UMLsec
Presenting the Profile

Jan Jürjens

Software & Systems Engineering
Informatics, TU Munich
Germany

juerjens@in.tum.de

http://www.jurjens.de/jan

Roadmap

Prologue

The profile

Security analysis

Security patterns

Case studies

Using Java security, CORBAsec

Outlook

A need for Security

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

Attacks threaten economical and physical well-being 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.

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.
- Designers often lack background in security.
- Security as an afterthought.

Even experts may fail:

- Needham-Schroeder protocol: published 1978
- attacks found 1981 (Denning-Sacco), 1995 (Lowe)

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": unsatisfactory.

- insecure (how much damage until discovered ?)
- disruptive (distributing patches costs money, destroys confidence, annoys customers)

Traditional formal methods: expensive.

- training people
- constructing formal specifications

Goal: Security by design

Consider security

- within industrial development context
- from early on.

"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

Unified Modeling Language (UML):

- visual modeling for OO systems
- different views on a system
- high degree of abstraction possible
- de-facto industry standard (OMG)
- standard extension mechanisms

UMLsec

UMLsec: extension for secure systems development.

Goals:

- evaluate UML specifications for vulnerabilities in design
- encapsulate established rules of prudent security engineering
- make available to developers not specialized in security
- consider security from early design phases, in system context
- make verification cost-effective

Roadmap

Prologue

The profile

Security analysis

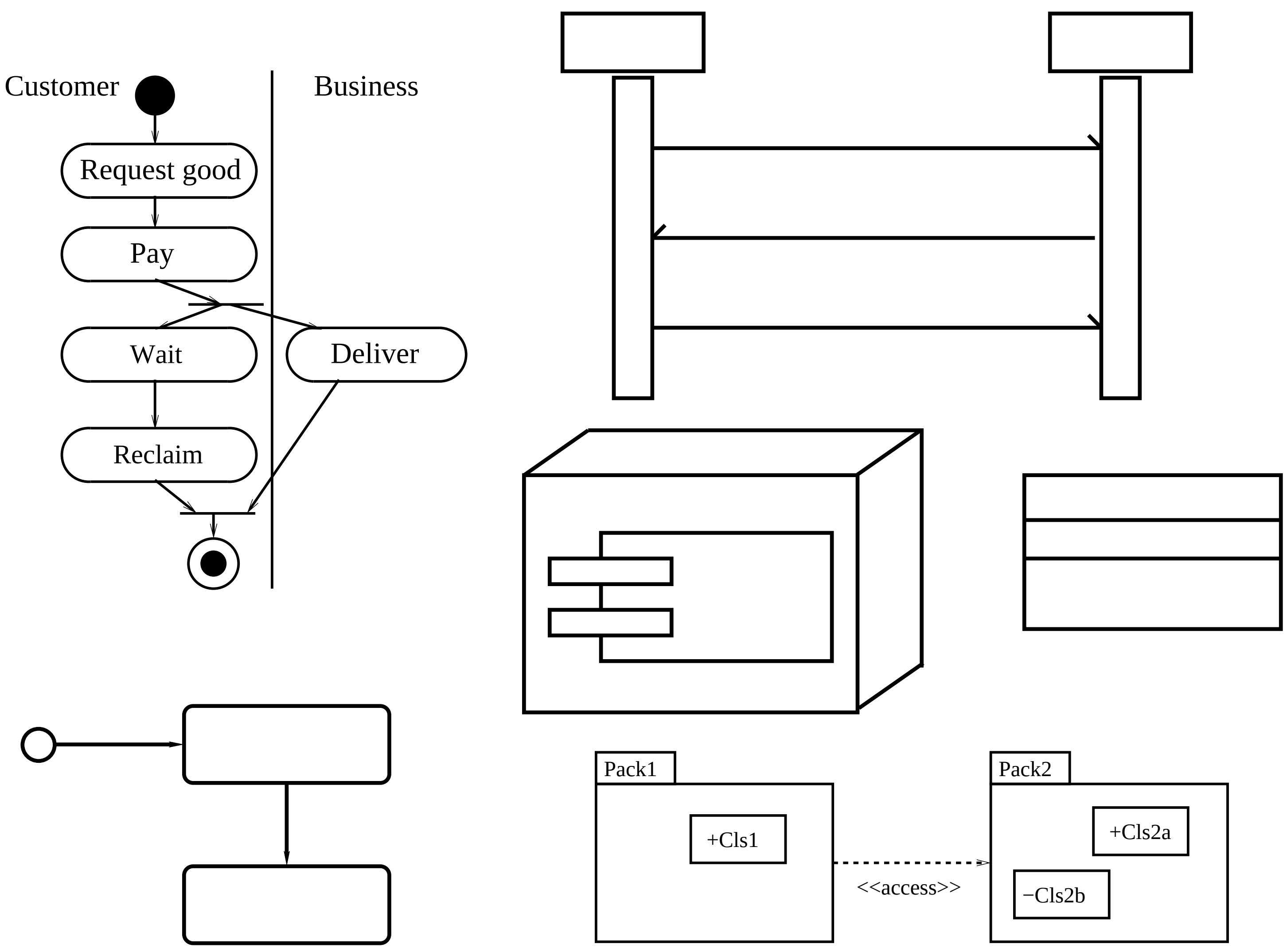
Security patterns

Case studies

Using Java security, CORBAsec

Outlook

A glimpse at UML



Used fragment of UML

Activity diagram: flow of control between system components

Class diagram: class structure of the system

Sequence diagram: interaction between components by message exchange

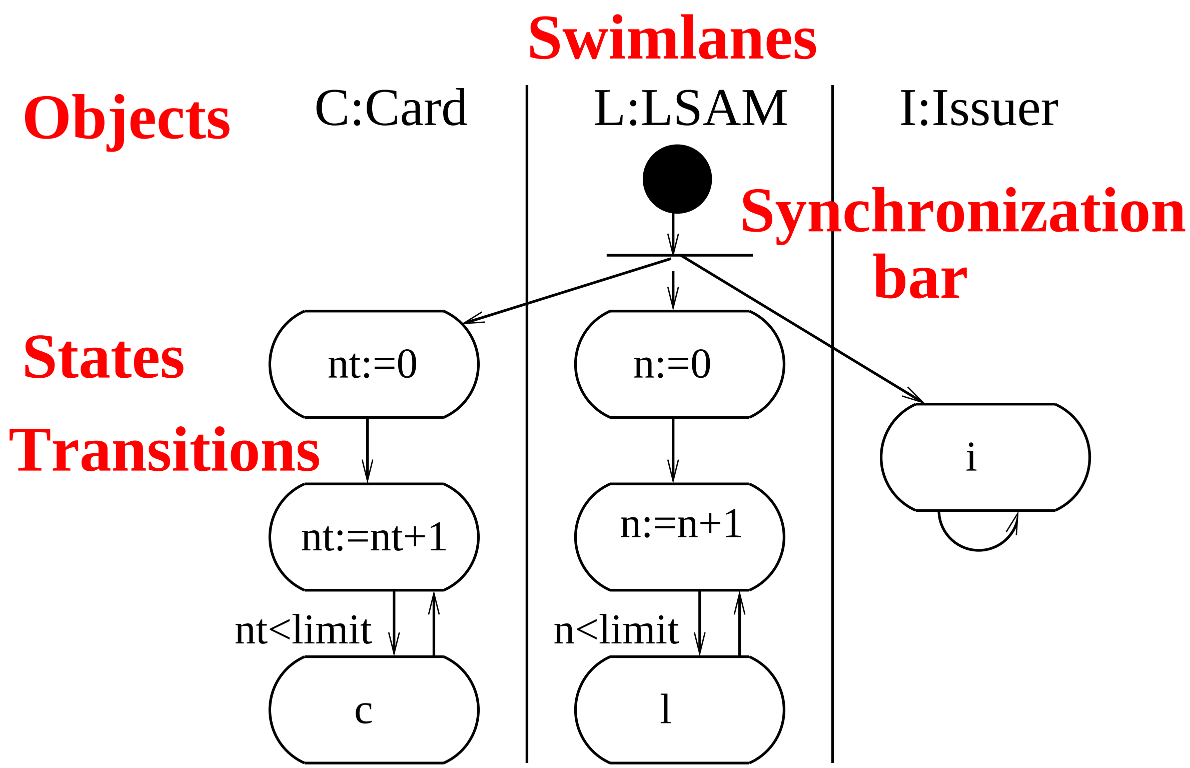
Statechart diagram: dynamic component behaviour

Deployment diagram: Components in physical environment

Package: collect system parts into groups

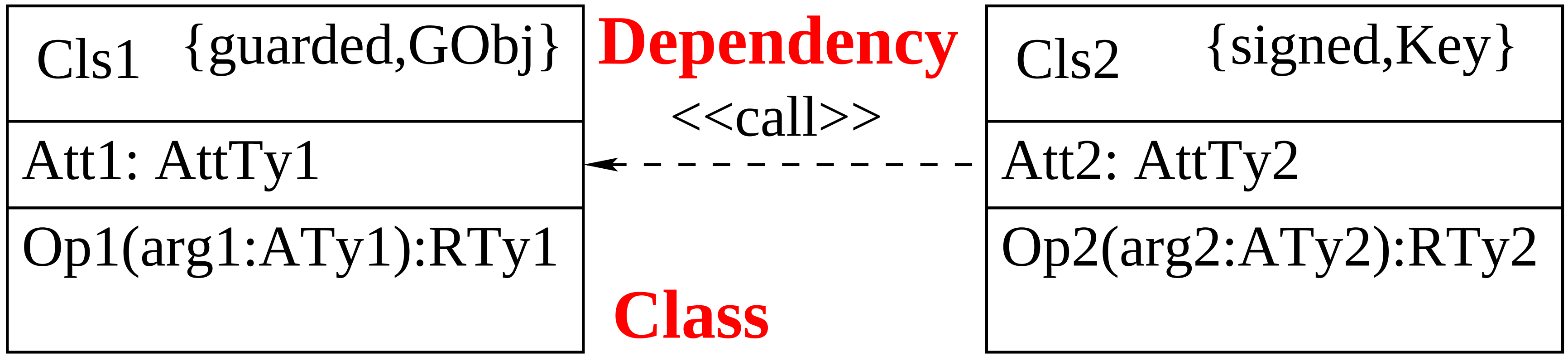
Current: UML 1.4 (released Feb. 2001)

UML run-through: Activity diagrams



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

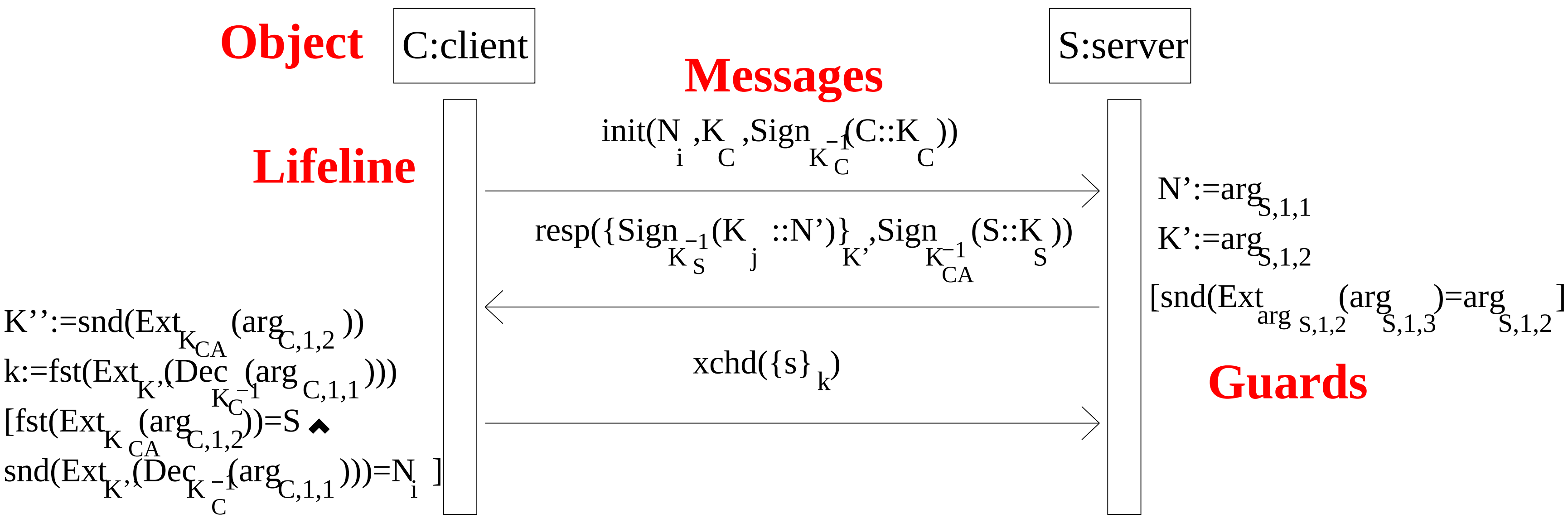
UML run-through: Class diagrams



Class structure of system.

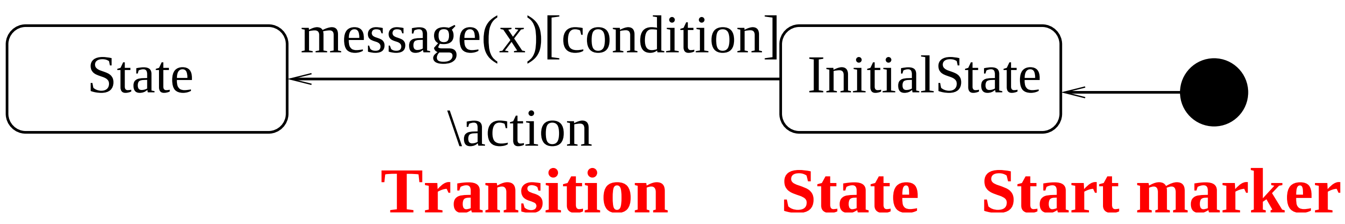
Classes with attributes and operations/signals; relationships between classes.

