Applications:
Pattern- and Component-based Development For Security-Critical Systems

Pattern- and Component-based Development For Security-Critical Systems

Security-critical systems are a particularly relevant application domain for modern software engineering approaches (such as pattern- and component-based systems):

- Software systems become increasingly security-critical, e.g. because of the increasing use of the Internet for security-critical (such as commercial) purposes
- Developers are still not suffiently educated in security and therefore many systems in practice are insecure (the situation is pretty much summarized by the picture...).

IT Security

[Weck 2003]

Aspects

Protection of the system, against attacks **Security**	Protection of environment, against faults **Safety**

Basic Goals

Integrity	Confiden-tiality	Availability	Account-ability	Nonrepu-diability

Stability	**Reability**
Robustness	**Maintainability**
Plausibility	**Correctness**
Trustability	**...**

Basic Functions

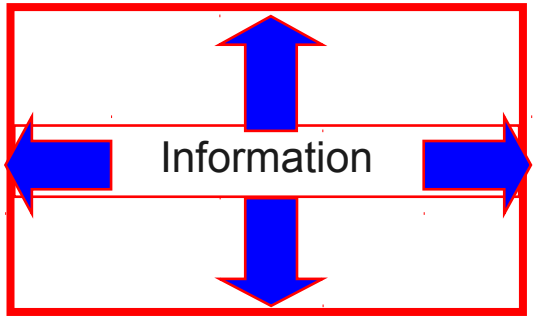
Identification Authentication	Authorization Rights managem.	Rights control	Logging	Fault tolerance	Control	...

Mechanisms

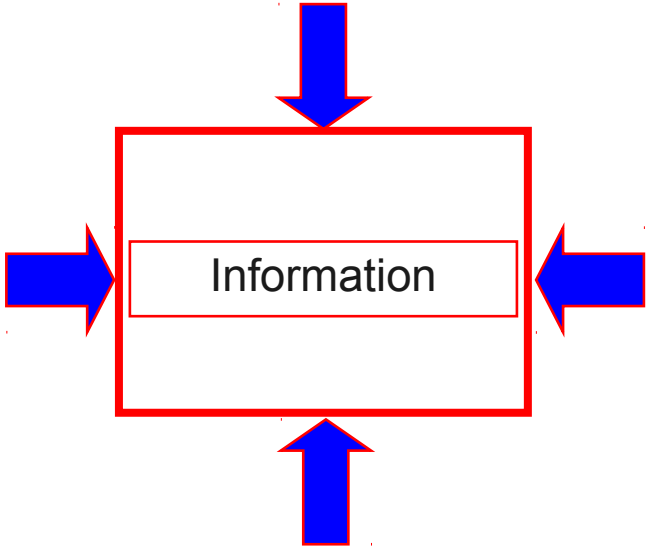
Chip cards Passwords	Separation of duty	Access Control discrete	 global	Crypto-graphy	Communic. protocols	Audit Logs	Redundancy	...

