Secure Enterprise Applications with UML

Jan Jürjens
(contributing P. Shabalin, S. Höhn, S. Meng)

Software & Systems Engineering
Informatics, TU Munich
Germany

juerjens@in.tum.de
http://www.jurjens.de/jan

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.

Problems

Many flaws found in designs of security-critical systems, 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 I

- Designing secure systems correctly is difficult.
  Even experts may fail:
  - Needham-Schroeder protocol (1978)
  - attacks found 1981 (Denning, Sacco), 1995 (Lowe)
- Designers often lack background in security.
- Security as an afterthought.

Causes II

Cannot use security mechanisms „blindly“:

- Security often compromised by circumventing (rather than breaking) them.
- Assumptions on system context, physical environment.

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

Difficulties

Exploit information spreads quickly.

No feedback on delivered security from customers.

Previous approaches

„Penetrate-and-patch“:

- insecure
- disruptive

Traditional formal methods: expensive.

- training people
- constructing formal specifications.

Goal: Security by design

Consider security

- from early on
- within development context
- taking an expansive view
- in a seamless way.

Secure design by model analysis.

Secure implementation by test generation.

Holistic view on Security

„An expansive view of the problem is most appropriate to help ensure that no gaps appear in the strategy“ (Saltzer, Schroeder 1975).

But „no complete method applicable to the construction of large general-purpose systems exists yet“ - since 1975.

Using UML

UML: unprecedented opportunity for high-quality critical systems development feasible in industrial context:

- De-facto standard in industrial modeling: large number of developers trained in UML.
- Relatively precisely defined (given the user community).
- Many tools in development (also for analysis, testing, simulation, transformation).

Challenges

- Adapt UML to critical system application domains.
- Correct use of UML in the application domains.
- Conflict between flexibility and unambiguity in the meaning of a notation.
- Improving tool-support for critical systems development with UML.

This tutorial

Background knowledge on using UML for critical systems development.

- UML basics, including extension mechanisms.
- Extensions of UML (UMLsec, UML-RT, ...)
- UML as a formal design technique.
- Tools.
- Case studies.

Concentrate on security-critical systems.

Roadmap

Prologue

UML

UMLsec: The profile

Security patterns

Case studies

Using Java security, CORBAsec

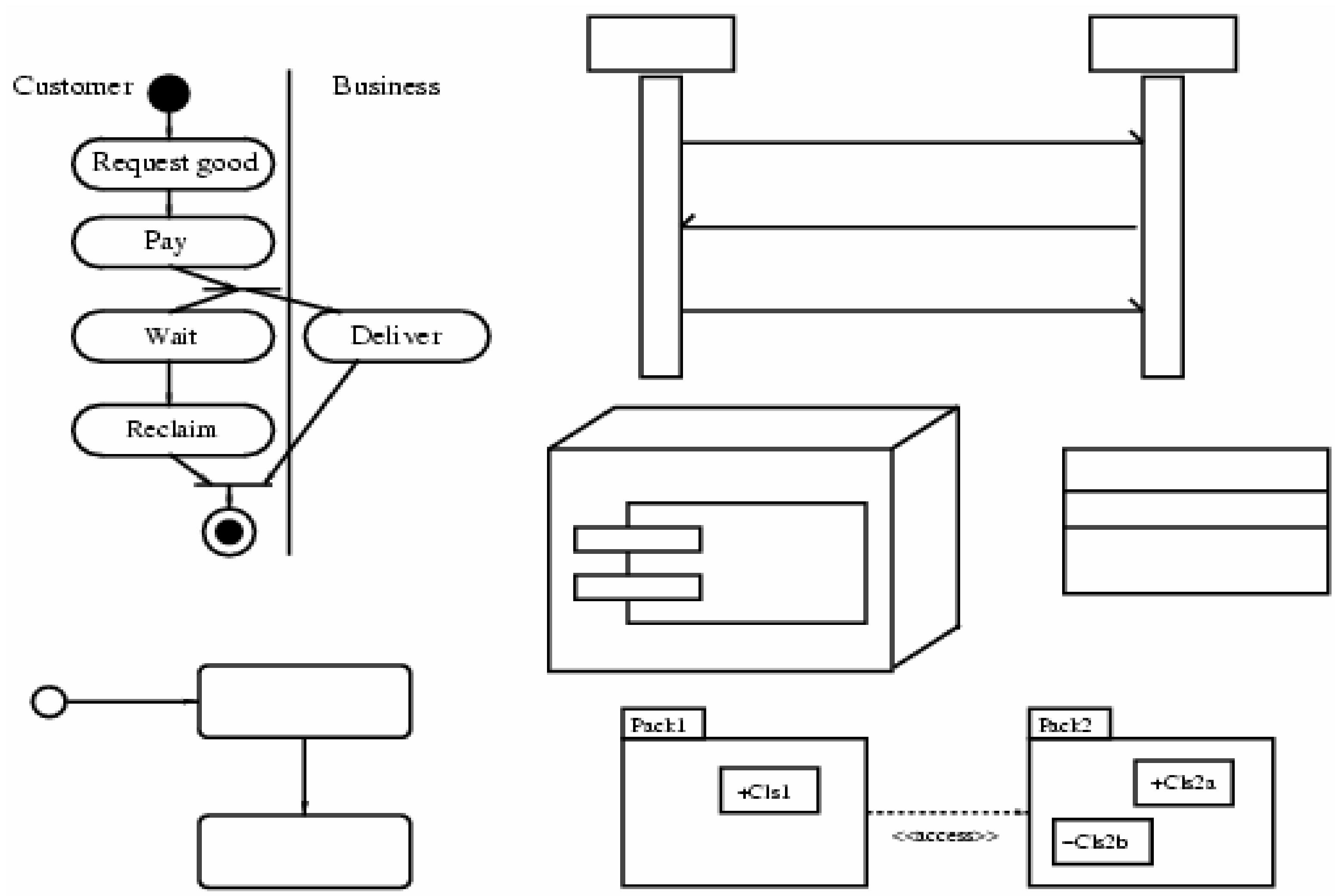
Tools

Using UML

Unified Modeling Language (UML):

- visual modelling language
- different views on a system
- high degree of abstraction possible
- de-facto industry standard (OMG)
- standard extension mechanisms

A glimpse at UML



Used fragment of UML

Activity diagram: flow of control between system components

Class diagram: data structure of the system

Sequence diagram: interaction between components by message exchange

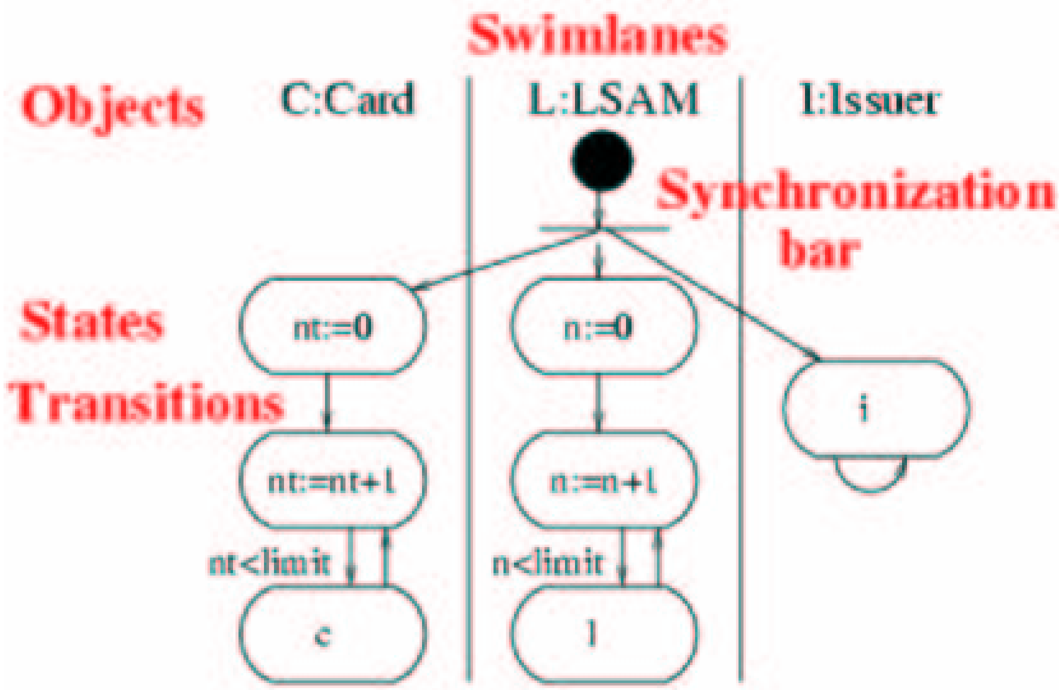
Statechart diagram: dynamic component behaviour

Deployment diagram: Components in physical environment

Package: collect system parts into groups

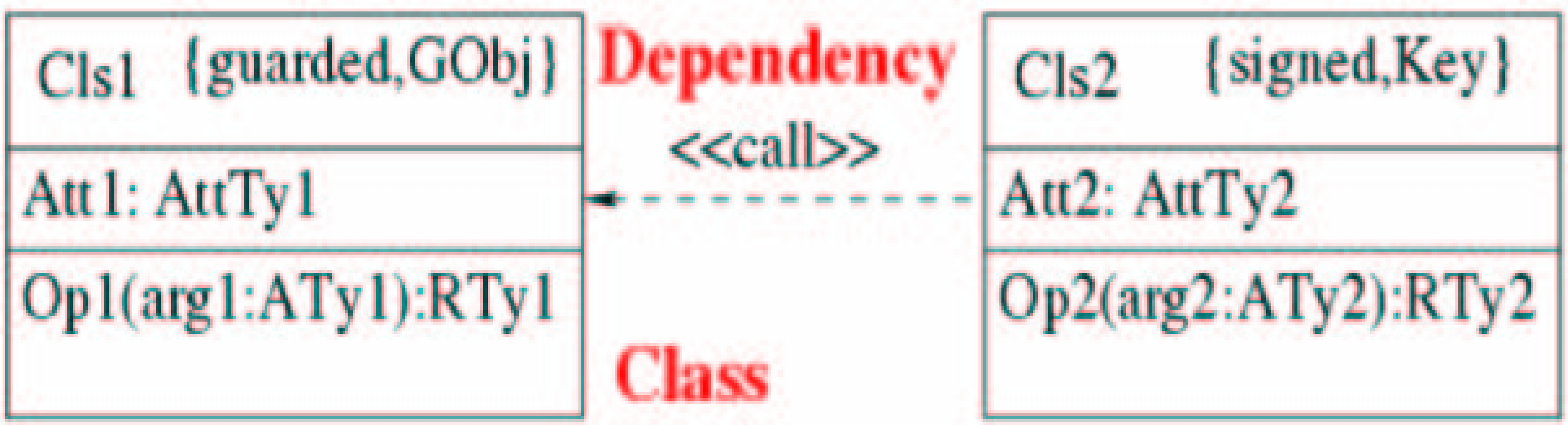
Current: UML 1.5 (released Mar 2003)

UML run–through: Activity diagrams



Specify the control flow between components within the system, at higher degree of abstraction than statecharts and sequence diagrams.

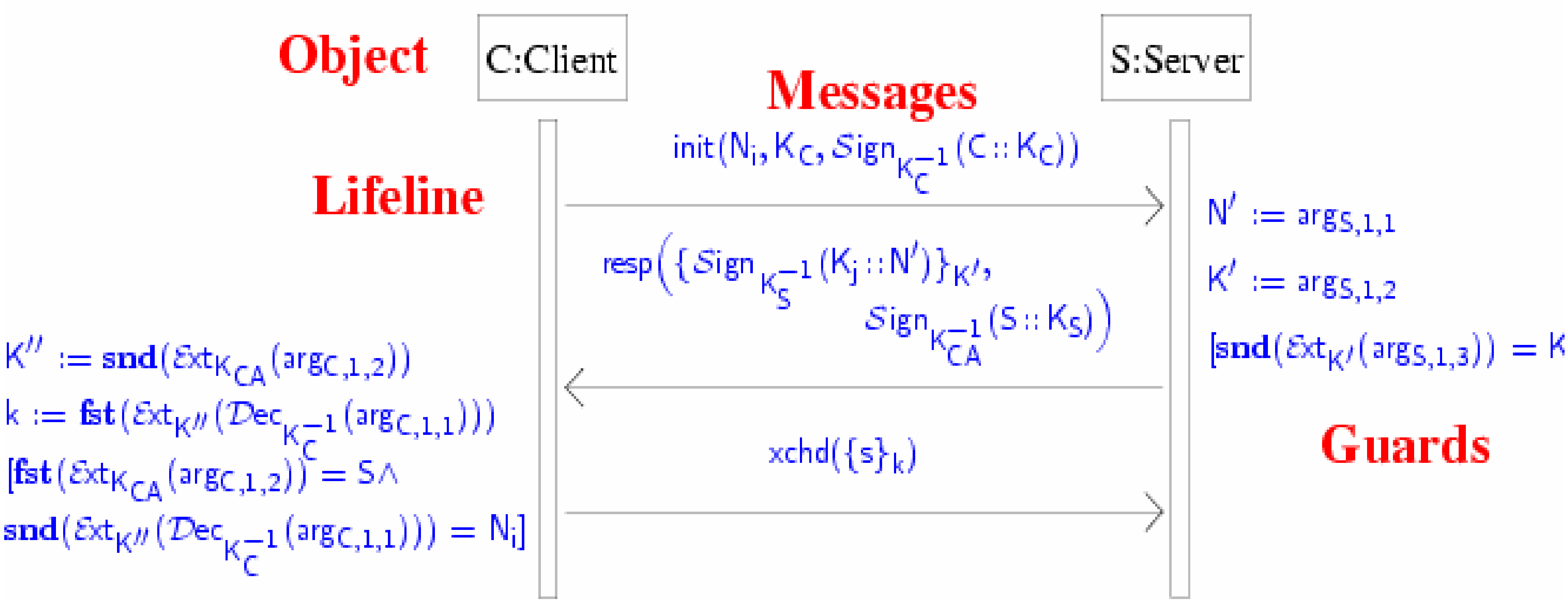
UML run-through: Class diagrams



Data structure of system.

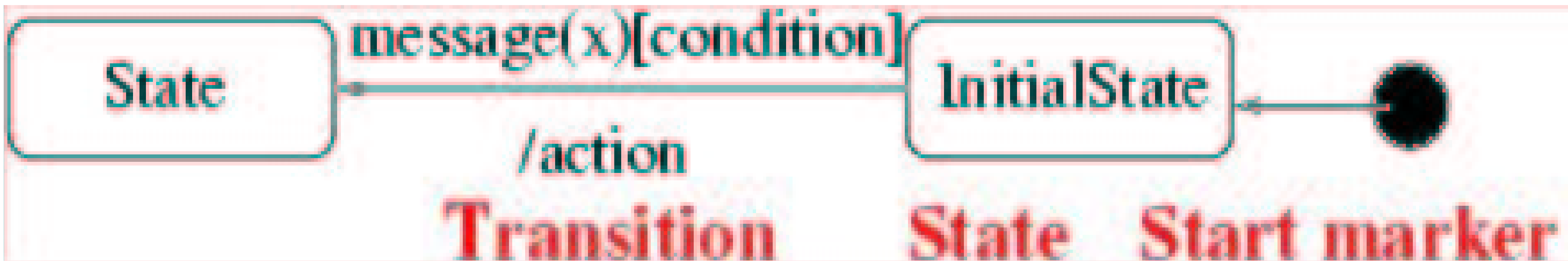
Components with attributes and operations/signals; relationships between components.

UML run-through: Sequence Diagrams



Describe interaction between system components via message exchange.

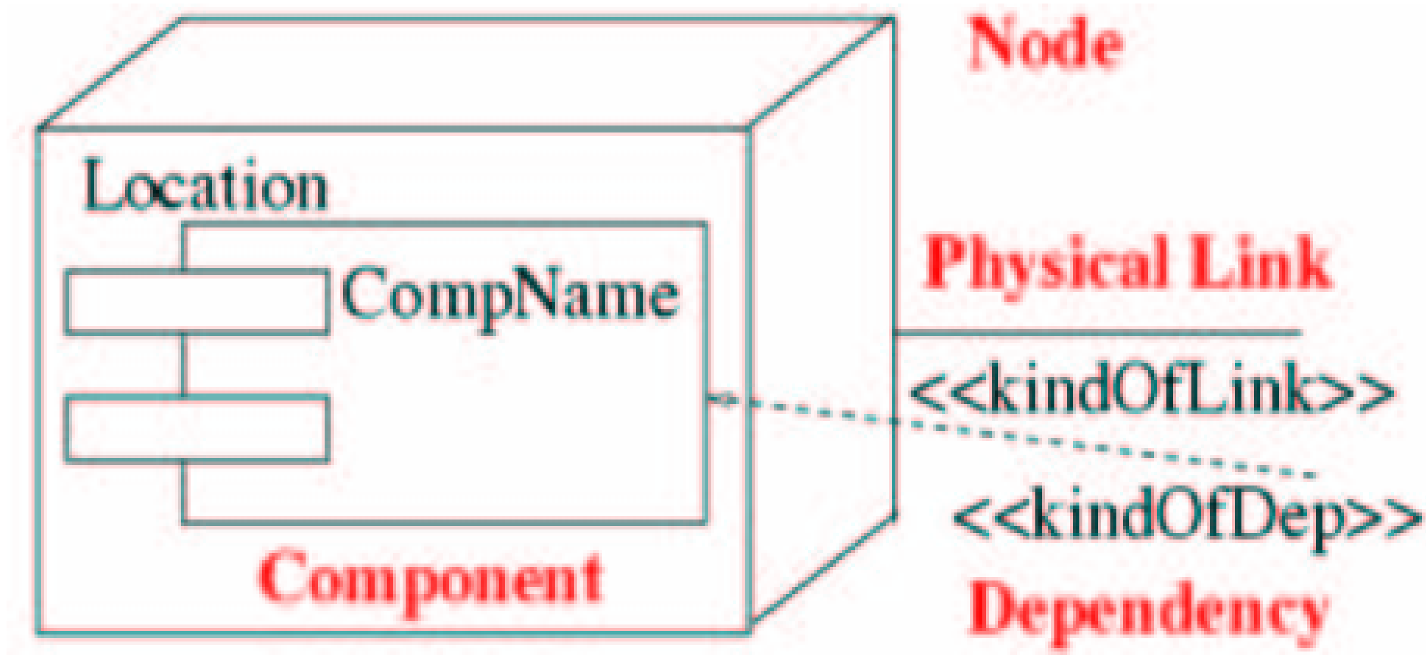
UML run-through: Statecharts



Dynamic behaviour of individual component.