UML run-through: Sequence diagrams



Describe interaction between objects or system components via message exchange.

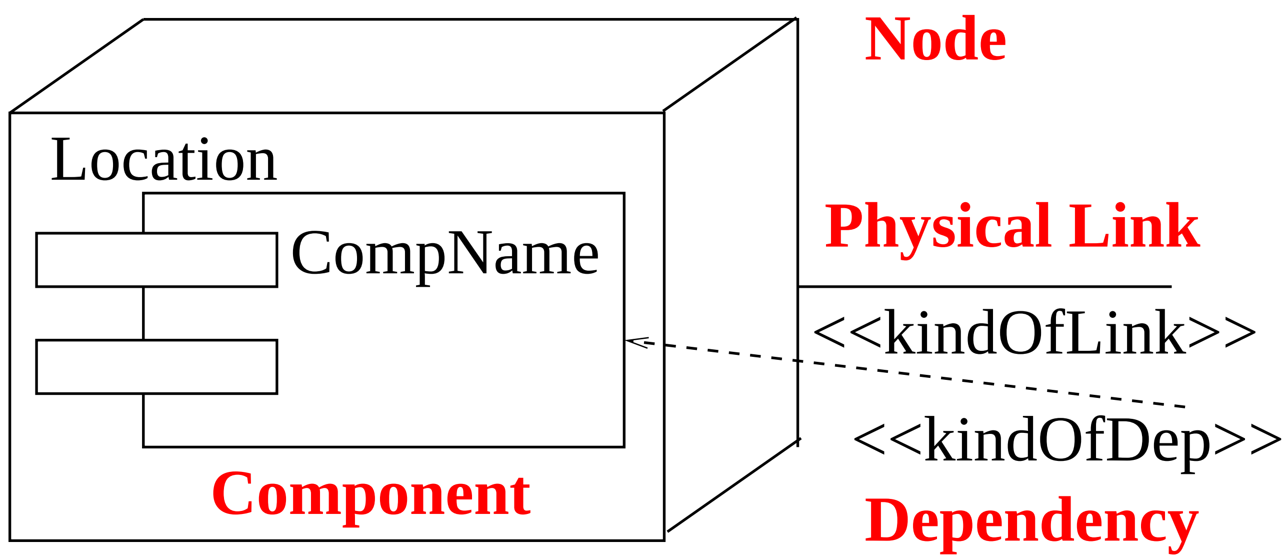
UML run-through: Statecharts



Dynamic behaviour of individual object.

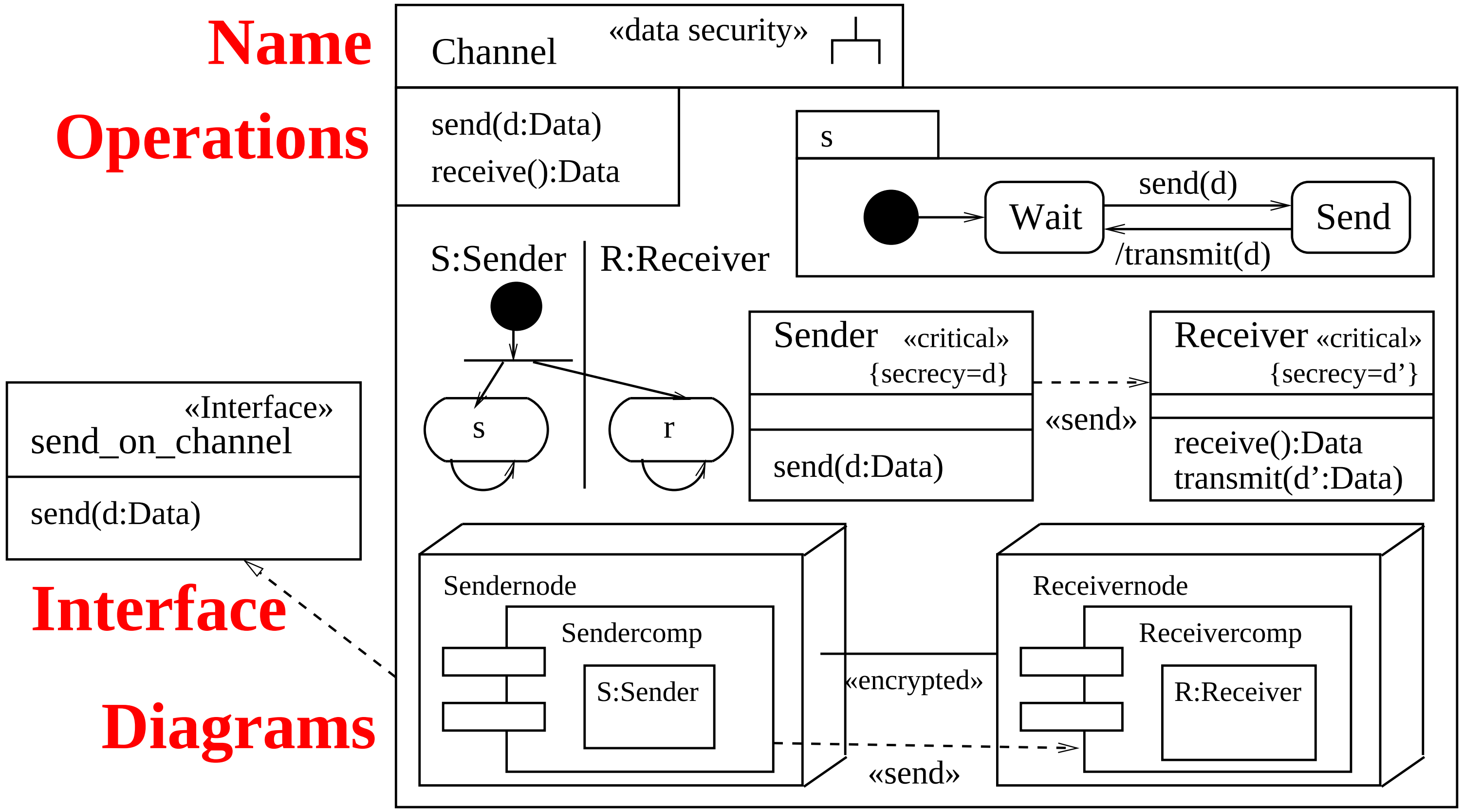
Input events cause state change and output actions.

UML run-through: Deployment diagrams



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

UML run-through: Packages



May be used to organize model elements into groups.

UML Extension mechanisms

Stereotype: specialize model element using $\langle\!\langle$ label $\rangle\!\rangle$

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

Constraint: refine semantics of stereotyped element

Profile: gather above information

The UMLsec profile

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

Use associated constraints to evaluate specifications and 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
smart card	node			smart card node
secure links	subsystem		dependency security matched by links	enforces secure communication links
secrecy	dependency			assumes secrecy
high	dependency			high sensitivity
secure dependency	subsystem		call,send respect data security	structural interaction data security
critical	class	secrecy		critical class
no down-flow	subsystem	high	prevents down-flow	information flow
no up-flow	subsystem	high	prevents up-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
provable	state state	action, ,cert	action is non-deniable	non-repudiation requirement
guarded access	subsystem		guarded objects accessed through guards	access control using guard objects
guarded	class	guard		guarded class

≪ Internet ≫, ≪ encrypted ≫, ≪ LAN ≫, ≪ smartcard ≫,...

Denote kinds of communication links resp. system nodes.

For adversary type A, stereotype s, have set Threats$_A(s) \subseteq$ {deleteall, deleteelt, readall, insertelt, access} of actions that adversaries are capable of. Default attacker:

Stereotype	Threats$_{default}()$
Internet	{deleteall,deleteelt,readall,insertelt}
encrypted	{deleteall}
LAN	$\emptyset$
wire	$\emptyset$
smart card	$\emptyset$
POS device	$\emptyset$
issuer node	$\emptyset$

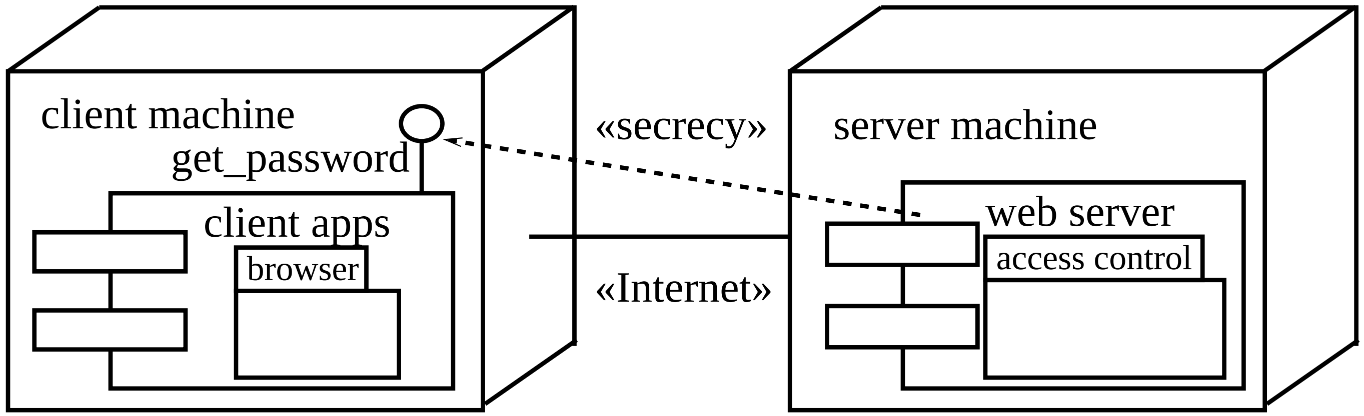
⟪ secure links ⟫

Ensures that security requirements on communication met by physical layer.

Constraint: for each dependency d with stereotype $s \in \{$⟪ secrecy ⟫, ⟪ integrity ⟫, ⟪ high ⟫$\}$ between components on nodes $n \neq m$, have communication link l between n and m with stereotype t such that

- if $s =$ ⟪ high ⟫: have $\mathsf{Threats}_A(t) = \emptyset$,
- if $s =$ ⟪ secrecy ⟫: have $\mathsf{readall} \notin \mathsf{Threats}_A(t)$
- if $s =$ ⟪ integrity ⟫: have $\mathsf{insertelt} \notin \mathsf{Threats}_A(t)$.

Example ⟪ secure links ⟫



Given the *default* adversary type, the constraint for the stereotype ⟪ securelinks ⟫ is violated:
According to the $\text{Threats}_{default}(Internet)$ scenario, the ⟪ Internet ⟫ communication link does not provide communication secrecy against the *default* adversary.

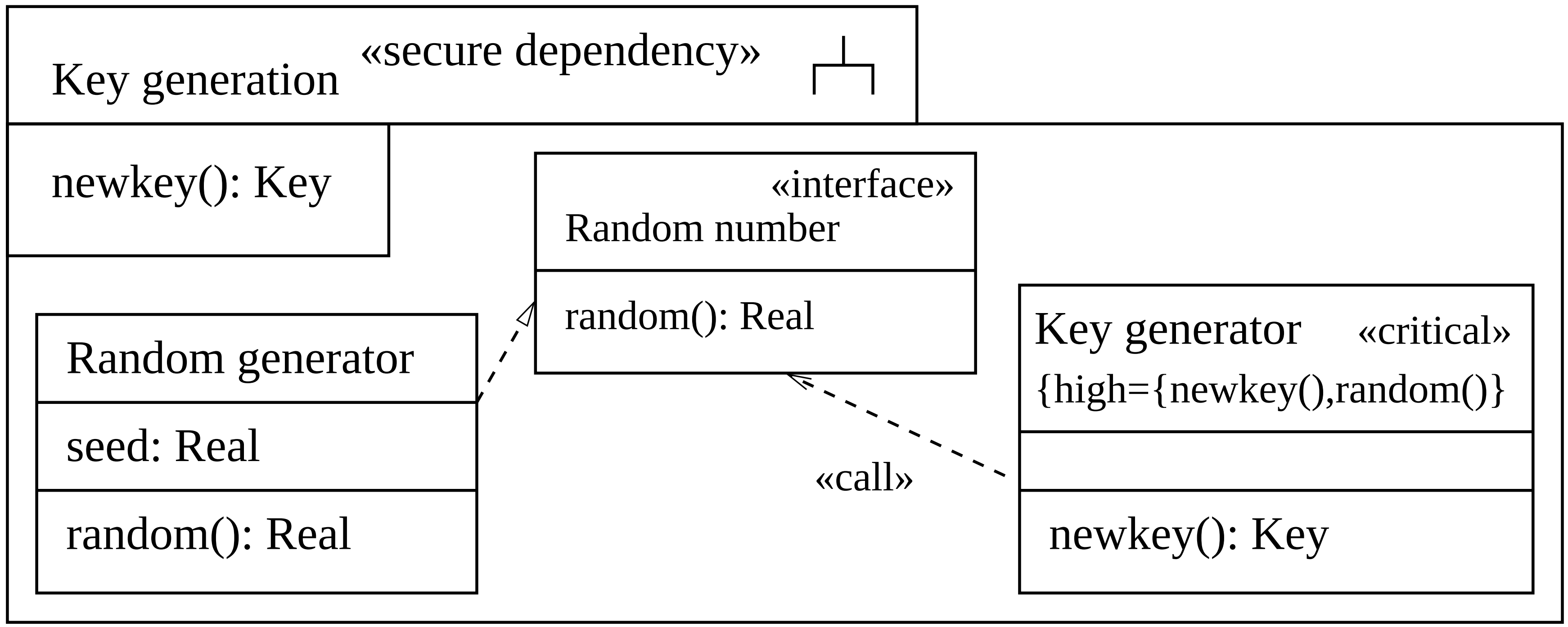
«secure dependency»

Ensures that «call» and «send» dependencies between components respect security requirements on communicated data given by tags {secrecy}, {integrity}, {high}.
Constraint: given «call» or «send» dependency from C to D:

- Any message n in D appears in {secrecy} (resp. {integrity} resp. {high}) in C if and only if does so in D.