Some Important Security Requirements I

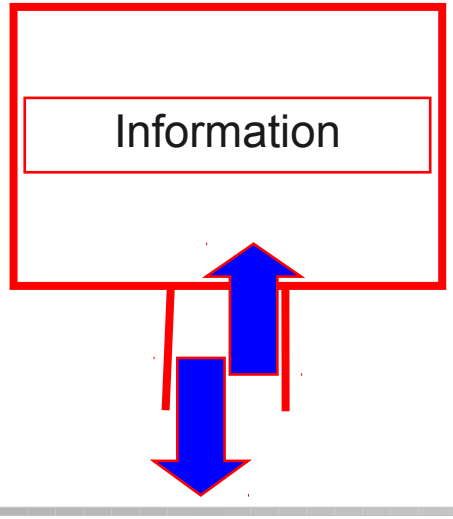
Secrecy



Integrity

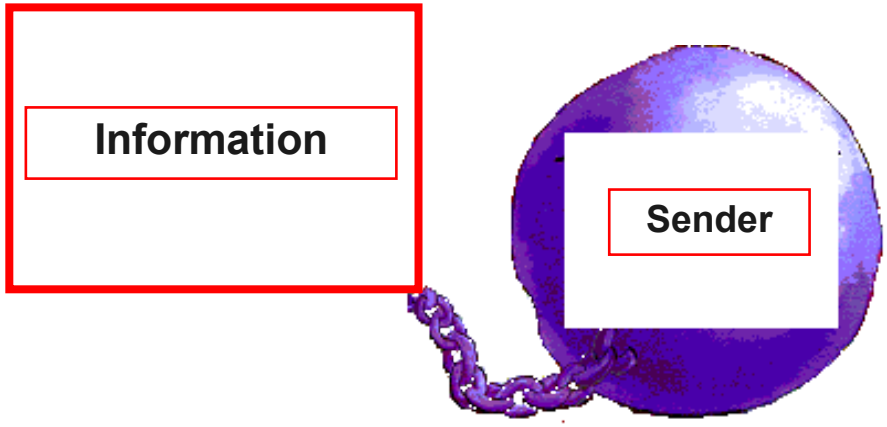


Availability

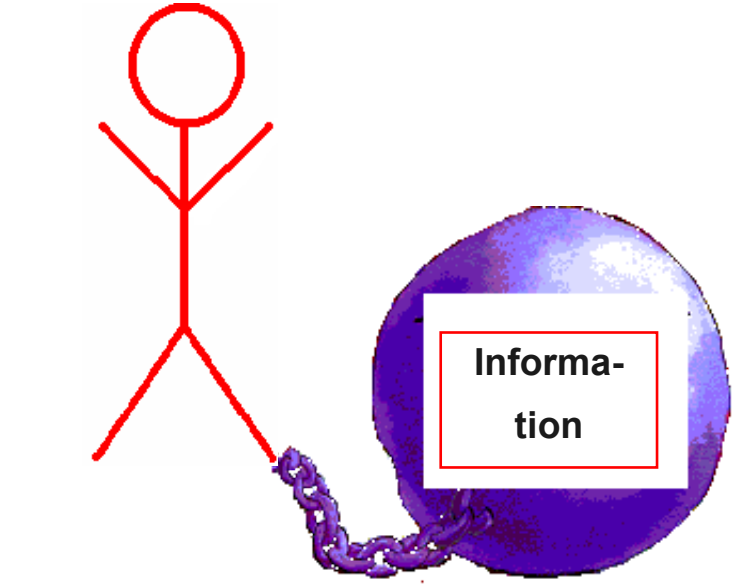


Some Important Security Requirements II

Authenticity



Non-repudiability



Sender

Modeling Security Patterns with UML

Requirements on UML Extension for Security I

Provide basic security requirements such as secrecy, integrity, authenticity.

Allow considering different threat scenarios depending on adversary strengths.

Allow including important security concepts (e.g. *tamper-resistant hardware*).

Allow incorporating security mechanisms (e.g. access control).

Requirements on UML Extension for Security II

Provide security primitives (e.g. (a)symmetric encryption).

Allow considering underlying physical security.

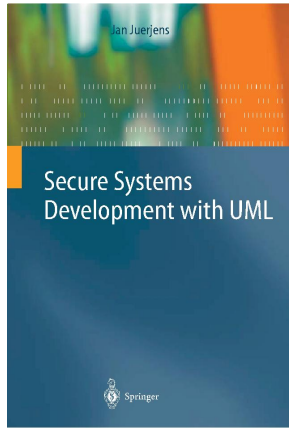
Allow addressing security management (e.g. secure workflow).

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

Model-based Security with UMLsec

Extension of the Unified Modeling Language (UML) for secure systems development.

- evaluate UML models for security
- encapsulate established rules of prudent secure engineering
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- can use in certification

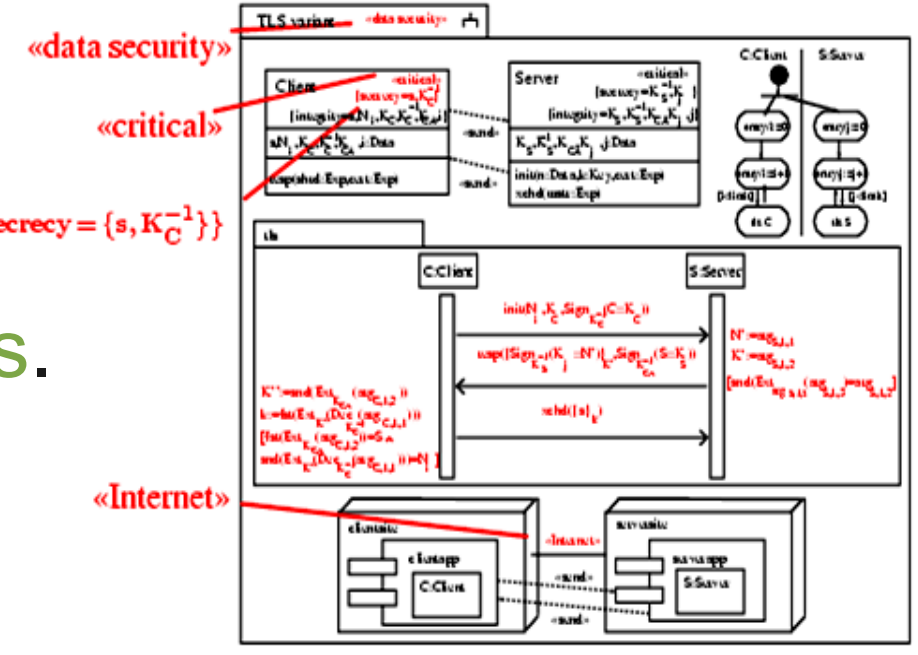


UMLsec

Insert recurring security requirements, adversary scenarios, security mechanisms as predefined markers.

