Foundations for Designing Secure Architectures

Jan Jürjens

Competence Center for IT Security

Software & Systems Engineering

TU Munich, Germany

juerjens@in.tum.de

http://www.umlsec.org

Foundations for Designing Secure Architectures

This talk: foundations for designing software architectures for secure systems, based on the Unified Modeling Language (UML).

Will use UML to model software-based systems, and in particular their architectures.

Will apply specialized tool-support to automatically evaluate the designs for potential security weaknesses.

(Based on tutorial at WICSA 2004.)

A Need for Security

Society and economies rely on computer networks for communication, finance, energy distribution, transportation...

Attacks threaten economical and physical integrity of people and organizations.

Interconnected systems can be attacked anonymously and from a safe distance.

Networked computers need to be secure.

Problem: Secure Architectures

Many flaws found in designs of security-critical system architectures, sometimes years after publication or use.

Spectacular Example (1997):

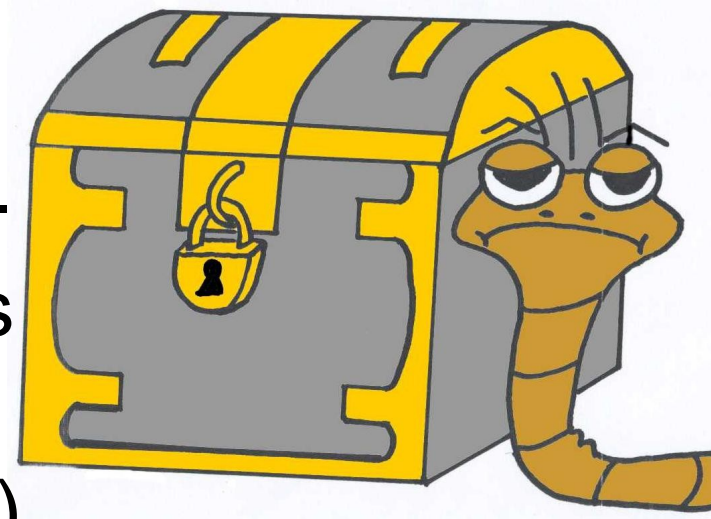
NSA hacker team breaks into U.S. Department of Defense computers and the U.S.electric power grid system. Simulates power outages and 911 emergency telephone overloads in Washington, D.C..

Causes

Security often compromised by circumventing security mechanisms within the architecture.

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (R. Needham).

Proposed Solution



Increase security of architectures with bounded investment in time, costs (crucial for industry).

- Consider architectural design artefacts arising in industrial development of security-critical systems (e.g. UML models).
- Tool-supported theoretically sound efficient automated security analysis.

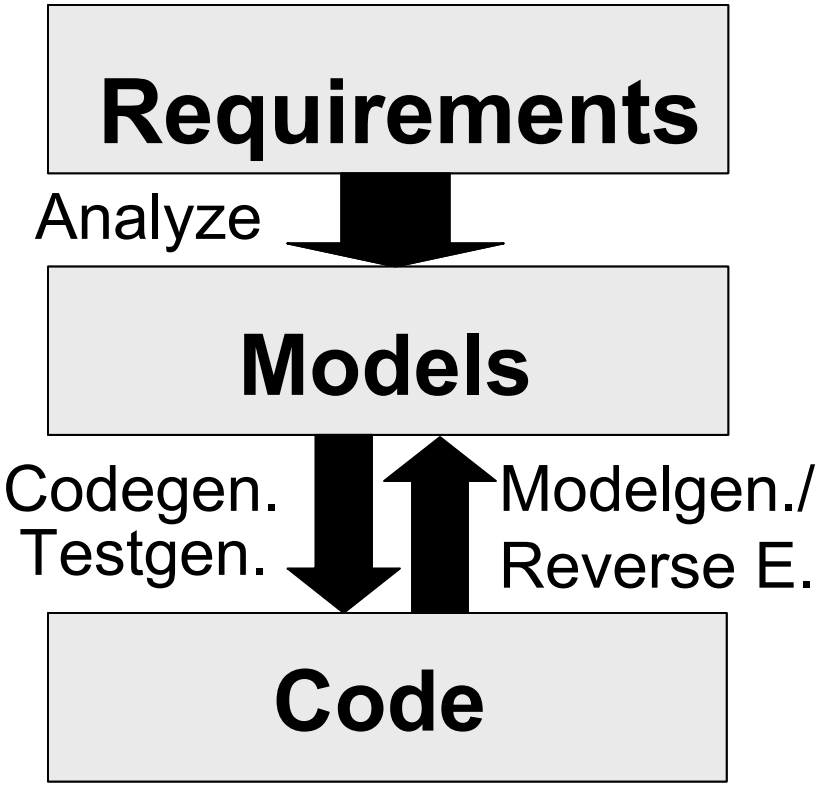
➔ *Model-based Security Engineering*

Model-based Security Engineering

Combined strategy:

- Analyze models automatically against security requirements.
- Generate code (or tests) from models automatically.
- Generate models from code to get changes (or analyze legacy systems).

Goal: model-based = source-based.



Idea notation-independent. Here: use UML.

Why UML ?

Seemingly de-facto standard in industrial modeling. Large number of developers trained in UML.

Increasingly used as architecture description language (ADL).

Relatively precisely defined (given the user community).

Many tools in development (also for code-generation, testing, reverse engineering, simulation, transformation).

UMLsec: Goals

Extension for secure systems development.

- evaluate UML specifications for weaknesses in design
- encapsulate established rules of prudent secure engineering as checklist
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- make certification cost-effective

UMLsec: How

Recurring security requirements, adversary scenarios, concepts offered as stereotypes with tags on component-level.

Use associated constraints to verify specifications using automated theorem provers and indicate possible weaknesses.

Ensures that UML specification provides desired level of security requirements.

Link to code via round-trip engineering etc.

≪Internet≫, ≪encrypted≫, …

Kinds of communication links resp. system nodes.

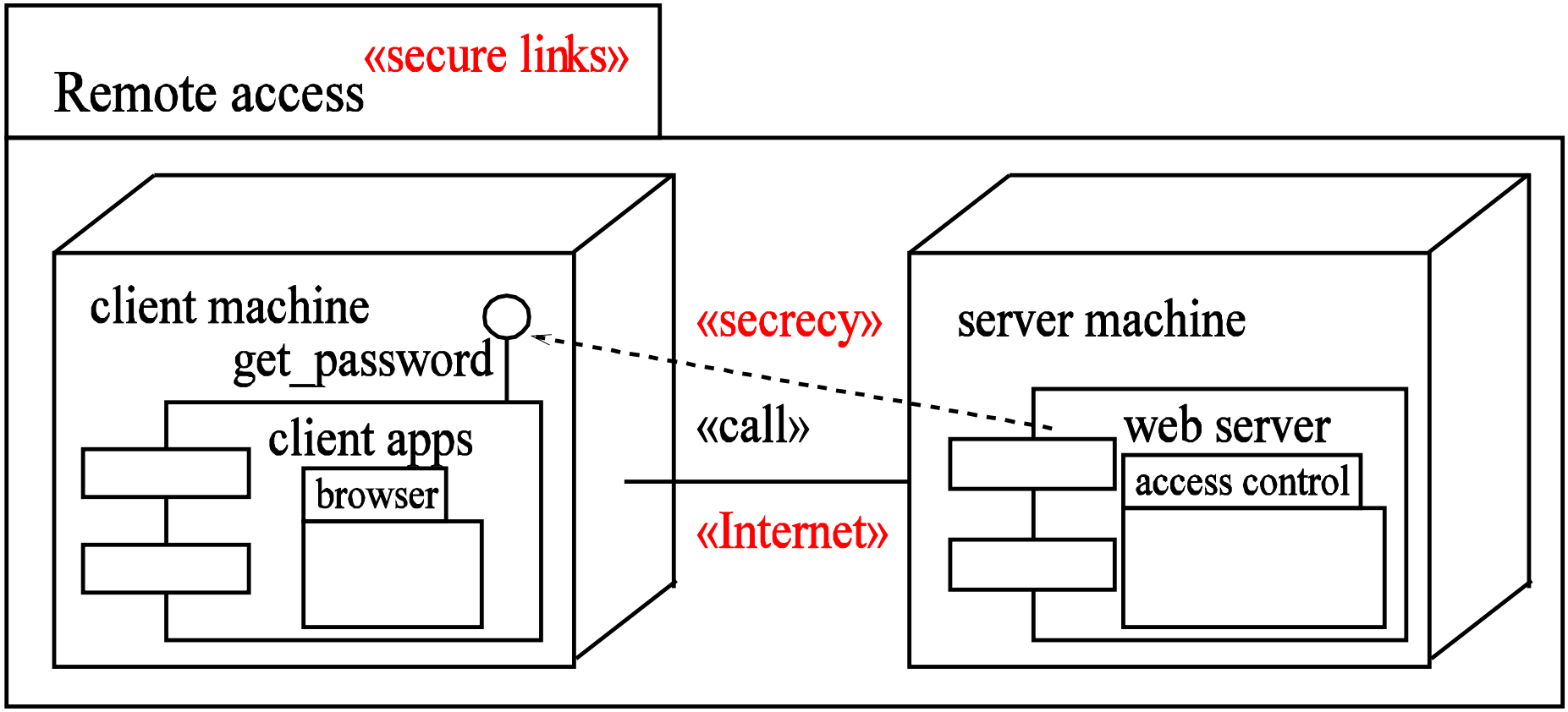
For adversary type A, stereotype s, have set