- If message in D appears in {secrecy} (resp. {integrity} resp. {high}) in C dependency stereotyped «secrecy» (resp. «integrity» resp. «high»).

Example «secure dependency»



Specification violates constraint for «secure dependency»: Randomgenerator and «call» dependency do not provide security levels for random() required by Keygenerator.

⟪ no down – flow ⟫, ⟪ no up – flow ⟫

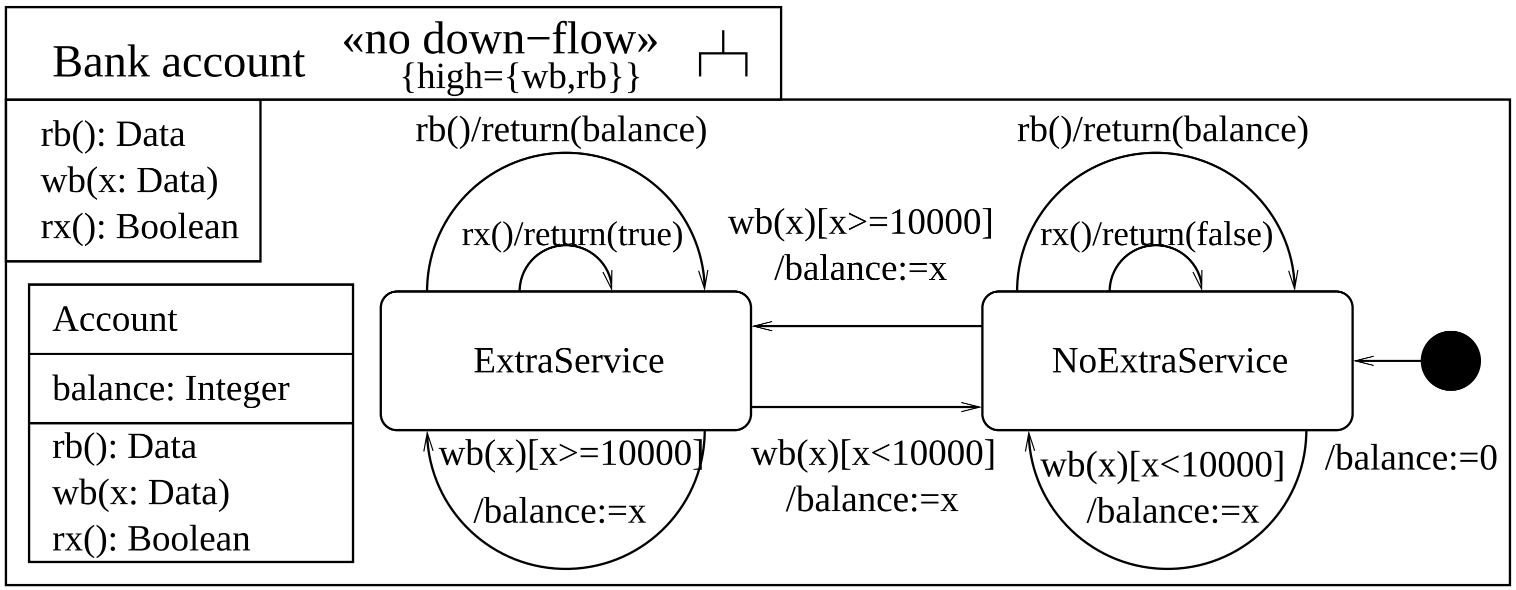
Enforce secure information flow.

Constraints:

⟪ nodown – flow ⟫: component prevents down-flow:
Value of any data specified in {high} may influence only the values of data also specified in {high}.

⟪ nodown – flow ⟫: component prevents up-flow:
Value of any data specified in {high} may be influenced only by the values of data also specified in {high}.

Example ⟨⟨ no down – flow ⟩⟩



Bank account object allows secret balance to be read with rb() and written with wb(x). Is in state ExtraService exactly if balance is over 10000. State can be queried with rx(). Does not provide ⟨⟨ no down – flow ⟩⟩: partial information about input of high wb() returned by non-high rx().

«data security»

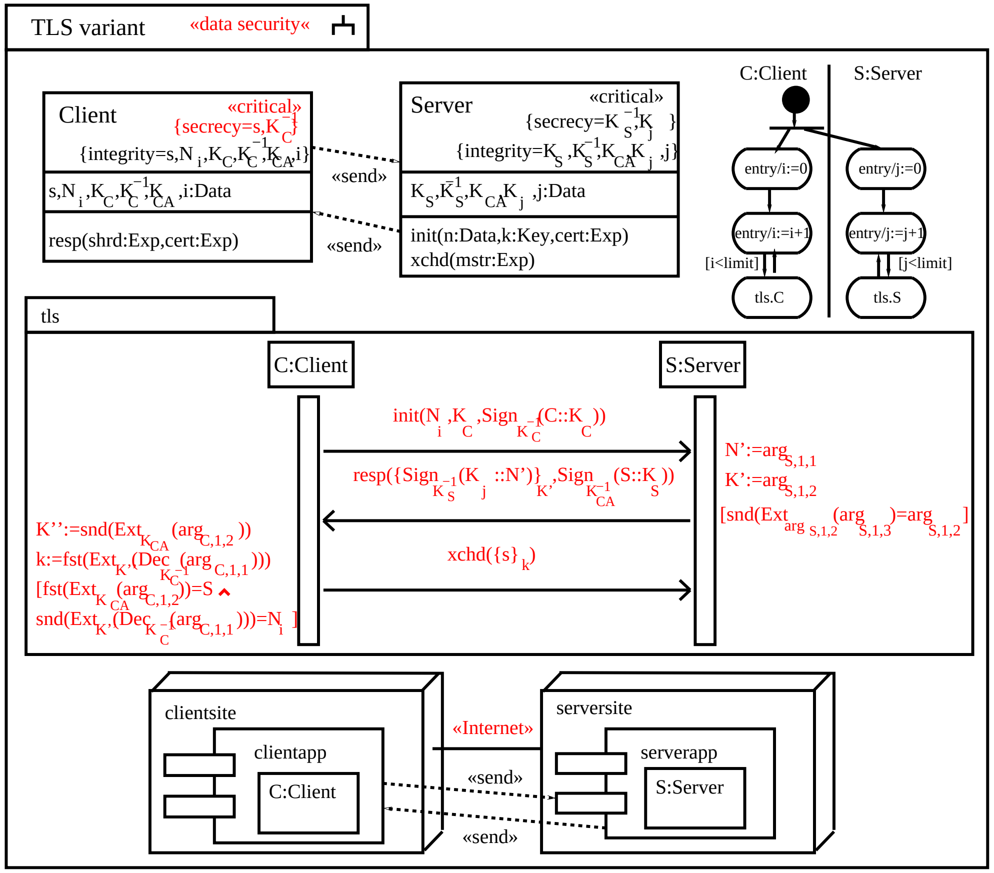
Ensures that data security requirements given by «critical» and associated tags respected with regard to threat scenario arising from the deployment diagram.

Constraints:

Secrecy of data designated {secrecy} preserved against adversaries of given type.

Integrity of data designated {integrity} preserved against adversaries of given type.

Example «data security»



Variant of TLS proposed at INFOCOM 1999.

Violates «datasecurity» (specifically, {secrecy} of s) against default adversary.

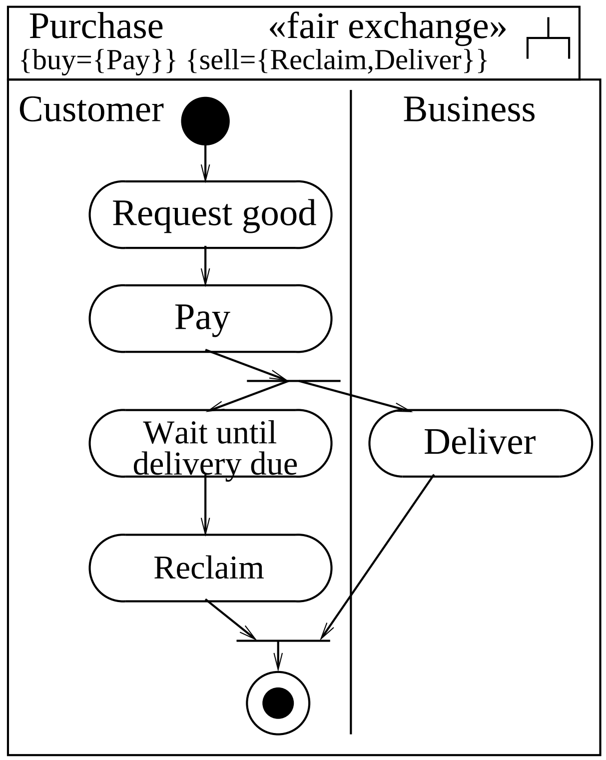
«fair exchange»

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 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.

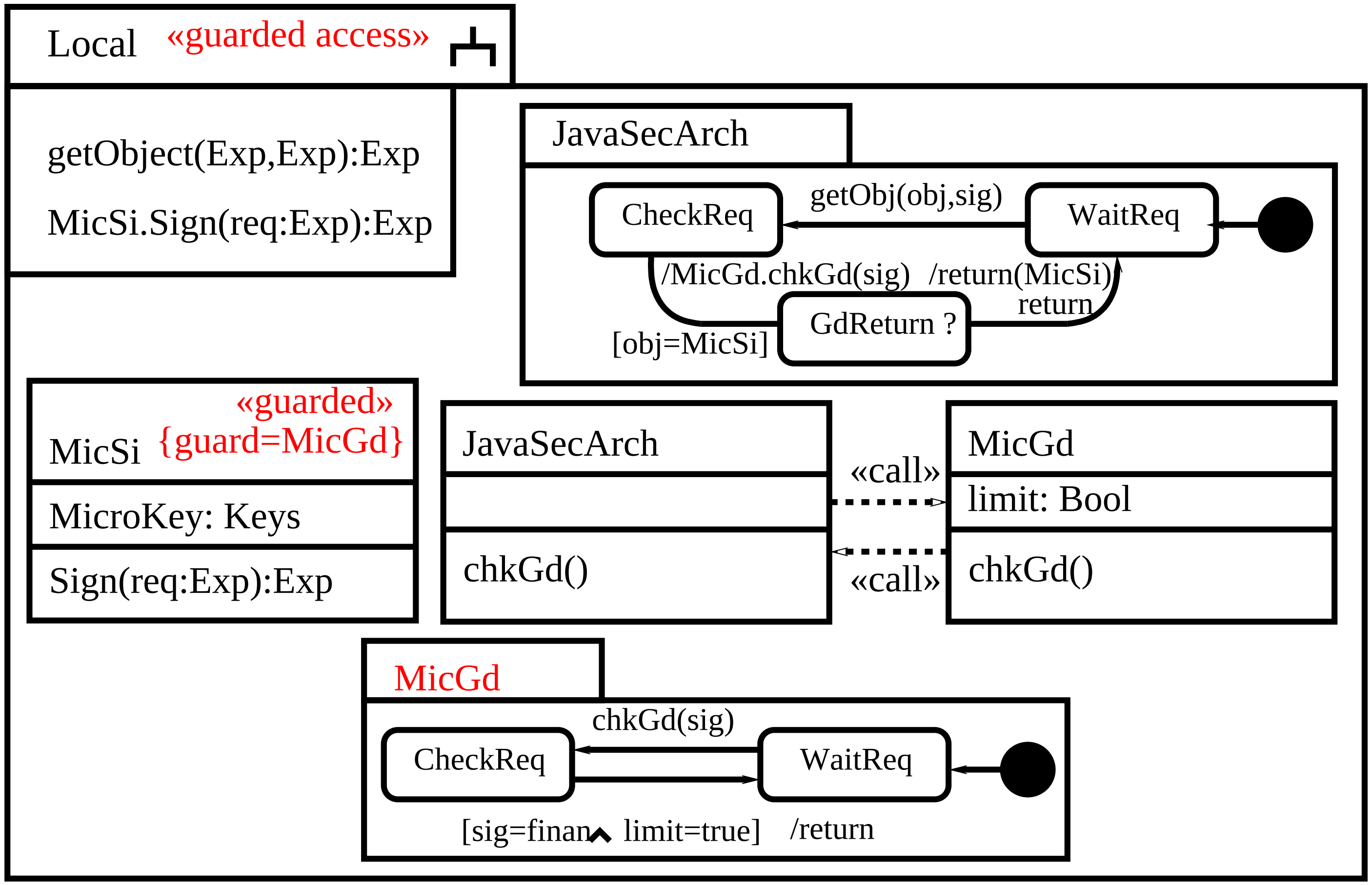
«guarded access»

Ensures that «guarded» classes only accessed through {guard} classes.