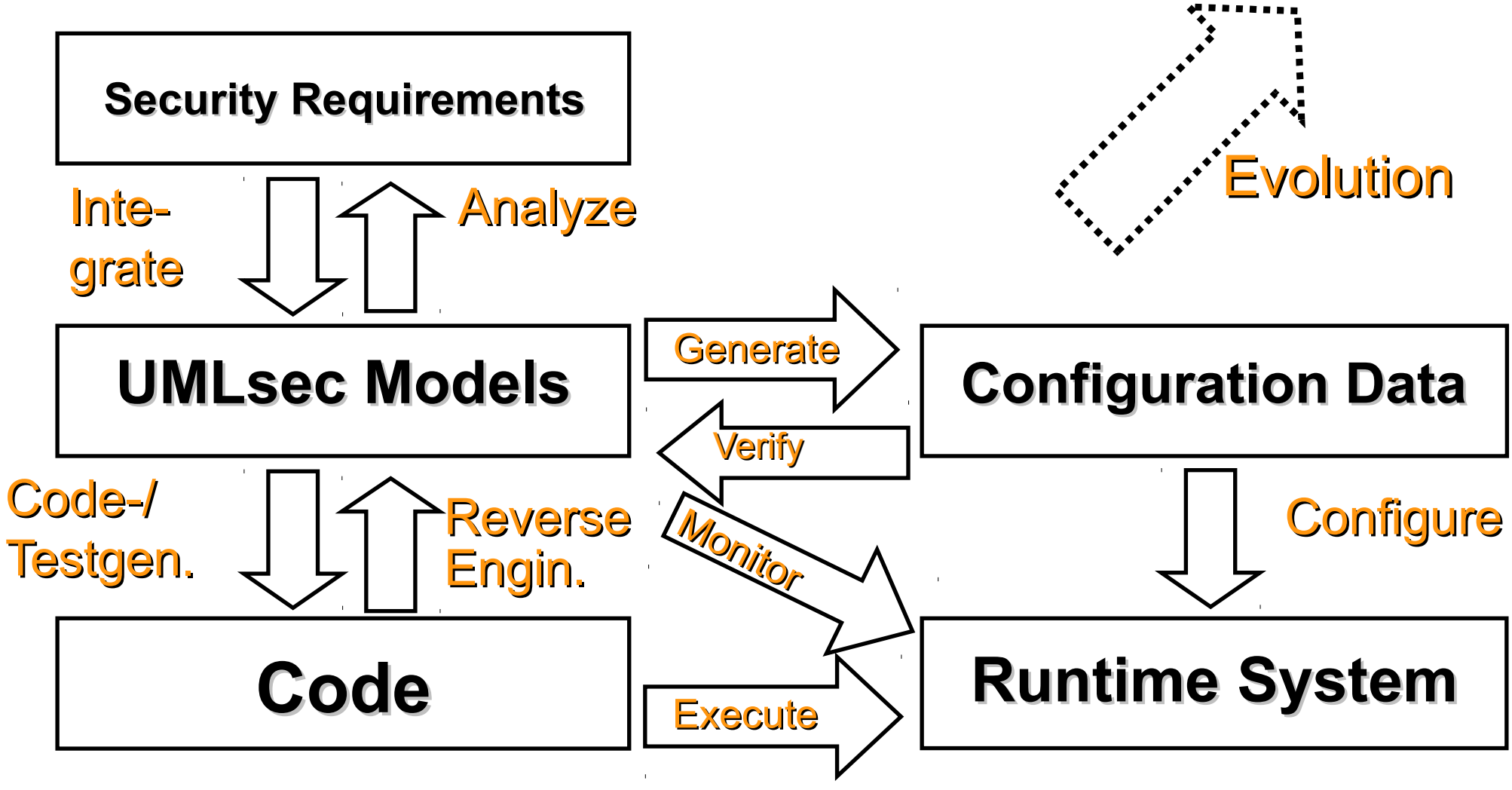
Use associated logical constraints to verify specifications using model checkers and ATPs based on formal semantics.

Ensures that UML specification enforces the relevant security requirements wrt Dolev-Yao type adversaries.



[FASE01,UML02,FOSAD05,ICSE05]

Model-based Security Engineering with UMLsec



What Does UMLsec Cover ?

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.

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

UMLsec as Integrating Formal Framework

Have formalizations of major security requirements in one integrated notation.