$Threats^{A}(s)$ {delete, read, insert, access} of actions that adversaries are capable of.

Default attacker:

Stereotype	$Threats^{default}(s)$
Internet	{delete, read, insert}
encrypted	{delete}
LAN	∅
smart card	∅

Example ≪secure links≫



Given default adversary type, is ≪secure links≫ provided ?

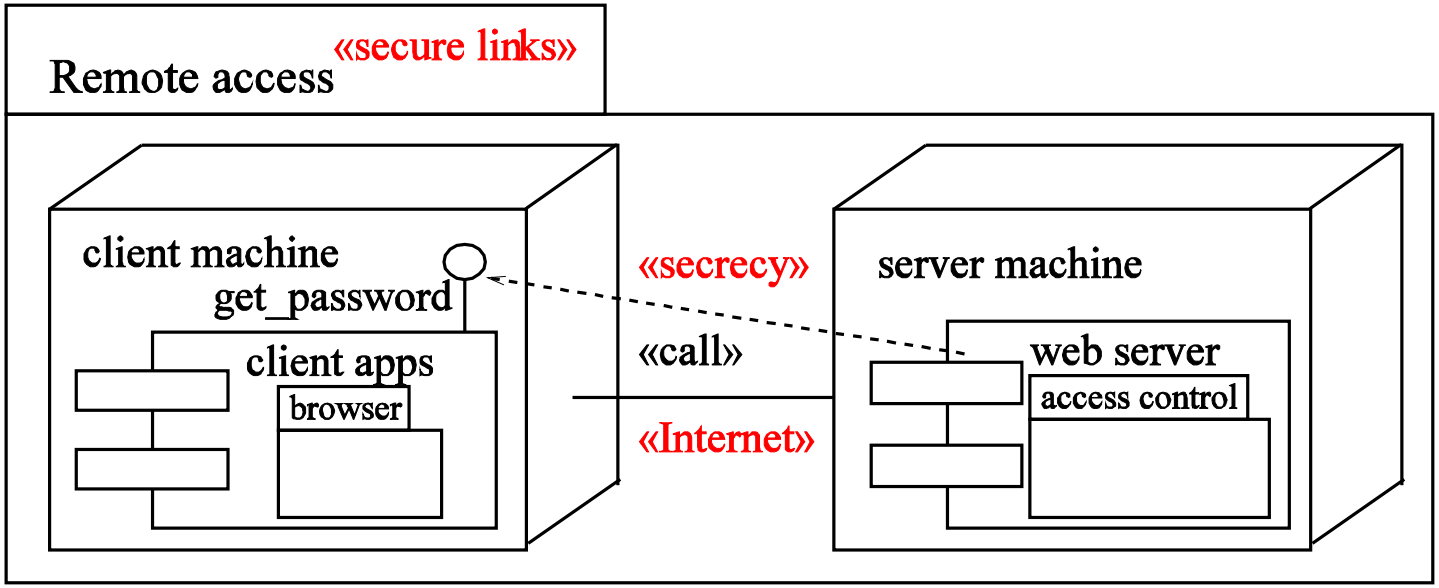
≪secure links≫

Ensures that physical layer meets security requirements on communication.

Constraint: for each dependency *d* with stereotype *s* {≪secrecy≫, ≪integrity≫} between components on nodes $n \neq m$, have a communication link *l* between *n* and *m* with stereotype *t* such that

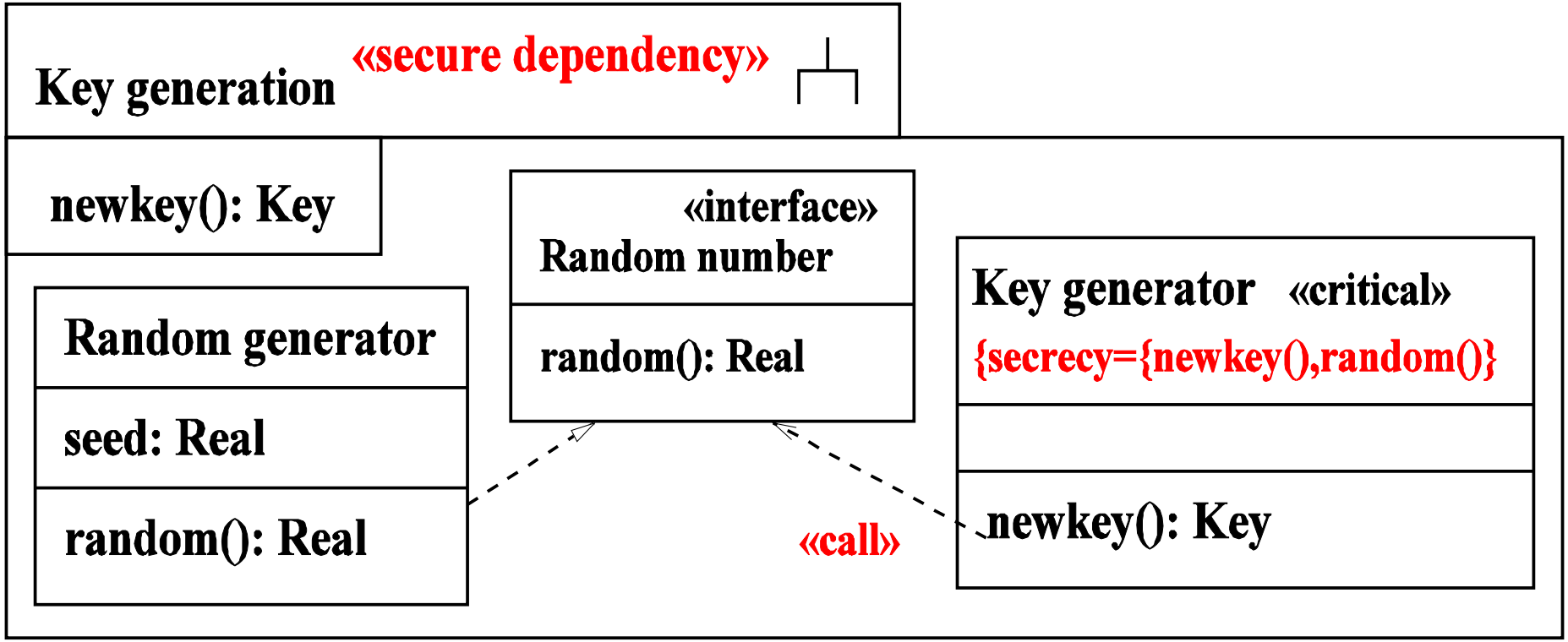
- if *s* = ≪secrecy≫: have read $\notin$ Threats_A *(t)*.
- if *s* = ≪integrity≫: have insert $\notin$ Threats_A *(t)*.

Example ≪secure links≫



Given default adversary type, constraint for stereotype ≪secure links≫ violated: According to the $\text{Threats}_{\text{default}}$(Internet) scenario, ≪Internet≫ link does not provide secrecy against default adversary.

Example ≪secure dependency≫



≪secure dependency≫provided ?