Constraints:

- names of «guarded» objects not publicly known
- each «guarded» class has {guard} class.

Example «guarded access»



Applets from internet bank and financial advisor need access to local financial data. Provides «guarded access».

Does UMLsec meet requirements ?

Security requirements: secrecy, integrity built in.

Threat scenarios: $\mathsf{Threats}_{adv}(ster)$: actions against stereotyped elements available to adversary.

Security concepts: E.g. ≪smart card≫ built in.

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

Security primitives: (A)symmetric encryption built in.

Physical security: Deployment diagrams, threat scenarios.

Security management: Use activity diagrams (e.g. ≪fair exchange≫).

Roadmap

Prologue

The profile

Security analysis

Security patterns

Case studies

Using Java security, CORBAsec

Outlook

Formal Semantics for UML: Why

Meaning of diagrams stated imprecisely in (OMG 2001).

Ambiguities problem for

- tool support
- establishing behavioral properites (e.g. security)

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

Formal semantics for UML: How

Diagrams in context (using subsystems).

Model actions and internal activities explicitly.

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

For UMLsec: include adversary arising from threat scenario in deployment diagram.

Use Abstract State Machines (pseudo-code).

Distributed Systems

Objects distributed over untrusted networks.

"Adversary" intercepts, modifies, deletes, inserts messages.

Cryptography provides security.

Security Analysis

Model classes of adversaries.

May attack different parts of the system in a specified way.

Example: *insider* attacker may intercept communication links in LAN.

To evaluate security of specification, execute jointly with adversary.

Abstract protocol descriptions I

Specify protocol participants as processes (following Dolev, Yao 1982).

In addition to expected participants, model attacker, who:

- may participate in protocol runs,
- knows some data in advance,
- may intercept and delete messages on some channels,
- may inject produced messages into some channels

Abstract protocol descriptions II

Keys are symbols, cryptoalgorithms are abstract operations.

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

Expressions

Exp: term algebra generated by **Var**$\cup$**Keys**$\cup$**Data** and

- $\_ :: \_$ (concatenation) and empty expression ε,
- $\{\_\}_\_$ (encryption)
- $\mathcal{D}ec\_(\_)$ (decryption)
- $\mathcal{S}ign\_(\_)$ (signing)
- $\mathcal{E}xt\_(\_)$ (extracting from signature)
- $\mathcal{H}ash(\_)$ (hashing)

by factoring out the equations $\mathcal{D}ec_{K^{-1}}(\{E\}_K) = E$ and $\mathcal{E}xt_K(\mathcal{S}ign_{K^{-1}}(E)) = E$ (for $K \in$ **Keys**).

Abstract adversary

Specify set $\mathcal{K}_A^0$ of initial knowledge of an adversary of type A.

To test secrecy of $M \in \mathbf{Exp}$ against attacker type A:

Jointly execute $\mathcal{S}$ and $\mathcal{A}$ where $\mathcal{A}$ is most powerful attacker of type A according to threat scenario from deployment diagram, with $M \notin \mathcal{K}_A^0$.
M is kept secret by $\mathcal{S}$ if M never thus output in clear.

Example: secrecy

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

Suppose component receives key K encrypted with its public key over communication link and 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.

Abstract adversary (alternative)

Define: $\mathcal{K}_A^{n+1}$ is the $\mathbf{Exp}$-subalgebra generated by $\mathcal{K}_A^n$ and the expressions received after $n+1$st iteration of the protocol.

Theorem. $\mathcal{S}$ keeps secrecy of M against attackers of type A iff there is no n with $M \in \mathcal{K}_A^n$.

Roadmap

Prologue

The profile

Security analysis

Security patterns

Case studies

Using Java security, CORBAsec

Outlook

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.

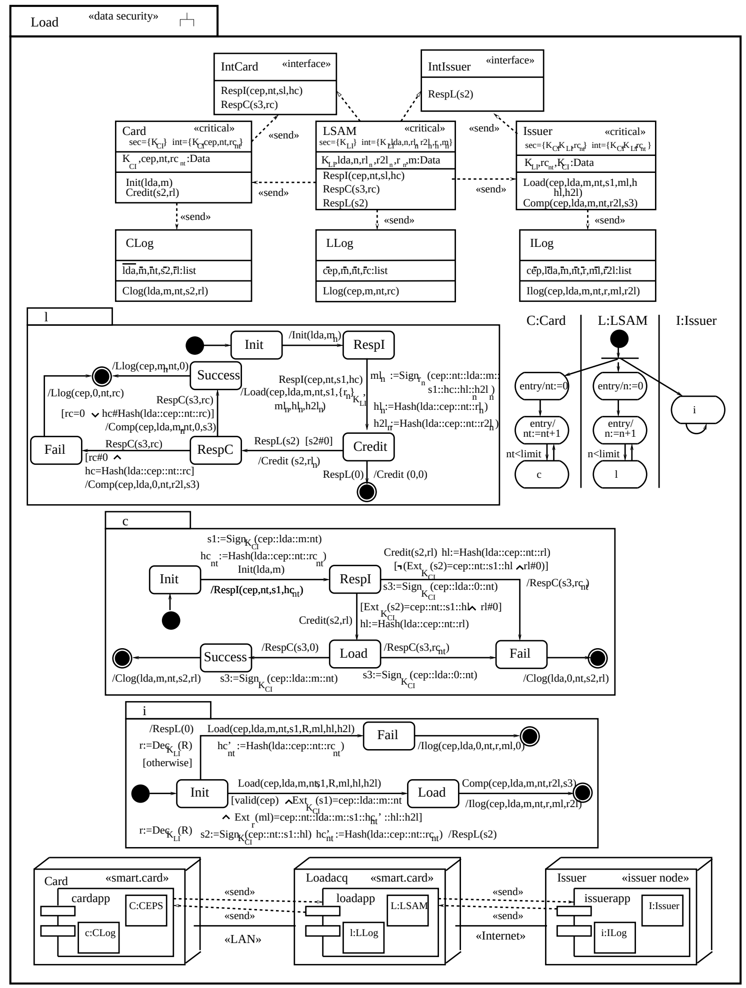
Method for reassurance may reduce this temptation.

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).



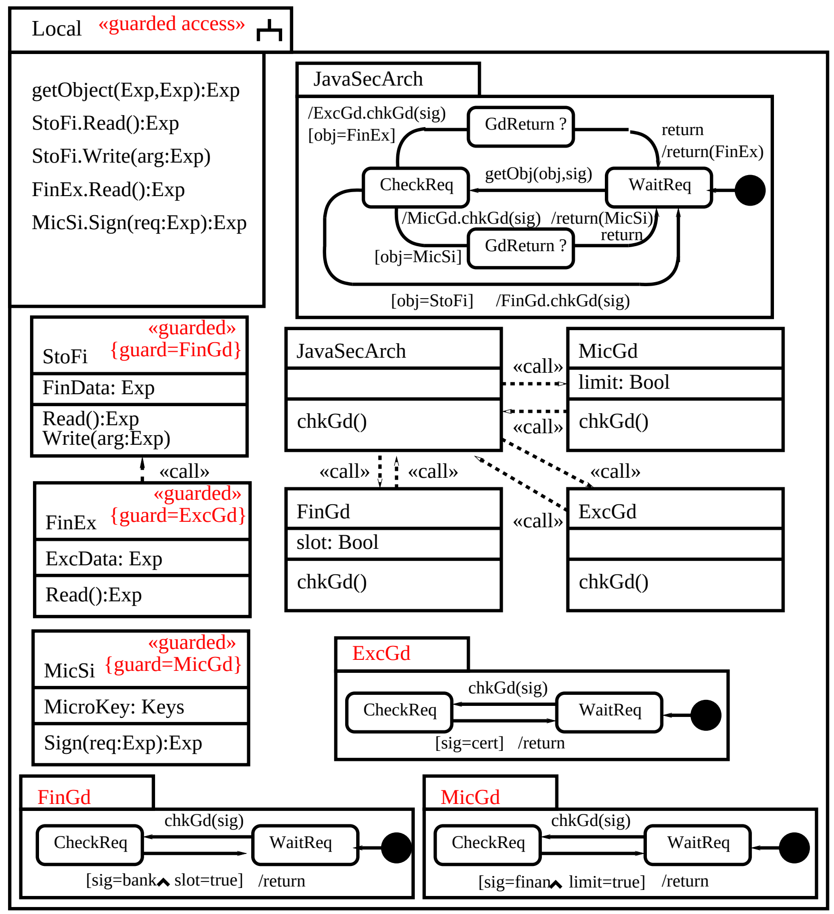
Complete mediation

Every access to every object must be checked for authority.

Can enforce principle e.g. in Java using guarded objects.

Ensure proper use of guards (prevent forgotten access checks).

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.

Specification satisfies separation of privilege wrt. privilege p if signature of two or more principals required to be granted p.

Formulate such requirements abstractly using activity diagrams.

Verify behavioural specifications wrt. these requirements.

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.

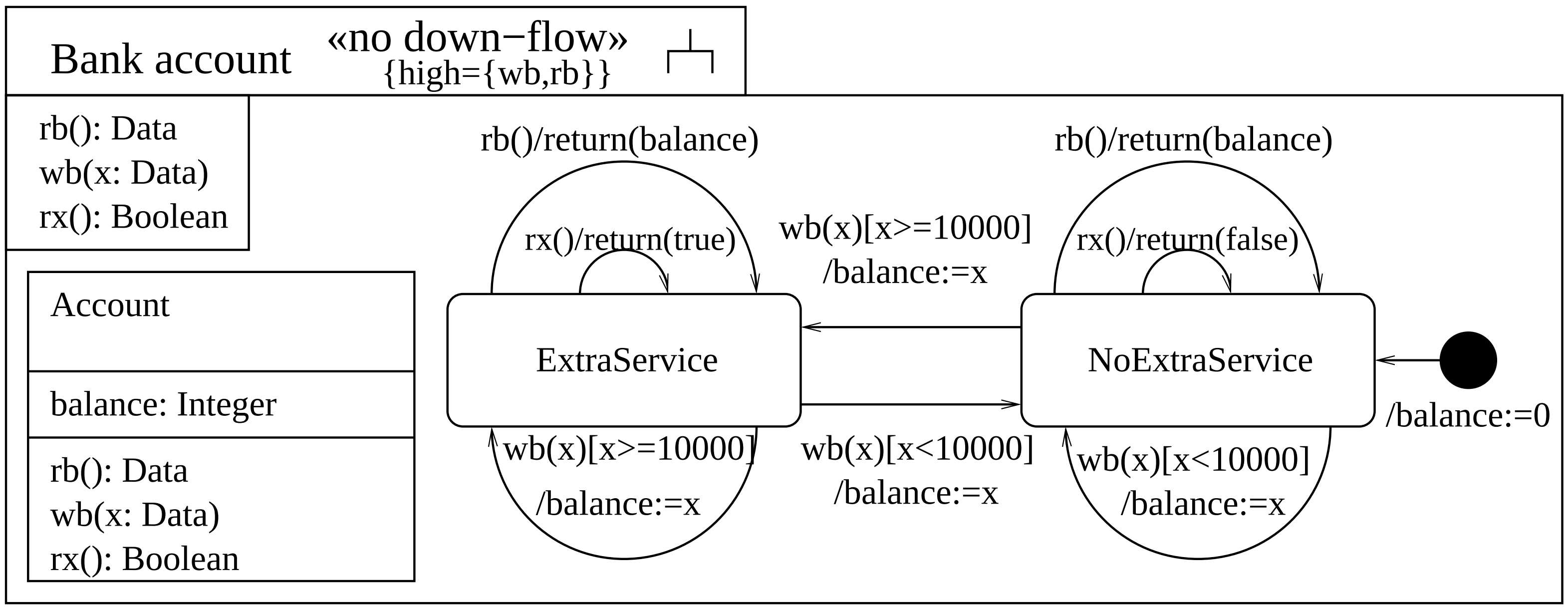
Violation of rules symptom of potential trouble; review design to be sure that trouble accounted for or unimportant.

Design principles reduce number and seriousness of flaws.

Security Patterns

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

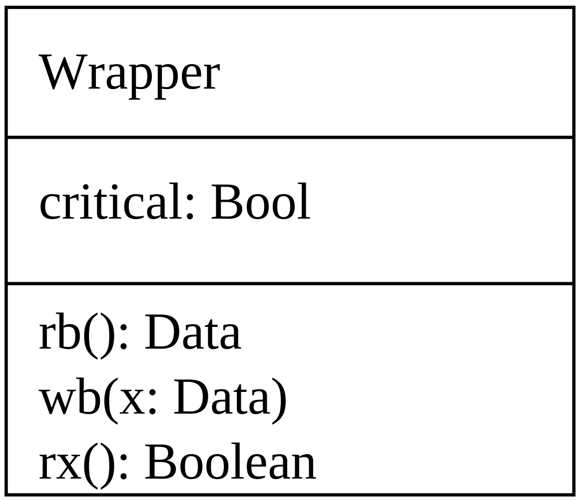
Example.