Want to relate / combine requirements; get modularity / composability, hierarchical decomposition, refinement, … :

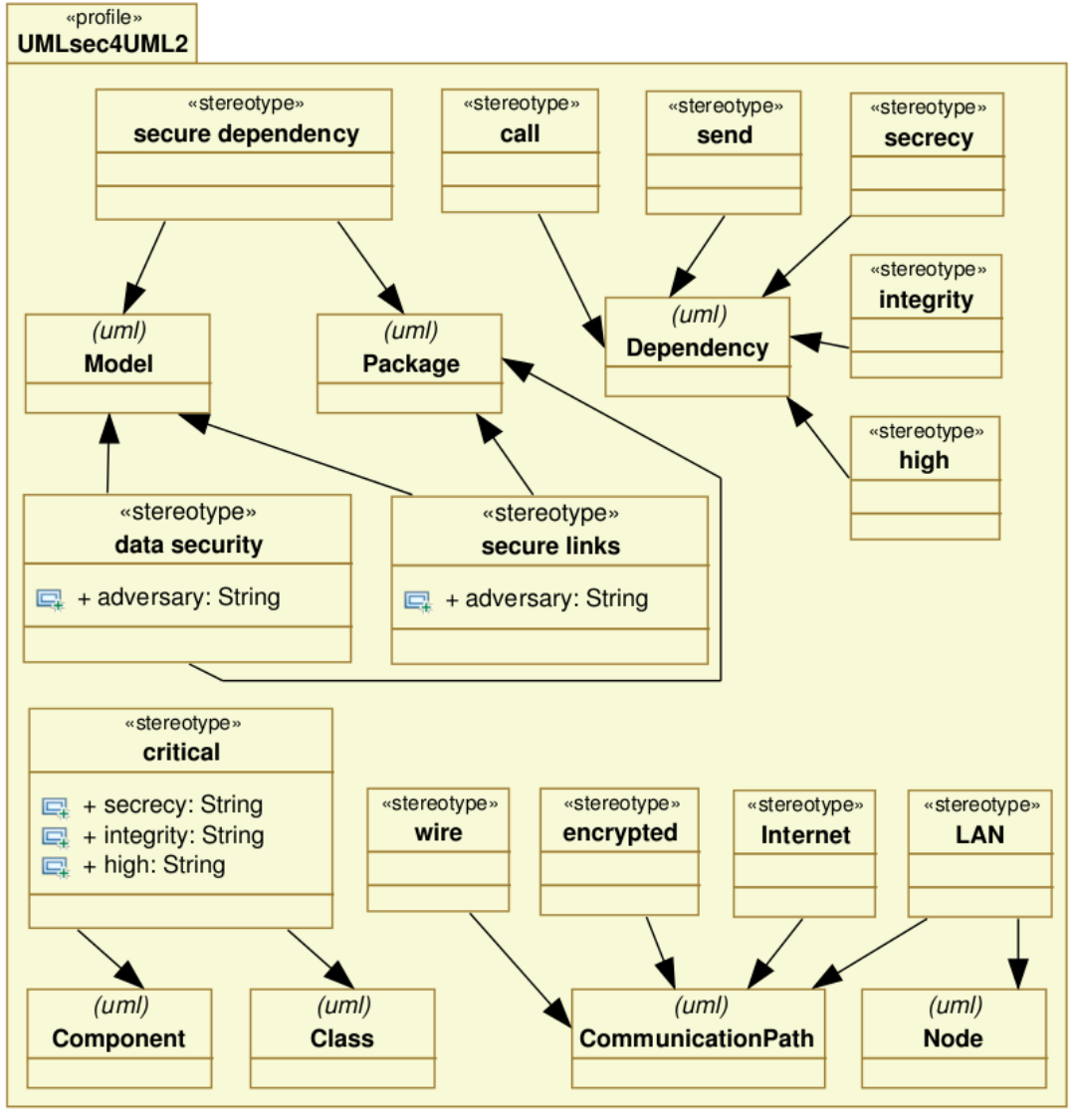
For example:

If system satisfies <<secure links>> and

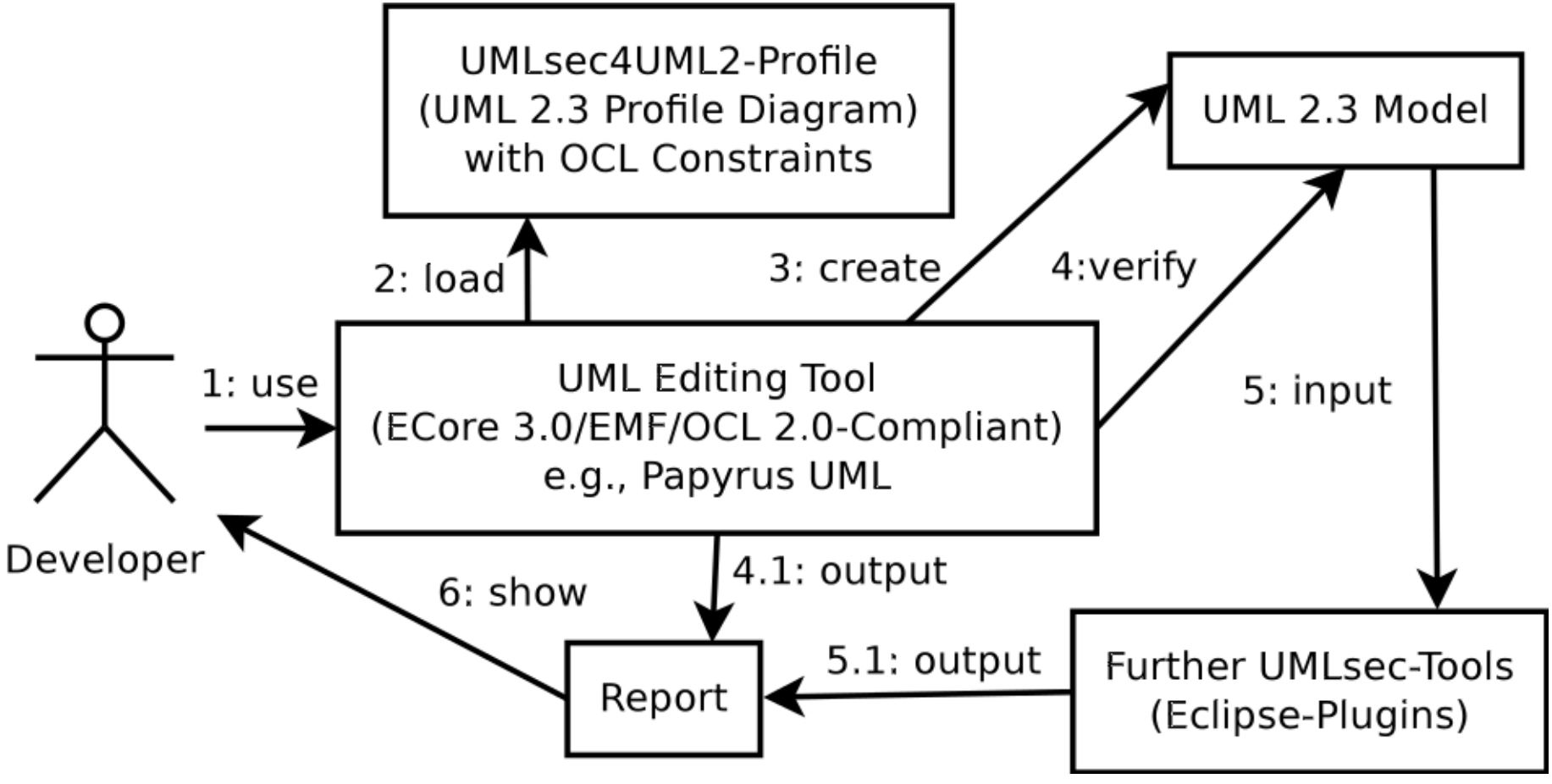
subsystems satisfy <<data security>> then

system satisfies <<data security>>.

