of $[[S]]_A$) {

if (expression in {cert} is given as output at a state S in e) then{ state in {action} appears as current state before S in e.

}

}

To avoid illegitimate repayment claims, in <<fair exchange>> example:

- Employ <<provable>> with regard to state Pay.
- Ensure that Reclaim payment action checks whether Customer can provide proof of payment.

role-based access control <<rbac>>

- Applicable to subsystems containing activity diagram
- Enforces rolebased access control in the business process specified in the activity diagram.

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

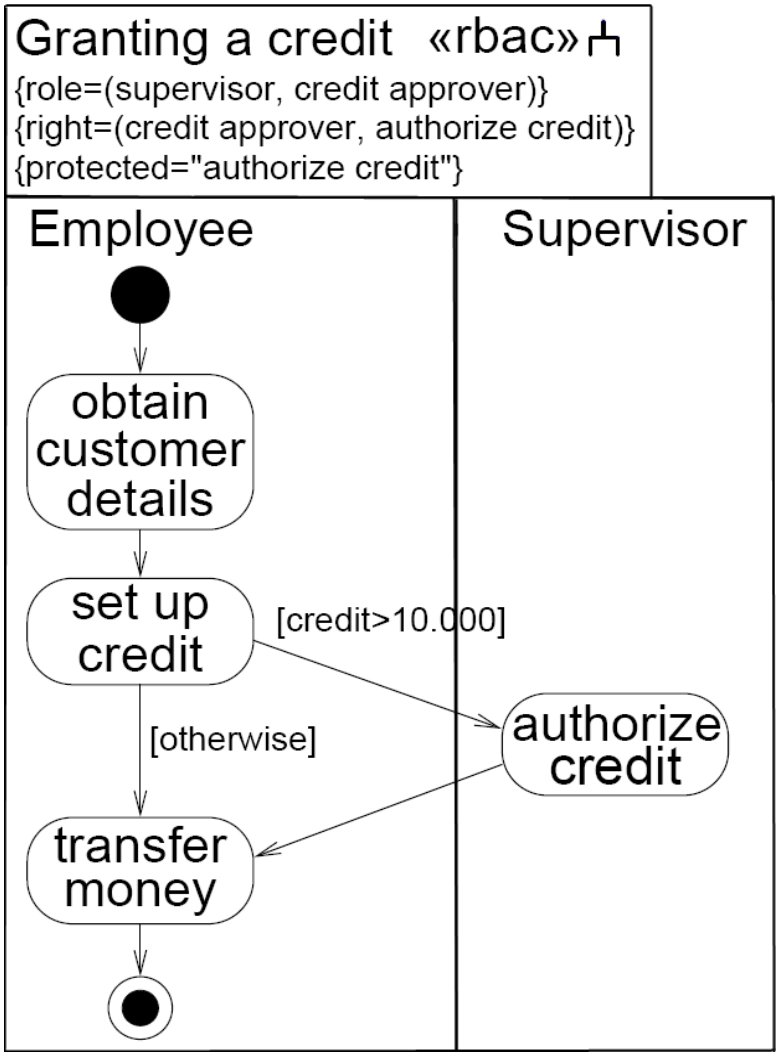
- {protected} contains states in the activity diagram, to which the access should be controled.
- {role} list of pairs (actor; role)
  - actor actor in activity diagram, role is a role.
- {right} has a list of pairs (role; right)
  - role is a role
  - right represents the right to access a protected resource.

Requires that actors in the activity diagram only perform activities for which they have the appropriate rights.

Role-based access control <<rbac>>

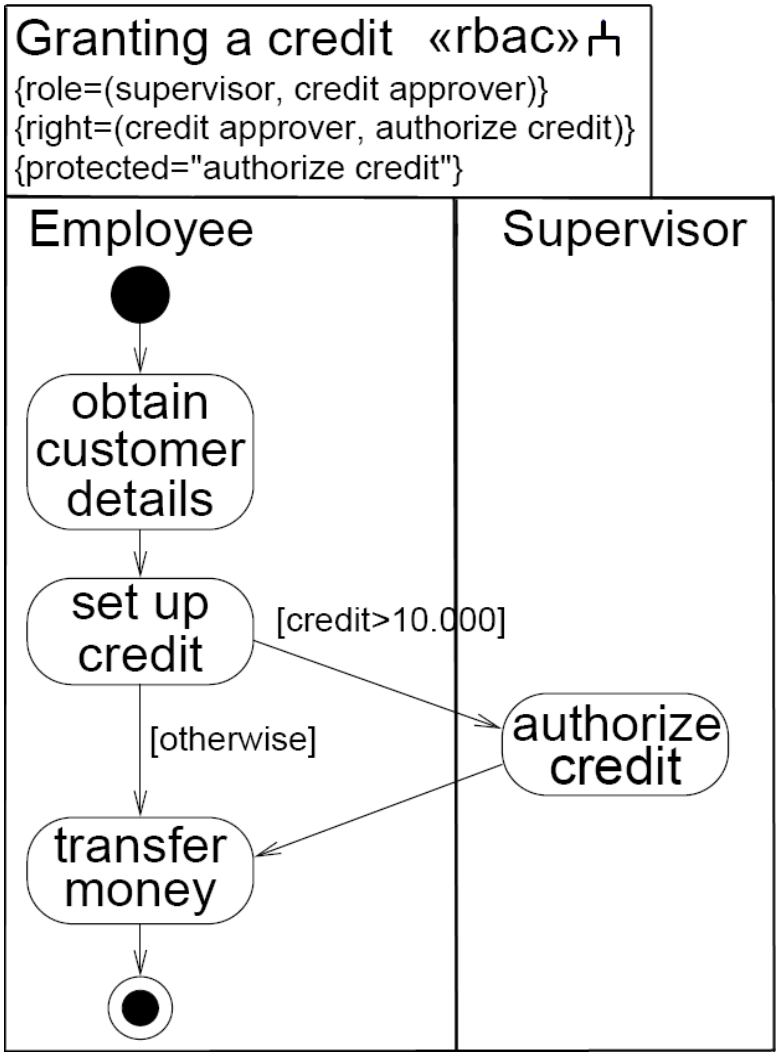
For a subsystem S, this is formalized as follows:

- For every actor A in S and every activity a in swimlane of A in the activity diagram in S, there exists a role R such that (A;R) is a value of {role} and (R; a) is a value of {right}.



Example: Role-based access control (<<rbac>>)

- Simplified part of a business process
  - credit is being set up for a customer of a bank.
- Bank employees have the right to set up credits.
- For large credits > e.g.10.000, supervisors have to authorize the credit before money is transferred.
- Protected resource: authorize credit activity
  - Supervisor, in her role of credit approver, has appropriate permission
- Diagram is correctly labeled <<rbac>>
  - the associated constraint is respected.



Example: Role-based access control (<<rbac>>)

- Example: Instance of the security principle of separation of privilege.
- Ensure that employee is not assigned two roles with associated privileges that are supposed to be separated.
- How to link access control to the level of the technical security architecture is demonstrated using the stereotype <<guarded access>>.

Communication Architecture (1)

- Internet, encrypted, LAN, wire, smart card, POS device, issuer node
  - On links (resp. nodes) in deployment diagrams: denote the respective kinds of communication links (resp. system nodes).
- Require that each link or node carries at most one of these stereotypes.
- For each adversary type A, we have a function $Threats_A(s)$ from

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

  to a set of strings

  $Threats_A(s) \subseteq$ {delete; read; insert; access}:

  - node stereotype s: $Threats_A(s) \subseteq$ {access}
  - link stereotype s: $Threats_A(s) \subseteq$ {delete; read; insert}.

Communication Architecture (2)

$Threats_A(s)$ specifies which kinds of actions an adversary of type A can apply to nodes or links stereotyped s.

Given UML subsystem S, function $Threats_A(s)$ gives rise to

- $threats^A_A(x)$
  - takes a node or link x and a type of adversary A
  - returns set of strings $threats^A_A(x) \subseteq \{delete; read; insert; access\}^2$.

Evaluate UML subsystems using their execution semantics[1] , by referring to the security framework using UML Machine Systems[2].

Examples for threat sets associated with some common adversary types are the default and insider attacker.

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.2
2 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.4

Communication Architecture (3)

Default attacker: outsider adversary with modest capability.

Ability:

- on an Internet link: read, delete, and insert messages.
- on an encrypted Internet link, (such as a virtual private network):
  - delete messages, without knowing their encrypted content, by bringing down a network server.
  - **not able** to read the plaintext messages or insert messages encrypted with the right key.
- Assume: encryption set up such that the adversary does not get hold of the secret key.
- No direct access to local area network (LAN) and therefore unable to eaves-drop on those connections, nor on wires connecting security-critical devices .
- Smart cards are assumed to be tamperresistant.
  - May not be against more sophisticated attackers.
- Unable to access POS devices or card issuer systems.

Communication Architecture (4)

- For adversary type A, *stereotype* s, has a set $Threats_A(s) \subseteq \{delete, read, insert, access\}$ of actions that adversaries are capable of.
- Default attacker: able to read, delete, insert and access messages on an Internet link.