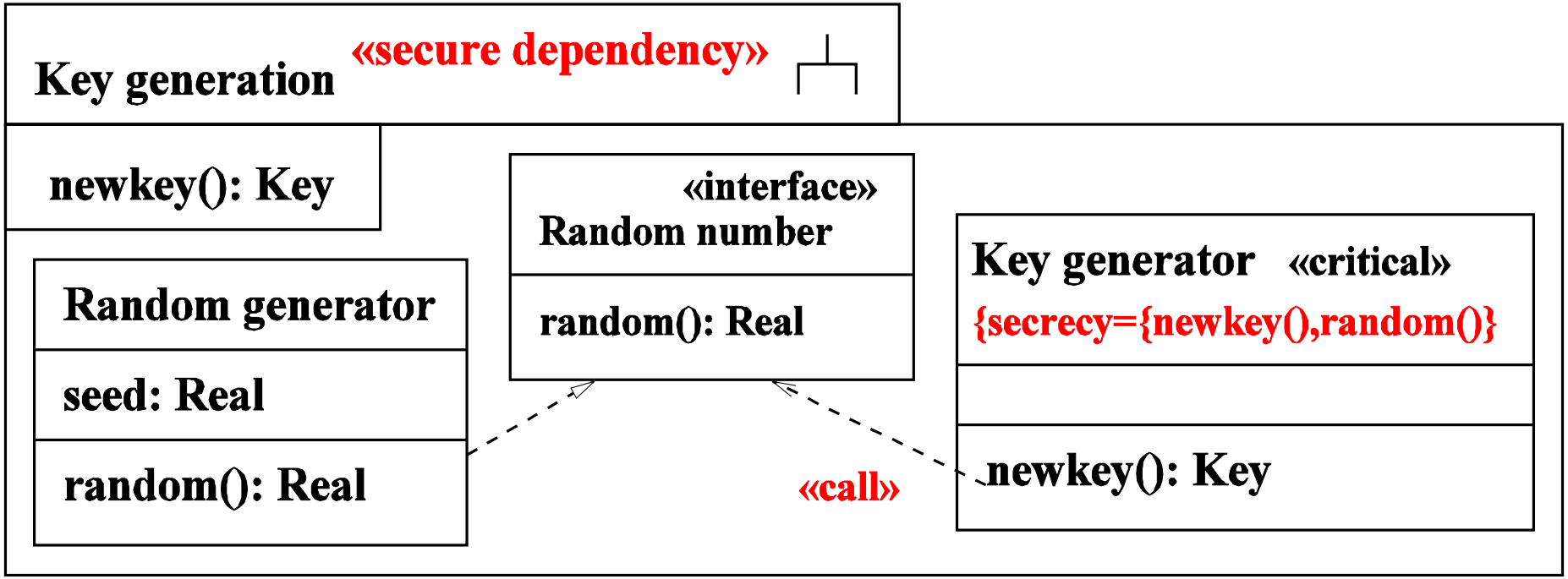
≪secure dependency≫

Ensure that ≪call≫ and ≪send≫ dependencies between components respect security requirements on communicated data given by tags {secrecy}, {integrity}.

Constraint: for ≪call≫ or ≪send≫ dependency from *C* to *D* (and similarly for {integrity}):

- Msg in *D* is {secrecy} in *C* if and only if also in *D*.
- If msg in *D* is {secrecy} in *C*, dependency stereotyped ≪secrecy≫.

Example ≪secure dependency≫



Violates ≪secure dependency≫: Random generator and ≪call≫ dependency do not give security level for random() to key generator.

Example ≪data security≫

«data security»

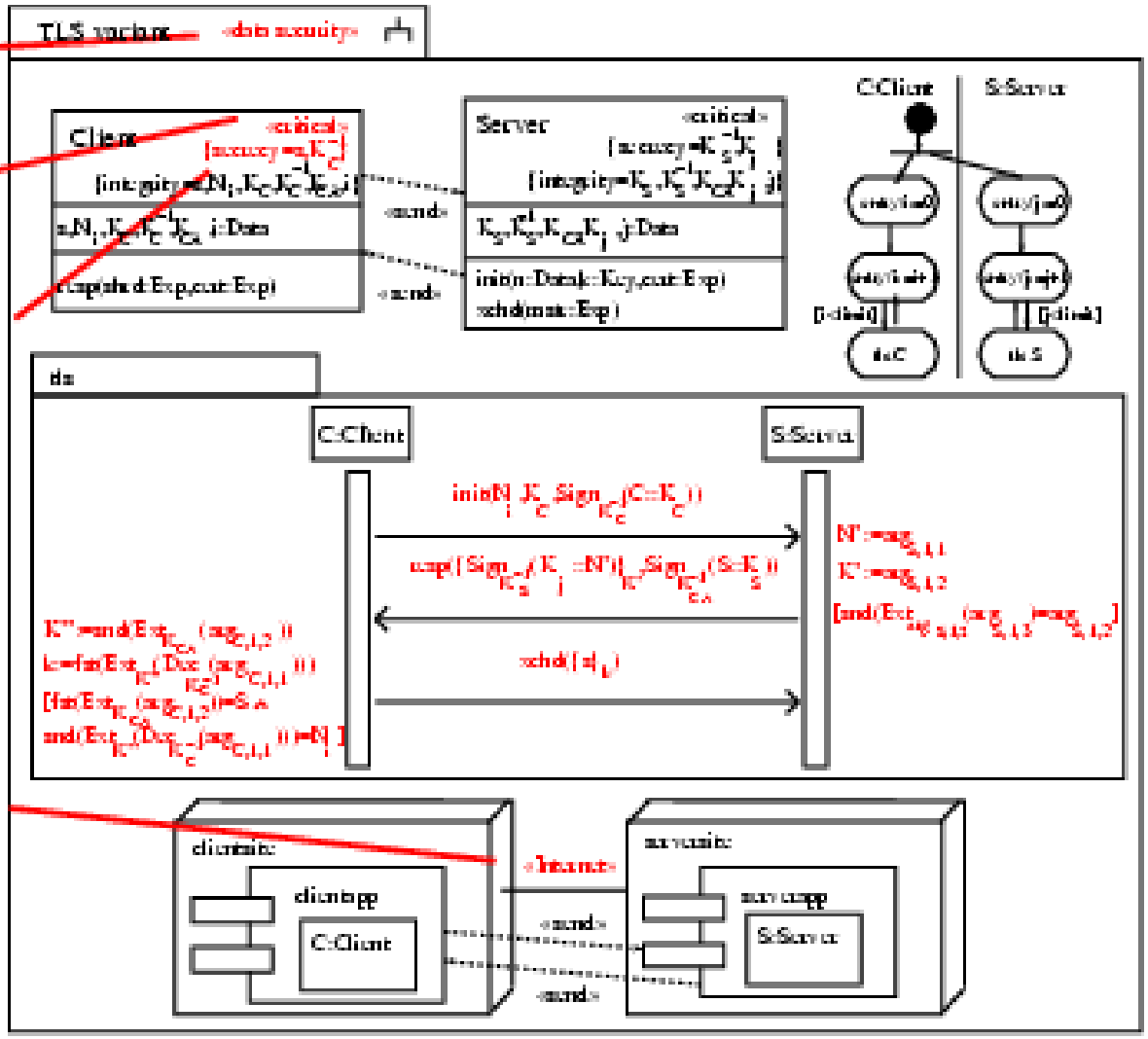
«critical»

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

Variant of TLS (INFOCOM`99).

≪data security≫ against default adversary provided ?

«Internet»



≪data security≫

Security requirements of data marked ≪critical≫ enforced against threat scenario from deployment diagram.

Constraints:

Secrecy of {secrecy} data preserved.

Integrity of {integrity} data preserved.