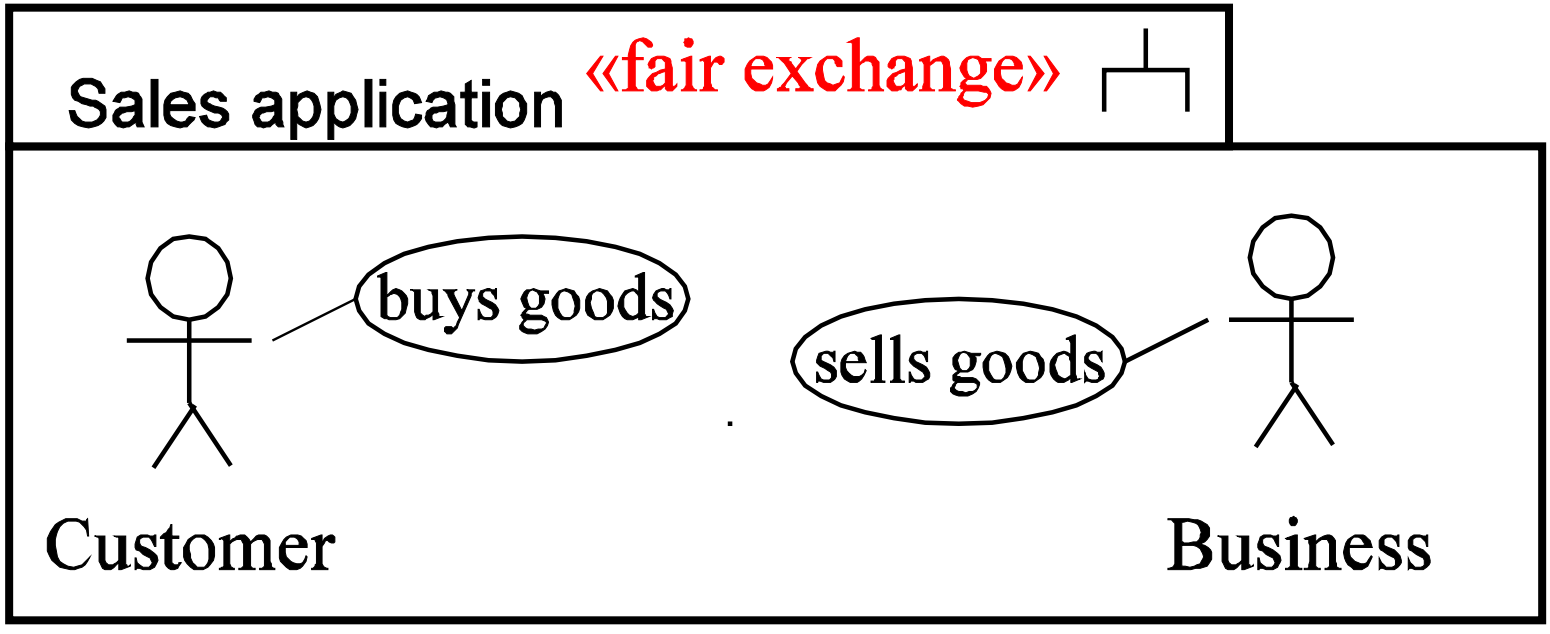
UMLsec Profile based on OCL



UMLsec4UML Workflow



Requirements with Use Case Diagrams



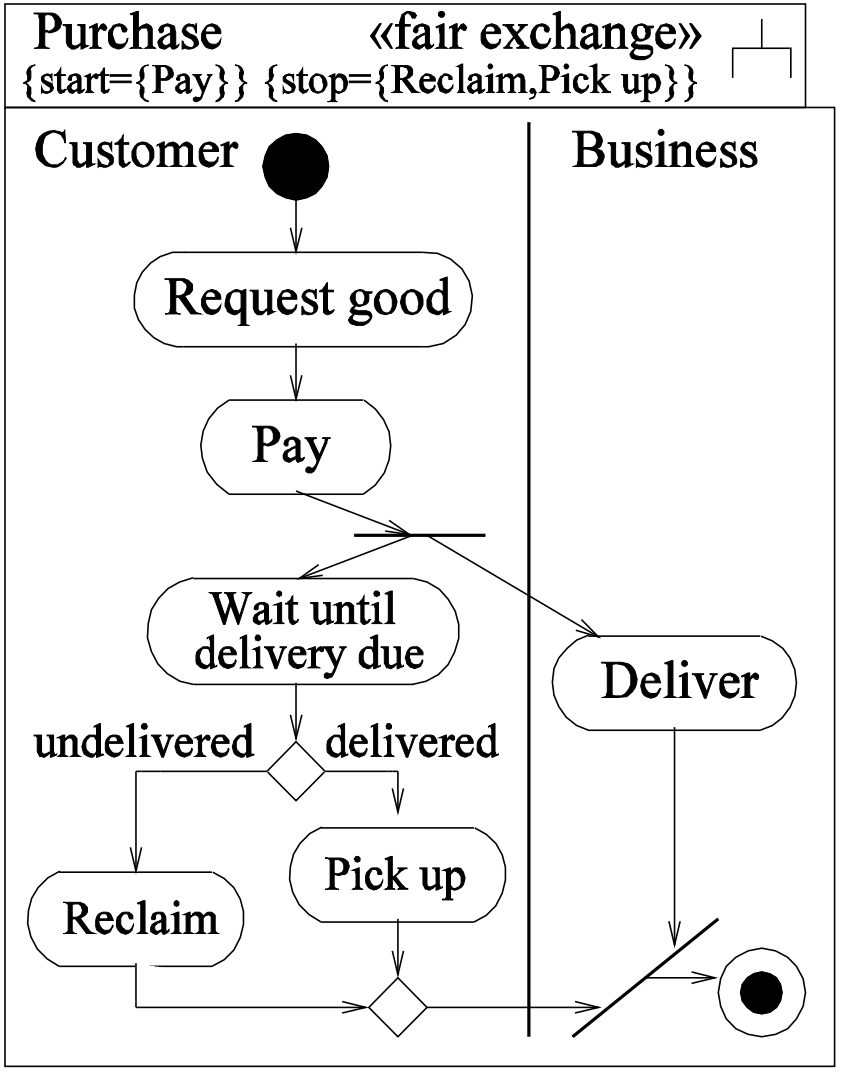
Capture security requirements in use case diagrams.

Constraint: need to appear in corresponding activity diagram.

Fair Exchange

Customer buys good from a business.

How can enforce fair exchange:
After payment, customer is eventually either delivered good or able to reclaim payment (or vc.vs.).



<<fair exchange>>

Ensures generic fair exchange condition.