Default attacker:

Stereotype s	$Threats_{default}(s)$
Internet	{delete, read, insert}
encrypted	{delete}
LAN	∅
smartcard	∅

Communication Architecture (5)

- Insider attacker, in the context of the electronic purse system[1].
- May access the encrypted Internet link.
  - knowing the corresponding key, and local system components.

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}

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 5.3

Dependencies

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

- Used on dependencies in static structure or component diagrams.
- Denote dependencies supposed to provide respective security requirement for the data, sent along them as arguments, return values of operations or signals.
- Used in the constraint for the stereotype <<secure links>>.

Critical Data <<critical>>

- Labels objects or subsystem instances containing data that is critical
- Tags: {secrecy}, {integrity}, {authenticity}, {fresh}, and {high}, representing the corresponding security requirements[1].
- {secrecy} names of expressions, attributes or message argument vari-ables of current object the secrecy of which is supposed to be protected; name of an operation is allowed to require that its arguments and return values should be kept secret.
- {integrity} has as values pairs (v;E)
  - v variable of object whose integrity should be protected
  - E set of acceptable expressions that may be assigned to v.

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

Critical Data

- {authenticity} contains pairs (a; o) of attributes of the <<critical>> object or subsystem
  - a stores the data whose authenticity should be provided and
  - o stores the origin of that data.
- {fresh} atomic data (elements of the set Data ⋃ Keys) that should be freshly generated.
- These constraints are enforced by the constraint of <<data security>> which labels subsystems that contain <<critical>> objects, as explained below.
- {high} names of messages that are supposed to be protected w.r.t. secure information flow, as enforced by <<no down-flow>> and <<no up-flow>>.
- Synchronous operations: return messages required to be protected.

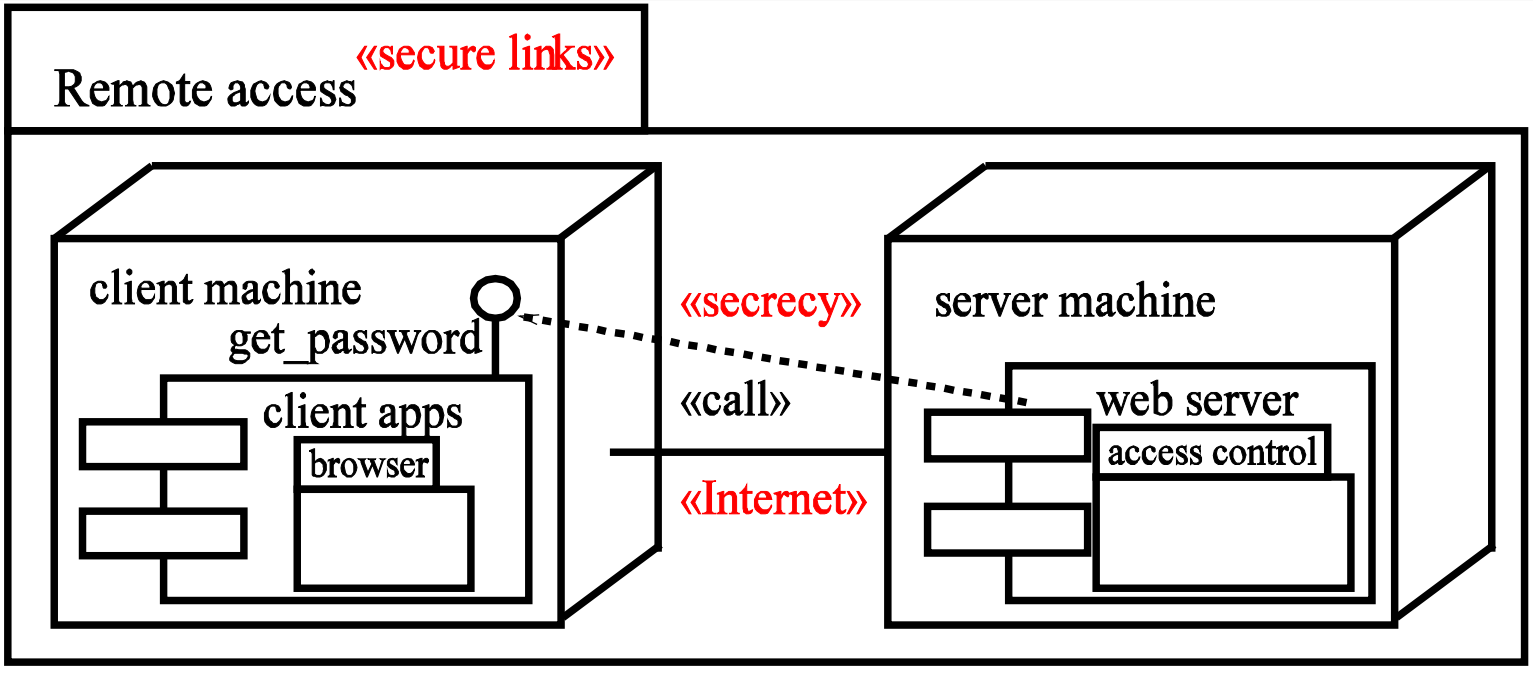
Secure Communication

- Together with the associated stereotypes <<secrecy>>, <<integrity>>, <<high>>, and <<critical>> one can describe different conditions for ensuring secure data communication with the following stereotypes:
  - <<secure links>>
  - <<secure dependencies>>
  - <<data security>>

Secure Communication

- <<secure links>>
  - Ensures that security requirements on the communication dependencies between components are supported by the physical situation, relative to the adversary model under consideration.
- <<secure dependencies>>
  - Ensures that the security requirements in different parts of a static structure diagram are consistent.
- <<data security>>
  - Ensures that security is enforced on the behavior level.
- One could for example merge the conditions of <<secure links>> and <<secure dependencies>> to give one stereotype.

Security at Architectural Level: Example

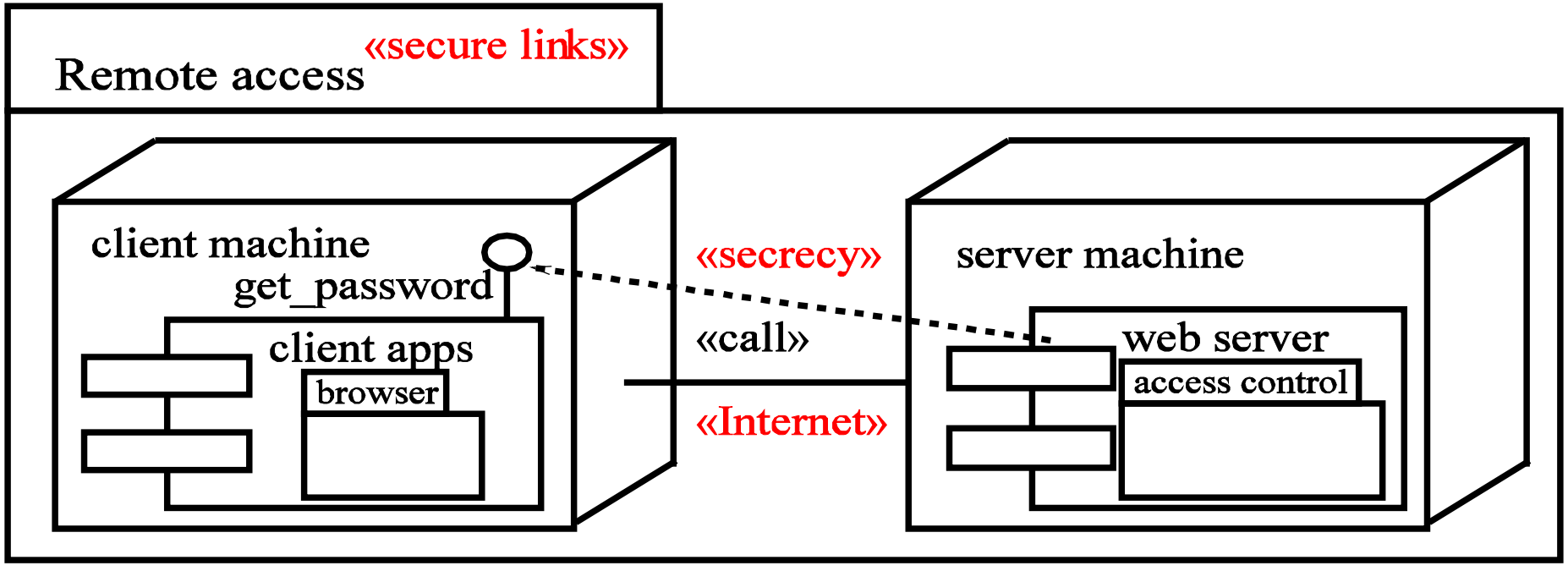


- Business application: part of an e-commerce system
- Supposed to be realized as web application.
- Payment transaction involves transmission of secret data over Internet links.
- <<secure links>> demands that security requirements on communication are met by physical layer.
- Architecture secure against default adversary ?