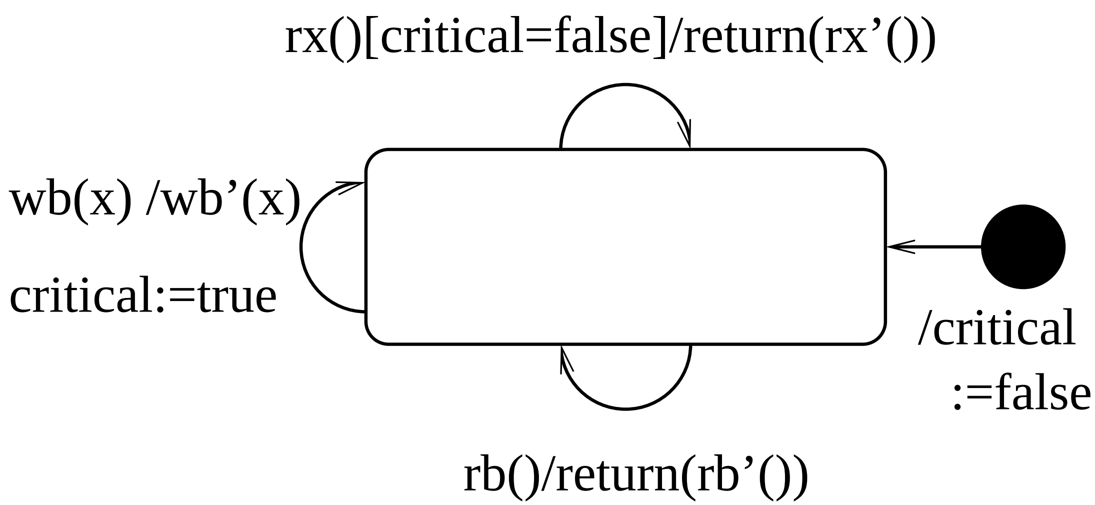


Problem: does not preserve security of account balance.

Solution: Wrapper Pattern

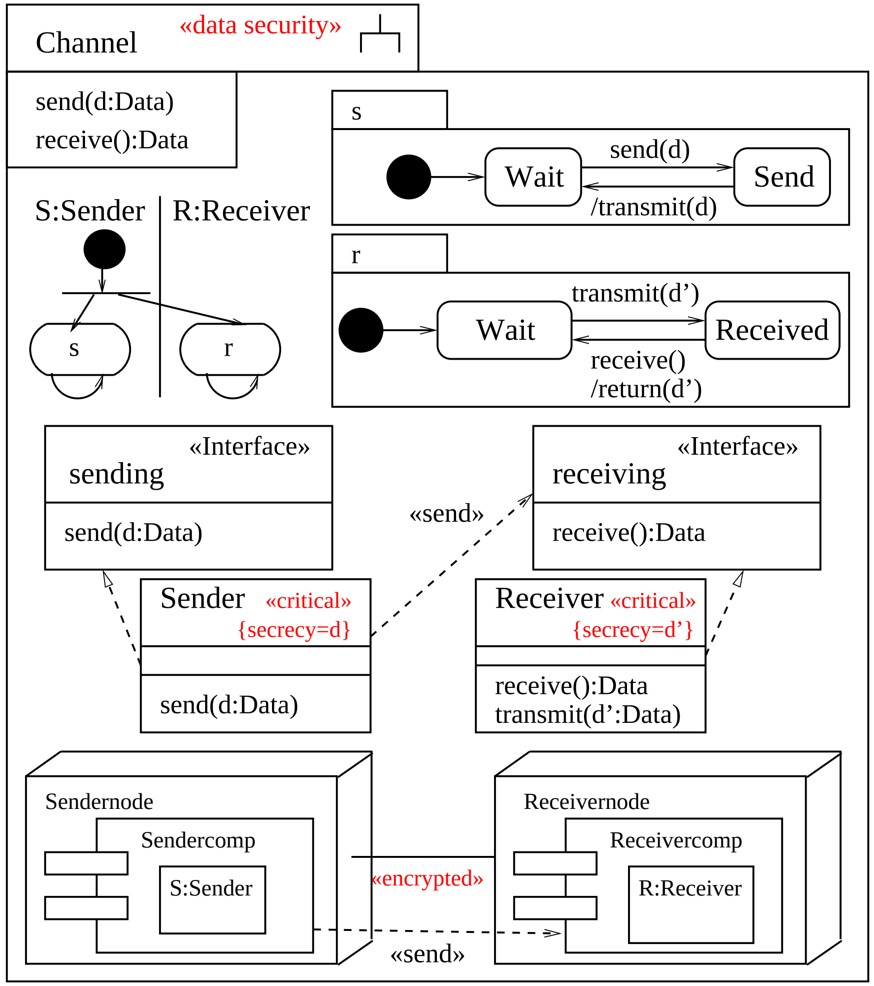
Technically, pattern application is transformation of specification.





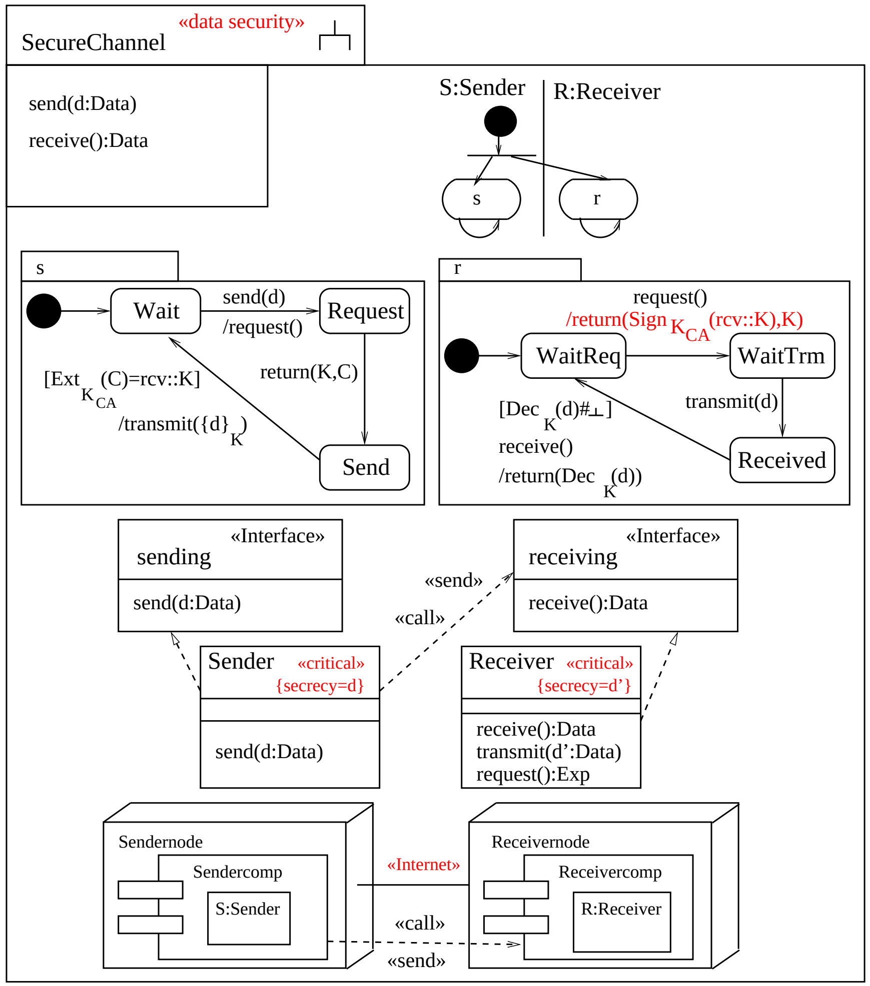
Use wrapper pattern to ensure that no low read after high write.
Can check this is secure (once and for all).

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

The profile

Security analysis

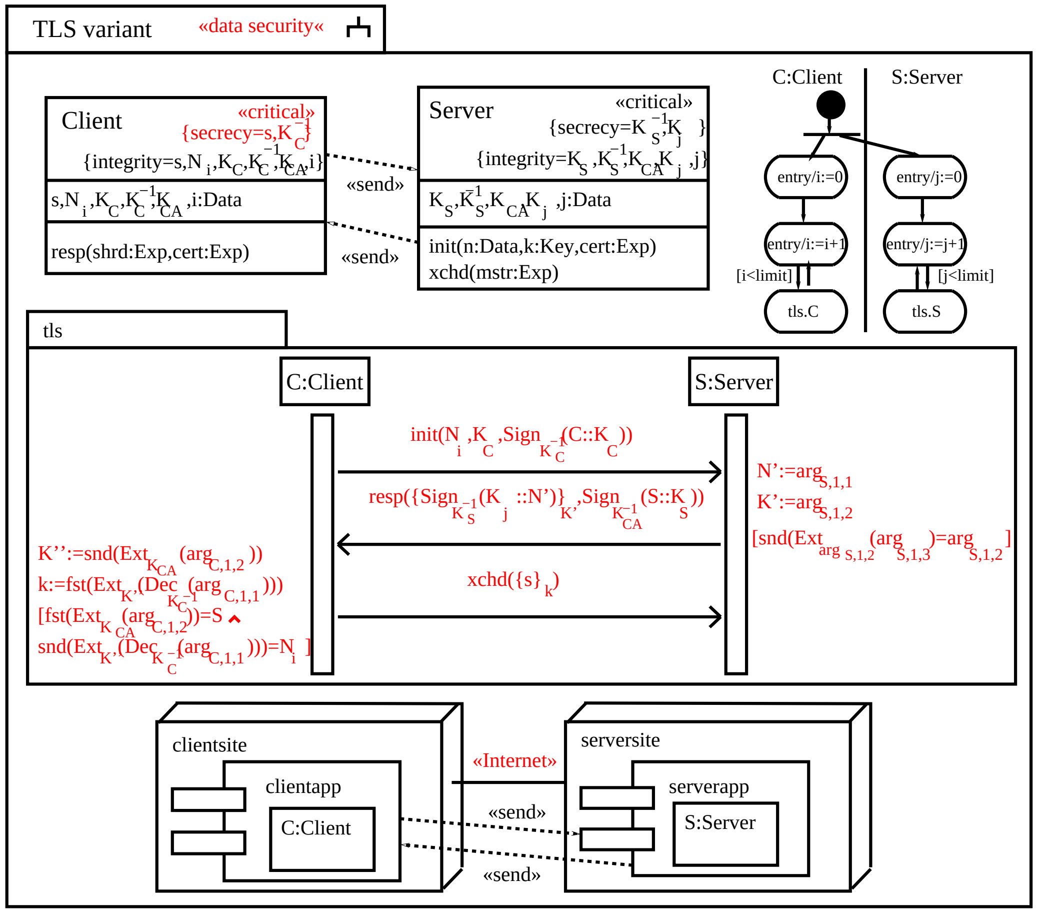
Security patterns

Case studies

Using Java security, CORBAsec

Outlook

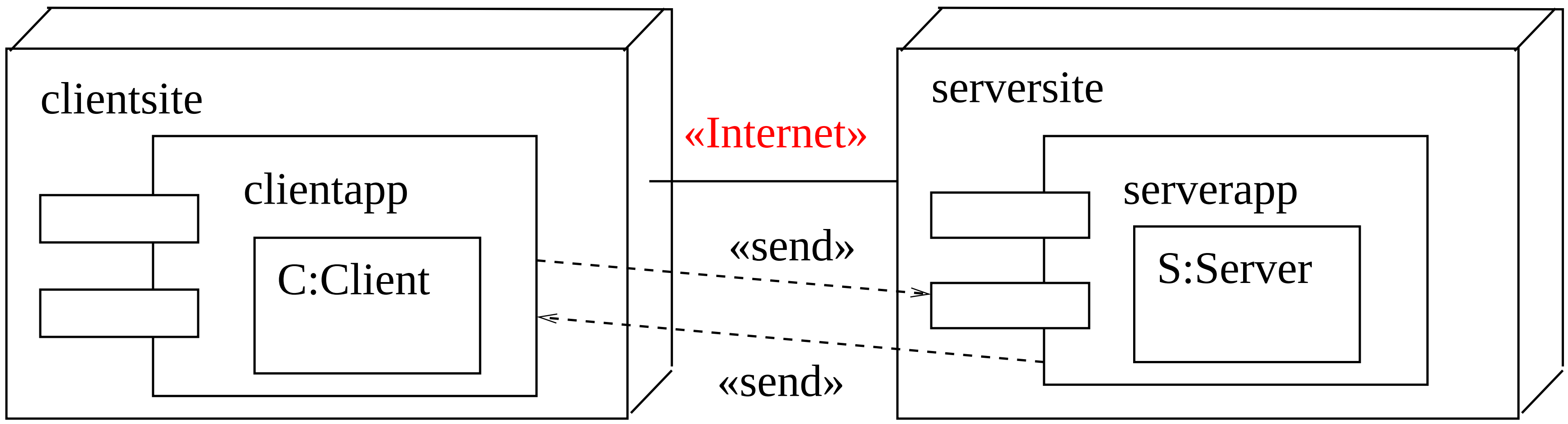
Example: Proposed Variant of TLS (SSL)



Apostolopoulos, Peris, Saha; IEEE Infocom 1999

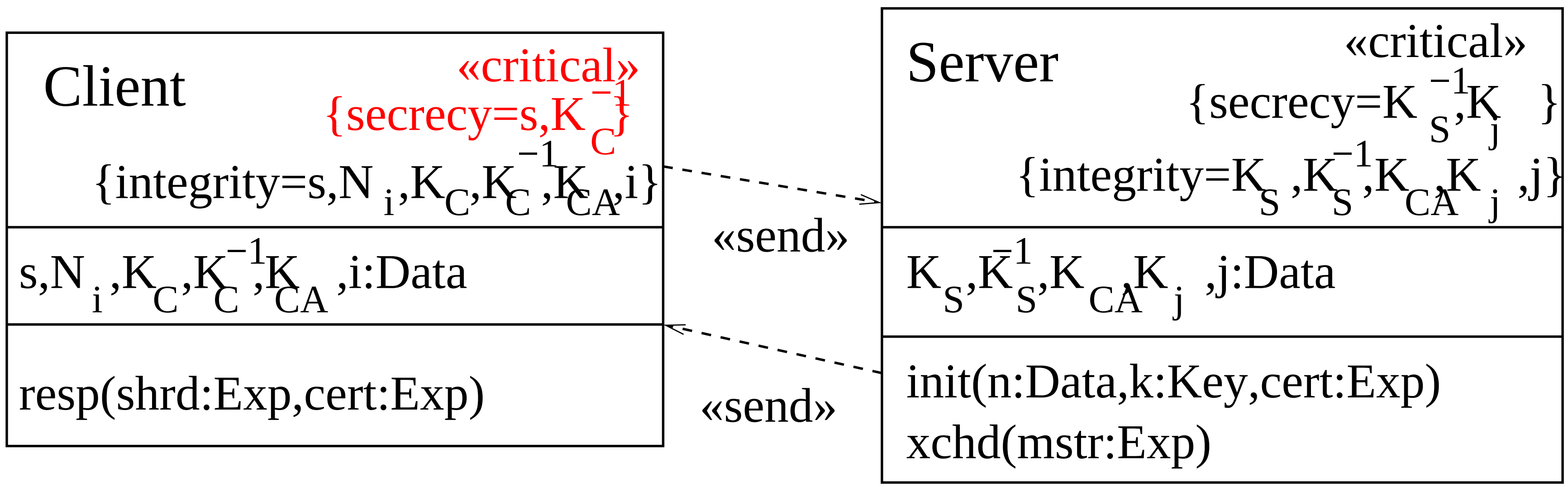
Goal: send secret s protected by session key K_j.