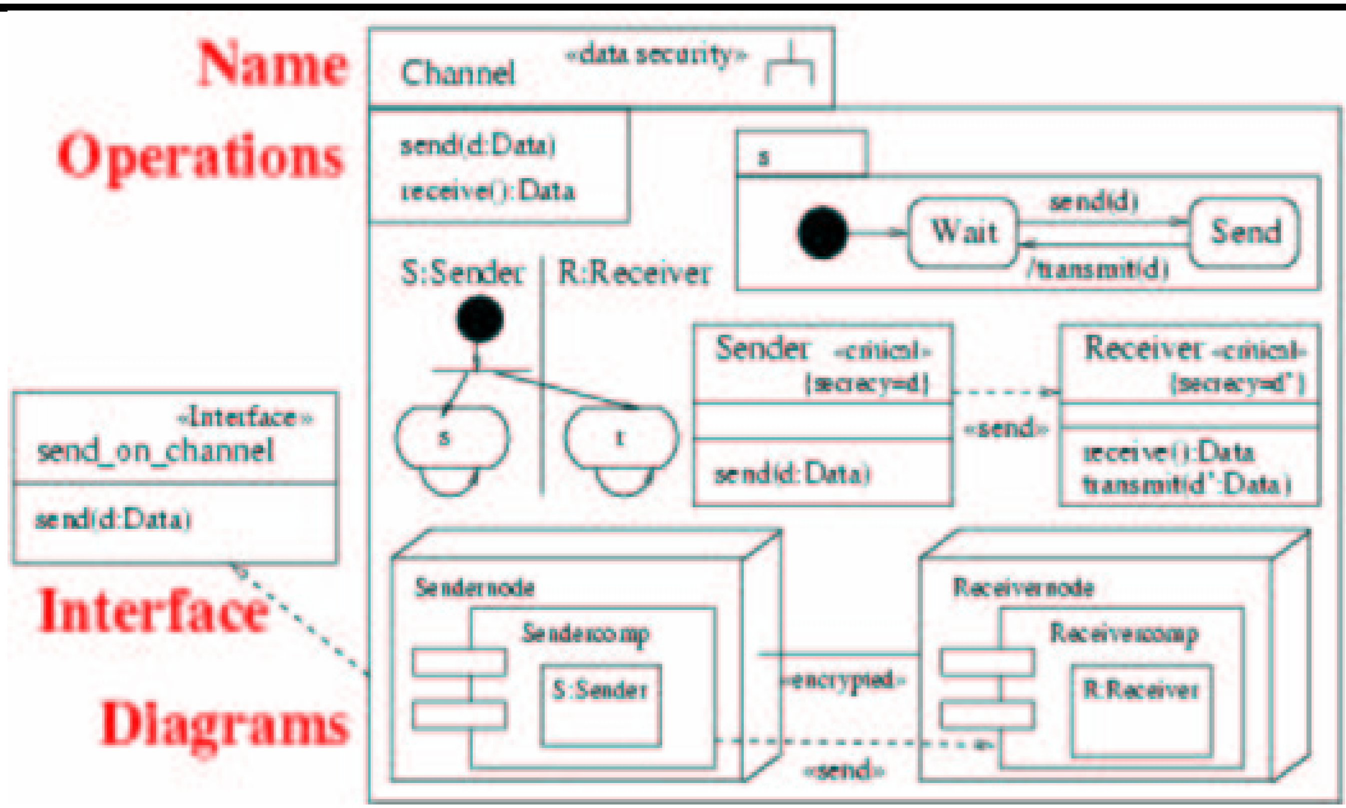
Input events cause state change and output actions.

UML run-through: Deployment diagrams



Describe the physical layer on which the system is to be implemented.

UML run-through: Package



May be used to organize model elements into groups.

UML Extension mechanisms

Stereotype: specialize model element using ≪label≫.

Tagged value: attach {tag=value} pair to stereotyped element.

Constraint: refine semantics of stereotyped element.

Profile: gather above information.

Roadmap

Prologue

UML

UMLsec: The profile

Security patterns

Case studies

Using Java security, CORBAsec

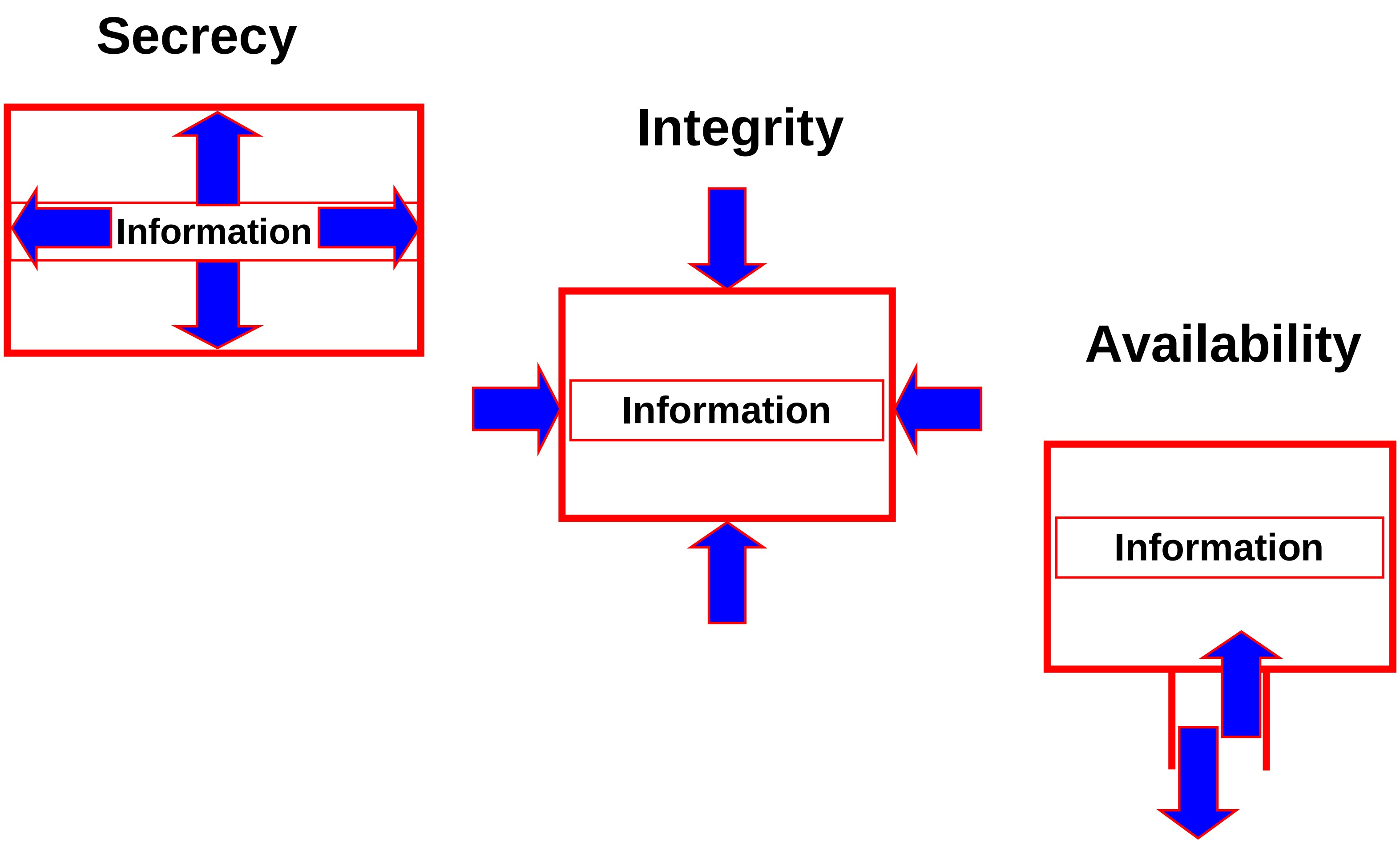
Tools

UMLsec

UMLsec: extension for secure systems development.

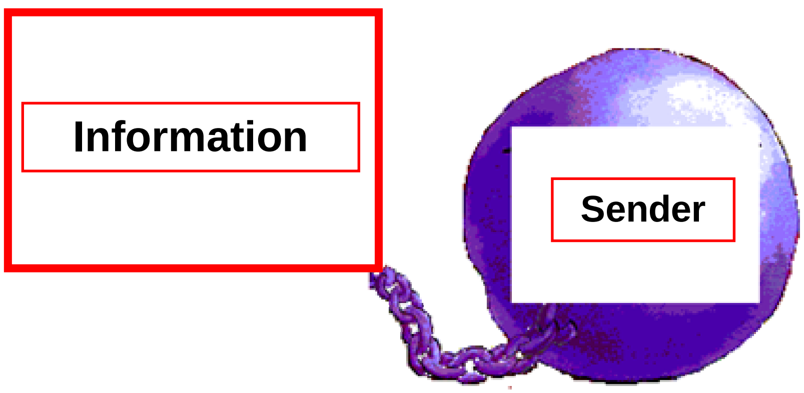
- evaluate UML specifications for vulnerabilities
- encapsulate security engineering patterns
- also for developers not specialized in security
- security from early design phases, in system context
- make certification cost-effective

Basic Security Requirements I

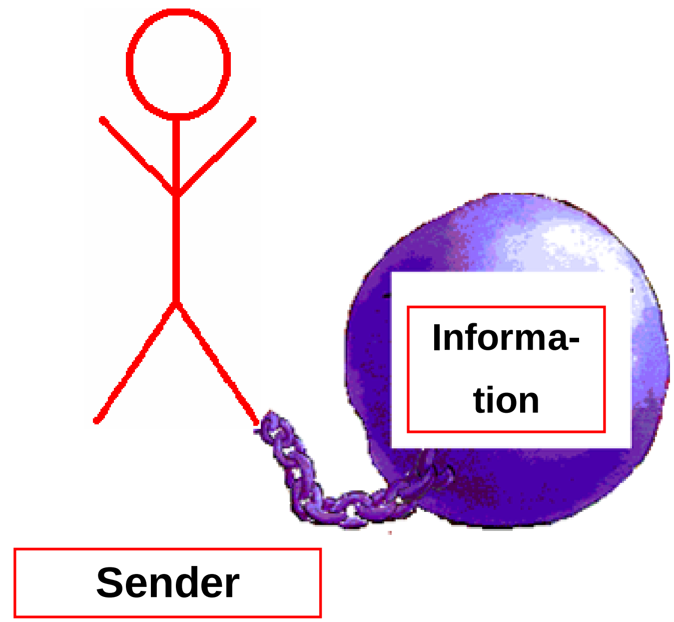


Basic Security Requirements II

Authenticity



Nonrepudiability



The UMLsec profile

Recurring security requirements as stereotypes with tags (secrecy, integrity,...).

Associated constraints to evaluate model, indicate possible vulnerabilities.

Ensures that stated security requirements enforce given security policy.

Ensures that UML specification provides requirements.

Requirements on UML extension for security I

Mandatory requirements:

- Provide basic security requirements such as secrecy and integrity.
- 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).

Requirements on UML extension for security II

- Provide security primitives (e.g. (a)symmetric encryption).
- Allow considering underlying physical security.
- Allow addressing security management (e.g. secure workflow).

Optional requirements: Include domain-specific security knowledge (Java, smart cards, CORBA, ...).

UMLsec: general ideas

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

UMLsec profile (excerpt)

Stereotype	Base class	Tags	Constraints	Description
Internet	link			Internet connection
secure links	subsystem		dependency security matched by links	enforces secure communication links
secrecy	dependency			assumes secrecy
secure dependency	subsystem		call, send respect data security	structural interaction data security
no down-flow	subsystem	high	prevents down-flow	information flow
data security	subsystem		provides secrecy, integrity	basic datasec requirements
fair exchange	package	start, stop	after start eventually reach stop	enforce fair exchange
guarded access	Subsystem		guarded objects acc. through guards.	access control using guard objects

≪Internet≫, ≪encrypted≫, …

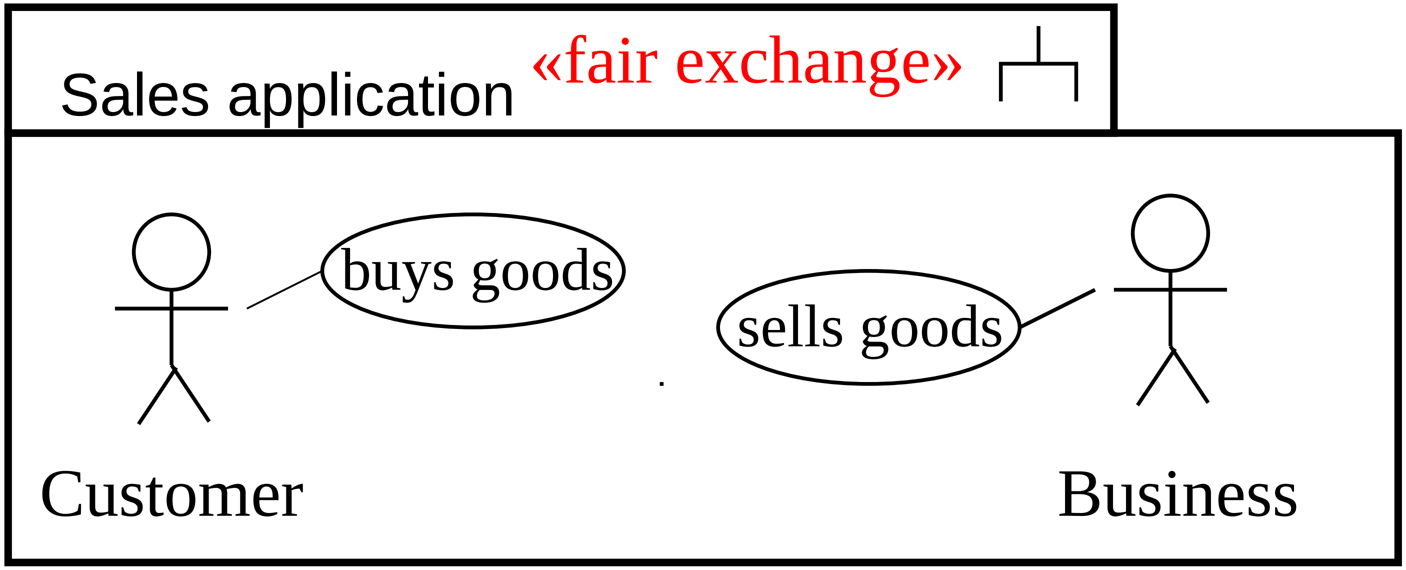
Kinds of communication links resp. system nodes.

For adversary type A, stereotype s, have set $\mathrm{Threats}_A(s) \in$ {delete, read, insert, access} of actions that adversaries are capable of.

Default attacker:

Stereotype	$\mathrm{Threats}_{default}()$
Internet	{delete, read, insert}
encrypted	{delete}
LAN	$\emptyset$
smart card	$\emptyset$

Requirements with use case diagrams



Capture security requirements in use case diagrams.

Constraint: need to appear in corresponding activity diagram.

≪fair exchange≫

Ensures generic fair exchange condition.

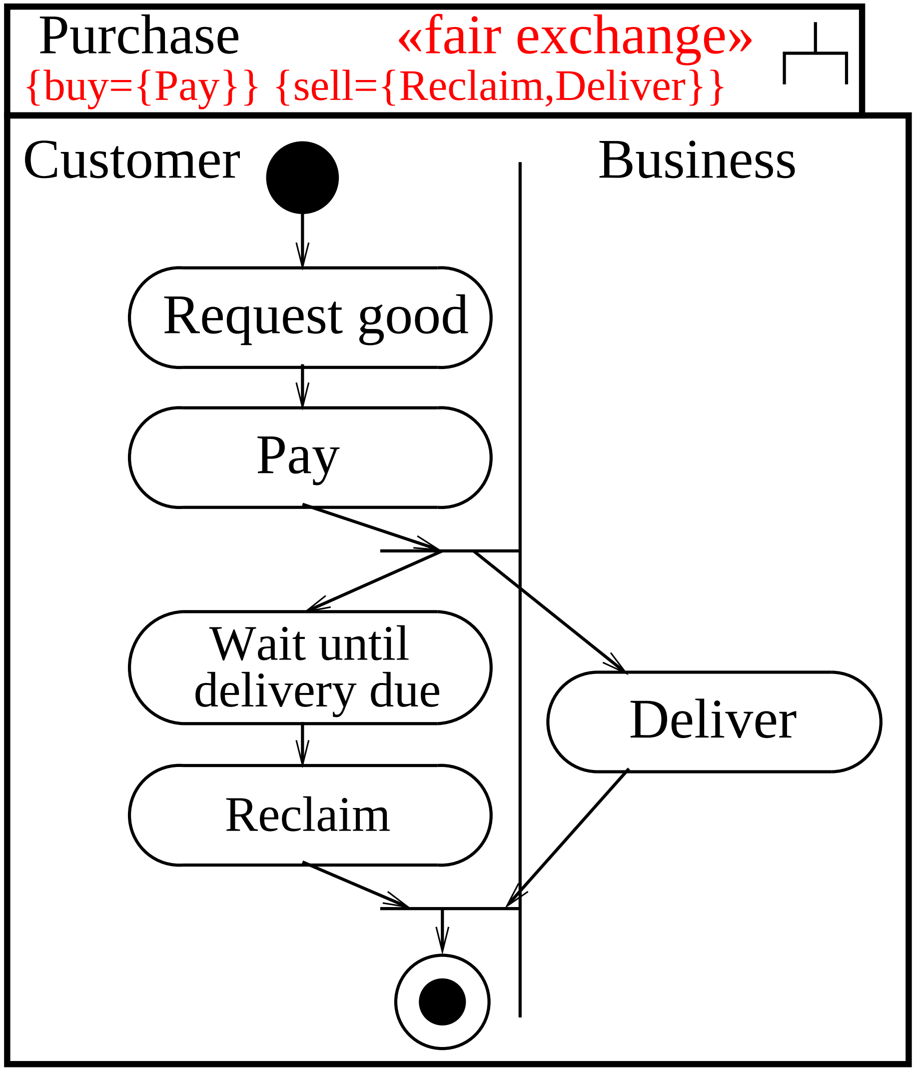
Constraint: after a {buy} state in activity diagram is reached, eventually reach {sell} state.

(Cannot be ensured for systems that an attacker can stop completely.)

Example ≪fair exchange≫

Customer buys a good from a business.