Stereotype <<secure links>>

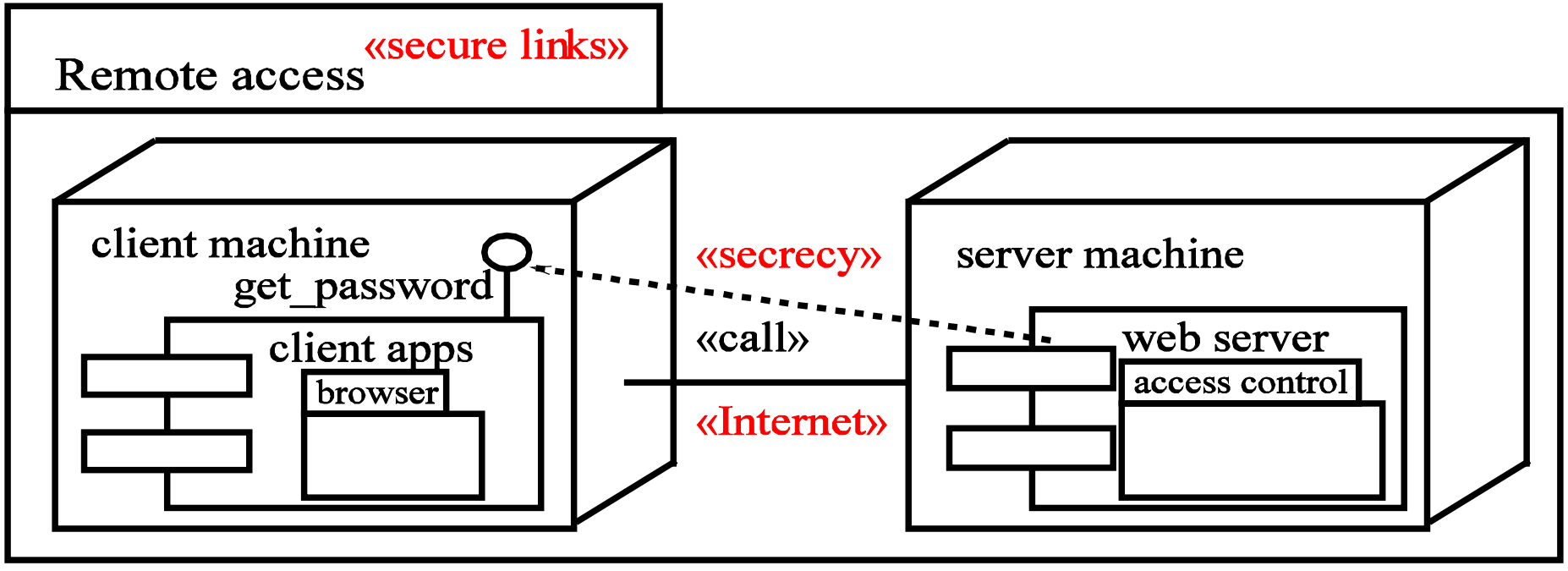
- Remember $threats_A^S(x) \subseteq \{delete; read; insert; access\}$ with UML subsystem S, node or link x and adversary A.
- Label subsystems containing static structure diagrams
- ensures that physical layer meets security requirements on communication.
- Constraint enforces that for each dependency d with stereotype s∈{<<secrecy>>, <<integrity>>, <<high>>} between subsystems or objects on different nodes m≠n, have a communication link l between m and n such that:
  - If s = <<high>> : have $threats_A^S(t) = \emptyset$
  - If s = <<secrecy>> : have $read \notin threats_A^S(t)$
  - If s = <<integrity>> : have $insert \notin threats_A^S(t)$

Revisit example <<secure links>>



Constraint for stereotype <<secure links>> fulfilled for default adversaries ?

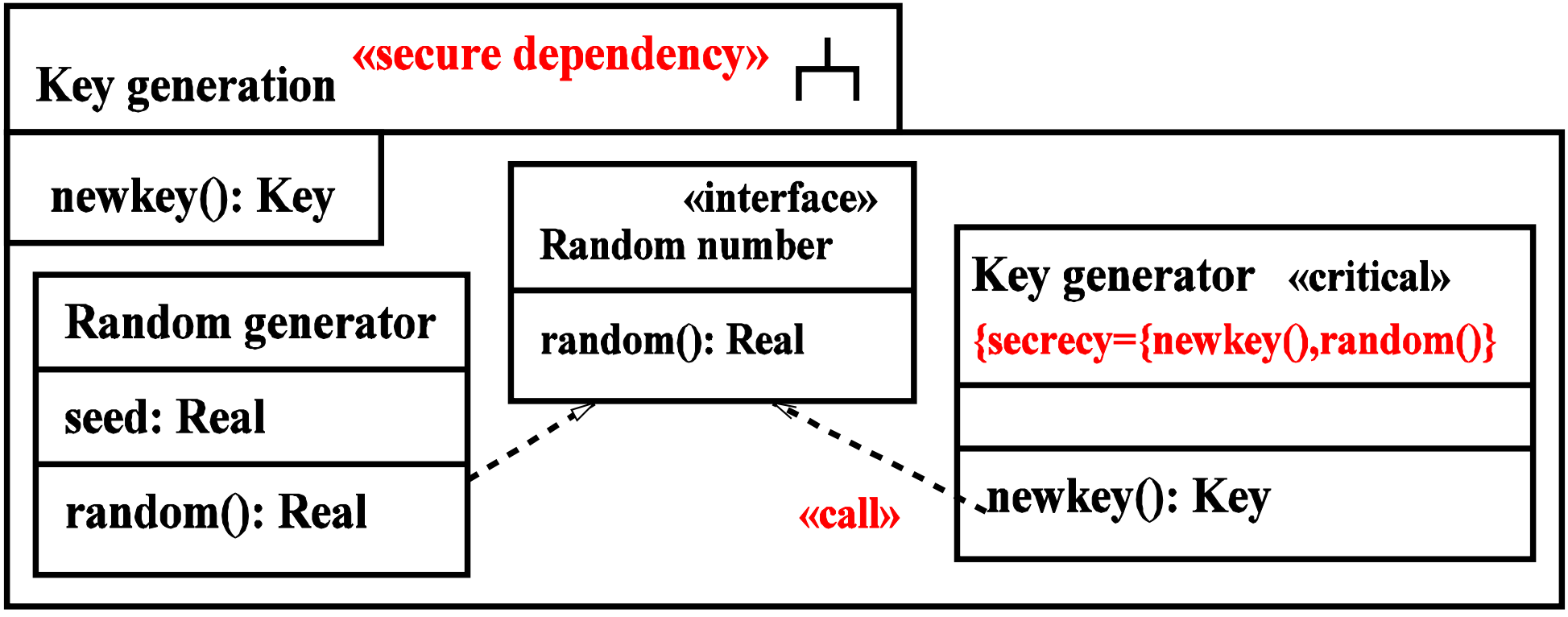
Revisit example <<secure links>>



Constraint for stereotype <<secure links>> fulfilled for default adversaries ?

- Intuitively: Internet connections do not provide secrecy against default adversaries.
- Technically: Constraint is violated because the dependency carries the stereotype <<secrecy>>, but for <<Internet>> of the corresponding link we have read $\in$ $threats_{default}$(Internet).

Security at Class Structure Level: Example

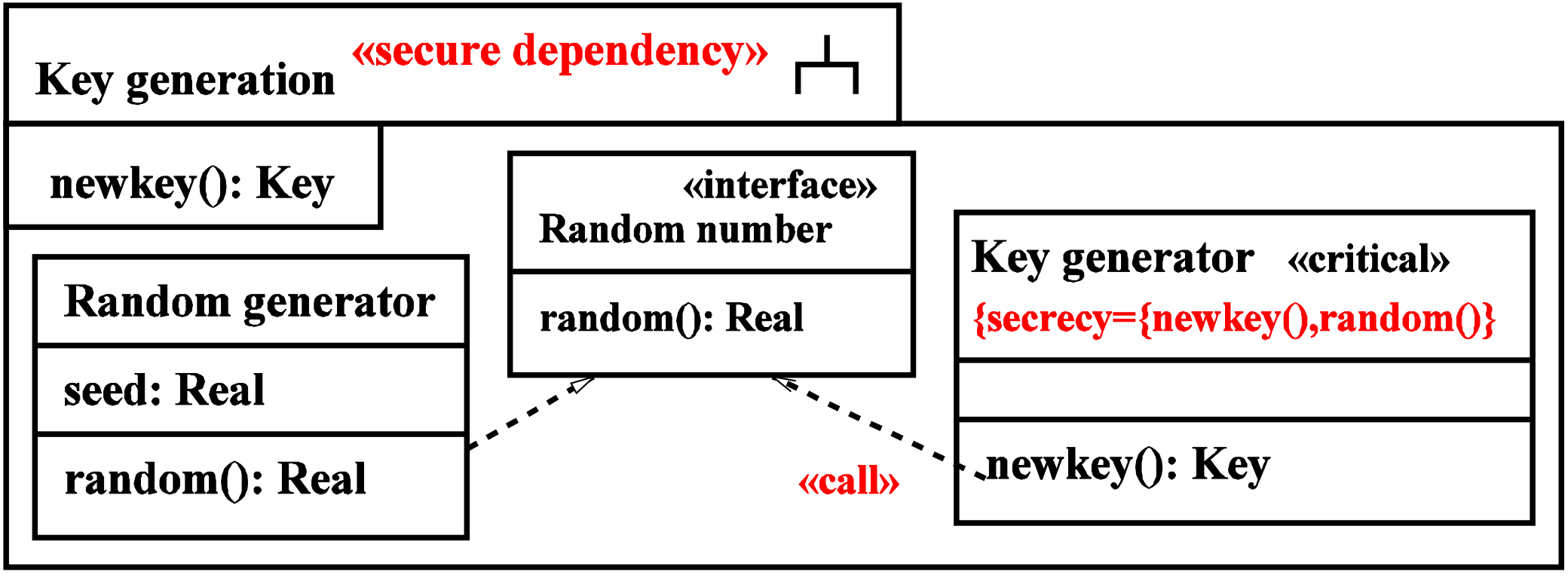


Security annotations consistent across class diagram ?