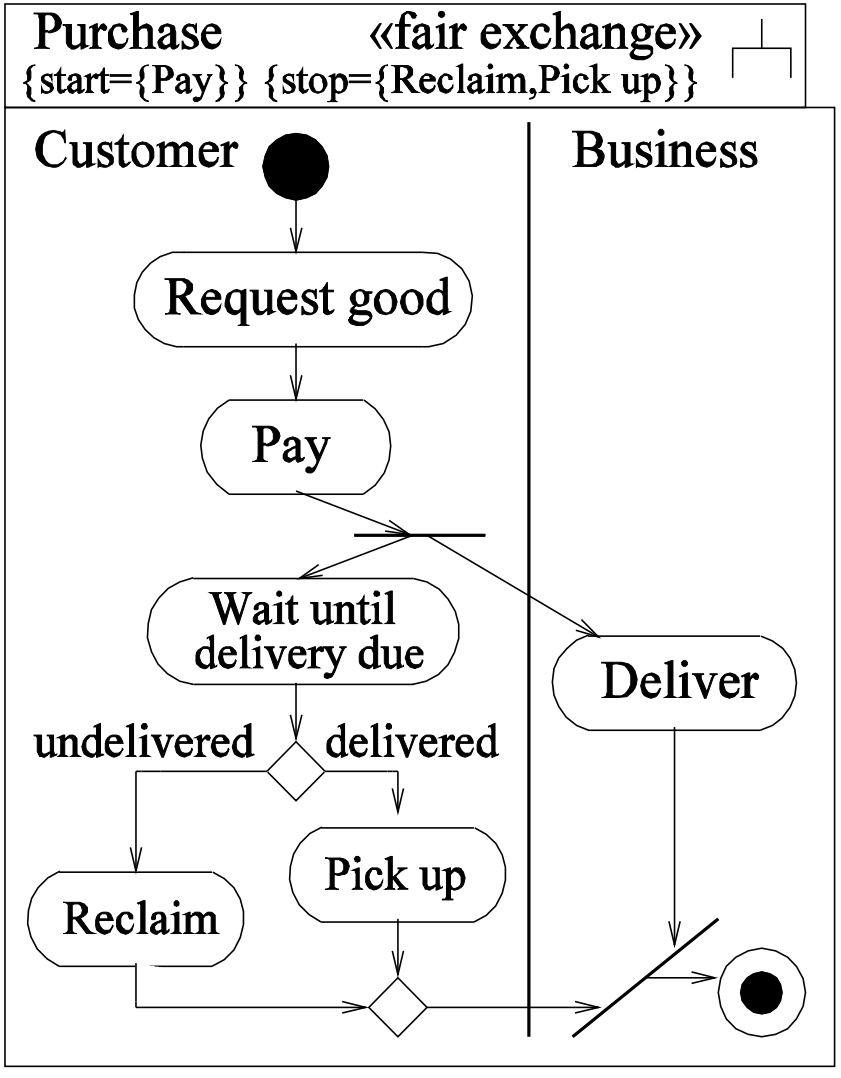
Constraint: after a {start} state in activity diagram is reached, eventually reach {stop} state.

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

Example

<<fair exchange>>

fulfilled if adversary cannot stop system: After payment, customer is eventually either delivered good or able to reclaim payment.



<<Internet>>, <<encrypted>>, …

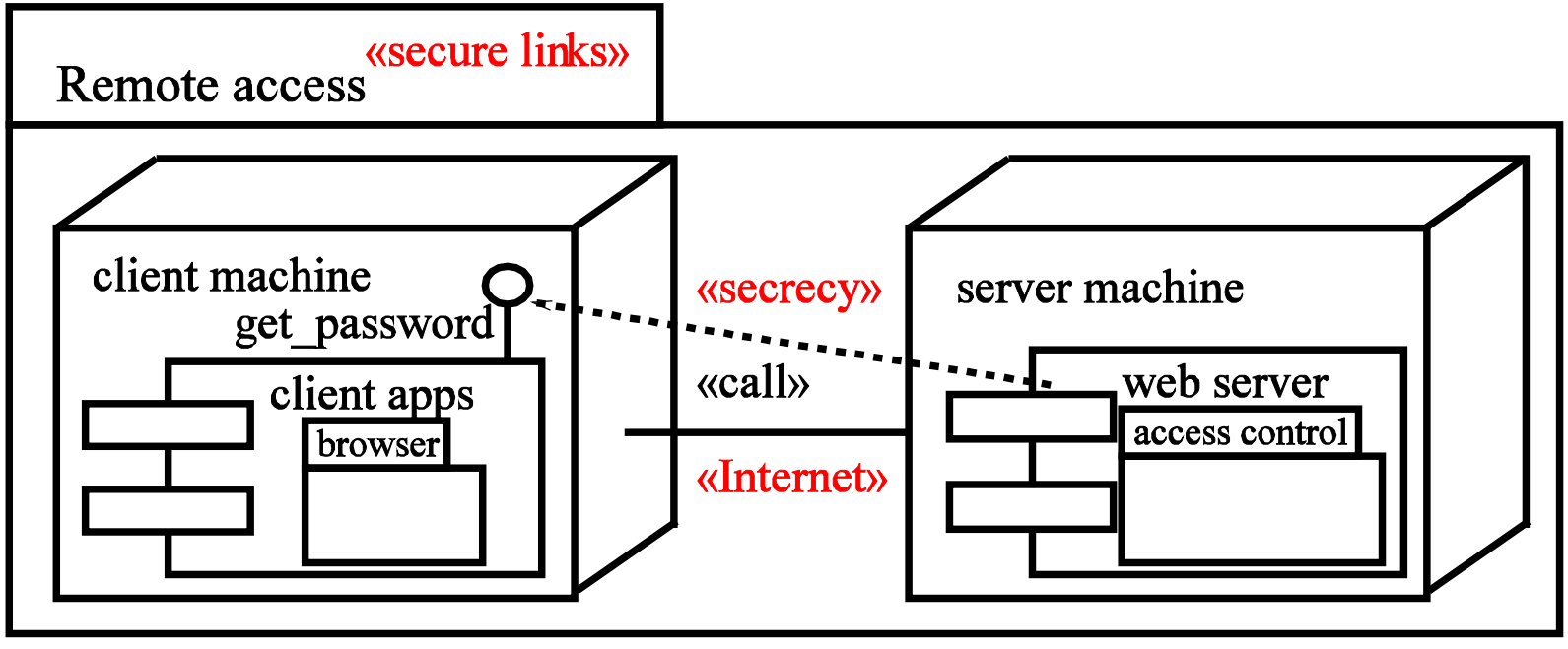
Kinds of communication links resp. system nodes.