Fair exchange means: after payment, customer is eventually either delivered good or able to reclaim payment.



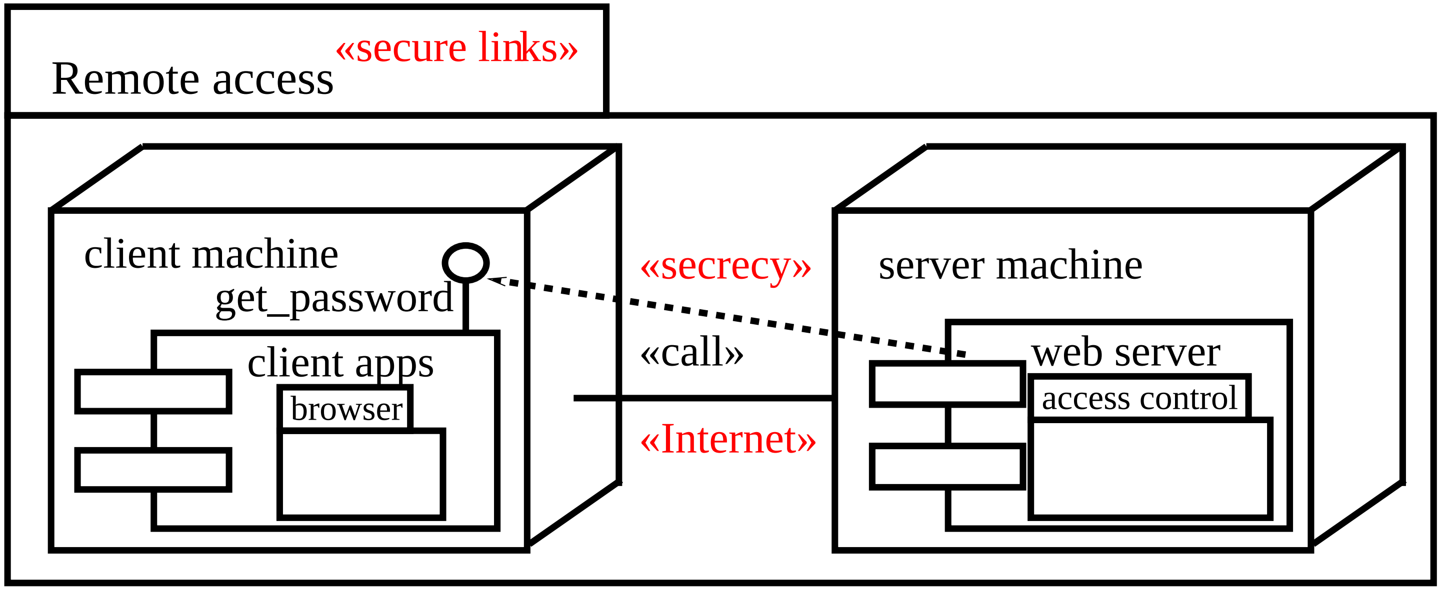
≪secure links≫

Ensures that physical layer meets security requirements on communication.

Constraint: for each dependency *d* with stereotype $s \in$ {≪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 $\notin$ $\mathrm{Threats}_A(t)$.
- if *s* = ≪integrity≫: have insert $\notin$ $\mathrm{Threats}_A(t)$.

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.

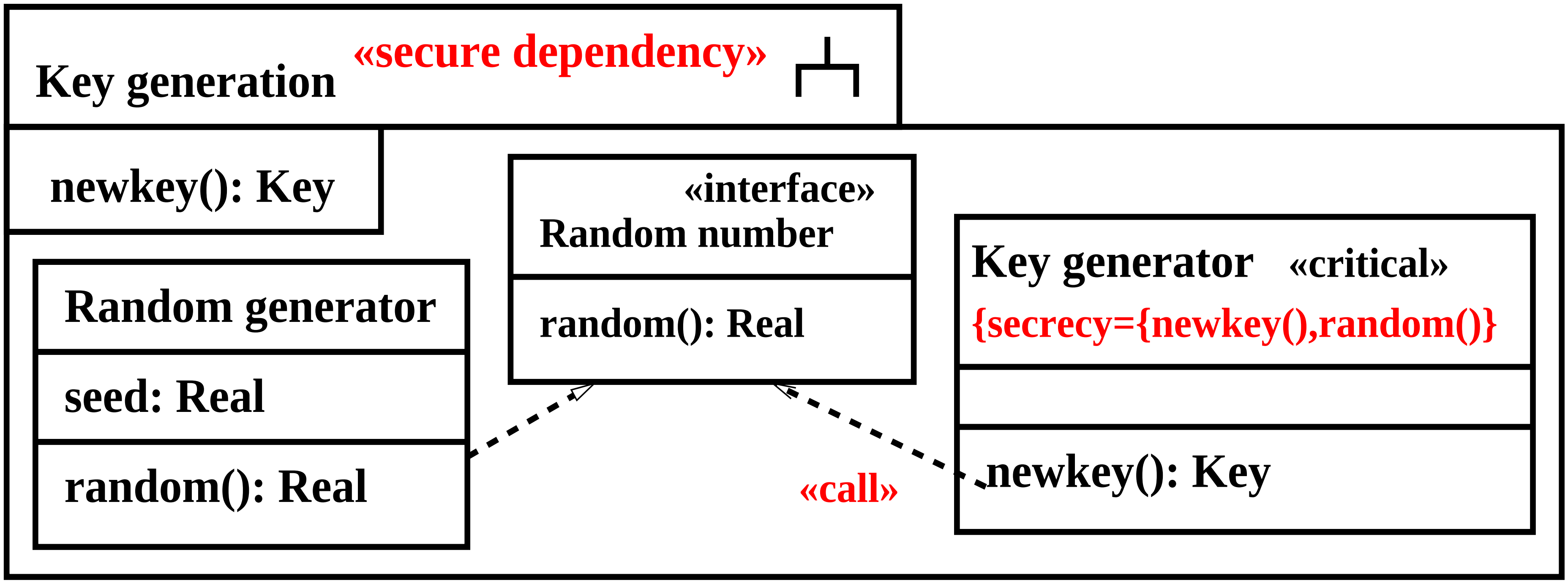
≪secure dependency≫

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 {secrecy}):

- 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.

≪no down–flow≫

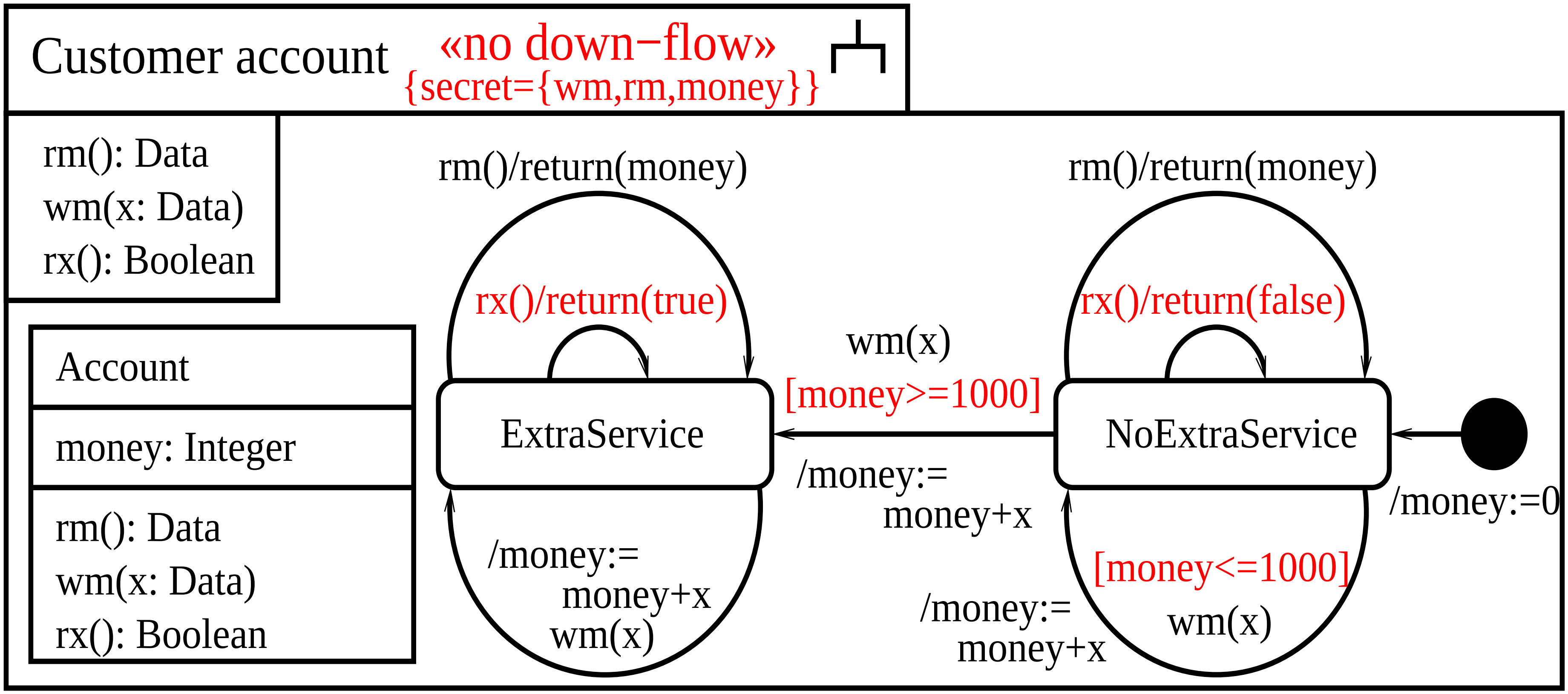
Enforce secure information flow.

Constraint:

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

Formalize by referring to formal behavioural semantics.

Example ≪no down-flow≫



≪no down–flow≫ violated: partial information on input of high wm() returned by non-high rx().

≪data security≫

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

Constraints:

Secrecy of {secrecy} data preserved.

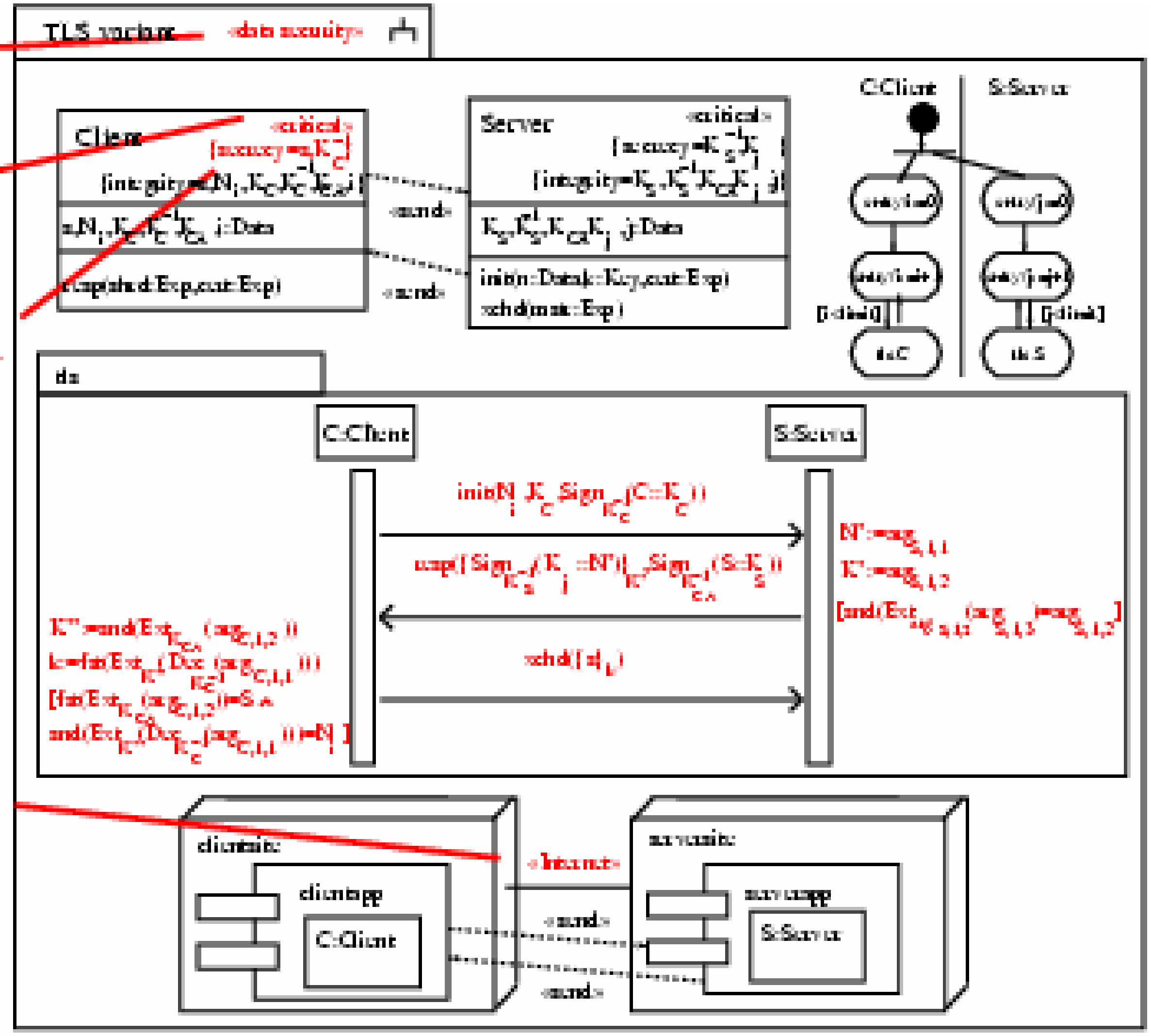
Integrity of {integrity} data preserved.

Example ≪data security≫

«data security»

«critical»

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



Variant of TLS (INFOCOM`99).

Violates {secrecy} of s against default adversary.

«Internet»

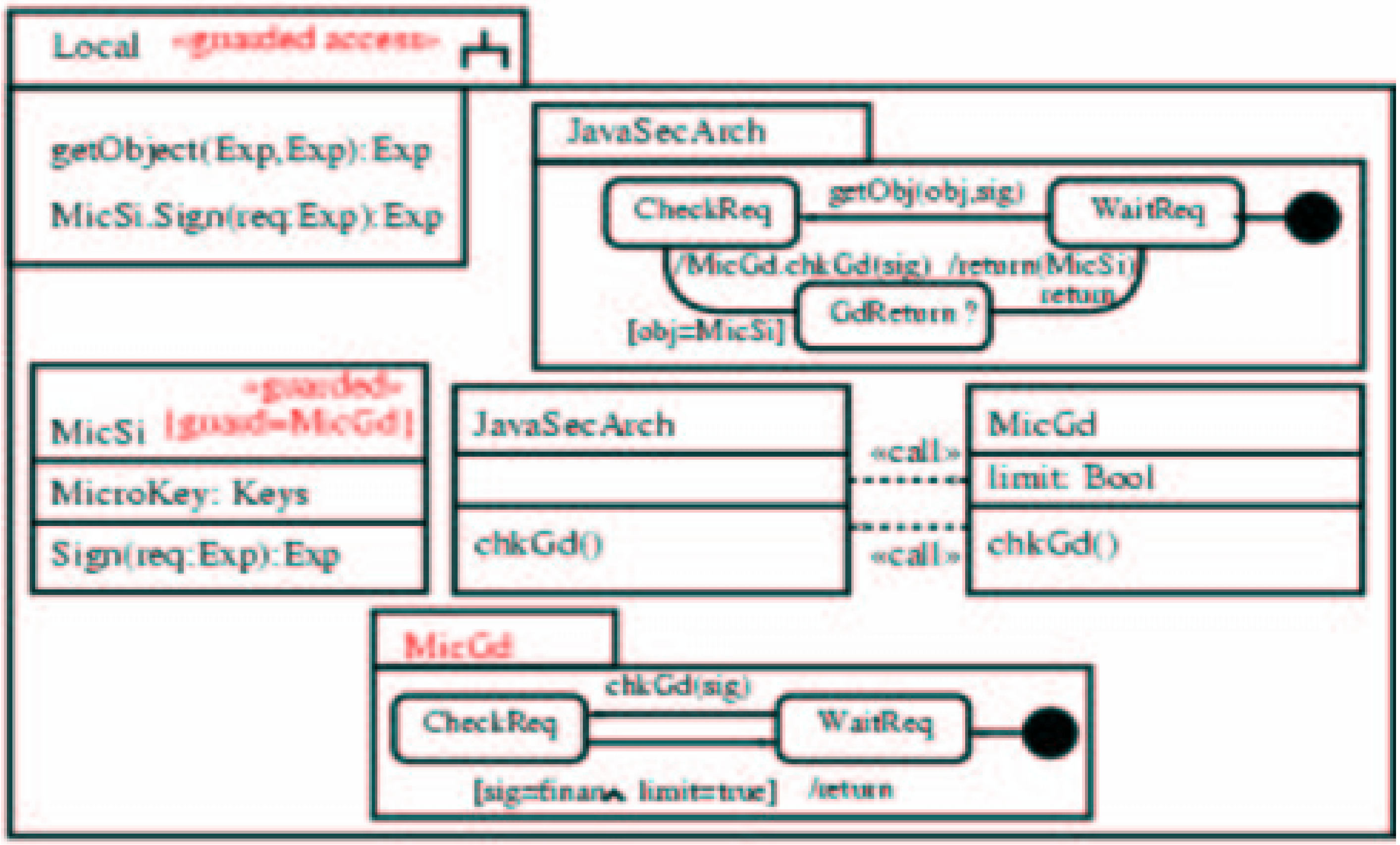
≪guarded access≫

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

Constraints:

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

Example ≪guarded access≫



Provides ≪guarded access≫:

Access to MicSi protected by MicGd.

Does UMLsec meet requirements?

Security requirements: ≪secrecy≫,…

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

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.

Roadmap

Prologue

UML

UMLsec: The profile

Security patterns: Rules, patterns

Case studies

Using Java security, CORBAsec

Tools

Rules of prudent security engineering

Saltzer, Schroeder (1975):

Design principles for security-critical systems.

Check how to enforce these with UMLsec.

Economy of mechanism

Keep the design as simple and small as possible.

Often systems made complicated to make them (look) secure.