Stereotype <<secure dependency>>

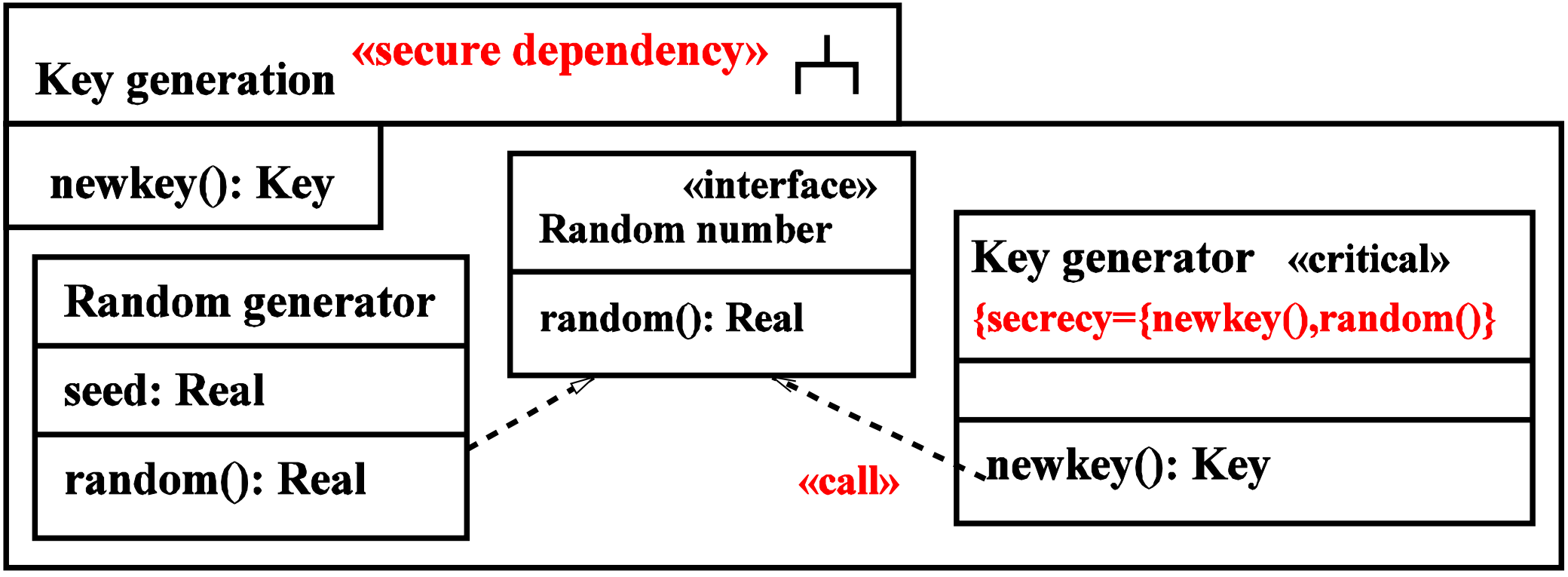
- Labels subsystems containing static structure diagrams
- Ensures: <<call>> and <<send>> dependencies between components respect security requirements on communicated data given by {secrecy}, {integrity} of the stereotype <<critical>>.
- More exactly, Constraint enforced is that if there is a <<call>> or <<send>> dependency from an object or subsystem C to an interface I of an object or subsystem D then the following conditions are fulfilled:
  - For any message name n in I, n ∈ {secrecy} (resp.{integrity}, {high}) in C if and only if it does so in D.
  - If a message name in I appears in {secrecy} (resp. {integrity}, {high}) in C then the dependency is stereotyped <<secrecy>> (resp.<<integrity>> resp. <<high>>)

Revisit example
<<secure dependency>>



<<secure dependency>> fulfilled or not ?

Revisit example <<secure dependency>>



- Violates <<secure dependency>>:
  - Random generator and <<call>> dependency do not give security property {secrecy} for random() required by key generator.

Stereotype <<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.
- Constraint associated with <<data security>> requires that these requirements are met w.r.t. the given adversary model.
- Formalization of this constraint discussed in detail in later section.

Stereotype <<data security>>

- Subsystem S stereotyped <<data security>> respects data security requirements by the stereotypes <<critical>> and the associated tags contained in the subsystem w.r.t. the threat scenario arising from the deployment diagram and adversary type A in {adversary}

More precisely: Constraint given by four conditions, which use the concepts of secrecy, integrity, authenticity, and freshness.

- secrecy: Subsystem preserves secrecy of data designated by {secrecy} against adversaries of type A.
- authenticity: For any (a,o) of {authenticity}, S provides the authenticity of the attribute a w.r.t. its origin o against adversaries of type A.

Stereotype <<data security>>

- integrity: {integrity} of <<critical>> with a value (v,E), the subsystem preserves the integrity of variable v against adversaries of type A, w.r.t. E of admissible expressions.
  - If E is omitted, integrity of v should be preserved w.r.t. the set of expressions that can be constructed from those in the specification of S.
  - Adversary should not be able to make the variable v take on a value previously known only to him.
- freshness: Within S stereotyped <<data security>>, any value data ∈ Data ∪ Keys tagged {fresh} in the relevant subsystem instance or object D stereotyped <<critical>> in S should be fresh in D.

Stereotype <<data security>>

Initial knowledge of the adversary may not contain the data values that, according to the tags of <<critical>>, should be guaranteed secrecy, integrity or authenticity:

- Cannot be achieved if the adversary knows this data initially.
- Further assumptions on the initial adversary knowledge can be specified.
- If admissible expressions or the intended origin of data in {integrity} and {authenticity} refer to expressions not locally known at the <<critical>> object where these tags are applied, one can associate these tags with the relevant <<data security>> stereotype.
- Assume that standard adversary not able to break encryption, but can exploit design flaws e.g. in a crypto protocol, for example by attempting so-called „man-in-the-middle“ attacks.

Stereotype <<data security>>

Note:

- Enough for data to be listed with a security requirement in one of the objects or subsystems contained in the subsystem to be required to fulfill the conditions.
- Several nested subsystems may each carry <<data security>>.
  - The conditions are required to hold w.r.t. each of them. Important to note when including one subsystem in another.

Secure Use of Cryptography: Example

Variant of the Internet security protocol TLS proposed in [APS99]

Goal:

- Secure channel over an untrusted communication link between a client and a server.
  - Provide secrecy and server authenticity, as specified by the {secrecy} and {authenticity}.
- To achieve this, some of local attributes have to satisfy {integrity} as well.
  - The adversary should not be able to make these attributes take on a value in his previous knowledge.