Example ≪data security≫

«data security»

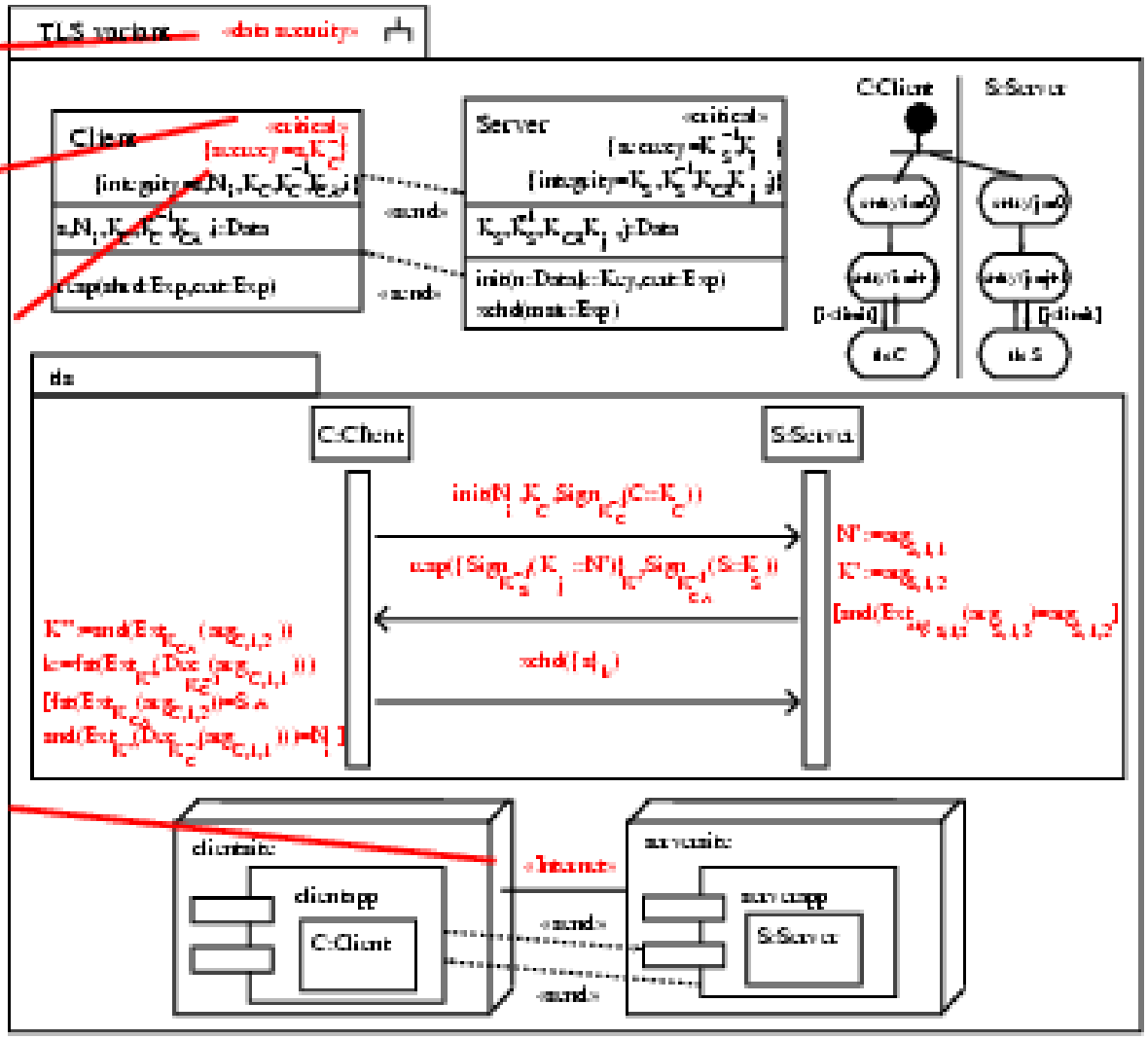
«critical»

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

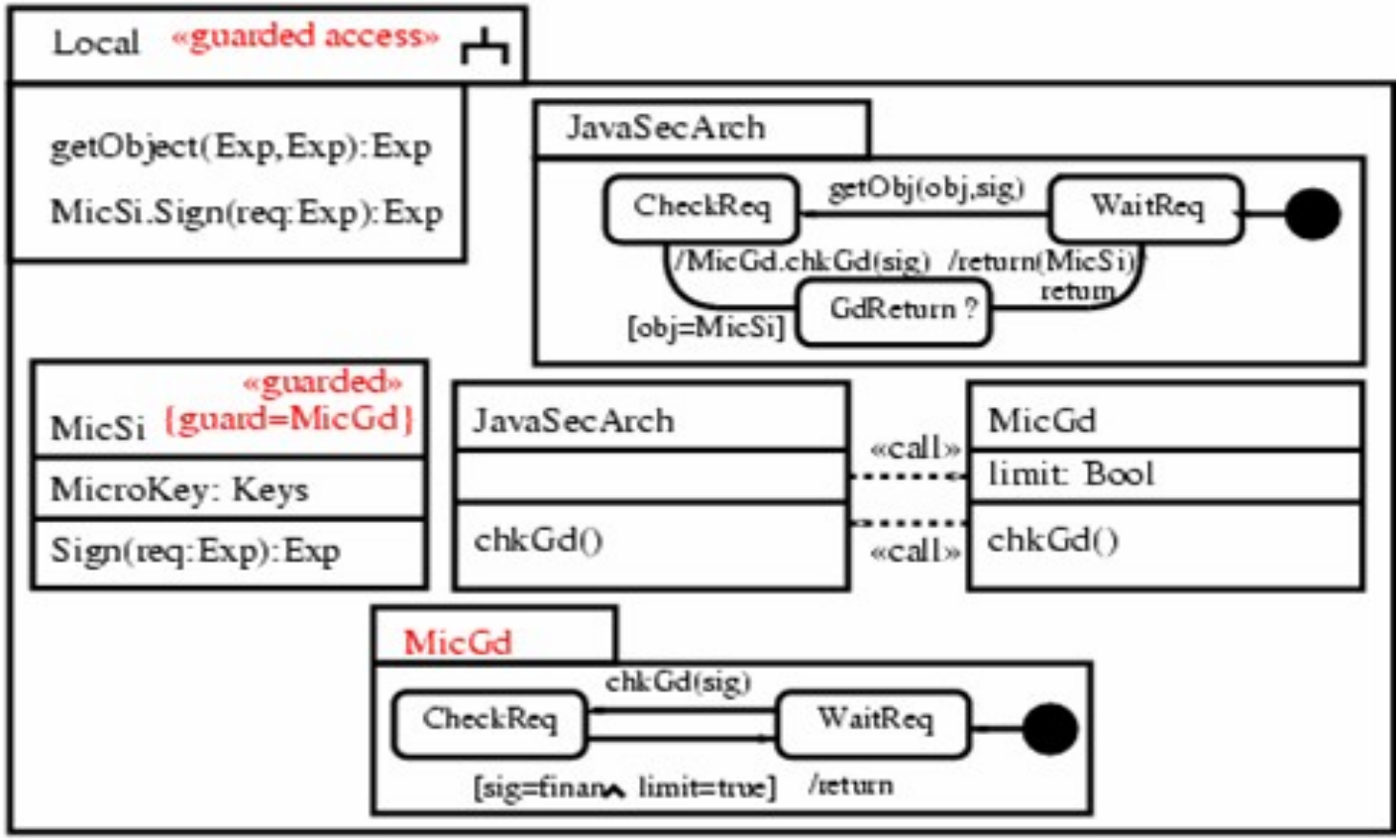
Variant of TLS (INFOCOM`99).

Violates {secrecy} of s against default adversary.

«Internet»



Example ≪guarded access≫



Provides ≪guarded access≫:

Access to MicSi protected by MicGd.

≪guarded access≫

Ensures that in Java, ≪guarded≫ classes only accessed through {guard} classes.

Constraints:

- References of ≪guarded≫ objects remain secret.
- Each ≪guarded≫ class has {guard} class.

Concepts covered by UMLsec

Security requirements: ≪secrecy≫,…

Threat scenarios: Use $\text{Threats}_{\text{adv}}(\text{ster})$.

Security concepts: For example ≪smart card≫.

Security mechanisms: E.g. ≪guarded access≫.

Security primitives: Encryption built in.

Physical security: Given in deployment diagrams.

Security management: Use activity diagrams.

Technology specific: Java, CORBA security.

Security Analysis

Following Dolev, Yao (1982): To analyze system, verify against attacker model from threat scenarios in deployment diagrams who

- may participate in some protocol runs,
- knows some data in advance,
- may intercept messages on some links,
- injects messages that it can produce in some links
- may access certain nodes.