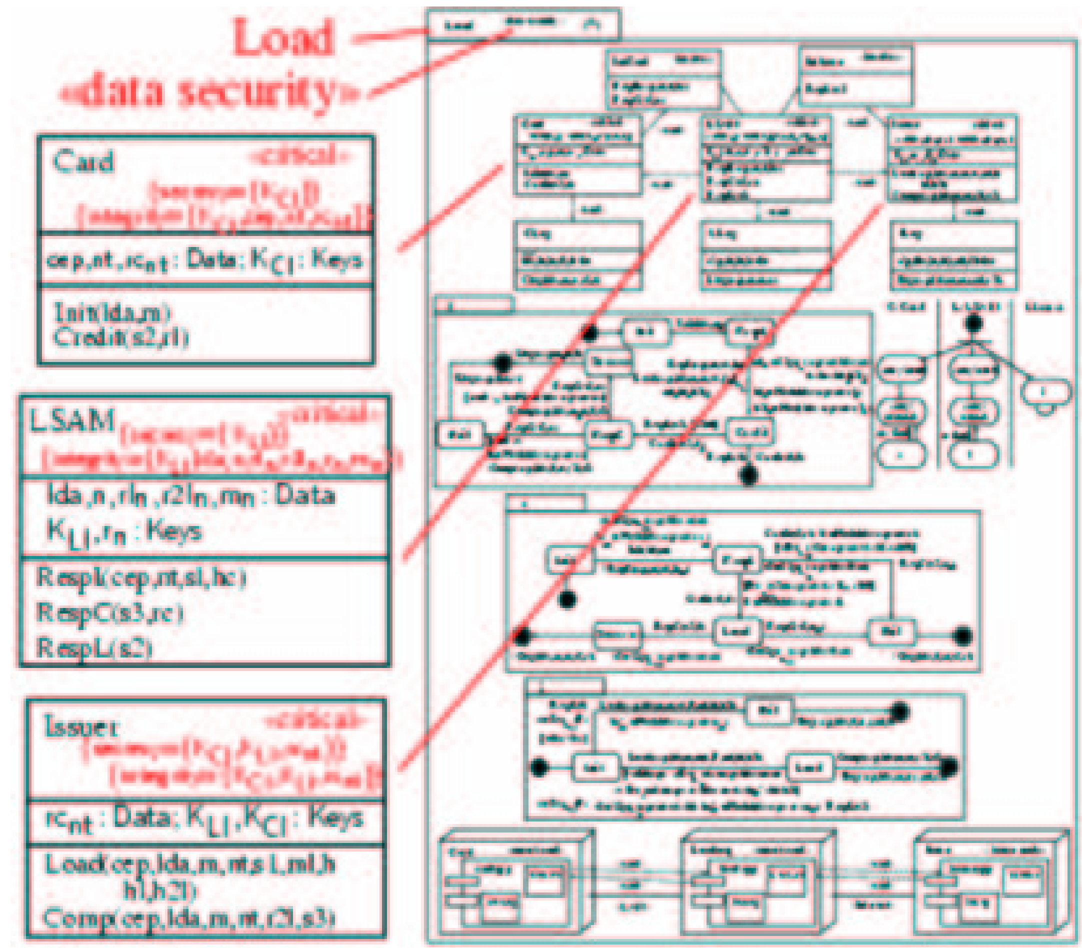
Method for reassurance may reduce this temptation.

Payoffs from formal evaluation may increase incentive for following the rule.

Fail-safe defaults

Base access decisions on permission rather than exclusion.

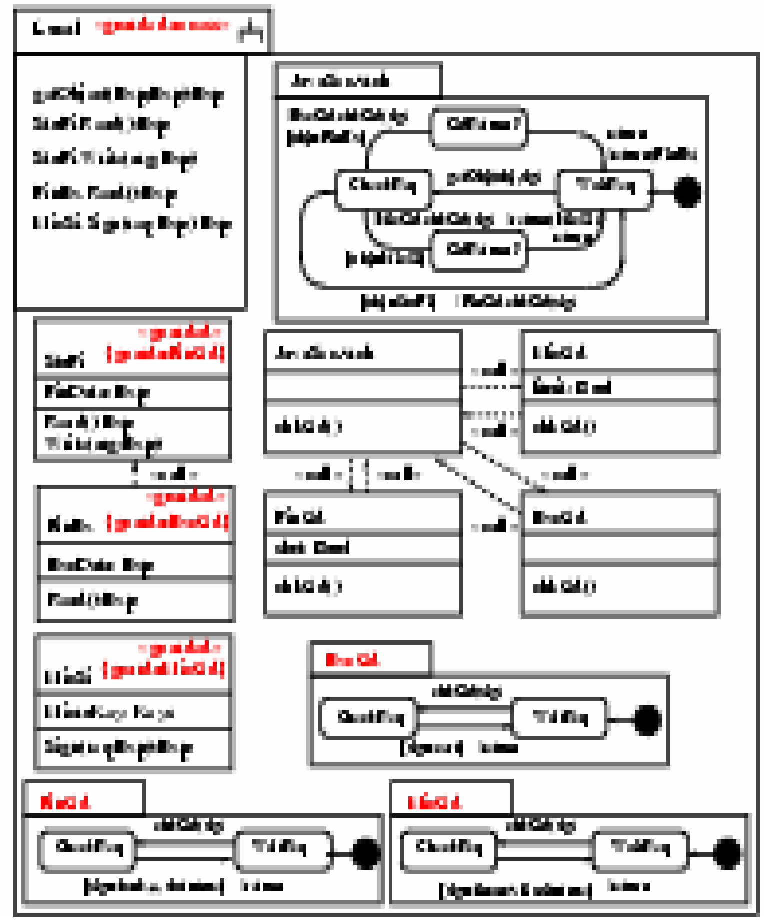
Example: secure log-keeping for audit control in Common Electronic Purse Specifications (CEPS).



Complete mediation

Every access to every object must be checked for authority.

E.g. in Java: use guarded objects. Use UMLsec to ensure proper use of guards.
More feasibly, mediation wrt. a set of sensitive objects.



Open design

The design should not be secret.

Method of reassurance may help to develop systems whose security does not rely on the secrecy of its design.

Separation of privilege

A protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.

Example: signature of two or more principals required for privilege. Formulate requirements with activity diagrams.

Verify behavioural specifications wrt. them.

Least privilege

Every program and every user of the system should operate using the least set of privileges necessary to complete the job.

Least privilege: every proper diminishing of privileges gives system not satisfying functionality requirements.

Can make precise and check this.

Least common mechanism

Minimize the amount of mechanism common to more than one user and depended on by all users.

Object-orientation:

- data encapsulation
- data sharing well-defined (keep at necessary minimum).

Psychological acceptability

Human interface must be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

Wrt. development process: ease of use in development of secure systems.

User side: e.g. performance evaluation (acceptability of performance impact of security).

Discussion

No absolute rules, but warnings.

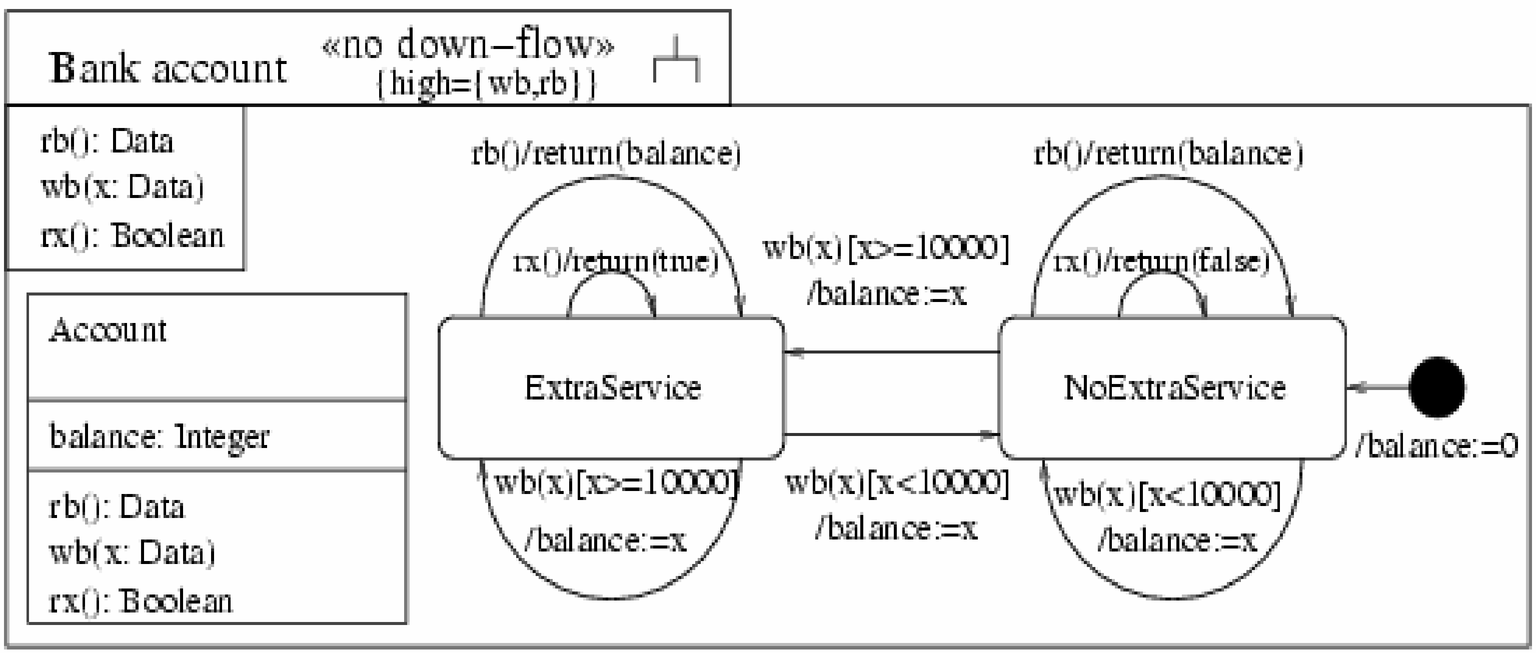
Violation of rules symptom of potential trouble; review design to be sure that trouble accounted for or unimportant.

Design principles reduce number and seriousness of flaws.

Security Patterns

Security patterns: use UML to encapsulate knowledge of prudent security engineering.

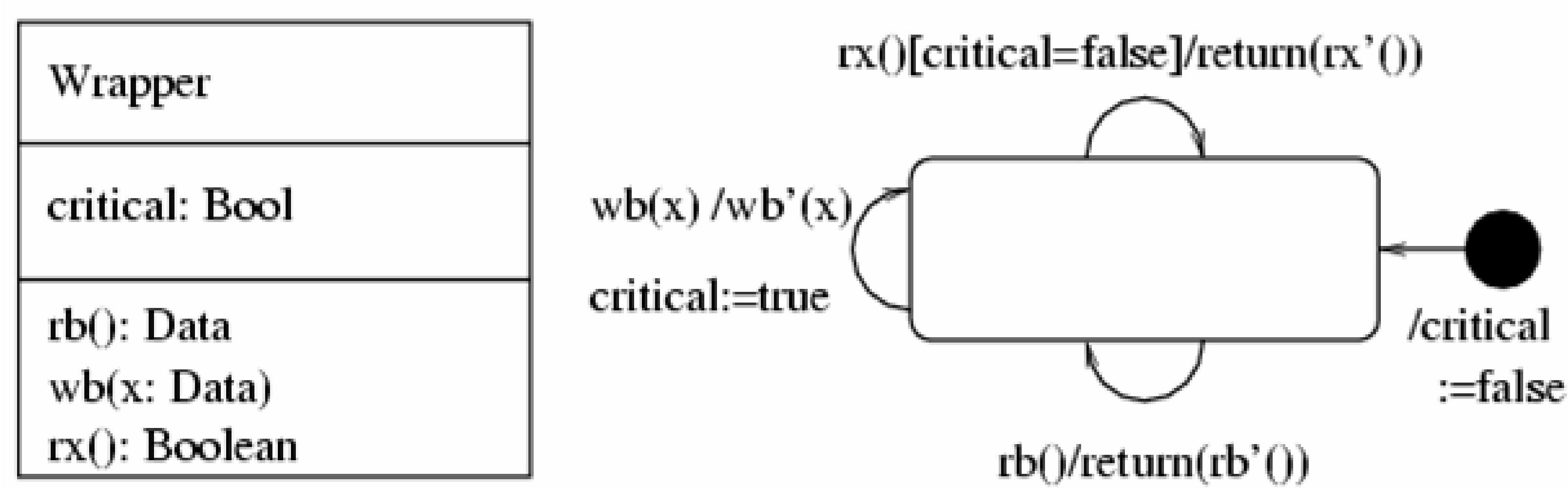
Example:



Does not preserve security of account balance.

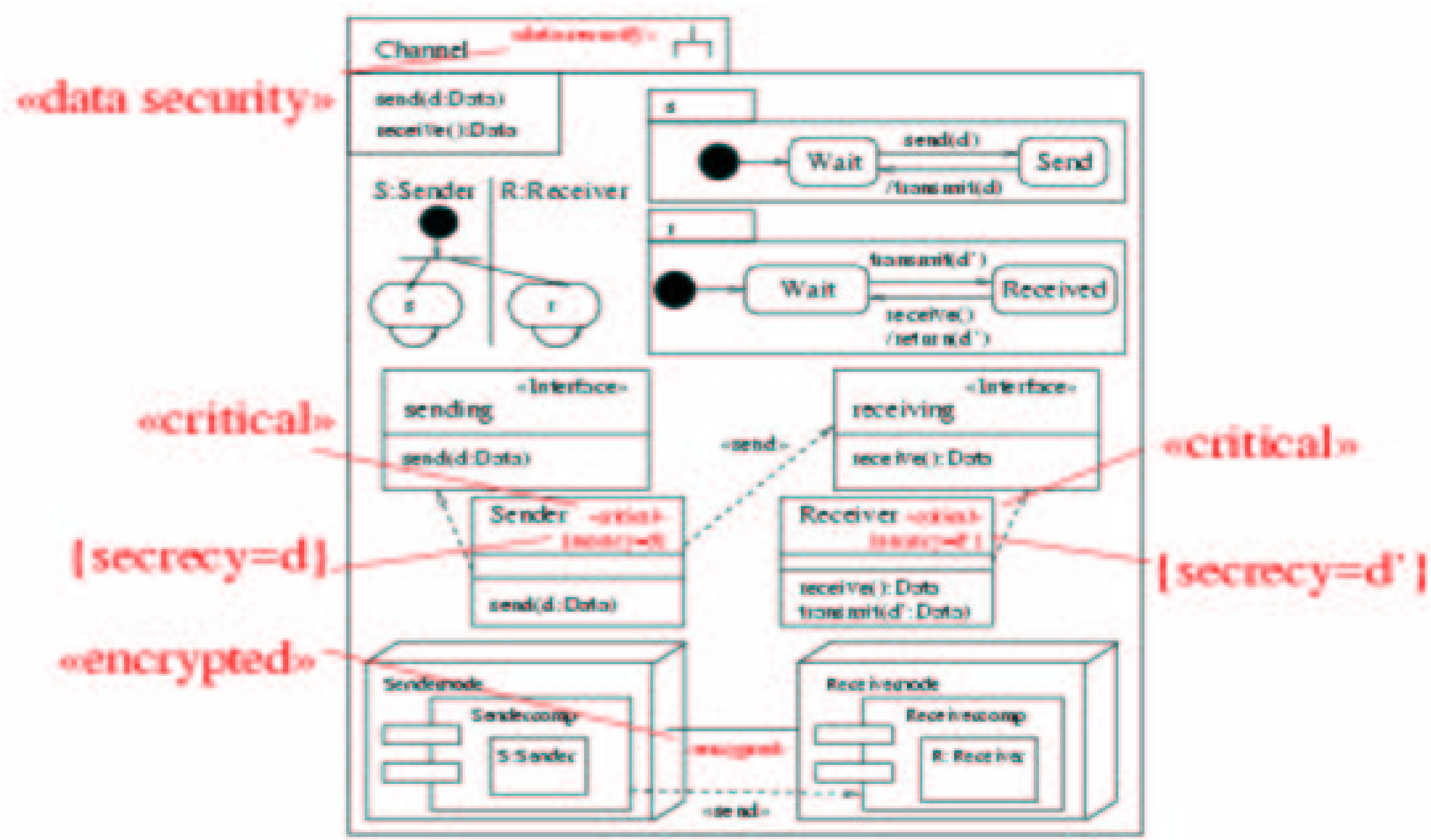
Solution: Wrapper Pattern

Technically, pattern application is transformation of specification.



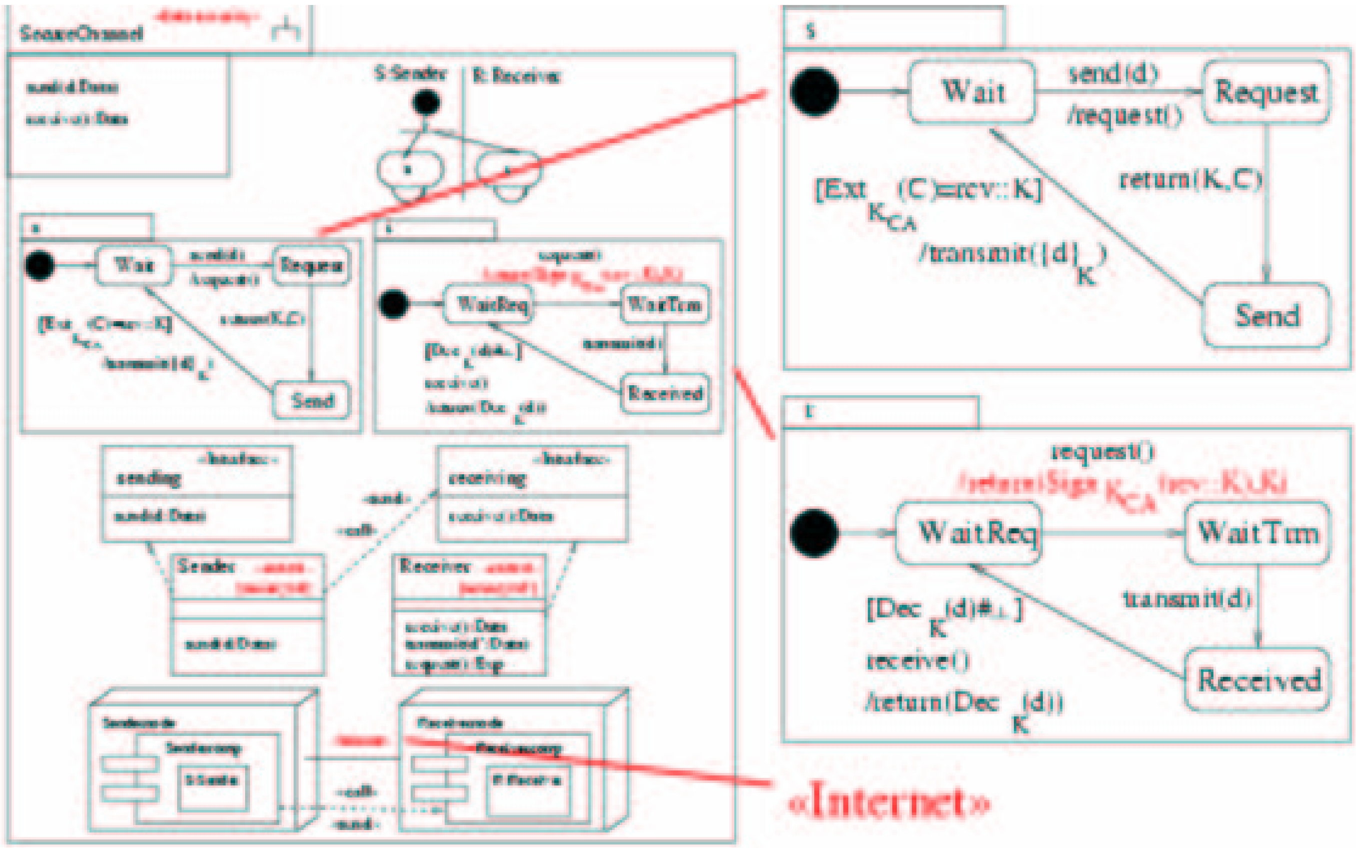
Use wrapper pattern to ensure that no low read after high write.
Can check this is secure (once and for all).