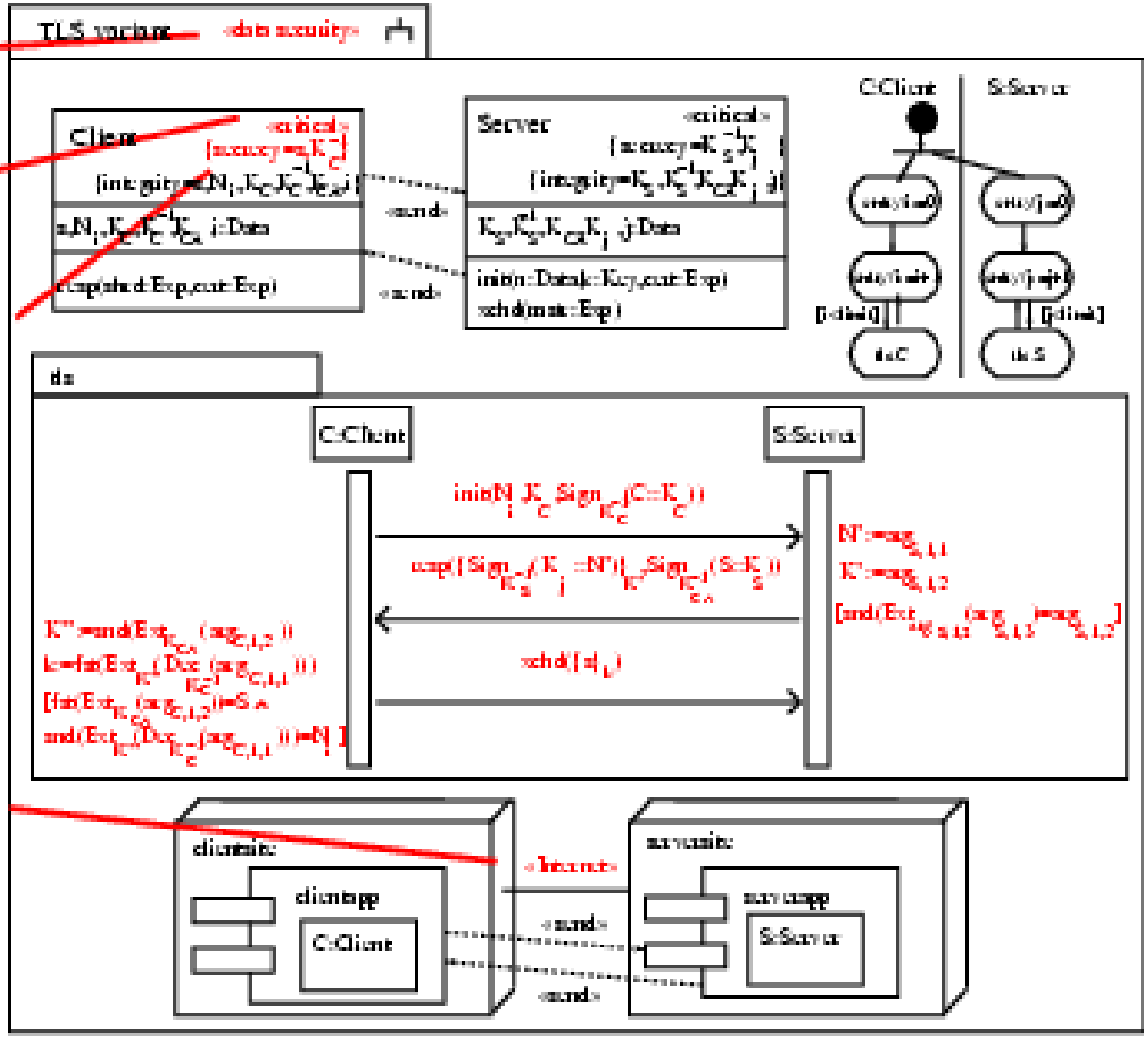
[APS99] V. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost? In Conference on Computer Communications (IEEE Infocom), pages 717-725. IEEE Computer Society, New York, March 1999.

Secure Use of Cryptography: Example

Variant of TLS (INFOCOM`99).

Goal is to exchange a secret session key K, using public keys K_C and K_S, which is then used to encrypt the secret data s before transmission.

Cryptoprotocol secure against default adversary?



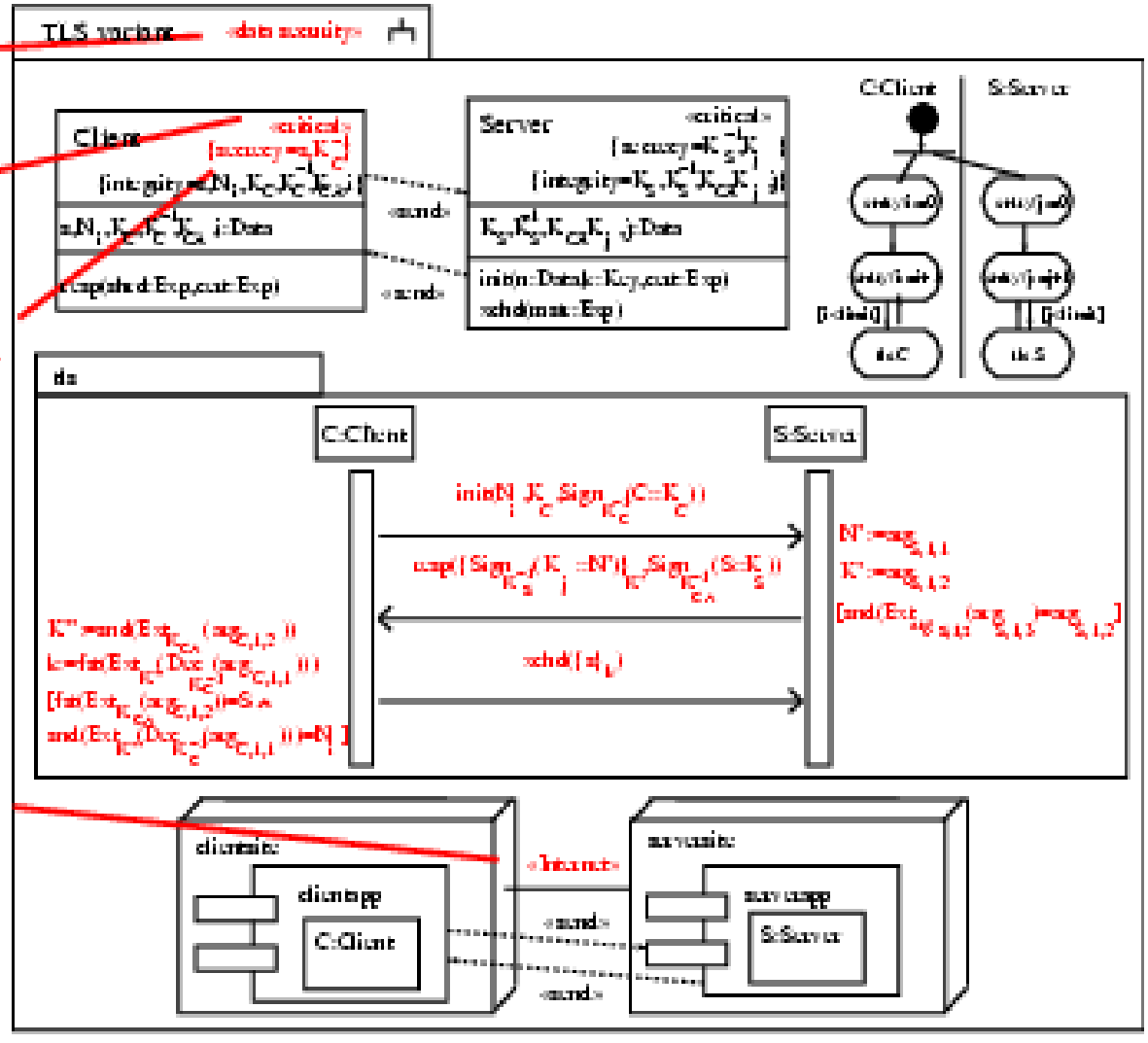
Revisit example (Variant of TLS) <<data security>>

«data security»

«critical»

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

«Internet»



Violates {secrecy} of s against default adversary.

More details later.

Secure Use of Cryptography: Possible extension

- Properties of secrecy, integrity, and authenticity are taken relative to the considered type of adversary.
- Default adversary is a principal external to the system;
- Adversaries as part of the system under consideration are possible giving adversary access to the relevant system components.
  - by defining $Threats_A(s)$ to contain access for the relevant stereotype s.
- E.g.: e-commerce protocol involving customer, merchant, and bank
  - goods being purchased is a secret known only to the customer and merchant, and not the bank.

Secure Use of Cryptography: Possible extension

- Formulated by marking relevant data as “secret“ and by performing a security analysis relative to the adversary model “bank“.
  - Adversary is given access to the bank component by defining Threats() function in a suitable way.
- Note: Adversary does not necessarily have access to the input queue of the system.
- May be sensible, e.g. to apply {secrecy} to a value received by the system from the outside.
- Condition associated with <<data security>> only ensures that stereotyped component keeps the values received by the environment secret.
- Make sure that the environment of the system part under consideration does not make these values available to the adversary either.

Secure Information Flow

- Alternative way of specifying secrecy-and integrity-like requirements.
- Protection against partial flow of information.
- Can be more difficult, especially when handling with encryption.
- Assign to each piece of the system data one of two security levels:
  - high, meaning highly sensitive or highly trusted.
  - low, meaning less sensitive or less trusted.

Given a set of messages H and a sequence m of event multi-sets, we write:

- m^H for the sequence of event multi-sets derived from those in m by deleting all events the message names of which are not in H.
- m_H for the sequence of event multi-sets derived from those in m by deleting all events the message names of which are in H.

Secure Information Flow: Background

Definition: Given a subsystem S and a set of high messages H,

we say:

- A prevents down-flow with respect to H if for any two sequences i; j of event multi-sets and any two output sequences $o \in [[S]]_A(i)$ and $p \in [[S]]A(j)$, $i_H = j_H$ implies $o_H = p_H$ and
- A prevents up-flow with respect to H if for any two sequences i; j of event multi-sets and any two output sequences $o \in [[S]]_A(i)$ and $p \in [[S]]_A(j)$, $i^H = j^H$ implies $o^H = p^H$.