TLS Variant: Physical view



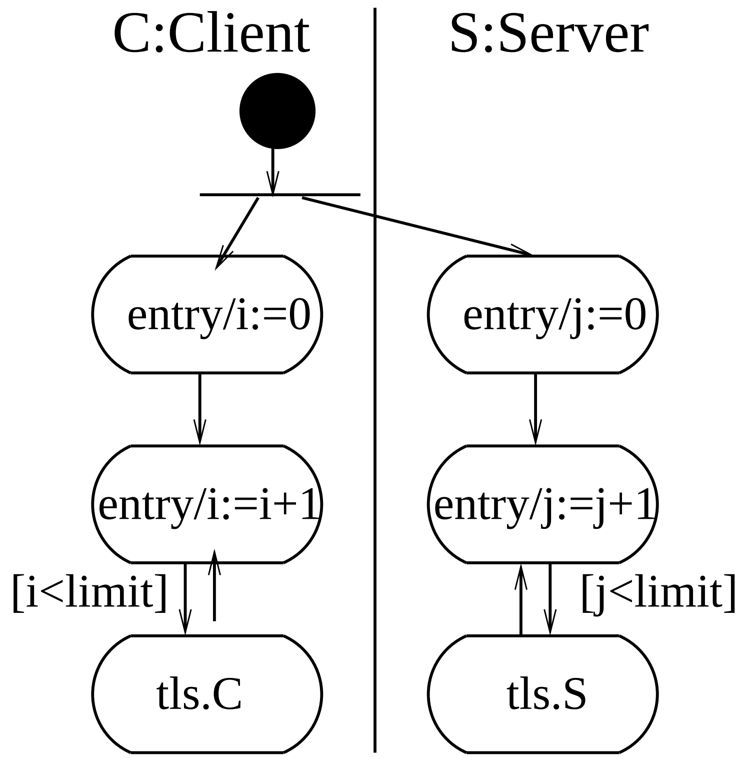
Deployment diagram.

TLS Variant: Structural view



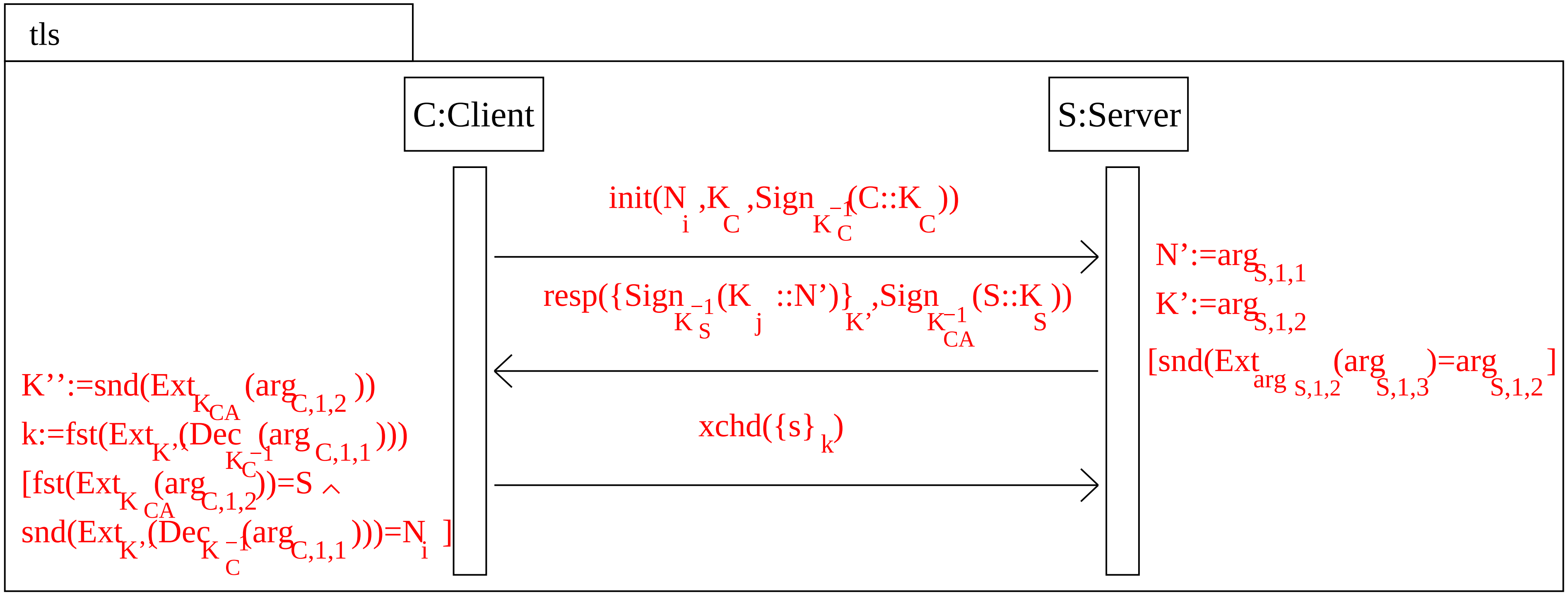
Class diagram.

TLS Variant: Coordination view



Activity diagram.

TLS Variant: Interaction view



Sequence diagram.

The flaw

Surprise: $\mathcal{S}$ does not keep secrecy of s against default adversaries with

$$\begin{aligned} \mathcal{K}_A^0 &= \{K_{CA}, K_C, K_S, C, S, \mathcal{S}ign_{K_{CA}^{-1}}(S :: K_S)\} \\ &\quad \cup \{\mathcal{S}ign_{K_{CA}^{-1}}(Z :: K_Z) : Z \in \mathbf{Data} \setminus \{S\}\}. \end{aligned}$$

Man-in-the-middle attack.

The attack

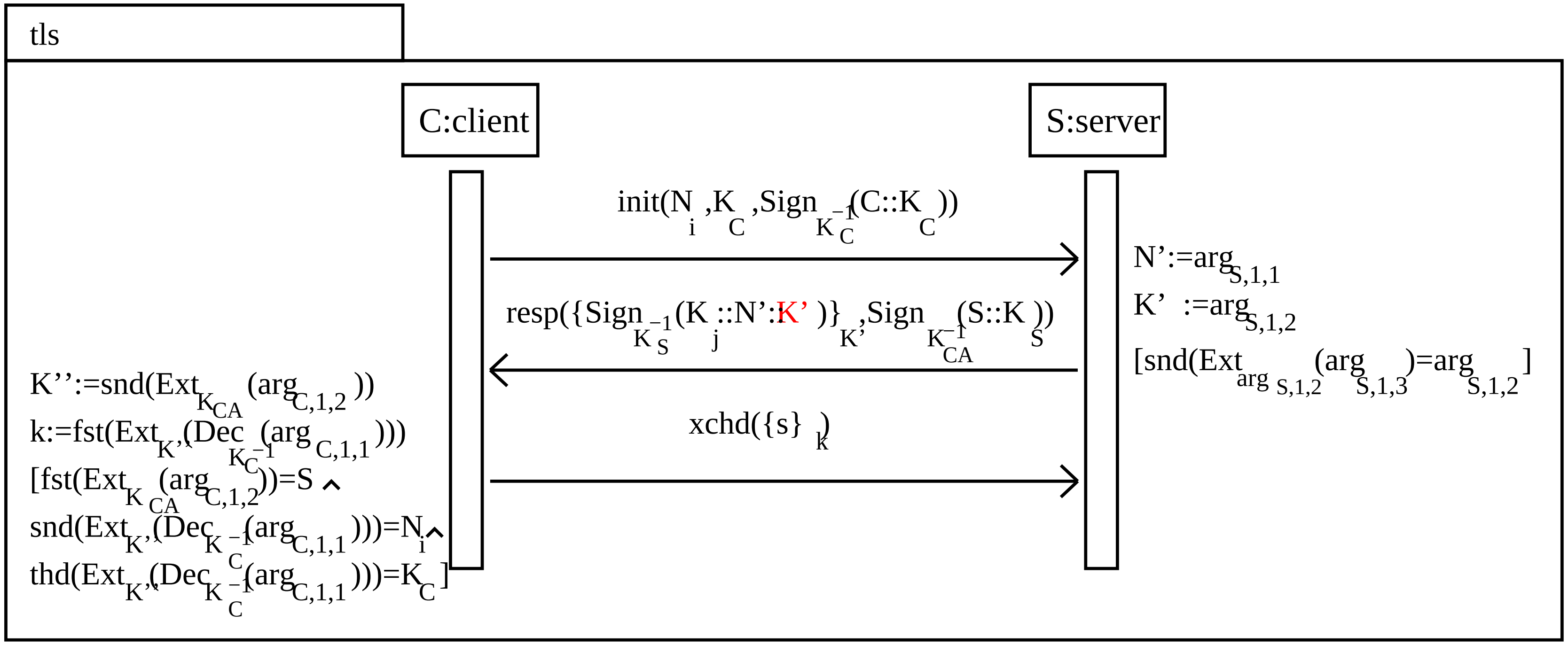
$$C \xrightarrow{N_i::K_C::\mathcal{S}ign_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::\mathcal{S}ign_{K_A^{-1}}(C::K_A)} S$$

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

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

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

The fix



Thm $\mathcal{S}'$ keeps secrecy of s against default adversaries with

$$\begin{aligned}\mathcal{K}_A^0 &= \{K_{CA}, K_C, K_S, C, S, \mathcal{S}ign_{K_{CA}^{-1}}(S :: K_S)\}\\ &\quad \cup\{\mathcal{S}ign_{K_{CA}^{-1}}(Z :: K_Z) : Z \in \mathbf{Data} \setminus \{S\}\}.\end{aligned}$$

Common Electronic Purse Specifications (CEPS)

Candidate for globally interoperable electronic purse standard.
Supported by 90 percent of electronic purse market.

Smart card contains account balance. Built-in chip performs cryptographic operations securing the transactions.

Goal: more fraud protection than credit cards (transaction-bound authentication).

Here: consider load protocol.

Load protocol

Consider unlinked, cash-based load transaction (on-line).

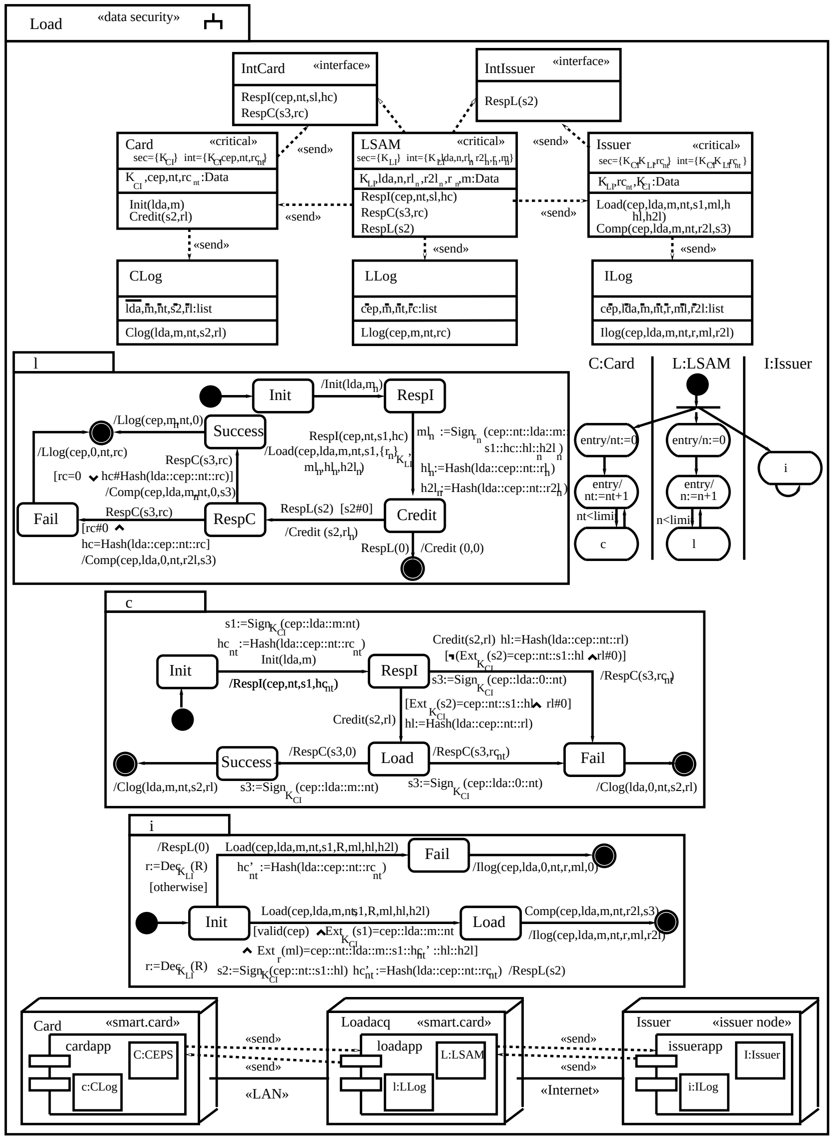
Load value onto card using cash at load device.