Secure channel pattern: problem



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

Secure channel pattern: (simple) solution



Exchange certificate and send encrypted data over Internet.

Roadmap

Prologue

UML

UMLsec: The profile

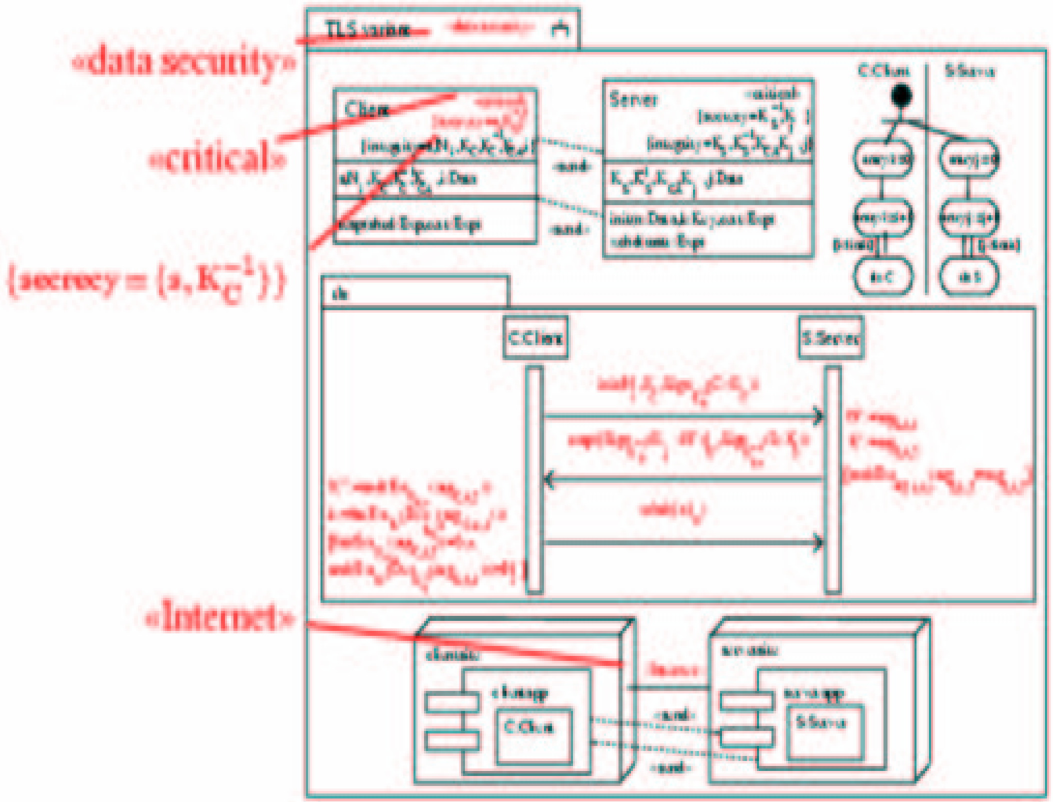
Security patterns

Case studies

Using Java security, CORBAsec

Tools

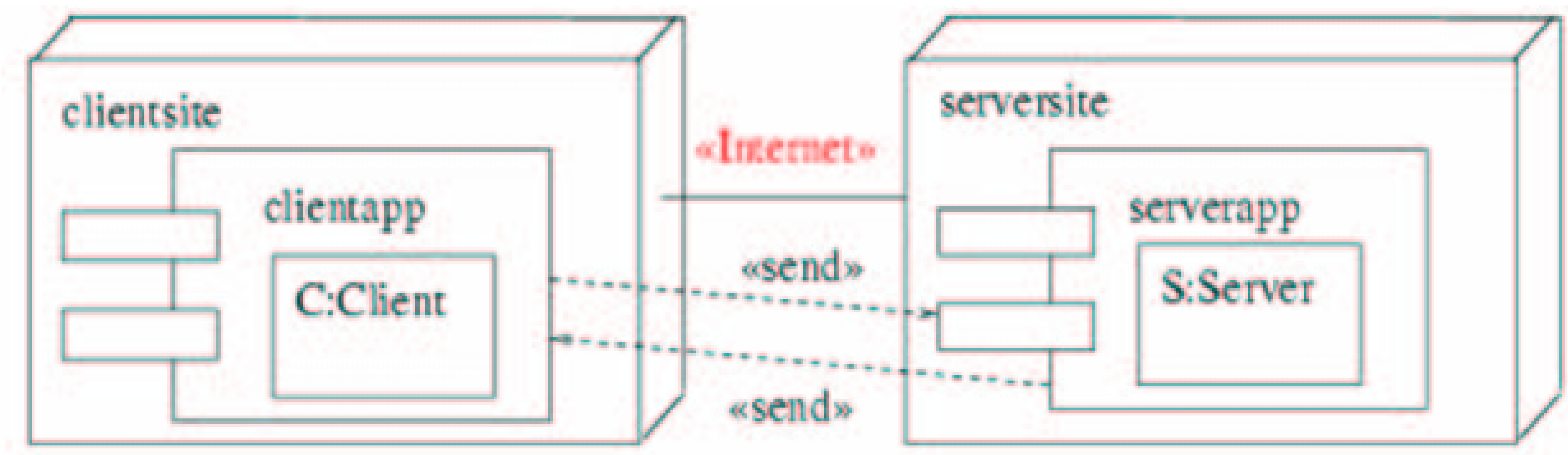
Example: Proposed Variant of TLS (SSL)



Apostolopoulos, Peris, Saha; IEEE Infocom 1999

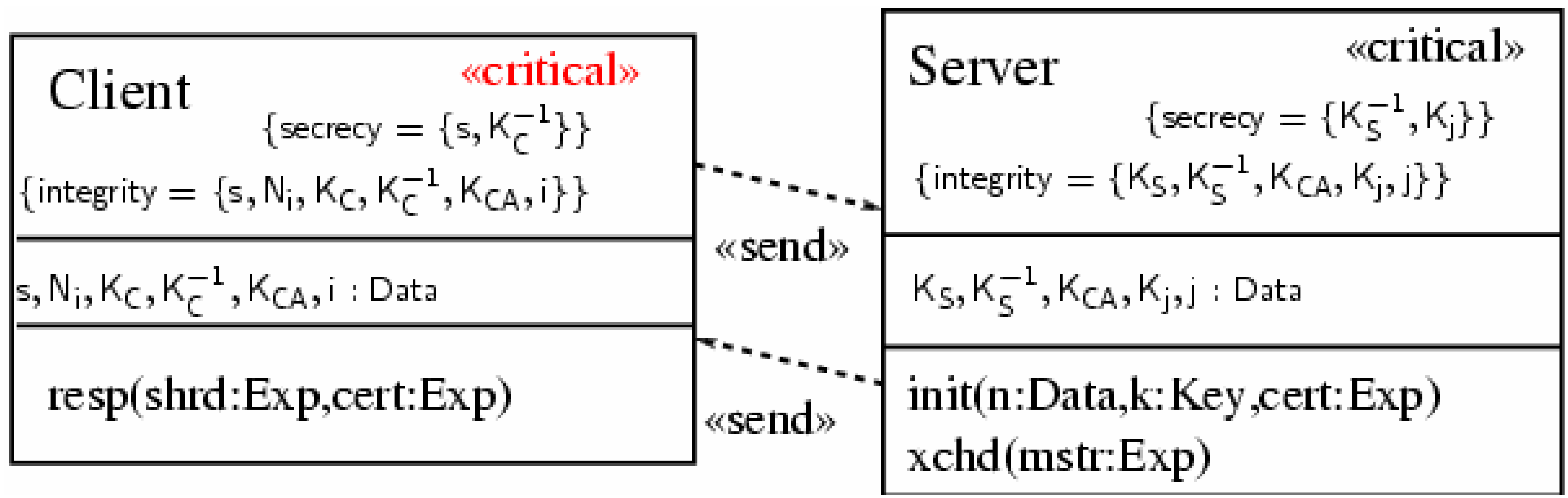
Goal: send secret *s* protected by session key K_j.

TLS Variant: Physical view



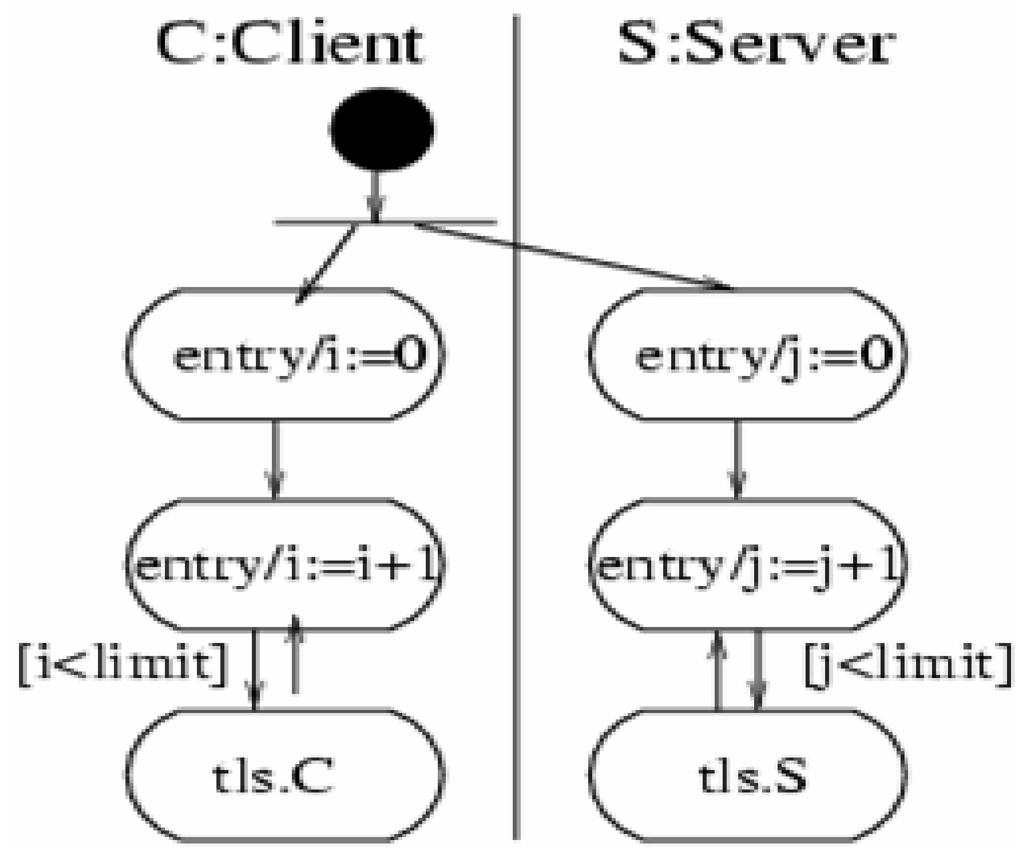
Deployment diagram.

TLS Variant: Structural view



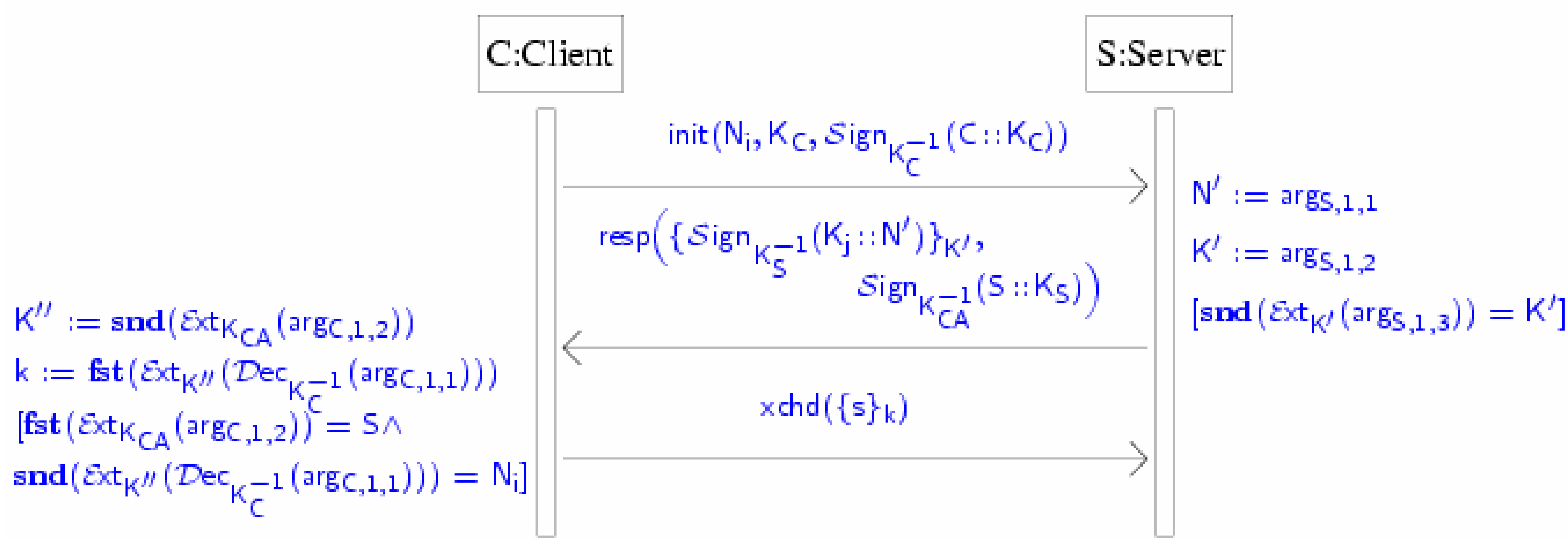
Class diagram

TLS Variant: Coordination view



Activity diagram.

TLS Variant: Interaction view



Sequence diagram.

The flaw

Surprise: *S* does not keep secrecy of *s* against *default* adversaries with

$$K_A^0 = \{K_{CA}, K_C, K_S, C, S, Sign_{K_{CA}^{-1}}(S :: K_S)\} \\ \cup \{Sign_{K_{CA}^{-1}}(Z :: K_Z) : Z \in Data \setminus \{S\}\}.$$

Man-in-the-middle attack.

The attack

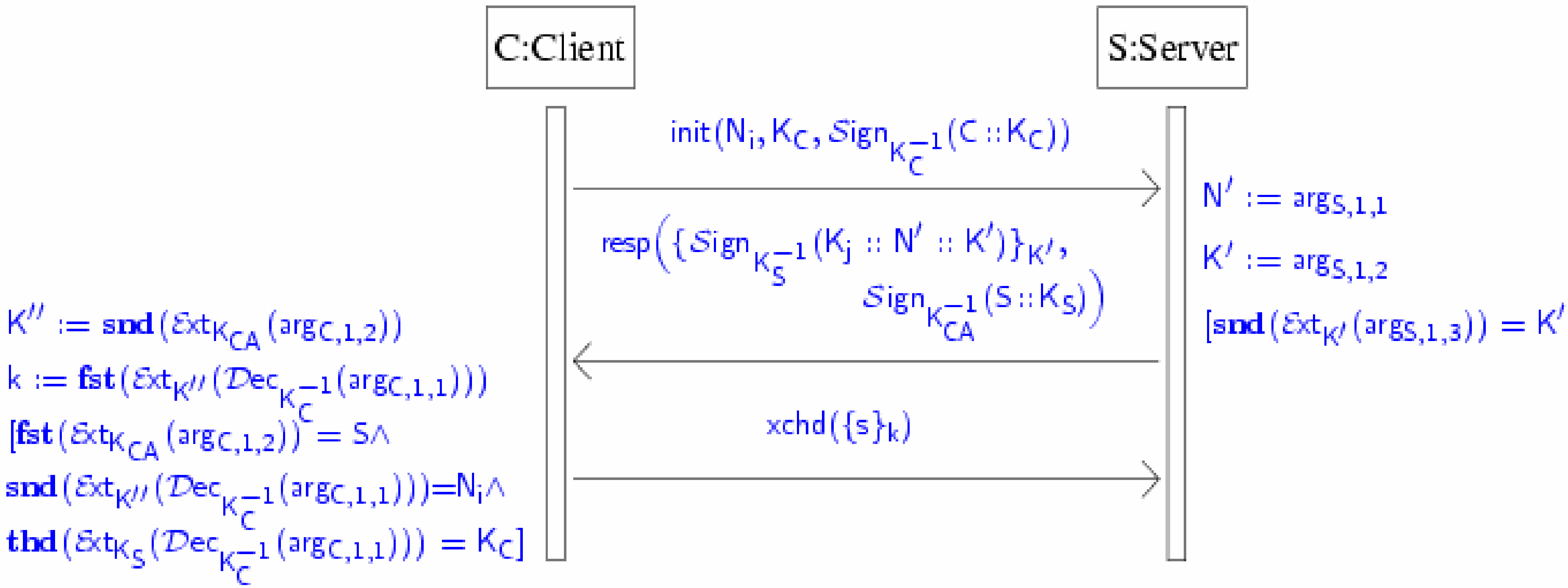
$$C \xrightarrow{N_i::K_C::Sign_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::Sign_{K_A^{-1}}(C::K_A)} S$$