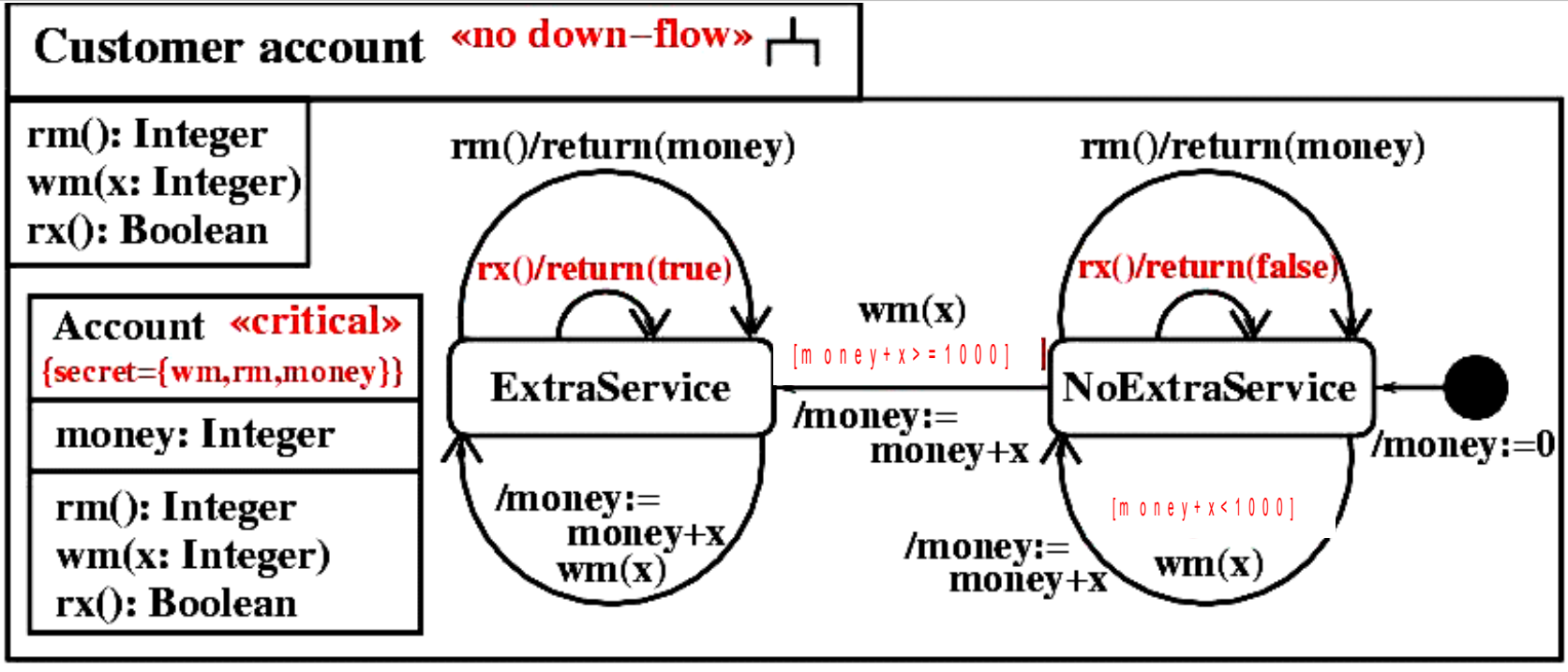
Secure Information Flow: Background

- Intuitively:
- Prevent down-flow: outputting a non-high (or low) message does not depend on high inputs.
  - rather stringent secrecy requirement for messages marked as high.
- Prevent up-flow: outputting a high value does not depend on low inputs.
  - stringent integrity requirement for messages marked as high.
- This notion is generalization of the original notion of non-interference for deterministic systems[1] to system models that are non-deterministic because of underspecification[2].

1 J. Goguen and J. Meseguer. Security policies and security models. In Symposium on Security and Privacy (S&P), pages 11{20. IEEEComputer Society, New York, 1982.

2 J. Jürjens. Principles for Secure Systems Design. PhD thesis, Oxford University Computing Laboratory, 2002.

Secure Information Flow: Example (1)

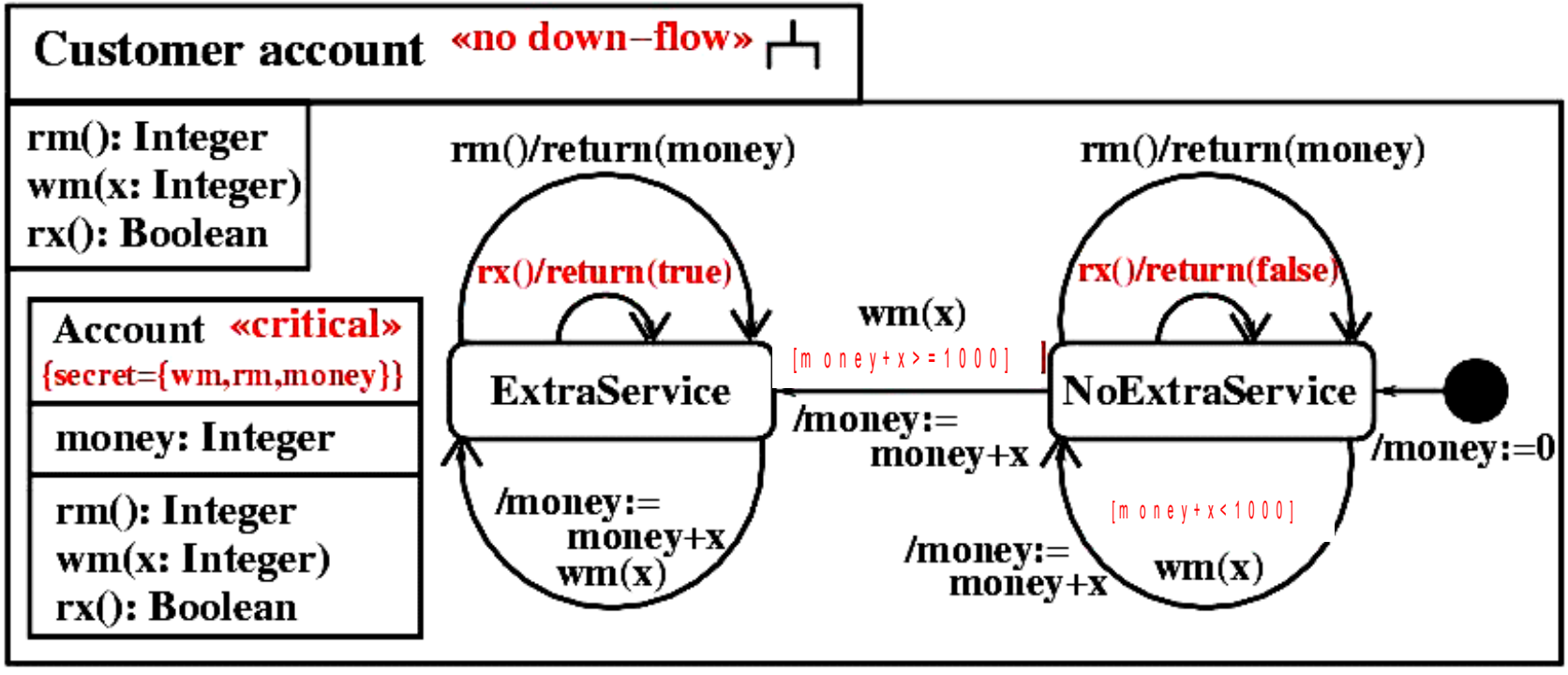


- Secret attribute money containing the amount of money spent by a given customer.
  - Can be read by rm(): return value is also secret.
  - Increase money with operation wm(x).
- When money exceeds 1000, goes into state ExtraService.
- Public operation rx() to check whether extra service should be provided.

Secure Information Flow

- Prevent the indirectly leakage out of any partial information about high data via non-high data, as specified by the stereotype <<no down-flow>>.
  - Enforces secure information flow by making use of {high} associated with <<critical>>.
- Intuitively: Value of any data specified in {secrecy} may influence only the values of data also specified in {secrecy}.
- More precisely, formalize by referring to formal behavioural semantics: Constraint for <<no down-flow>> (resp. <<no up-flow>>) is that UML machine Exec[[S]] for subsystem S prevents down-flow (resp. up-flow) with respect to messages specified in <<high>> and their return messages.
- E.g. for privacy reasons, it may be important that the observable information on the customer account allows no conclusion about the money spent.