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

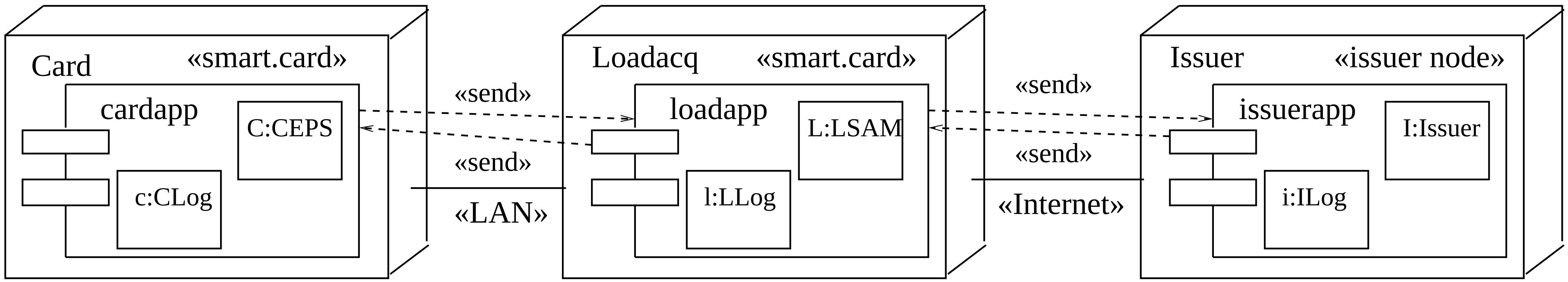
Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.

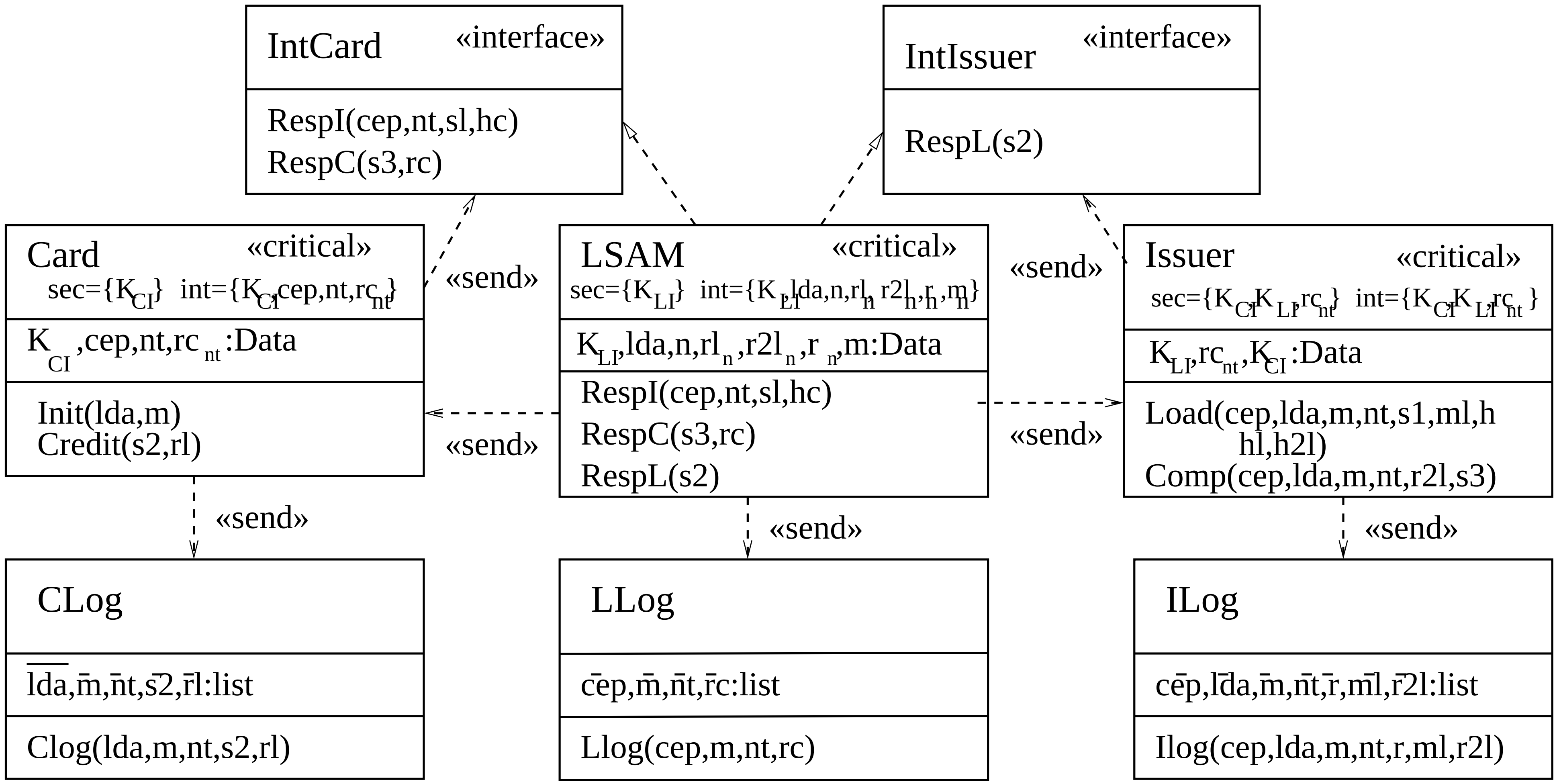
Load protocol



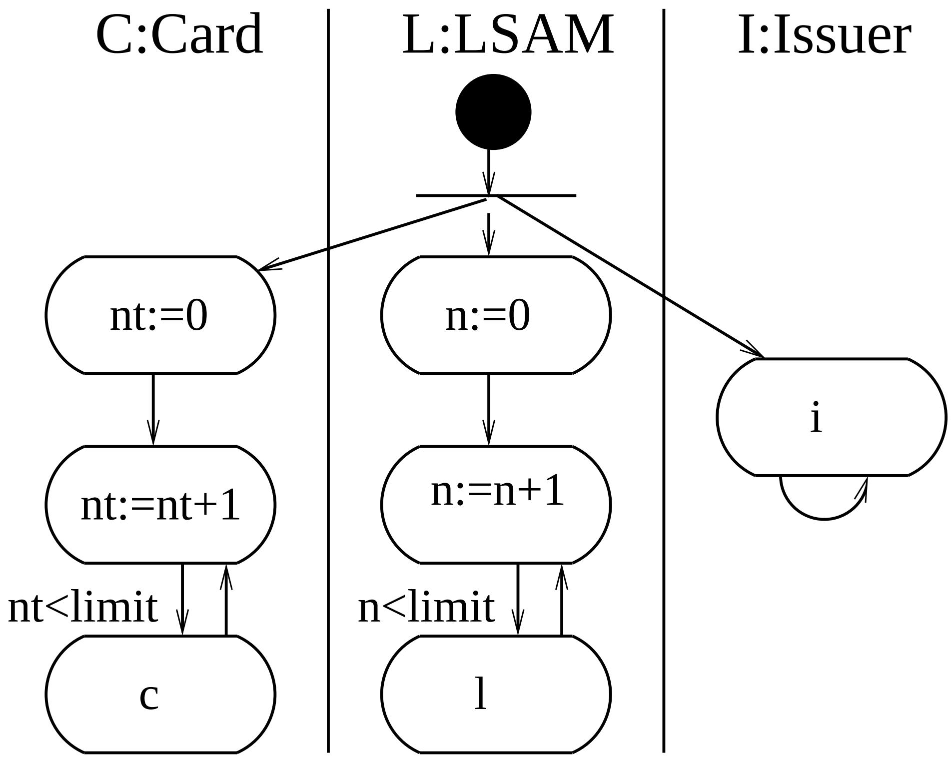
Load protocol: Physical view



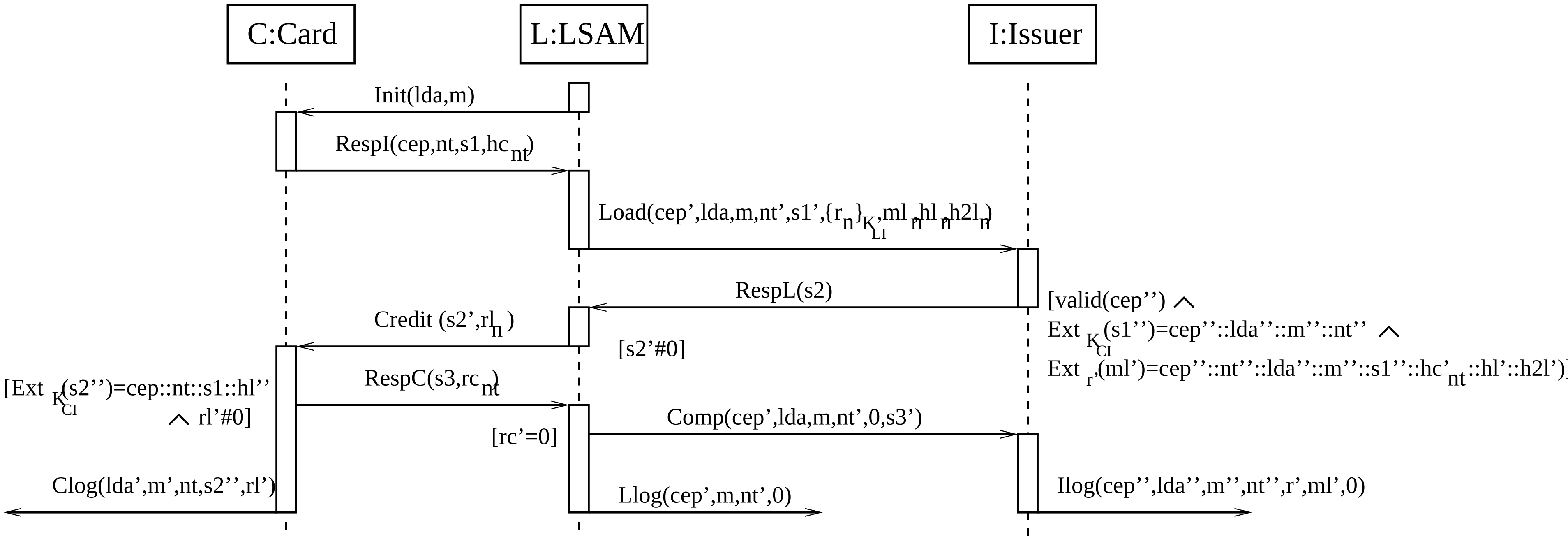
Load protocol: Structural view



Load protocol: Coordination view



Load protocol: Interaction view



$(lda',m'):=args_{C,1}$
$s1:=Sign_{K_{CI}}(cep::lda'::m'::nt)$
$hc_{nt}:=Hash(lda'::cep::nt::rc_{nt})$
$s3:=Sign_{K_{CI}}(cep::lda::m'::nt)$
$(s2'',rl'):=args_{C,2}$
$hl'':=Hash(lda'::cep::nt::rl')$

$s2':=arg_{L,2,1}$
$(s3',rc'):=args_{L,3}$

$(cep',nt',s1',hc'):=args_{L,1}$
$hl_n:=Hash(lda::cep'::nt'::rl_n)$
$h2l_n:=Hash(lda::cep'::nt'::r2l_n)$
$ml_n:=Sign_{r_n}(cep'::nt'::lda::m::s1'::hc'::hl_n::h2l_n)$

$(cep'',lda'',m'',nt'',s1'',R,ml',hl',h2l'):=args_{I,1}$
$r':=Dec_{K_{LI}}(R)$
$s2:=Sign_{K_{CI}}(cep''::nt''::s1''::hl')$
$hc'_{nt}:=Hash(lda''::cep''::nt''::rc_{nt})$

Security Threat Model

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

Could 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.
May 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
$ml_n = \mathcal{S}ign_{r_n}(cep :: nt :: lda :: m_n :: s1 :: hc_{nt} :: hl_n :: h2l_n)$
and card C possesses rl_n, where
$h_n = \mathcal{H}ash(lda :: cep :: nt :: rl_n)$. Then after protocol either of following hold:

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

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

Flaw

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

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

Verify this version wrt. above conditions.

Roadmap

Prologue

The profile

Security analysis

Security patterns

Case studies

Using Java security, CORBAsec

Outlook

Java Security

Originally (JDK 1.0): sandbox.

Too simplistic and restrictive.

JDK 1.2/1.3: more fine-grained security architecture
(access 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.