$$A \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_A}::Sign_{K_{CA}^{-1}}(S::K_S)} S$$

$$C \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_C}::Sign_{K_{CA}^{-1}}(S::K_S)} A$$

$$C \xrightarrow{\{s\}_{K_j}} A \xrightarrow{\{s\}_{K_j}} S$$

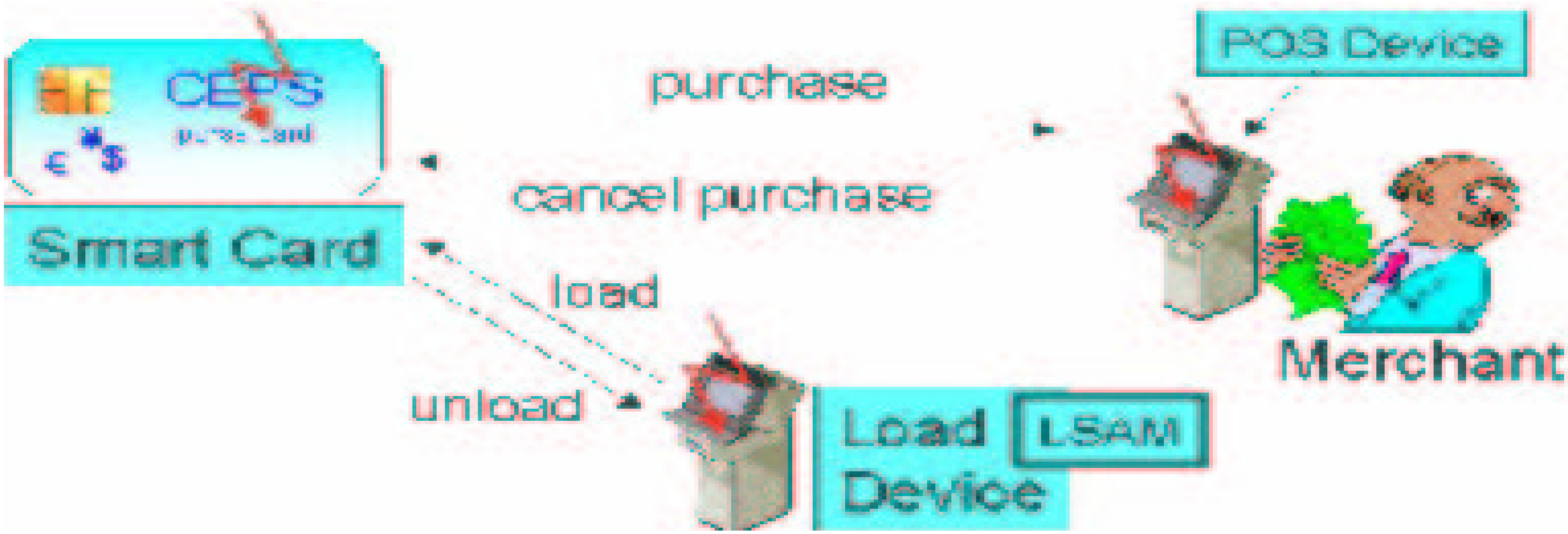
The fix



Thm: S' keeps secrecy of s against default adversaries with

$$K_A^0 = \{K_{CA}, K_C, K_S, C, S, Sign_{K_{CA}^{-1}}(S :: K_S)\}$$
$$\cup \{Sign_{K_{CA}^{-1}}(Z :: K_Z) : Z \in Data \backslash \{S\}\}.$$

Common Electronic Purse Specifications



Global electronic purse standard (90% of market).

Smart card contains account balance. Chip performs cryptographic operations securing the transactions.

More fraud protection than credit cards (transaction-bound authorisation).

FAIRPAY

Load protocol

Unlinked, cash-based load transaction (on-line).

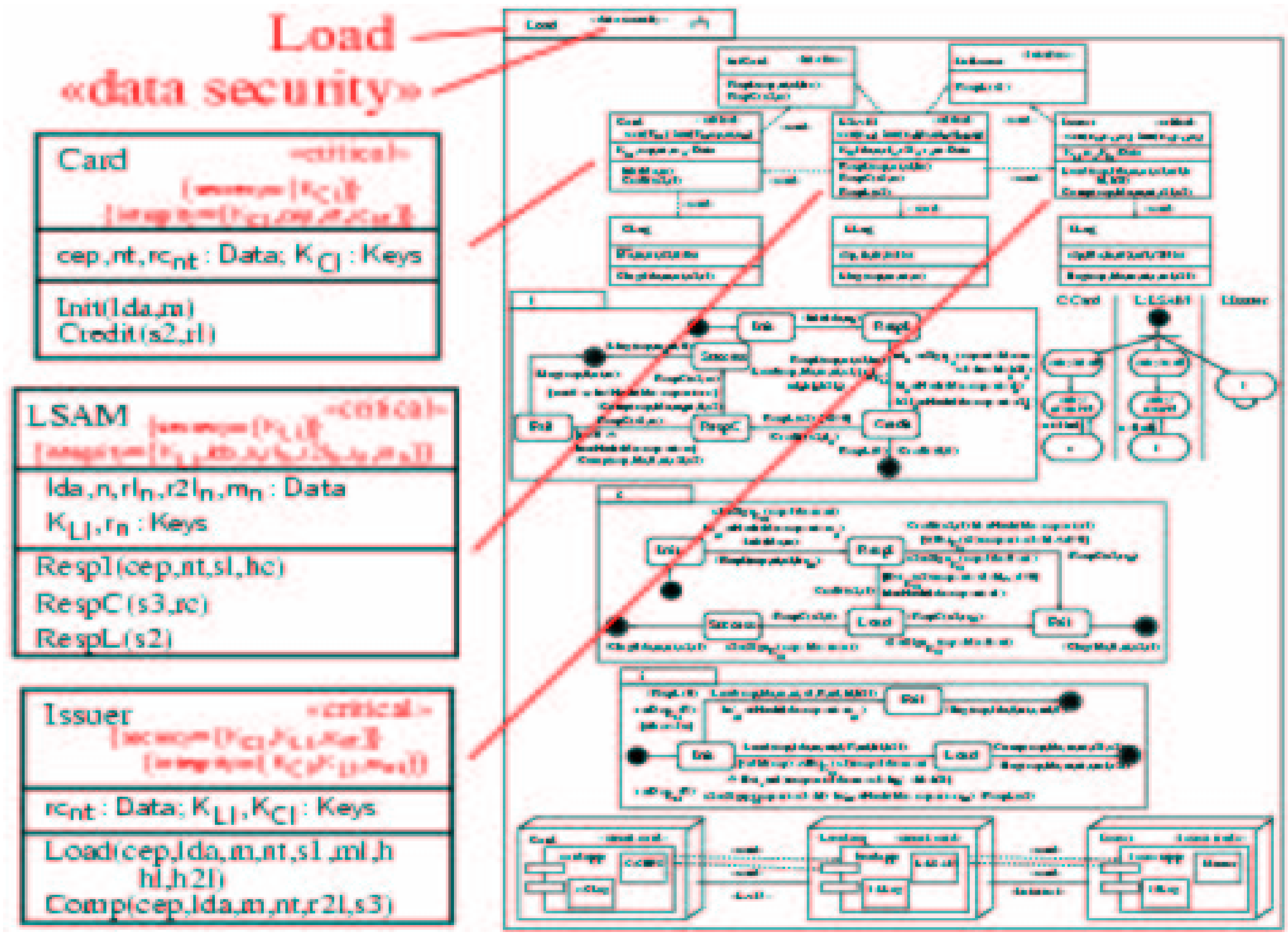
Load value onto card using cash at load device.

Load device contains Load Security Application Module (LSAM): secure data processing and storage.

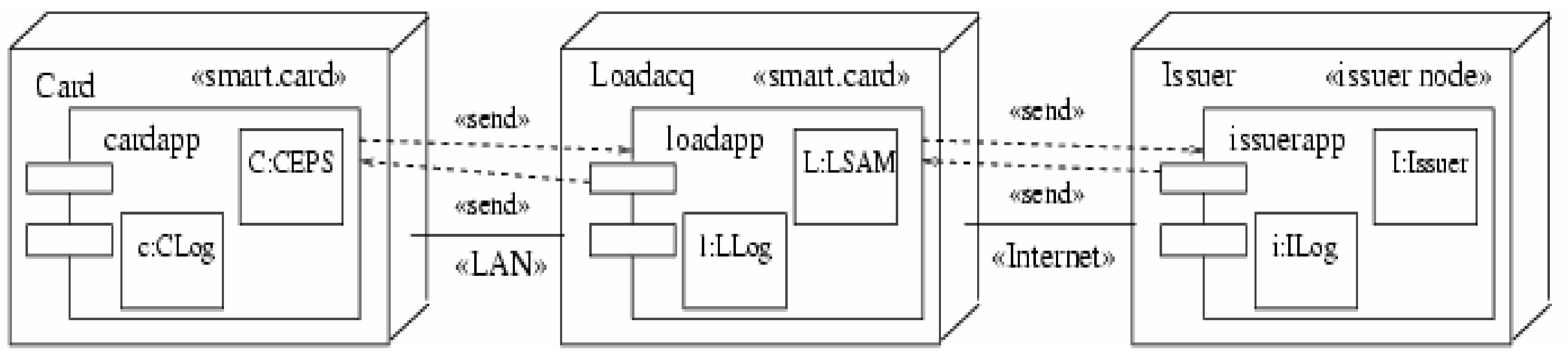
Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.

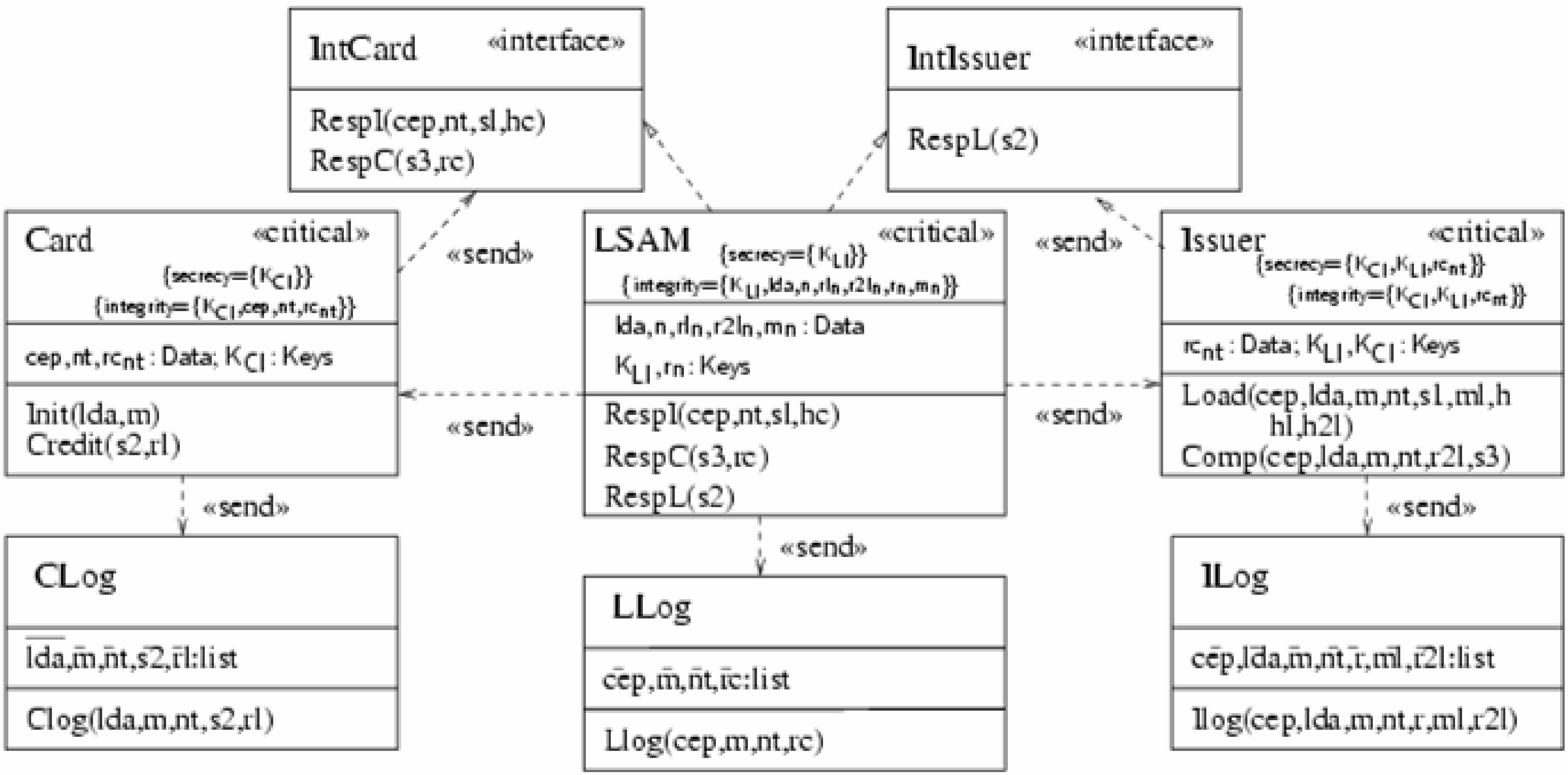
Load protocol



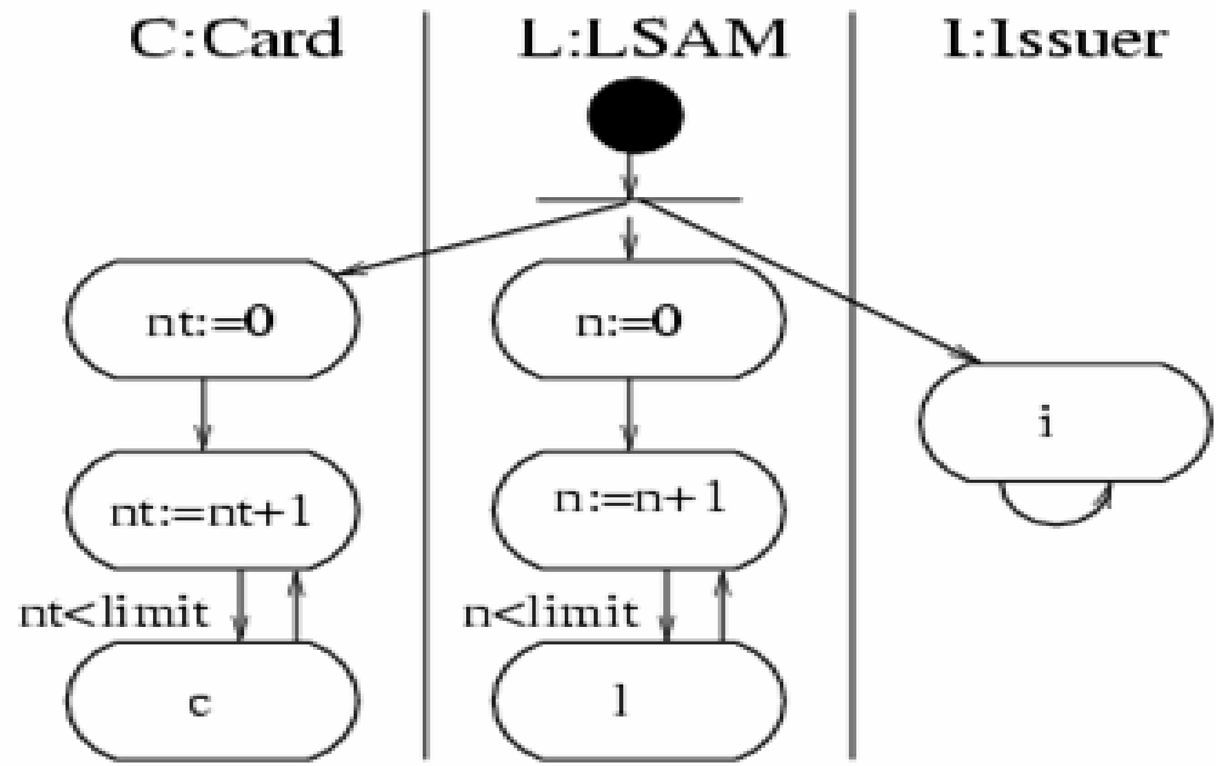
Load protocol: Physical view



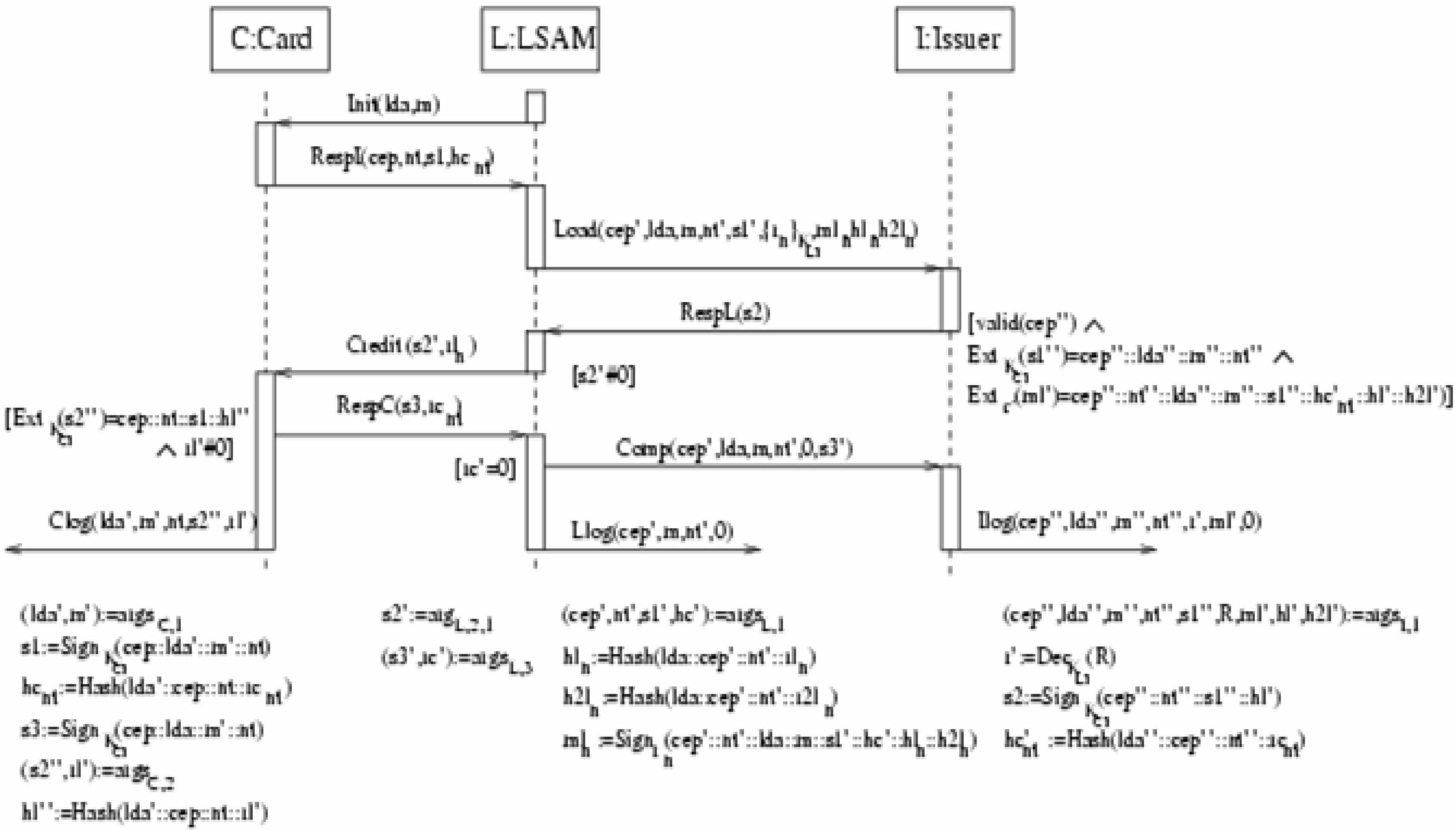
Load protocol: Structural view



Load protocol: Coordination view



Load protocol: Interaction view



Security Threat Model

Card, LSAM, issuer security module assumed tamper-resistant.