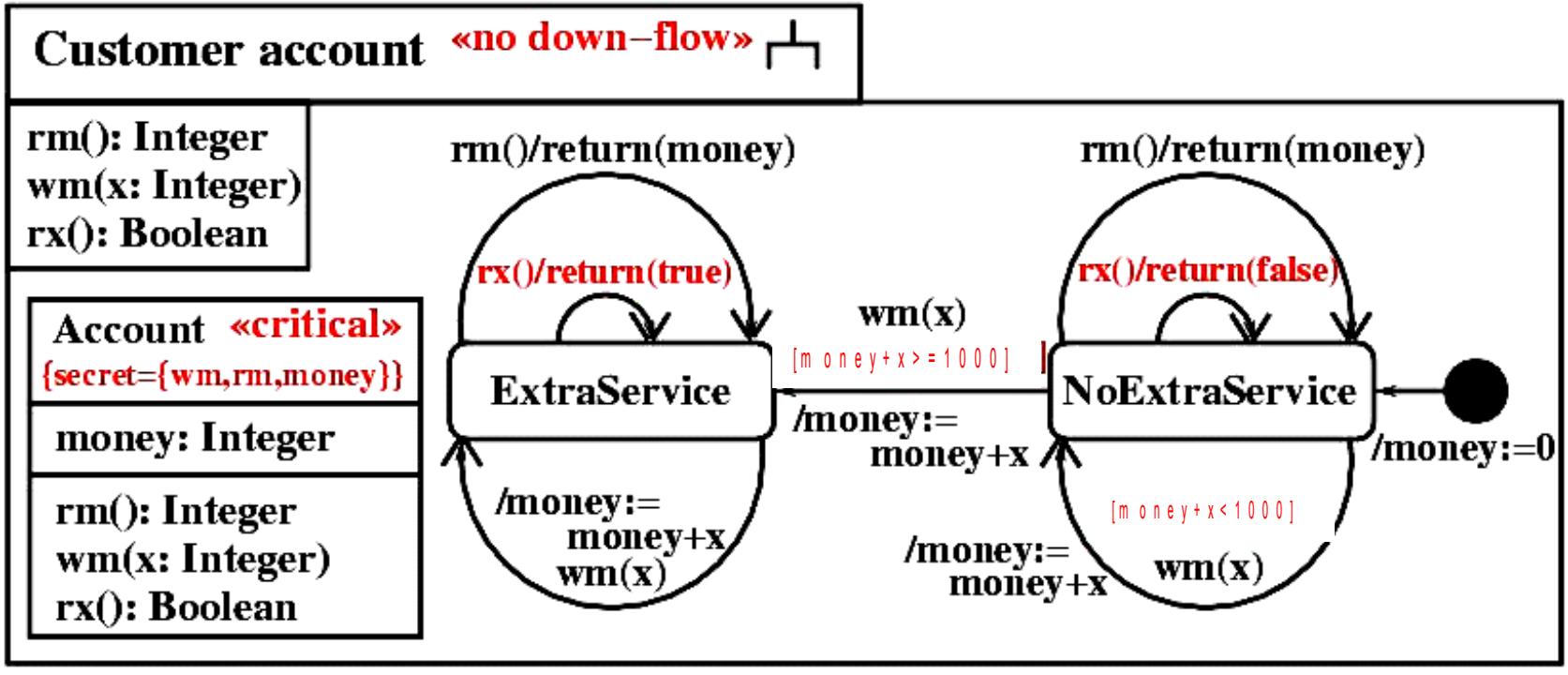
Secure Information Flow: Example (2)



Now we use the stereotype <<no down-fow >> to indicate that the object should not leak out any information about secret data, such as the money attribute.

No partial leakage of secrets ?

Secure Information Flow: Example (3)



- <<no down-fow>> indicates that the object should not leak out any information about secret data, such as the money attribute.
- Violation of the constraint associated with <<no down-flow>>:
  - partial information about the input of the high operation wm() leaked out via the return value of the non-high operation rx().

Secure Information Flow: Example (4)

How the underlying formalism captures the security flaw using the previous definition:

- sequences i; j of input multi-sets
- sequences $o \in [[A]](i)$ and $p \in [[A]](j)$ of output multi-sets of the UML Machine A giving the behavior of the considered statechart
- with $i_H = j_H$ and $o_H \neq p_H$, where H is the set of high messages.
- Consider the sequences
  - i := ({{wm(0)}} ; {{rx()}})
  - j := ({{wm(1000)}} ; {{rx()}}).

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

Secure Information Flow: Example (5)

Given $i_H = (\{\{\ \}\}, \{\{\ rx()\ \}\}) = j_H$.

Definition of the behavioral semantics of statecharts[1], brings the output multi-sets:

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

=>

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

meaning that the constraint associated with <<no down-flow>> is violated.

Can be detected automatically with the tool support provided for UMLsec[2].

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.2
2 http://umlsec.de/

Guarded Objects
<<guarded access>>

- Each object in the subsystem stereotyped <<guarded>> can only be accessed through the objects specified by {guard} attached to the <<guarded>> object.
- Formally: assume name $\notin K^p_A$ for adversary type A under conside-ration and each name name of an instance of a <<guarded>> object, meaning that a reference is not publicly available.
- Assume: for each <<guarded>> object there is a statechart specification of an object whose name is given in {guard}.
- To model passing of references.

Guarded Objects
<<guarded access>>

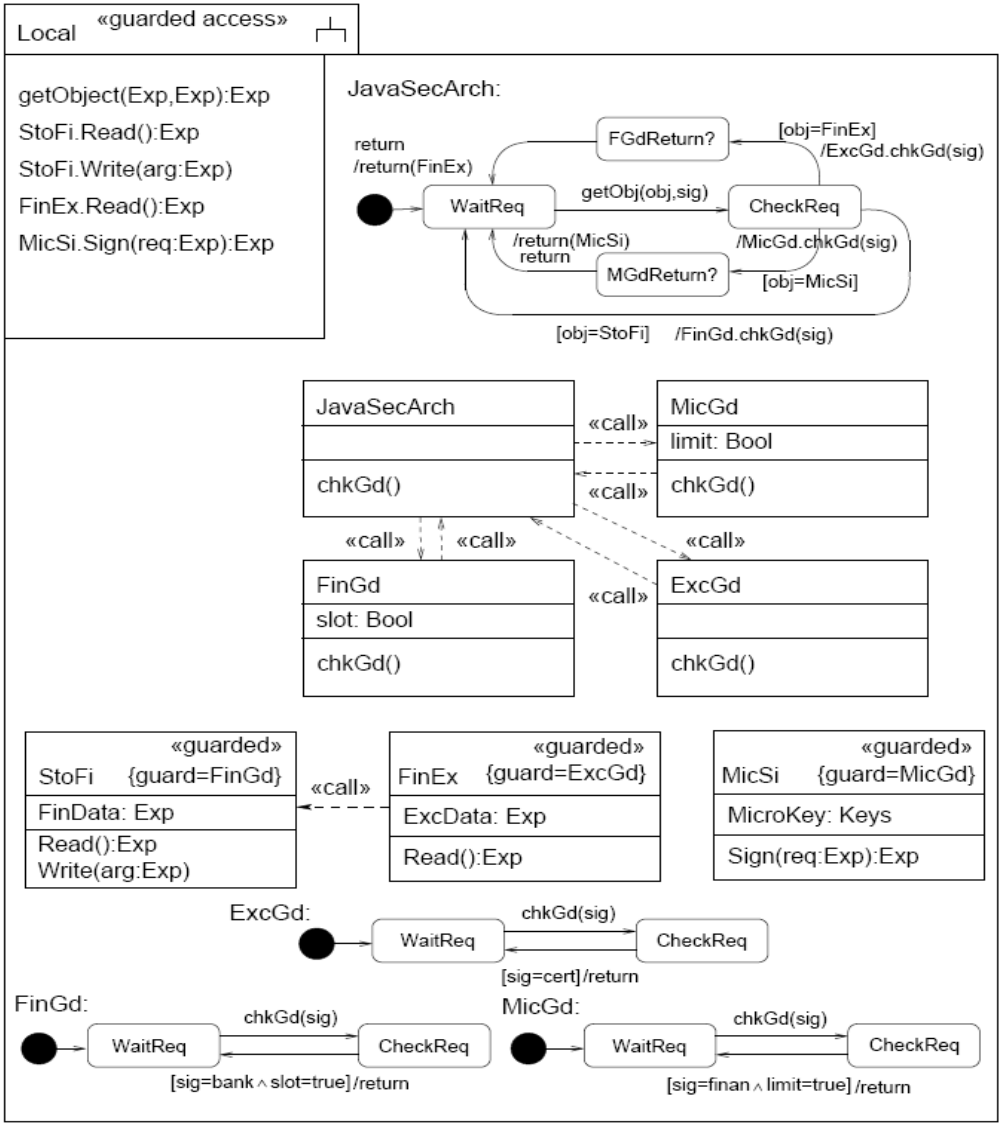
Example:

- Illustration with a web-based financial application.
- Two institutions offer services over the Internet to local users:
  - Internet bank, Bankeasy
  - financial advisor, Finance.
- Use these services:
  - Local client needs to grant the applets certain privileges.
- Access to local financial data is realized using GuardedObjects.