Secure Architecture Patterns

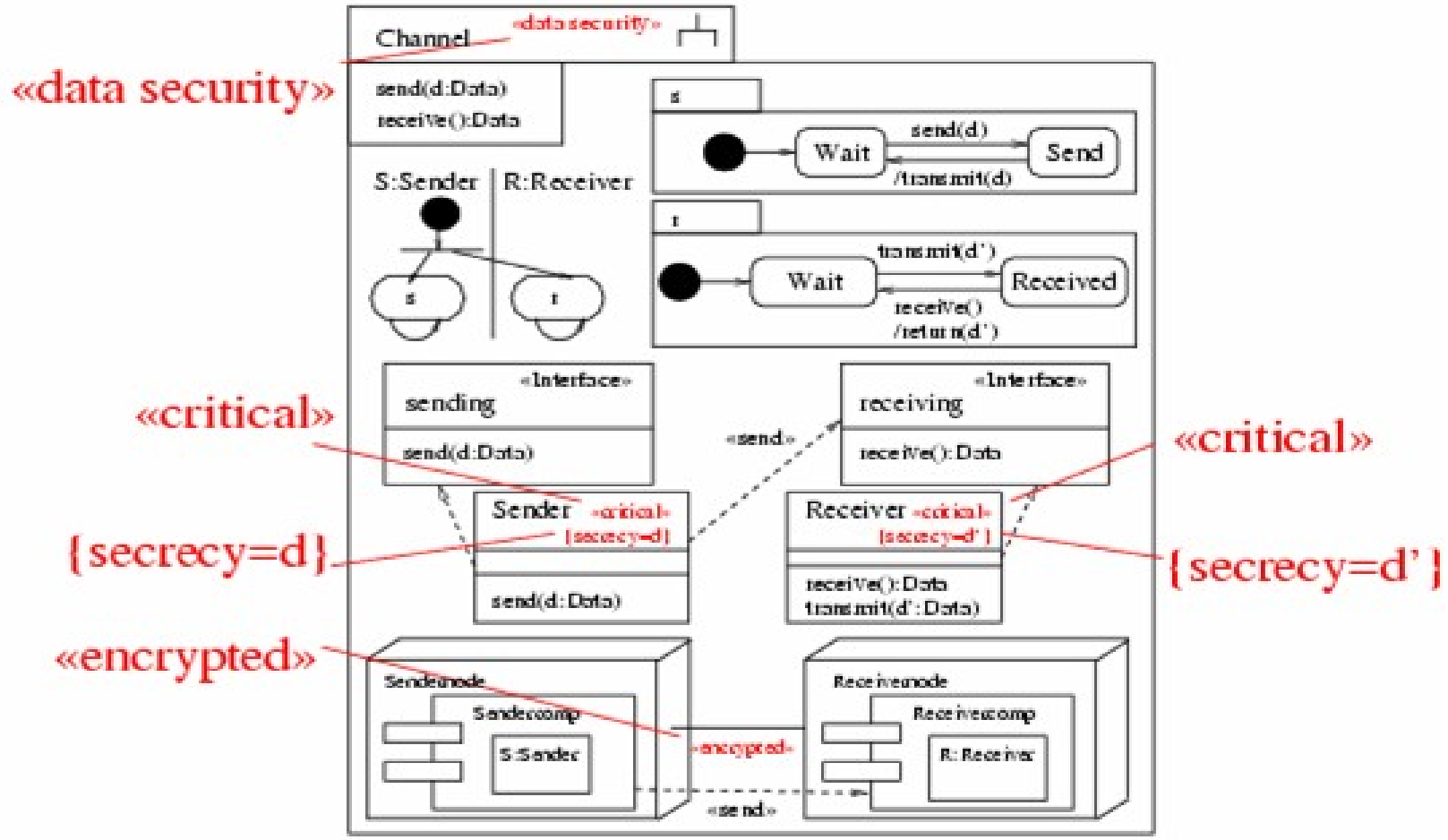
Architectural design patterns (Buschmann et al. 1996). Apply to security.

Example: Architecural primitive: Secure channel.

- Define a secure channel abstraction.
- Define concrete secure channel (protocol).
- Show simulates the abstraction.

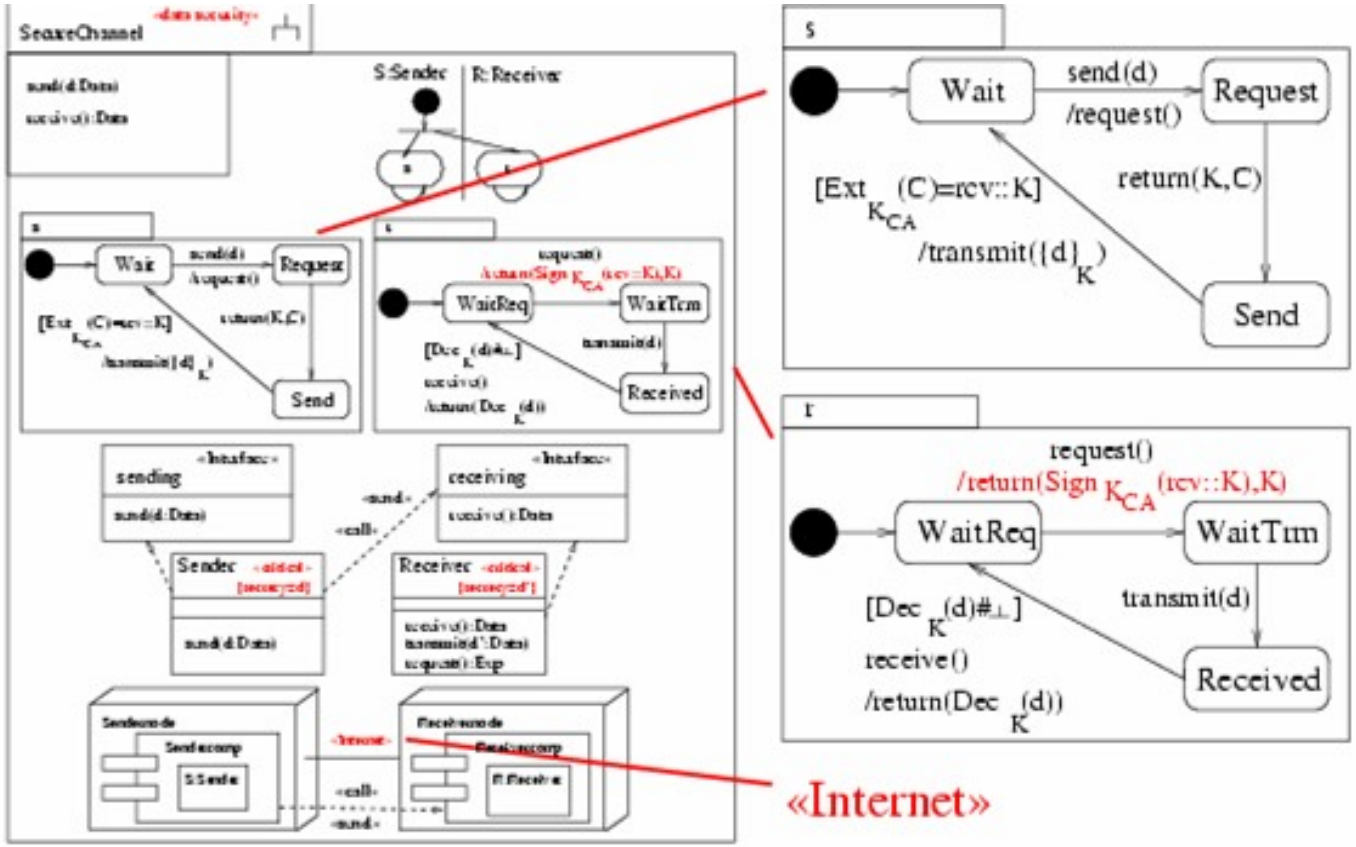
Give conditions under which it is secure to substitute channel abstractions by concrete protocols.

Secure Channel Pattern: Problem



To keep *d* secret, must be sent encrypted.

Secure Channel Pattern: (Toy) Solution



Exchange certificate and send encrypted data over Internet.

Secure Channel Abstraction

Show concrete channel is delayed equivalent to abstract channel, given a reasonable adversary model.