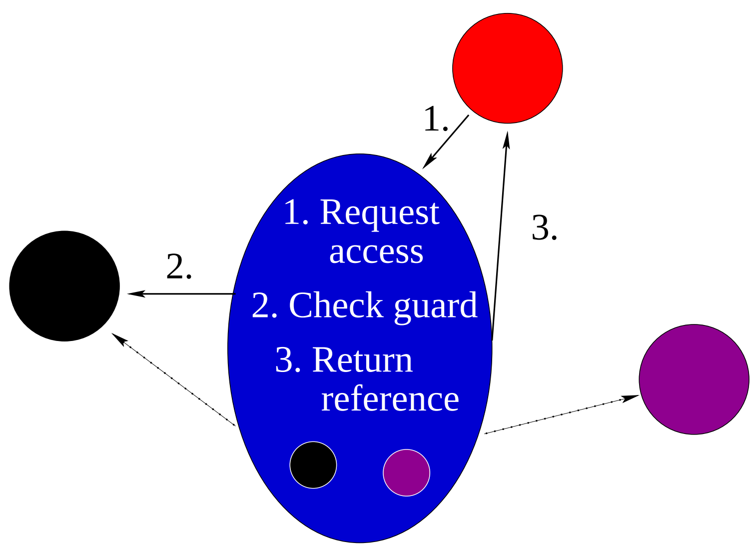
Similarly, need confidentiality.

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).

$\leadsto$ 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

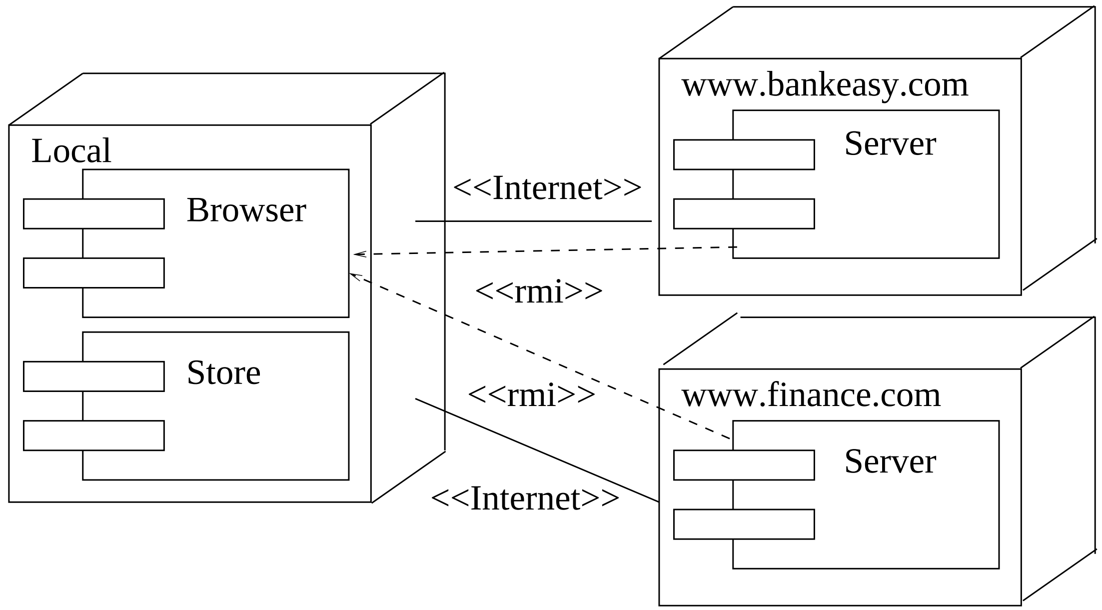
Theorem.

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

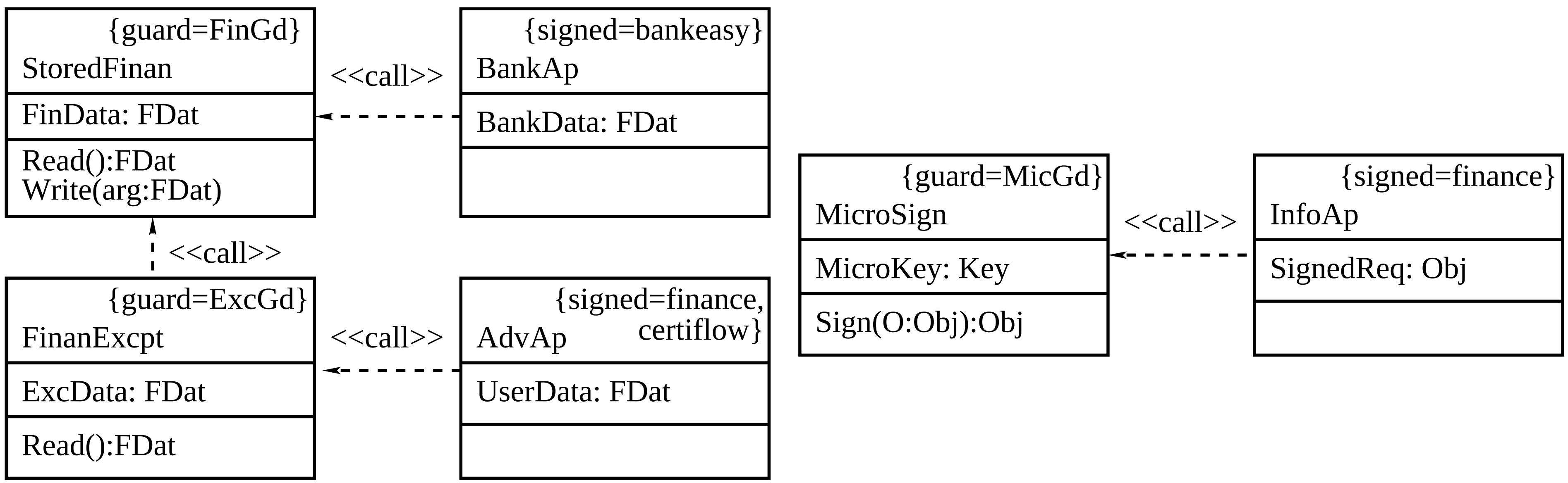


Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain privileges (step 1).

- 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.

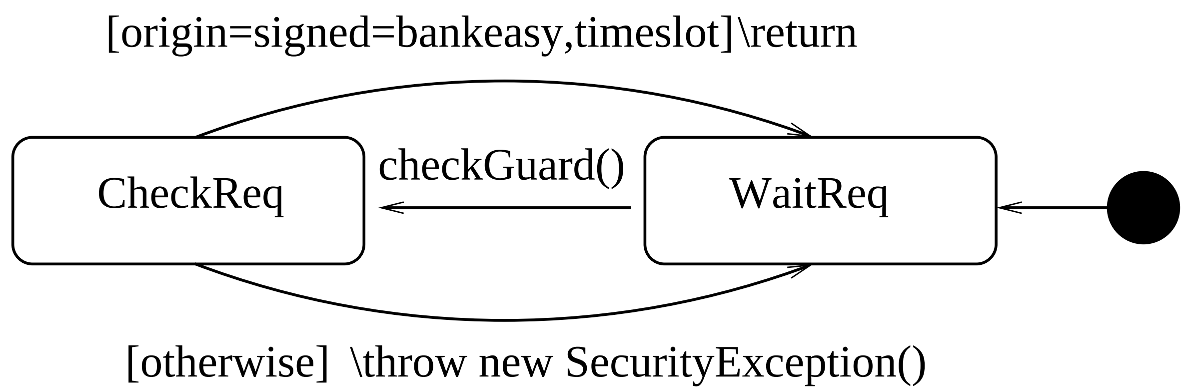
Financial Application: Class diagram

Sign and seal objects sent over Internet for integrity and confidentiality.
GuardedObjects control access.

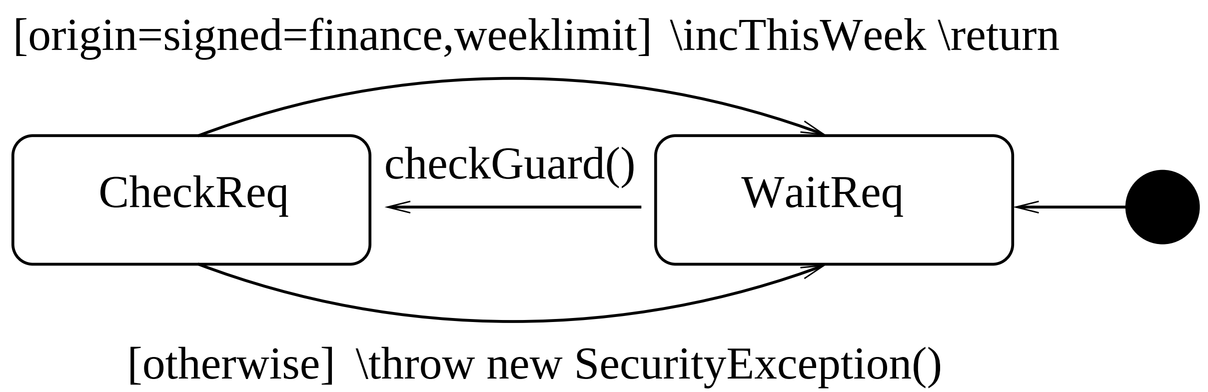


Financial Application: Guard objects (step 2)

timeslot true between 1pm and 2pm.



weeklimit true until access granted five times; incThisWeek 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.

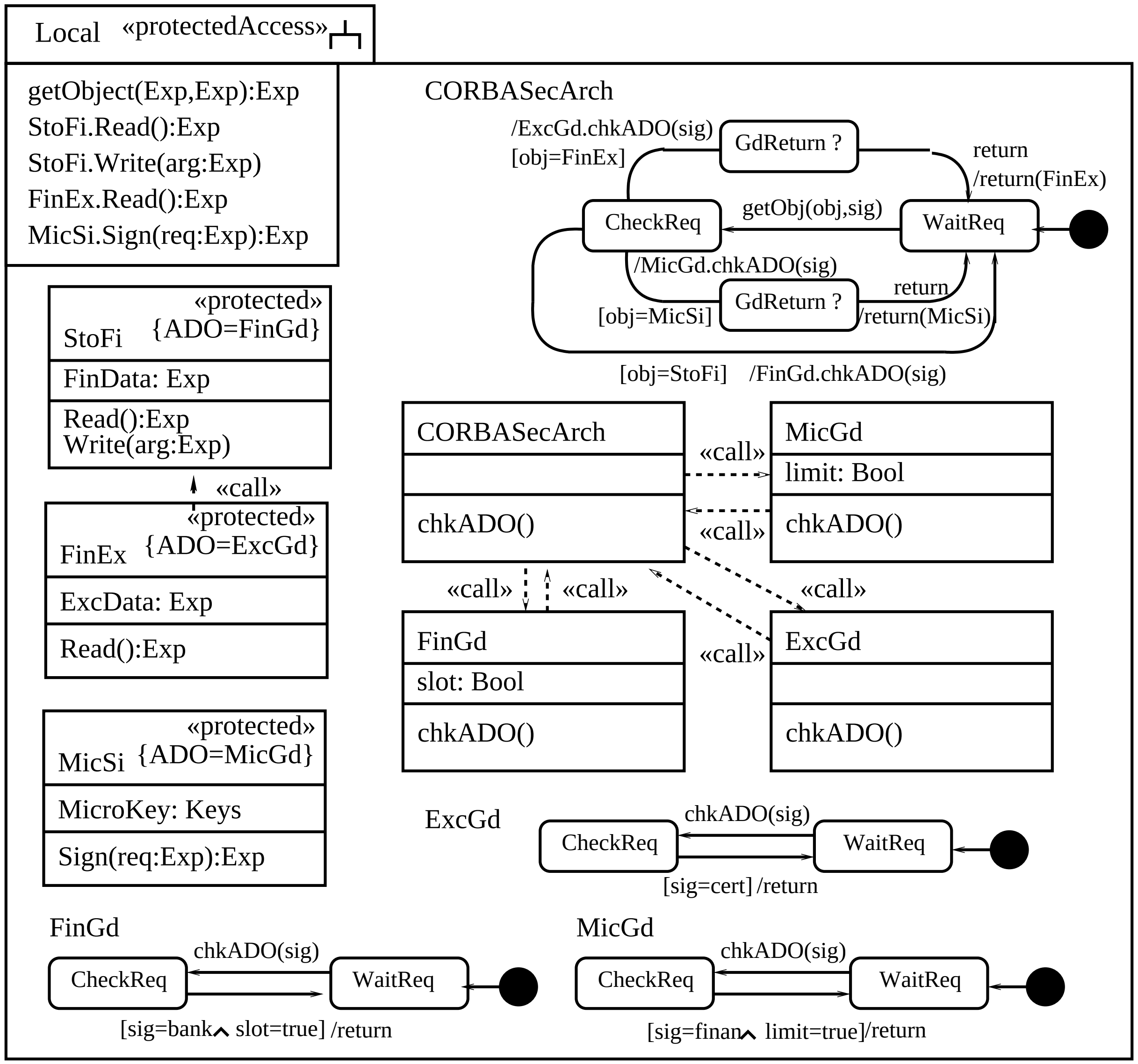
CORBA access control

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



Roadmap

Prologue

The profile

Security analysis

Security patterns

Case studies

Using Java security, CORBAsec

Outlook

Tool support

Drawing tool (Rational Rose, . . .)

Link via XMI (XML Metadata Interchange) to:

Analysis tool (AutoFocus)

- test-sequence generation
- verification
- code generation

Resources

Slides, papers etc.:

http://www4.in.tum.de/~umlsec

My homepage:

http://www.jurjens.de/jan

Stop.

Thanks for your attention !