Guarded Objects <<guarded access>>

Simplified relevant part of Java Security Architecture

- Receives requests for object references
- Forwards them to the guard objects of the three guarded objects.
- <<guarded>> objects StoFi, FinEx, and MicSi can only be accessed through their associated guard.
  - Subsystem instance fulfills the condition associated with <<guarded access>> w.r.t. default adversaries.



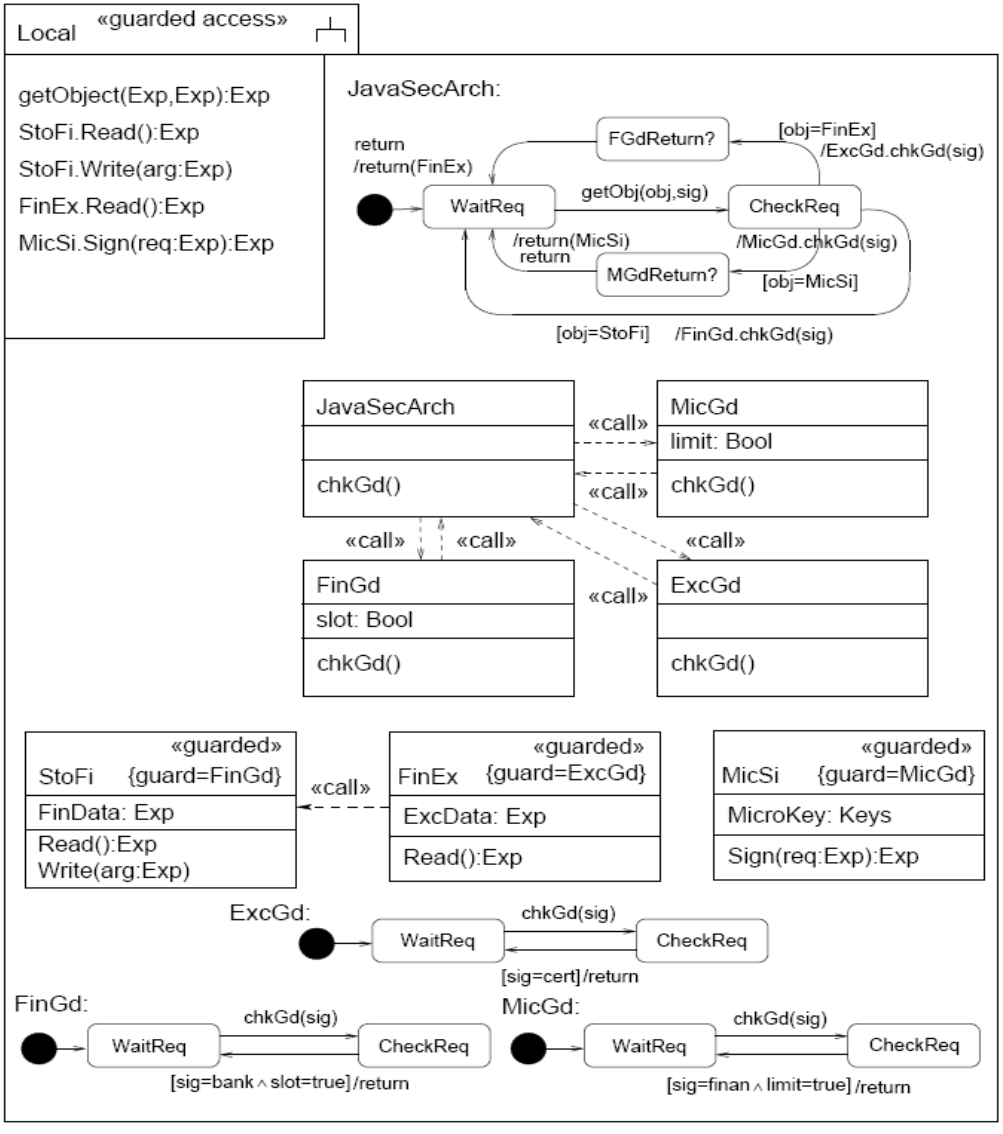
Guarded Objects Example

Access controls realized by Guard objects FinGd, ExpGd, and MicGd.

- Behavior is specified.

Applets signed by the bank

- Read and write the financial data stored in the local database, but only between 1 pm and 2 pm.
- Enforced by the FinGd guard object.
  - Condition slot is fulfilled if and only if the time is between 1 pm and 2 pm.



Guarded
<<guarded>>

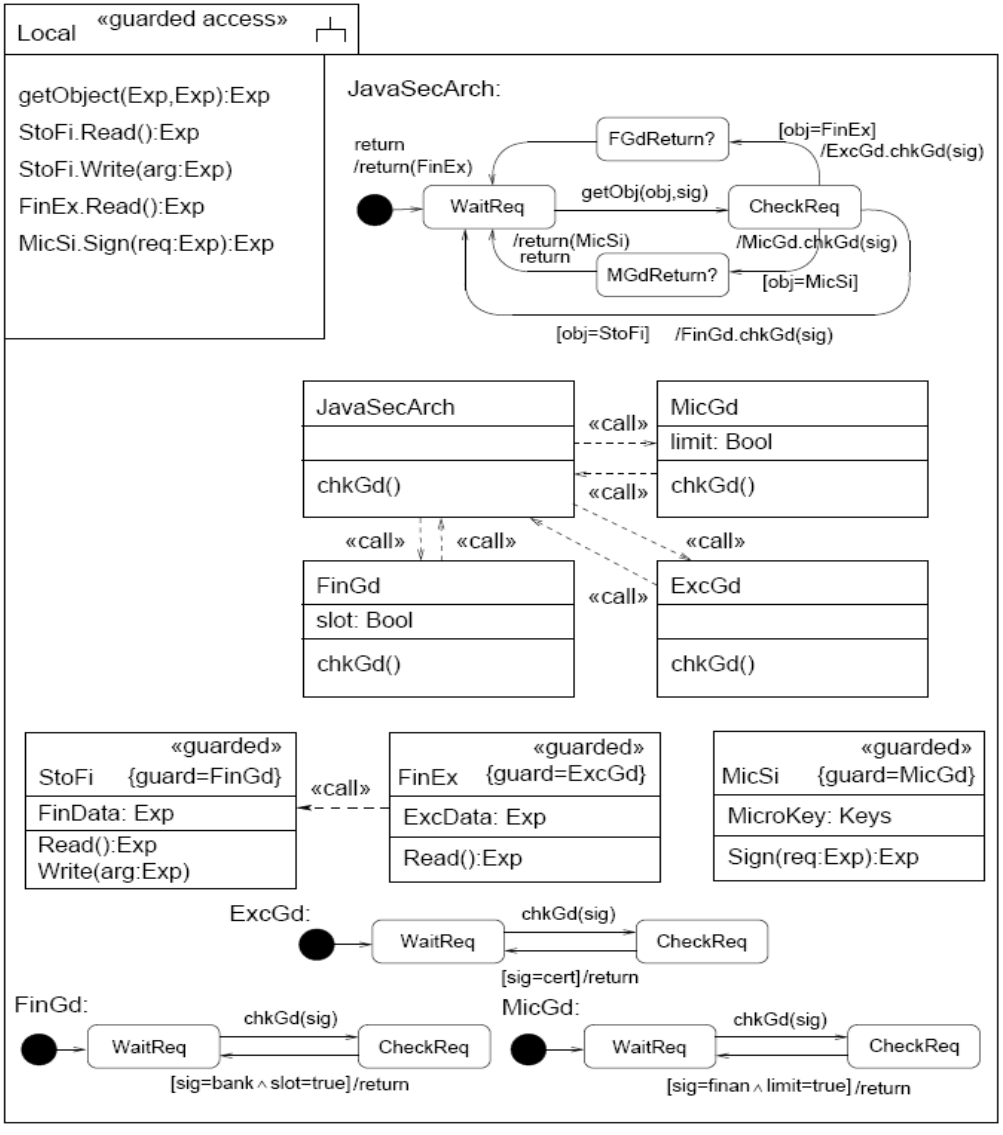
- <<guarded>> labels objects in scope of <<guarded access>> that are supposed to be guarded.[1]
- Tag:
  - {guard} name of the correspon-ding guard object.
- <<guarded>> objects

  StoFi, FinEx, MicSi

  protected by the {guard} objects

  FinGd, ExpGd, MicGd

  respectively.



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

Does UMLsec Meet Requirements ?

- Security requirements: Formalizations of basic security requirements provided via stereotypes, such as <<secrecy>>, etc.
- Threat scenarios: using the formal semantics and depending on the modeled underlying physical layer via the sets $Threats_{adv}(ster)$ of actions available to the adversary of kind *adv*.
- Security concepts: For example <<smart card>>.
- Underlying physical security:
  - Addressed by <<secure links>> in deployment diagrams.
- Security primitives:
  - Either built in, such as encryption, or
  - Can be treated, such as security protocols.
- Security managements: Use activity diagrams.

Summary: 3.3 UMLsec

- General Ideas
- Stereotypes
- Communication Architecture
- Critical Data
- Secure Communication
- Secure Information Flow