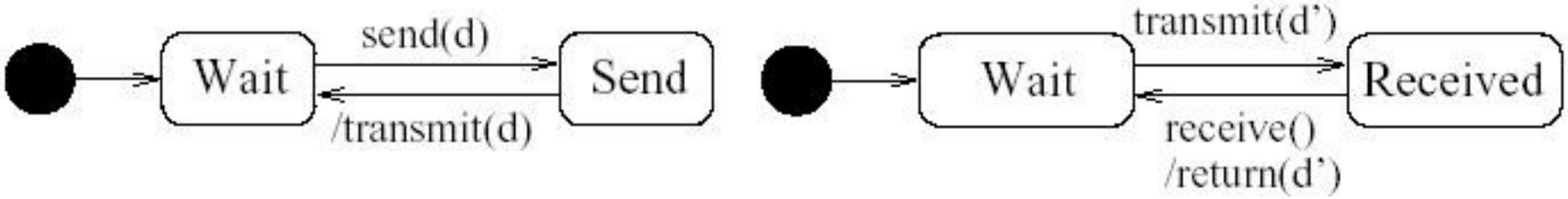
Use notion of delayed bisimulation.

Secure Channel Ideal

„Ideal“ of a secure channel:

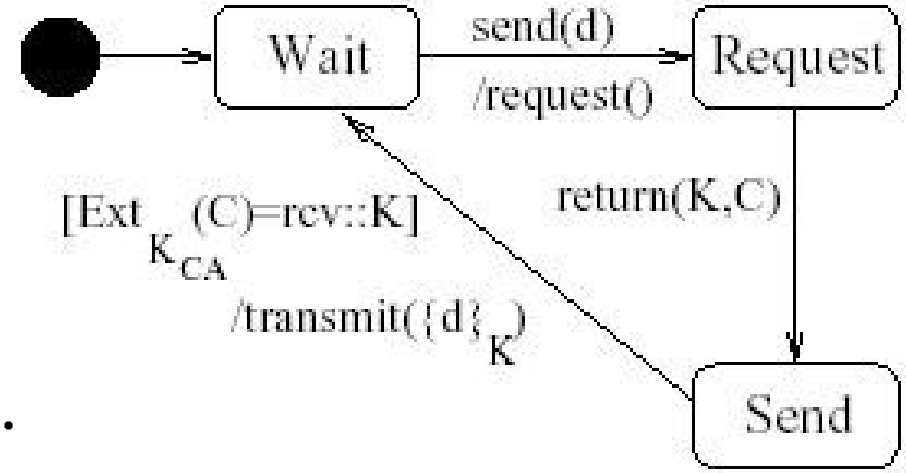
$$S = send(d).\overline{transmit}(s).S$$
$$R = transmit(d).\overline{receive}(d).R$$

Take $S \otimes^{\mathcal{I}} R$ for $\mathcal{I}$:={send,receive} as secure channel abstraction. Trivially secure in absence of adversaries.



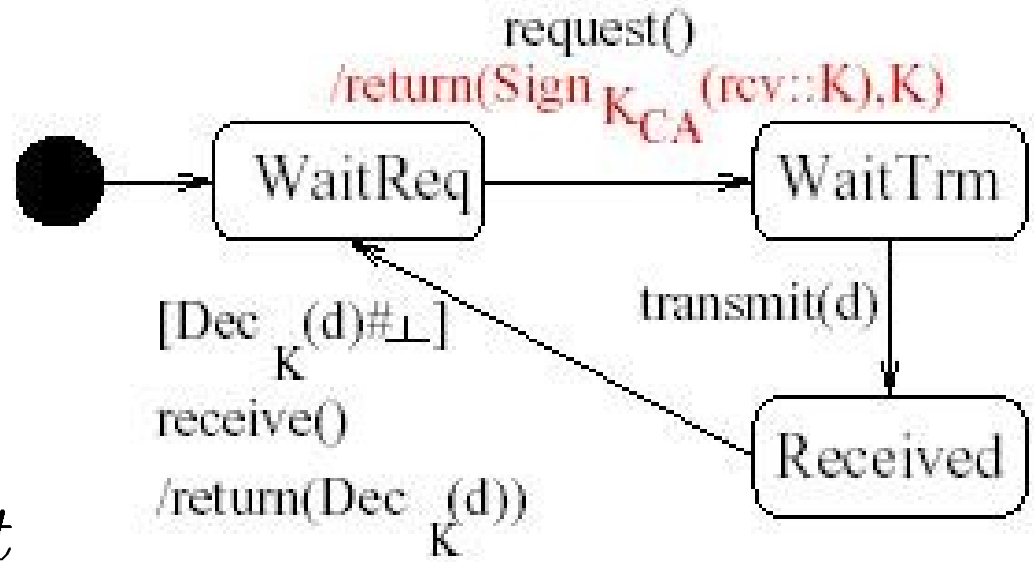
Concrete Secure Channel

Simple security protocol: encrypt under exchanged session key



$$
\begin{aligned}
S' \quad := \quad & \mathbf{iter}_i(send(d).\\
& \bar{request}.return(C).\\
& \mathbf{if}\ \mathbf{head}(\mathcal{E}xt_{K_R^{-1}}(\mathcal{D}ec_{K_S^{-1}}(C))) \in\\
& \mathbf{Keys} \wedge \mathbf{tail}(\mathcal{E}xt_{K_R}(\mathcal{D}ec_{K_S^{-1}}(C))) = j\\
& \mathbf{then}\ \bar{transmit}(\{d :: i\}_K)
\end{aligned}
$$

Concrete Secure Channel II



$$
\begin{aligned}
R' \;:=\; & \mathbf{iter}_j(request \\
& \bar{return}(\{\mathcal{S}ign_{K_R^{-1}}(K_j :: j)\}_{K_S}. \\
& transmit(E).receive. \\
& \mathbf{if}\ \mathbf{tail}(\mathcal{D}ec_{K_j}(E)) = j \\
& \mathbf{then}\ \bar{return}(\mathbf{head}(\mathcal{D}ec_{K_j}(E)))
\end{aligned}
$$

Bisimulation

A binary relation R on processes is a bisimulation iff $P\ R\ Q$ implies that for all actions α,

- if $P \rightarrow^{\alpha} P'$ then exists $Q \rightarrow^{\alpha} Q'$ with $P'\ R\ Q'$ and
- if $Q \rightarrow^{\alpha} Q'$ then exists $P \rightarrow^{\alpha} P'$ with $P'\ R\ Q'$.

P, Q are bisimilar if there exists a bisimulation R with $P\ R\ Q$.

Faithful representation ?

Is $(R'||S')\otimes^{\mathcal{S}}\mathcal{A}$ bisimilar to $S\otimes^{\mathcal{S}}R$?

No: delay possible. But:

Theorem. Suppose A does not know the messages *send, receive* nor any value in $\{K(S)^{-1},K(R)^{-1}\}\cup\{K_n,\{x::n\}_{Kn}:x\in Exp\wedge n\in\mathbb{N}\}$ nor $Sign_{K(R)}^{-1}(K'::n)$ unless $K'=K_n$. Then $(R'||S')\otimes^{\mathcal{S}}A$ is bisimilar to $(S\otimes^{\mathcal{S}}R)\otimes A$.

Theorem. $(R'||S')$ preserves secrecy of d against such $\mathcal{A}$.

Layered Security Architectures

Layer *on top* uses security *below*.

client authenticity

→

confidentiality, integrity, server authenticity

→

=

confidentiality, … + client authenticity

→

?

Security properties additive ?

Here: Bank application

Security analysis of web-based banking application, to be put to commercial use (clients fill out and sign digital order forms).

Layered security protocol (first layer: SSL protocol, second layer: client authentication protocol)

Security requirements:

- confidentiality
- authenticity

Insight

Layered Security Architectures indeed additive wrt. security properties in this particular case.

Generalize to classes of systems and security requirements.

Java Security Architecture

Originally (JDK 1.0): sandbox.

Too simplistic and restrictive.

JDK 1.2/1.3: more fine-grained security control, signing, sealing, guarding objects, . . .)

BUT: complex, thus use is error-prone.

Java Security policies

Permission entries consist of:

- protection domains (i. e. URL's and keys)
- target resource (e.g. files on local machine)
- corresponding permissions (e.g. read, write, execute)

Signed and Sealed Objects

Need to protect integrity of objects used as authentication tokens or transported across JVMs.

A SignedObject contains an object and its signature.