Intercept communication links, replace components.

Possible attack motivations:

- **Cardholder**: charge without pay
- **Load acquirer**: keep cardholder's money
- **Card issuer**: demand money from load acquirer

May coincide or collude.

Audit security

No direct communication between card and cardholder. Manipulate load device display.

Use post-transaction settlement scheme.

Relies on secure auditing.

Verify this here (only executions completed without exception).

Security conditions (informal)

Cardholder security If card appears to have been loaded with *m* according to its logs, cardholder can prove to card Issuer that a load acquirer owes *m* to card issuer.

Load acquirer security Load acquirer has to pay *m* to card issuer only if load acquirer has received *m* from cardholder.

Card issuer security Sum of balances of cardholder and load acquirer remains unchanged by transaction.

Load acquirer security

Suppose card issuer *I* possesses $ml_n = Sign_{rn}(cep::nt::lda::m_n::s1::hc_{nt}::hl_n::h2l_n)$ and card *C* possesses rl_n, where $hl_n = Hash(lda::cep::nt::rl_n)$.

Then after execution either of following hold:

- Llog(cep,lda,m_n,nt) has been sent to I:LLog (so load acquirer L has received and retains m_n in cash) or
- Llog $(cep, lda, 0, nt)$ has been sent to I : LLog (so L returns m_n to cardholder) and L has received rc_{nt} with $hc_{nt} = Hash(lda::cep::nt::rc_{nt})$ (negating ml_n).

"ml_n provides guarantee that load acquirer owes transaction amount to card issuer" (CEPS)

Flaw

Theorem. *L* does not provide load acquirer security against adversaries of type insider with $K_A^{fd} = \{cep, lda, m_n\}$.

Modification: use asymmetric key in ml_n, include signature certifying hc_{nt}.

Verify this version wrt. above conditions.

Further applications

- Analysis of multi-layer security protocol for web application of major German bank
- Analysis of SAP access control configuration for major German bank
- Risk analysis of critical business processes for Basel II / KontraG
- Risk analysis of digital control systems in nuclear power plants
- …

Roadmap

Prologue

UML

UMLsec: The profile

Security patterns

Case studies

Using Java security, CORBAsec

Tools

Java Security

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.

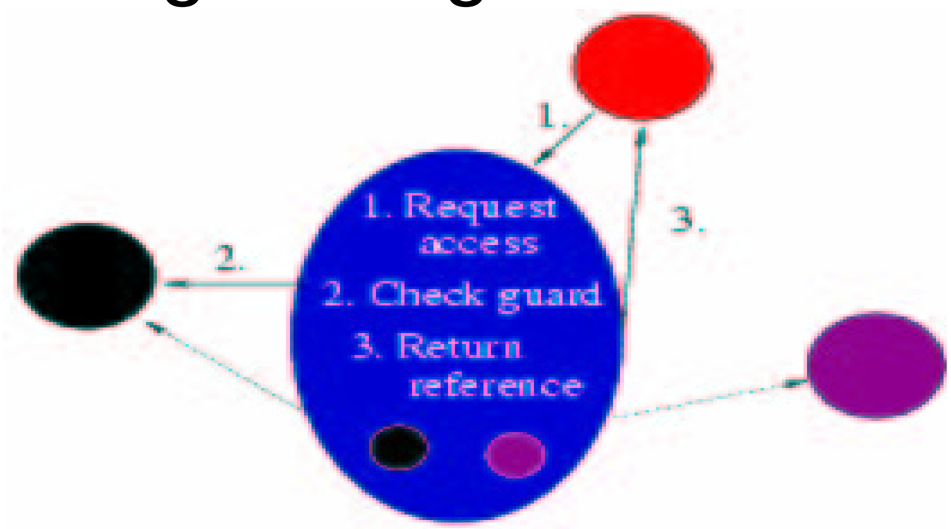
Similarly, need confidentiality.

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).

$\rightarrow$ Difficult to see which objects are granted permission.

$\Rightarrow$ 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

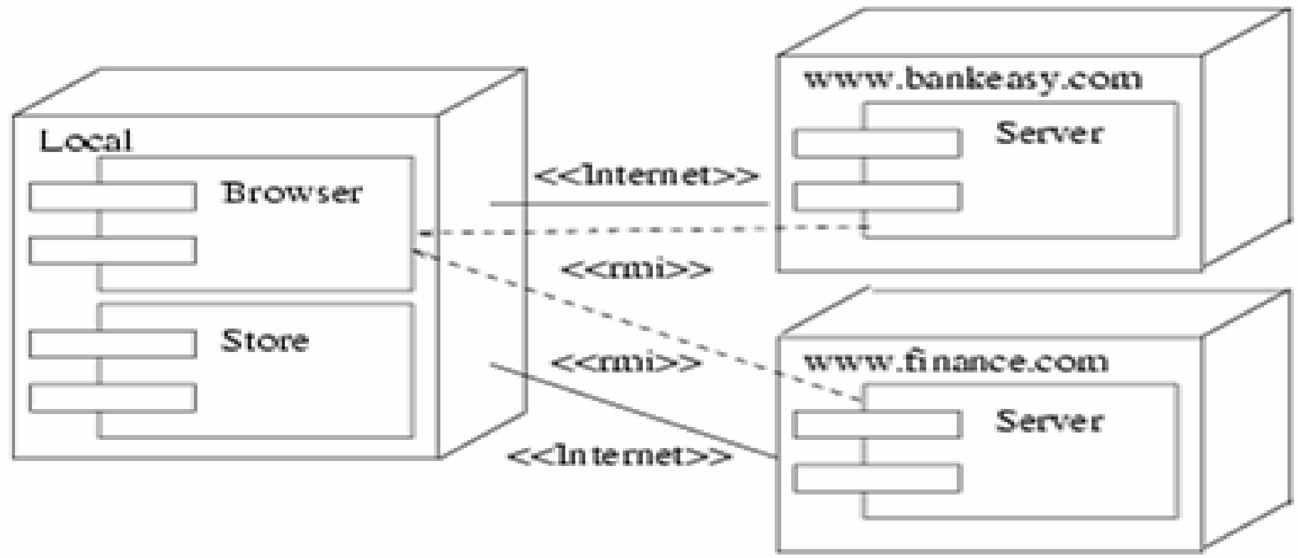
Theorem.

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



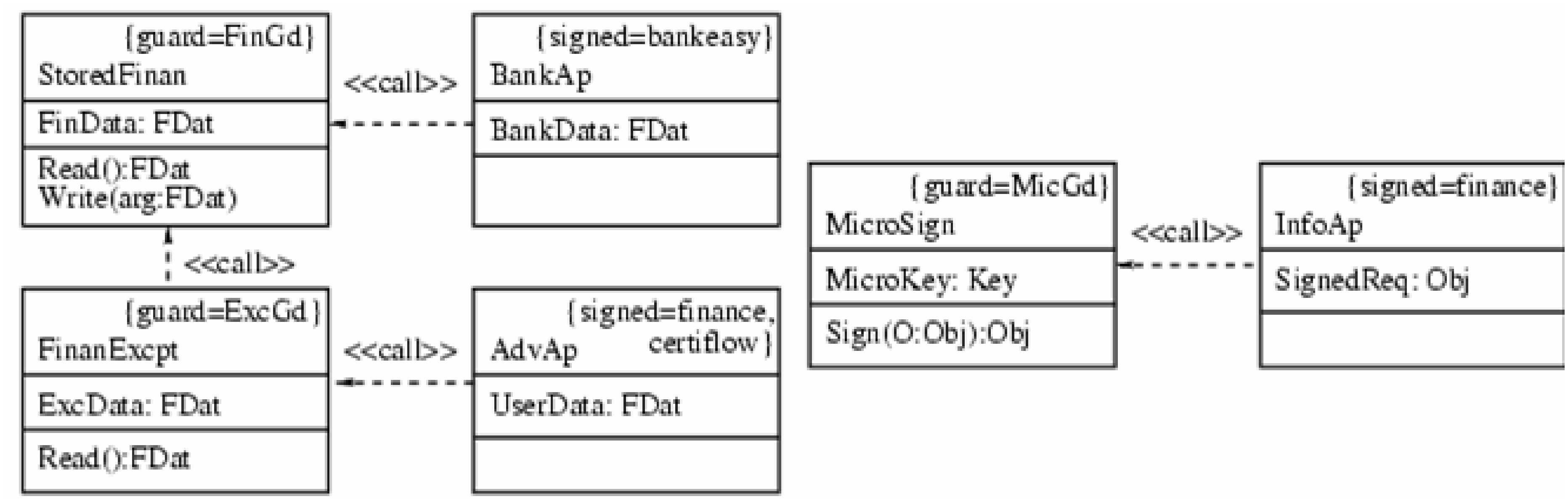
Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain Privileges (step1).

- 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

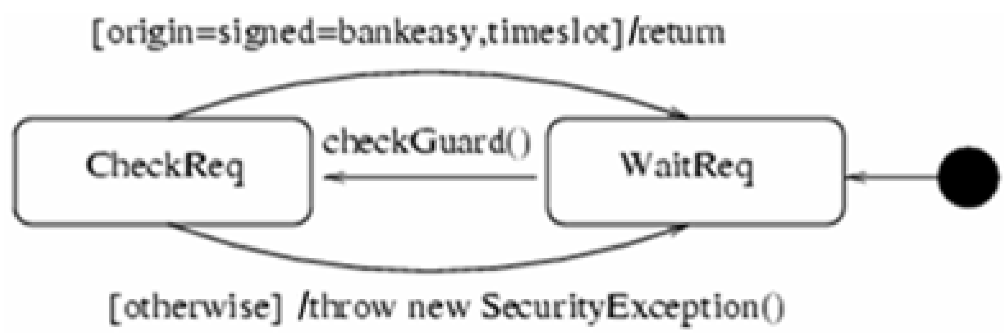
Sign and seal objects sent over Internet for Integrity and confidentiality.

GuardedObjects control access.

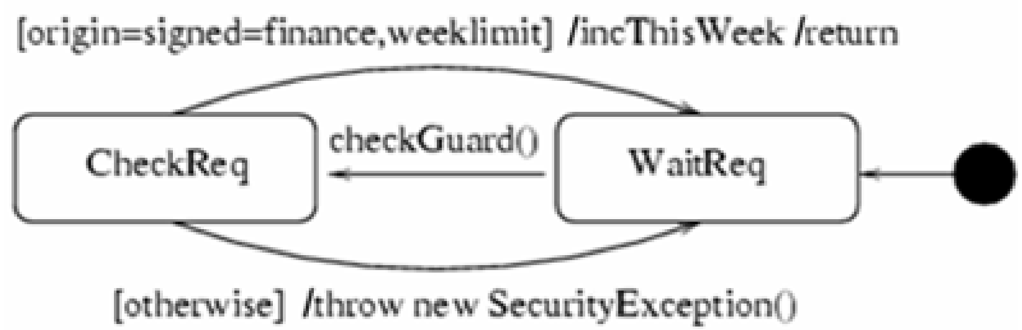


Financial Application: Guard objects (step 2)

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

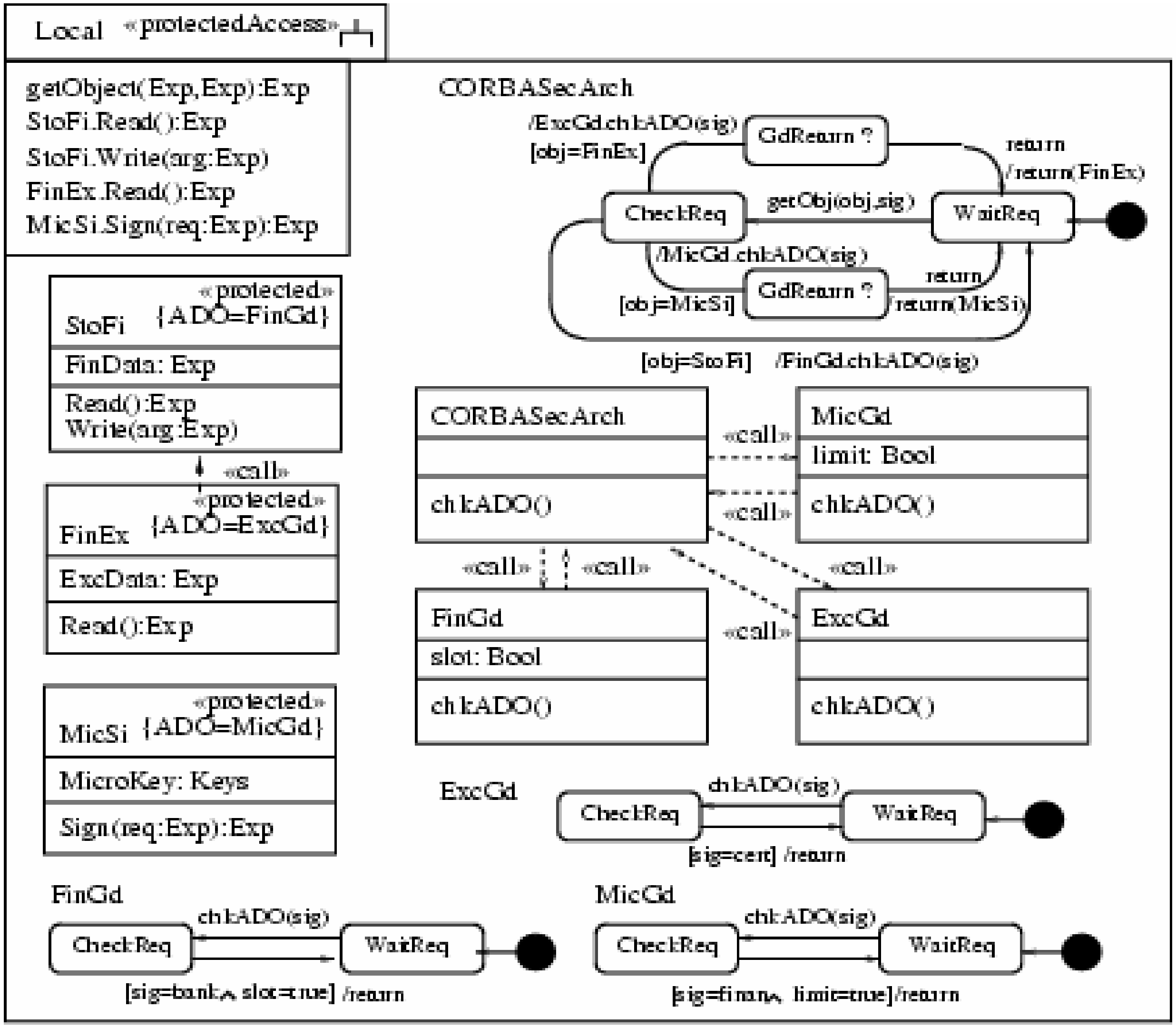
Object invocation access policy controls access of a client to a certain object via a certain method.

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.

Example: CORBA access control with UMLsec



Further Applications

- Analysis of multi-layer security protocol for web application of major German bank
- Analysis of SAP access control configurations for major German bank
- Risk analysis of critical business processes (for Basel II / KontraG)
- ...

Roadmap

Prologue

UML

UMLsec: The profile

Security patterns

Case studies

Using Java security, CORBAsec

Tools

Security Analysis

Model classes of adversaries.

May attack different parts of the system according to threat scenarios.

Example: insider attacker may intercept communication links in LAN.

To evaluate security of specification, simulate jointly with adversary model.

Security Analysis II

Keys are symbols, crypto-algorithms are abstract operations.

- Can only decrypt with right keys.
- Can only compose with available messages.
- Cannot perform statistical attacks.

Abstract adversary

Specify set K_A^0 of initial knowledge of an adversary of type *A*.

To test secrecy of *M* $\in$ Exp$\backslash K_A^0$ against attacker type *A*: Execute *S* with most powerful attacker of type *A* according to threat scenario from deployment diagram.

M kept secret by *S* if *M* never output in clear.

Example: secrecy

Component sending $\{m\}_K::K \in \mathrm{Exp}$ over Internet does not preserve secrecy of m or K against default attackers the Internet. Component sending (only) $\{m\}_K$ does.

Suppose component receives key K encrypted with its public key, sends back $\{m\}_K$. Does not preserve secrecy of m against attackers eavesdropping on and inserting messages on the link, but against attackers unable to insert messages.

➔

Tool-support: Concepts

Meaning of diagrams stated informally in (OMG 2003).

Ambiguities problem for

- tool support
- establishing behavioral properties (safety, security)

Need precise semantics for used part of UML, especially to ensure security requirements.

Formal semantics for UML

Diagrams in context (using subsystems).

Model actions and internal activities explicitly.

Message exchange between objects or components (incl. event dispatching).

Include adversary model arising from physical environment in deployment diagram.

Use Abstract State Machines (pseudo-code).