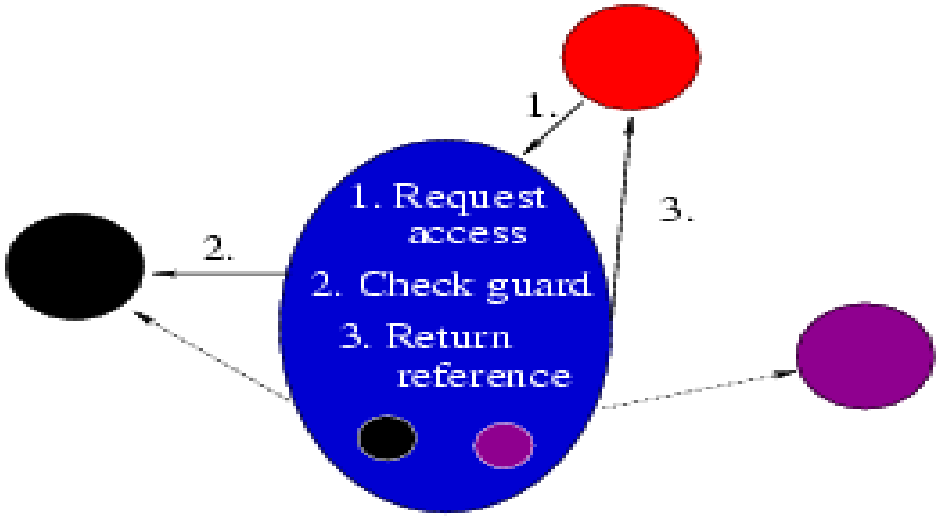
Similarly, need confidentiality.

A SealedObject is an encrypted object.

Guarded Objects

java.security.GuardedObject protects access to other objects.

- access controlled by getObject method
- invokes checkGuard method on the java.security.Guard that is guarding access
- If allowed: return reference. Otherwise: SecurityException



Problem: Complexity

- Granting of permission depends on execution context.
- Access control decisions may rely on multiple threads.
- A thread may involve several protection domains.
- Have method doPrivileged() overriding execution context.
- Guarded objects defer access control to run-time.
- Authentication in presence of adversaries can be subtle.
- Indirect granting of access with capabilities (keys).

→ Difficult to see which objects are granted permission.

⇒ use UMLsec

Design Process

(1) Formulate access control requirements for sensitive objects.

(2) Give guard objects with appropriate access control checks.

(3) Check that guard objects protect objects sufficiently.

(4) Check that access control is consistent with functionality.

(5) Check mobile objects are sufficiently protected.

Reasoning

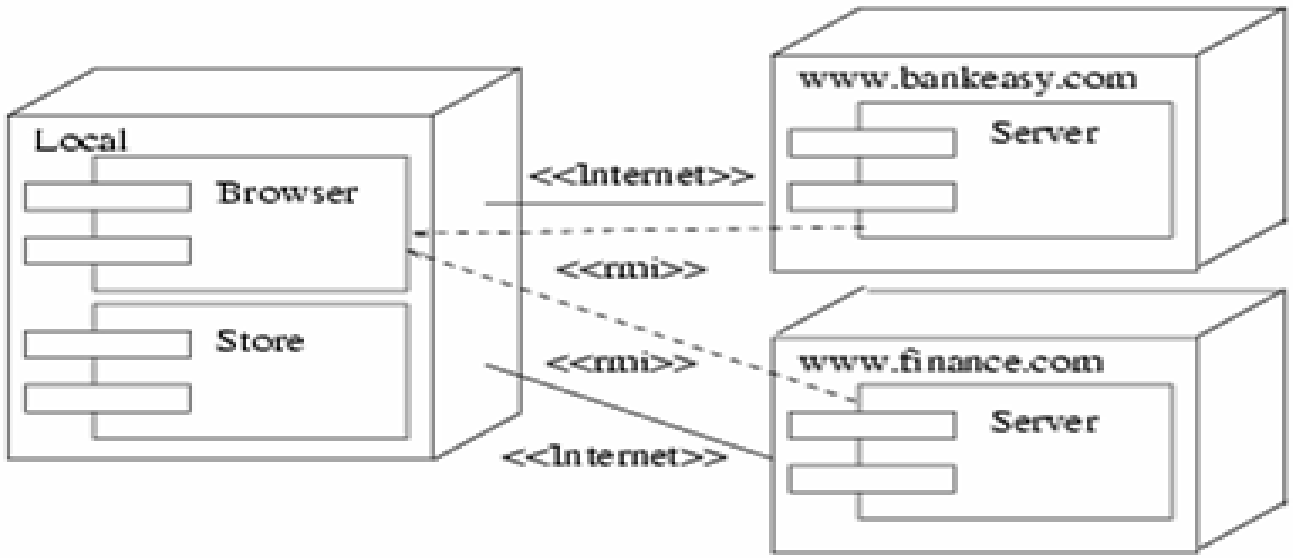
Theorem.

Suppose access to resource according to Guard object specifications granted only to objects signed with K.

Suppose all components keep secrecy of K.

Then only objects signed with K are granted access.

Example: Financial Application



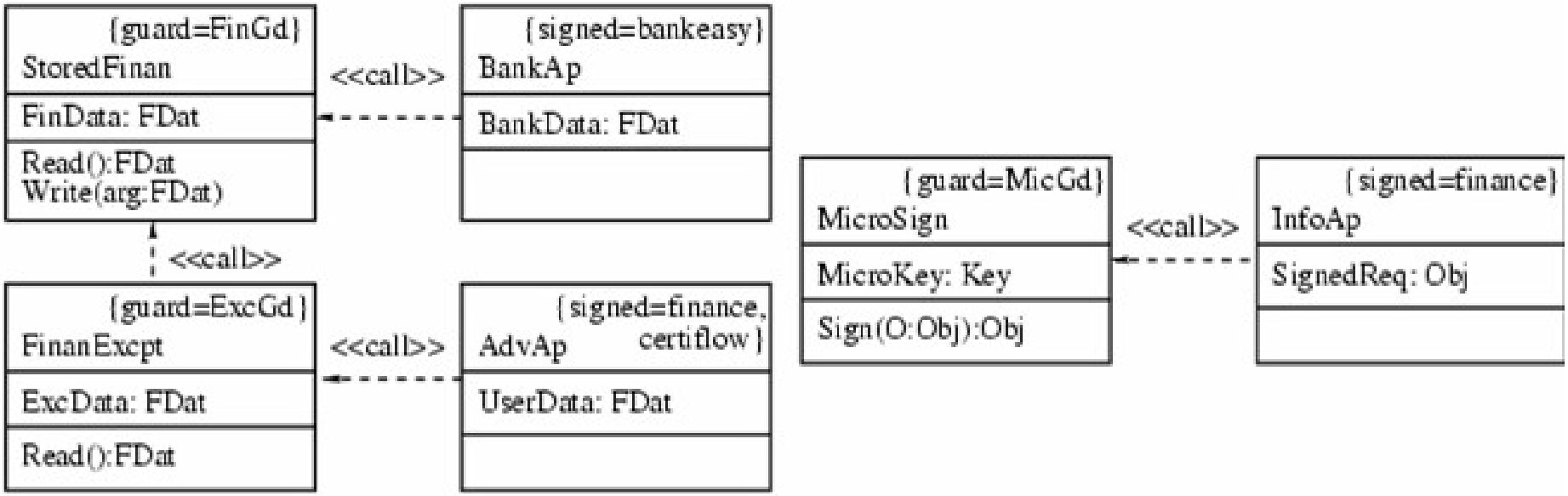
Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain Privileges (step1).

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

Financial Application: Class diagram

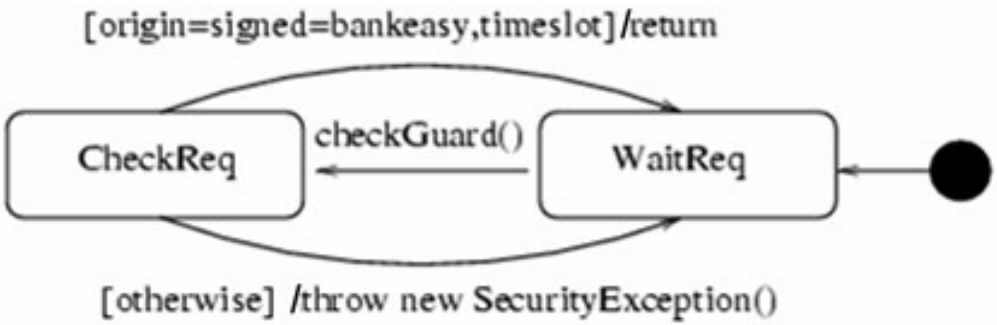
Sign and seal objects sent over Internet for Integrity and confidentiality.

GuardedObjects control access.

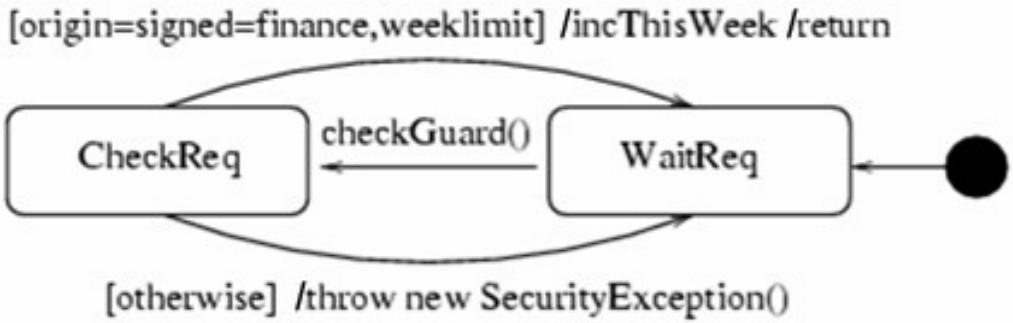


Financial Application: Guard objects (step 2)

timeslot true between 1pm and 2pm.



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



Financial Application: Validation

Guard objects give sufficient protection (step 3).

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

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

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

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

CORBA access control

Object invocation access policy controls access of a client to a certain object via a certain method.

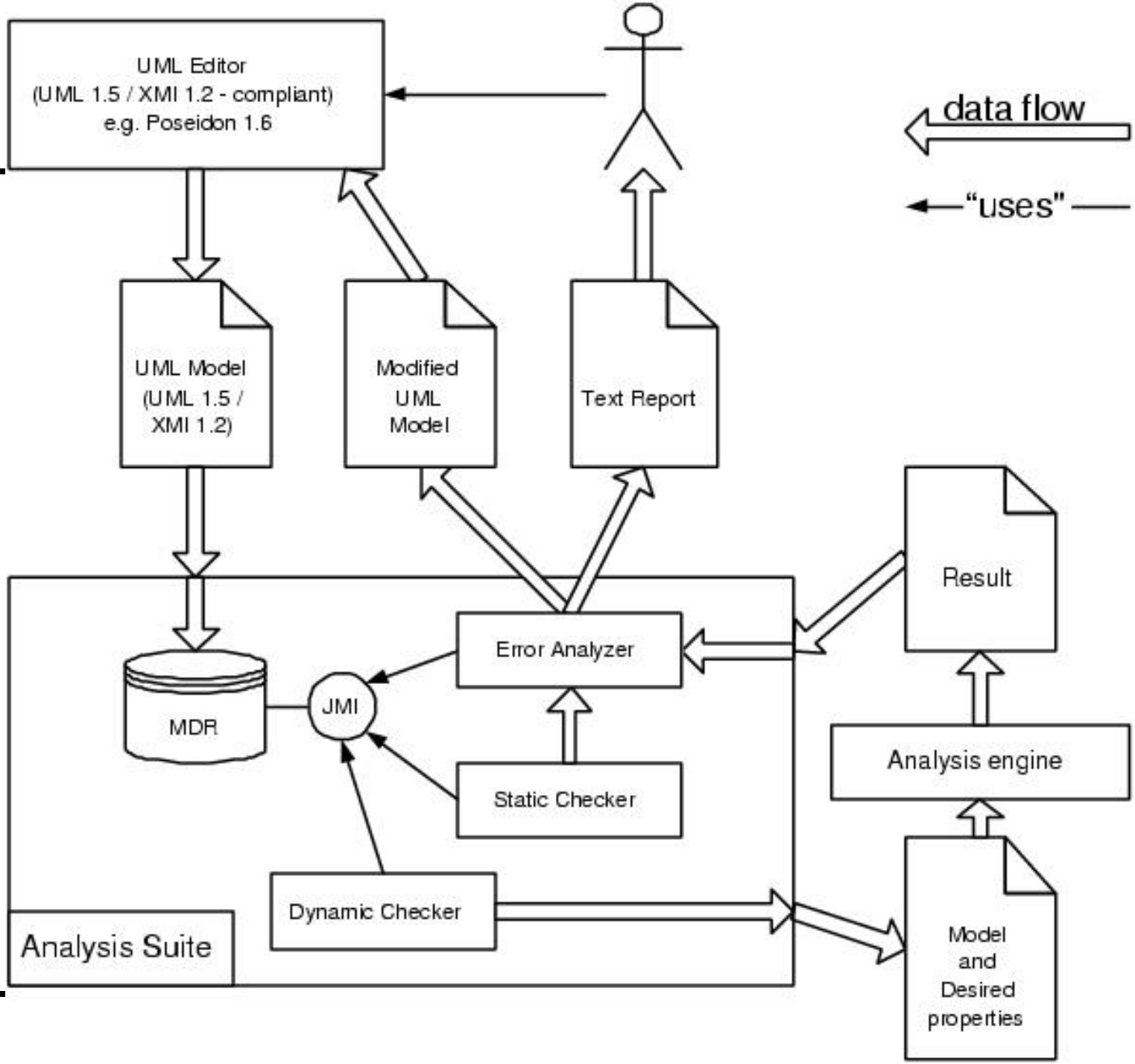
Realized by ORB and Security Service.

Use access decision functions to decide whether access permitted. Depends on

- called operation,
- privileges of the principals in whose account the client acts,
- control attributes of the target object.

Some Further UMLsec Applications

Secure Design Principles by Saltzer, Schroeder
Variant of the Internet security protocol TLS (SSL)
Common Electronic Purse Specification
Biometric authentication protocol for German Telekom
Analysis of SAP access control configurations for German bank
Telematic automobile emergency application of German car company
Electronic signature architecture of German insurance company
Electronic purse for Oktoberfest

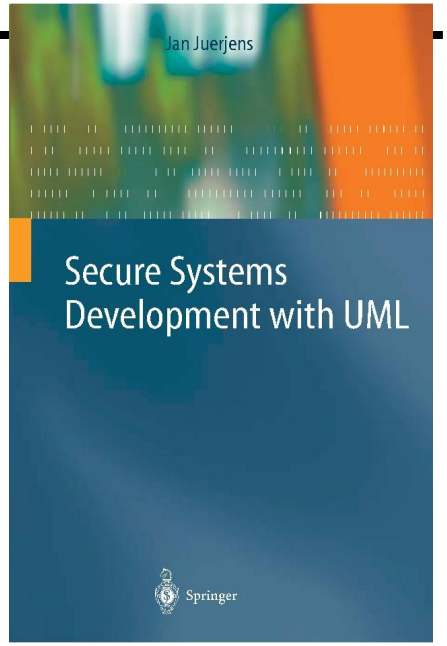

UML Editor
(UML 1.5 / XMI 1.2 - compliant)
e.g. Poseidon 1.6
data flow
"uses"
UML Model
(UML 1.5 /
XMI 1.2)
Modified
UML
Model
Text Report
Result
Error Analyzer
MDR
JMI
Static Checker
Analysis engine
Dynamic Checker
Analysis Suite
Model
and
Desired
properties

Conclusions

Model-based Security Engineering using UML:

- formally based approach
- automated tool support
- industrially used notation
- integrated approach (source-code, configuration data)

Resources



Jan Jürjens, Secure Systems Development with UML, Springer 04 (get Oct.)

Tutorials: Sept.: SAFECOMP (Potsdam), ASE (Linz), NODe (Erfurt). Oct.: UML (Lisabon). Nov.: ISSRE (Bretagne).

Spring School: May 2005, Carlos IV Univ. Madrid

Workshops: CSDUML@UML04, WITS05@POPL05

More information (papers, slides, tool etc.):
http://www.umlsec.org