Tool-supported analysis

Choose drawing tool for UML specifications.

Commercial modelling tools: so far mainly syntactic checks and code-generation.

Analyze specifications via XMI (XML Metadata Interchange).

UML Drawing Tools

Wide range of existing tools.

Consider some, selected under following Criteria (Shabalin 2002):

- Support for all (UMLsec-) relevant diagram types.
- Support for custom UML extensions.
- Availability (test version, etc).
- Prevalence on the market.

Selected Tools

- Rational Rose. Developed by major participant in development of UML; market leader.
- Visio for Enterprise Architect. Part of Microsoft Developer Studio .NET.
- Together. Often referenced as one of the best UML tools.
- ArgoUML. Open Source Project, therefore interesting for academic community. Commercial variant Poseidon.

Comparison

Evaluated features:

Support for custom UML extensions.

- Model export; standards support; tool interoperability.
- Ability to enforce model rules, detect errors, etc.
- User interface quality.
- Possibility to use the tool for free for academic institutions.

Rational Rose (Rational Software Corporation)

One of the oldest on the market.

+ Free academic license.

+ Widely used in the industry.

+ Export to different XMI versions.

- Insufficient support for UML extensions (custom stereotypes yes; tags and constraints no).

- Limited support for checking syntactic correctness.

- Very inconvenient user interface. Bad layout control.

- Lack of compatibility between versions and with other Rational products for UML modelling.

Together from TogetherSoft

Widely used in the development community. Very good round-trip engineering between the UML model and the code.

+ Free academic license.

+ Written in Java, therefore platform-independent.

+ Nice, intuitive user interface.

+ Export to different XMI versions; recommendations which for which tool.

- Insufficient support for UML extensions (custom stereotypes yes; tags and constraints no).

Visio from Microsoft Corporation

Has recently been extended with UML editing support.

+ Good user interface.

+ Full support for UML extensions.

+ Very good correspondence to UML standard. Checks dynamically for syntactic correctness; suggestions for fixing errors.

- No free academic license.

- Proprietary, undocumented file format; no export to XMI or other tools.

- No round-trip engineering support. No way back after code generation.

ArgoUML / Poseidon

ArgoUML: Open Source Project. Commercial extension Poseidon (Gentleware), same internal data format.

+ Open Source.
+ Written in Java, therefore platform-independent.
+ XMI default model format.
+ Solid mature product with good UML specification support.

Tool-supported analysis

Commercial modelling tools: so far mainly syntactic checks and code-generation.

Goal: more sophisticated analysis; connection to verification tools.

Several possibilities:

- General purpose language with integrated XML parser (Perl, …)
- Special purpose XML parsing language (XSLT, …)
- Data Binding (Castor; XMI: e.g. MDR)

Data-binding with MDR

Extracts data from XMI file into Java Objects, following UML 1.4 meta-model.

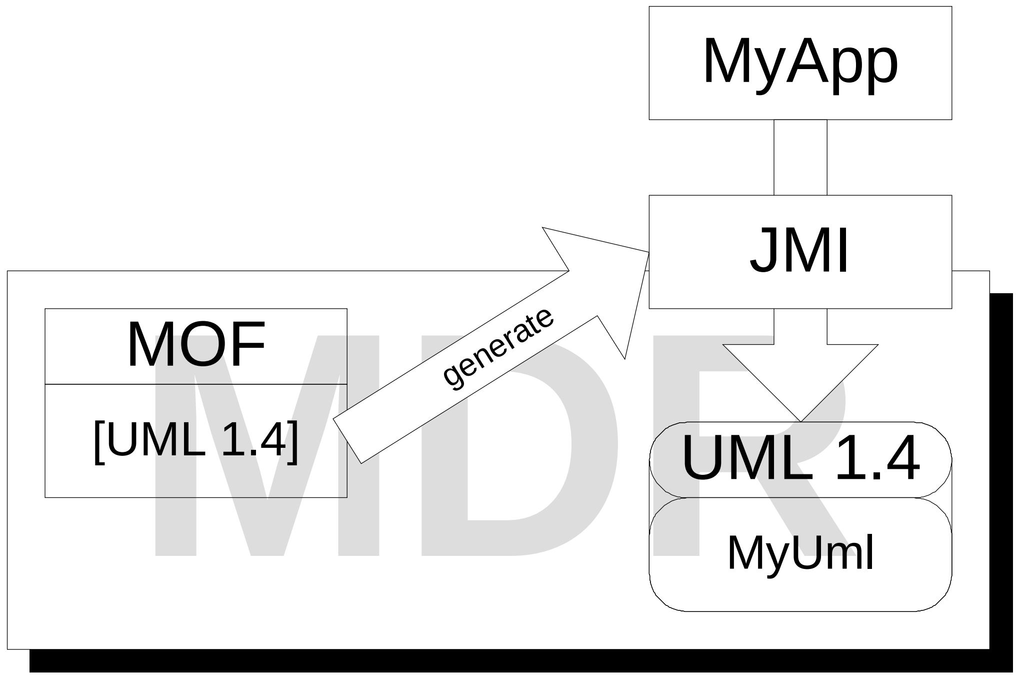
Access data via methods.

Advantage: No need to worry about XML.

Definition

- MDR = MetaData Repository
  - Load and Store a MOF Metamodel
  - Instantiate and Populate a Metamodel
  - Generate a JMI (Java Metadata Interface) Definition for a Metamodel
  - Access a Metamodel Instance

UML Processing



MDR Standards

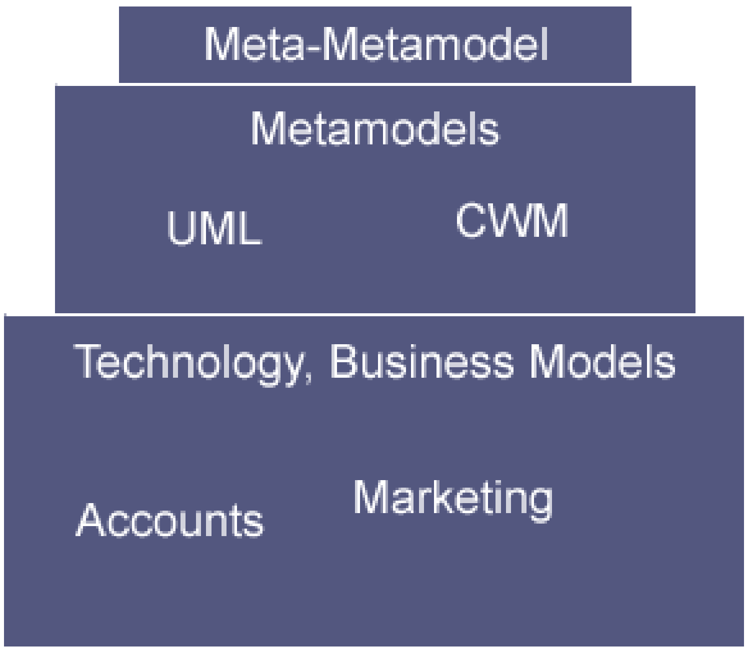
- MOF (Meta Object Facility)

  Abstract format for describing metamodels
- XMI (XML Metadata Interchange)

  Defines XML format for a MOF metamodel
- JMI (Java Metadata Interface)

  Defines mapping from MOF to Java

MOF Architecture



- Meta-Metamodel (M3)
  - defined by OMG
- Metamodels (M2)
  - user-defined
  - e.g. UML 1.5, MOF, CWM
  - can be created with uml2mof
- Business Model (M1)
  - instances of Metamodels
  - e.g. UML class diagram
- Information (M0)
  - instance of model
  - e.g. implementation of UML modelled classes in Java

MOF (Meta Object Facility)

OMG Standard for Metamodeling

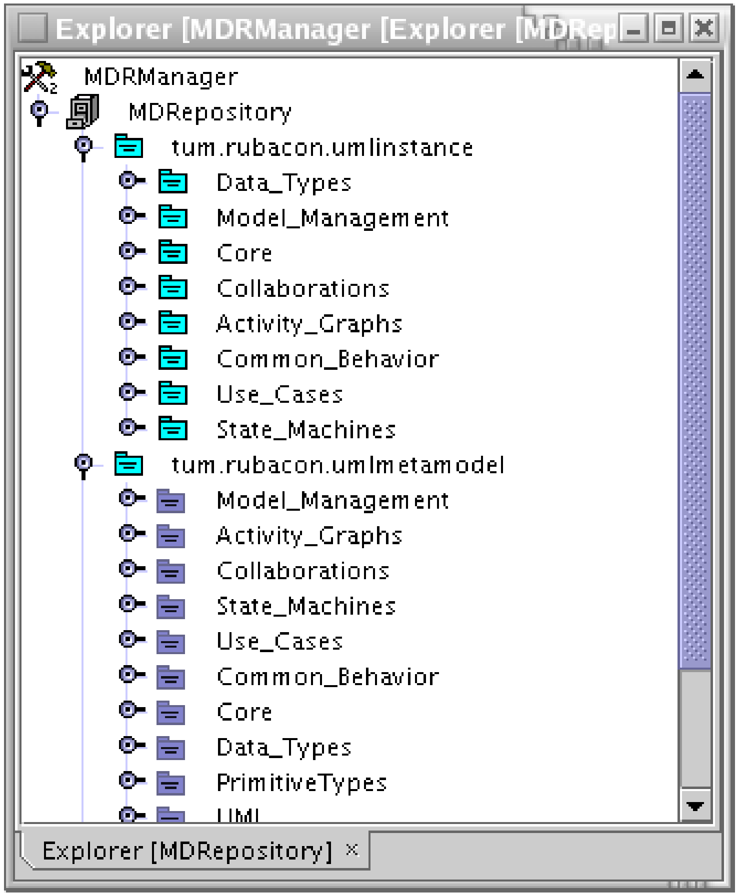
Meta-Metamodel	MetaClass, MetaAssociation - MOF Model
Metamodel	Class, Attribute, Dependency - UML (as language), CWM
Model	Person, House, City - UML model
Data	(Bob Marley, 1975) (Bonn) - Running Program

JMI: MOF Interfaces

- IDL mapping for manipulating Metadata
  - API for manipulating information contained in an instance of a Metamodel
  - MOF is MOF compliant!
  - Metamodels can be manipulated by this IDL mapping
  - JMI is MOF to Java mapping
  - JMI has same functionality
- Reflective APIs
  - manipulation of complex information
  - can be used without generating the IDL mapping
  - MDR has implemented these interfaces

Netbeans MDR-Explorer

- Part of Netbeans IDE
- Browse Repositories
- Create Instances
- Load XMI Data
- Generate JMI Interfaces
- Shows
  - Extents
  - Metamodels
  - Instances



MDR Repository: Loading Models

- Metamodel is instance of another Metamodel
- Loading Model = Loading Metamodel
- Needed Objects:
  - MDRepository
  - MofPackage
  - XMISaxReaderImpl

- Java Code-Snippet:

```
MDRepository rep;
UmlPackage uml;
// Objekte erzeugen:
rep =

    MDRManager.getDefault().getDefaultRepository()
    ;
reader =
  (XMISaxReaderImpl)Lookup.getDefault().lookup(
           XmiReader.class);

// loading extent:
uml = (UmlPackage)rep.getExtent(„name“);

// creating Extent:
uml = (UmlPackage)rep.createExtent(„name“);

// loading XMI:
reader.read(„url“, MofPackage);,
```

MDR Repository: Reading Data

- Requires open Repository and Package
- Requires JMI Interfaces
- Problem: where is the data I need?
- To find Objects:
  - open Model in MDR-Explorer
  - browse to the desired Element
  - use the getter Functions to retrieve the element

- Example: Loading UML Class:

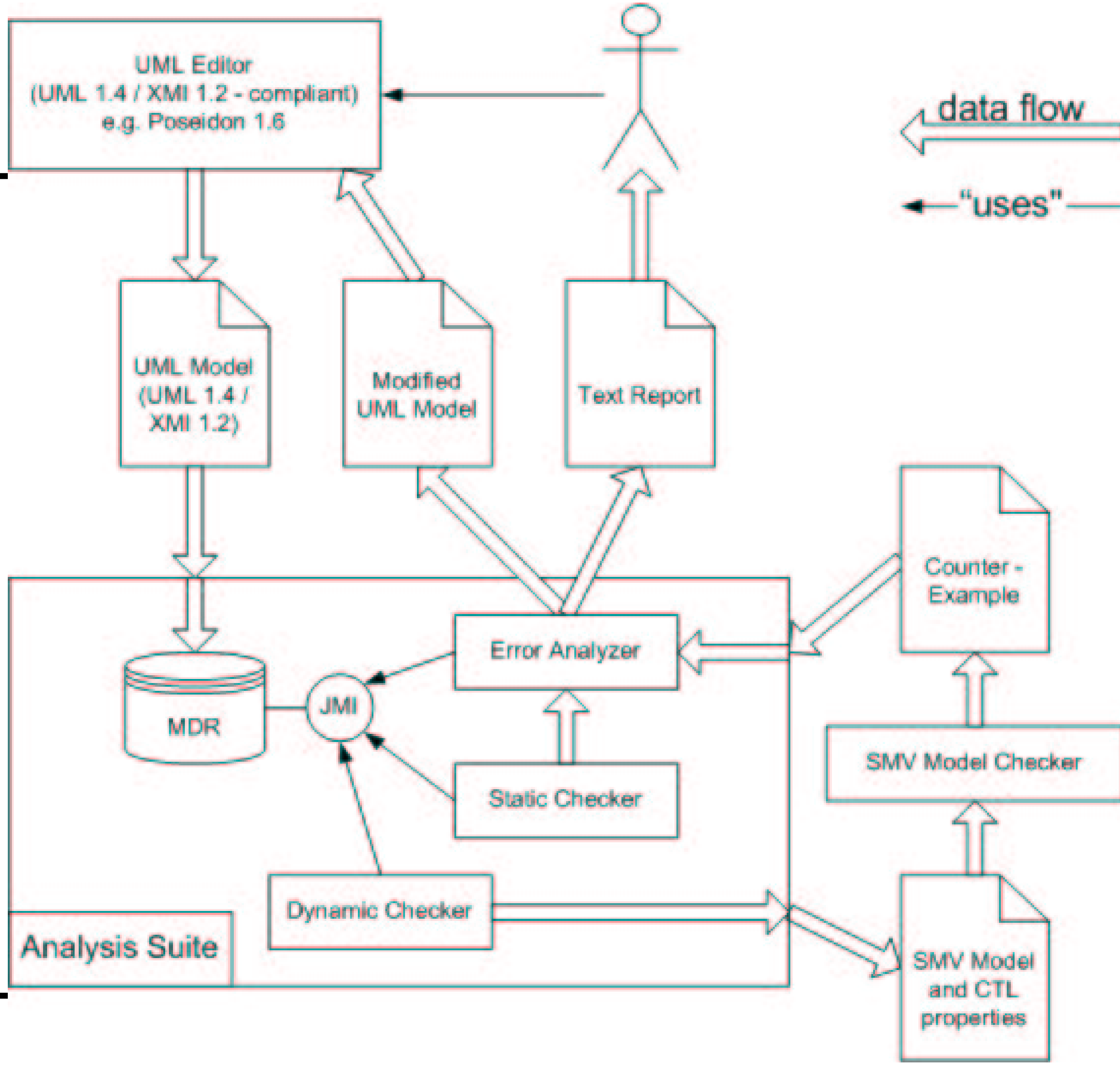
```
Iterator it =
  uml.getCore().getUmlClass(
  ).refAllOfClass().iterator
  ();

while (it.hasNext()) {
   UmlClass uc =
  (umlClass)it.next();

   // .. do anything with
  UmlClass ..
}
```

Connection with analysis tool

Industrial CASE tool with UML-like notation: AUTOFOCUS (http://autofocus.informatik.tu-muenchen.de)

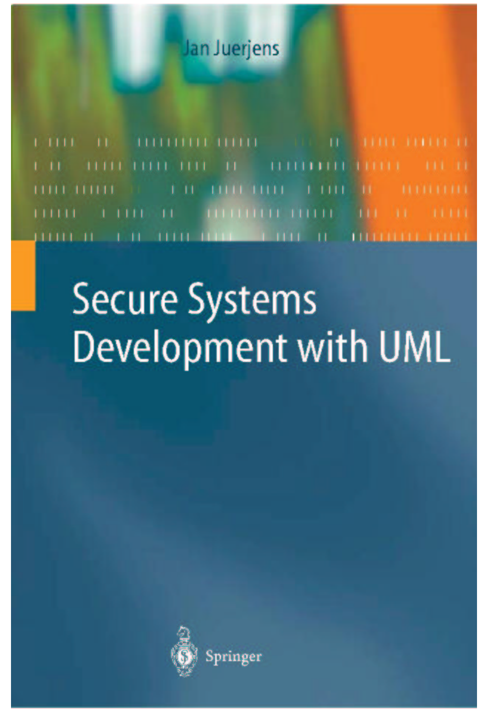
- Simulation
- Validation (Consistency, Testing, Model Checking)
- Code Generation (e.g. Java, C, Ada)
- Connection to Matlab

Connect UML tool to underlying analysis engine.


UML Editor
(UML 1.4 / XMI 1.2 - compliant)
e.g. Poseidon 1.6
data flow
"uses"
UML Model
(UML 1.4 /
XMI 1.2)
Modified
UML Model
Text Report
Counter -
Example
Error Analyzer
MDR
JMI
Static Checker
SMV Model Checker
Dynamic Checker
Analysis Suite
SMV Model
and CTL
properties

Some resources

Book: Jan Jürjens, Secure Systems Development with UML, Springer-Verlag, due 2003



Follow-on Tutorials: Sept: FME (Pisa), FDL (Frankfurt), SAFECOMP (Edinburgh), FORTE (BERLIN); Oct: Informatik (Frankfurt)

Special SoSyM issue on Critical Systems Development with UML

CSDUML'03 @ UML'03 conference (Oct. in SFO)

More information (slides etc.):
http://www4.in.tum.de/~juerjens/csdumltut
(user Participant, password Iwasthere)

Finally

We are always interested in industrial challenges for our tools, methods, and ideas to solve practical problems.
More info: http://www4.in.tum.de/~secse

Contact me here or via Internet.

Thanks for your attention !

BREAK ! (until 3.30 pm)

Note:

We are always interested in industrial challenges for our tools, methods, and ideas to solve practical problems.

More info: http://www4.in.tum.de/~secse

Contact me here or via Internet.

Roadmap

Prologue

UML

UMLsec: The profile

Security patterns

Case studies

Using Java security, CORBAsec

Tools