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

Default attacker:

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

Secure Architecture



Architecture secure against default adversary ?

<<secure links>>

Ensures that physical layer meets security requirements on communication against a given attacker of type *A*.

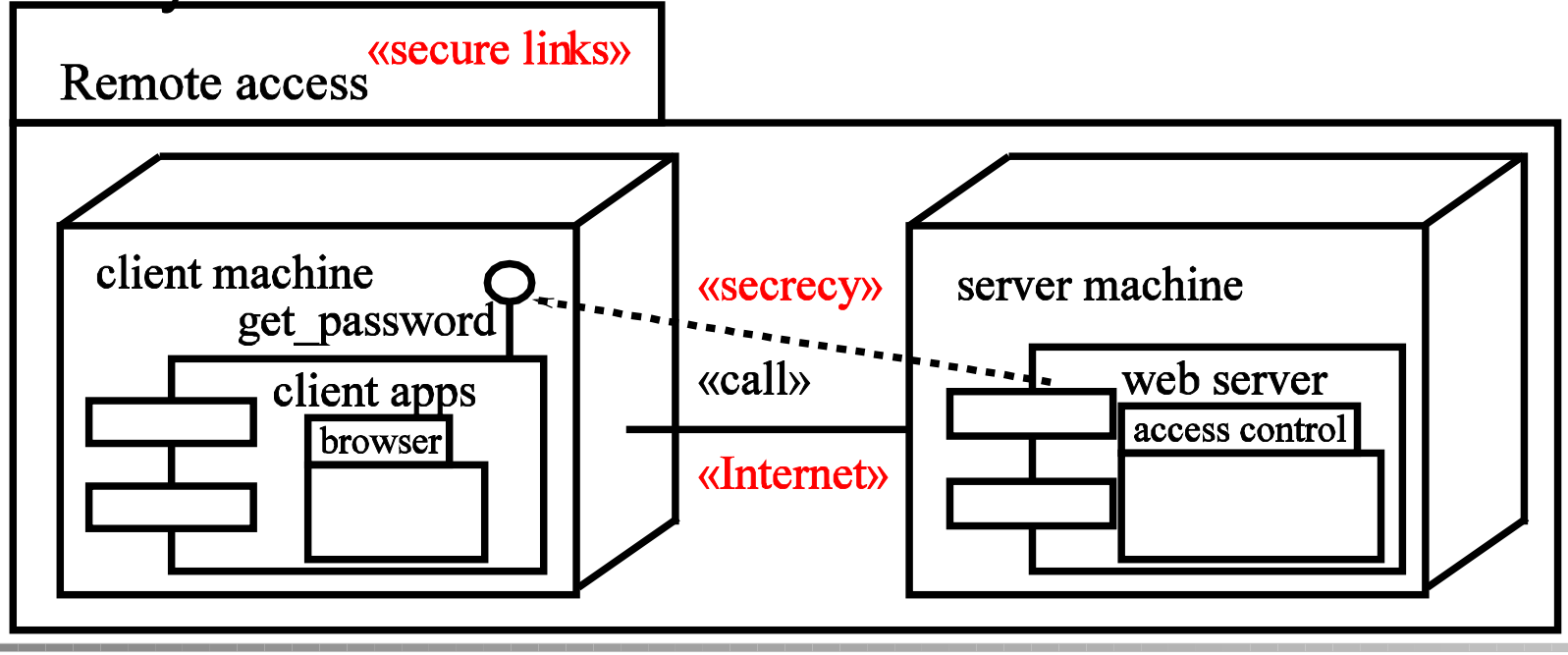
Constraint: for each dependency *d* with stereotype *s* ∈ {<<secrecy>>, <<integrity>>} between components on nodes *n*≠*m*, the communication is over a communication link *l* between *n* and *m* with stereotype *t* such that

- if *s* = <<secrecy>>: have read ∉ Threats_A *(t)*.
- if *s* = <<integrity>>: have insert ∉ Threats_A *(t)*.

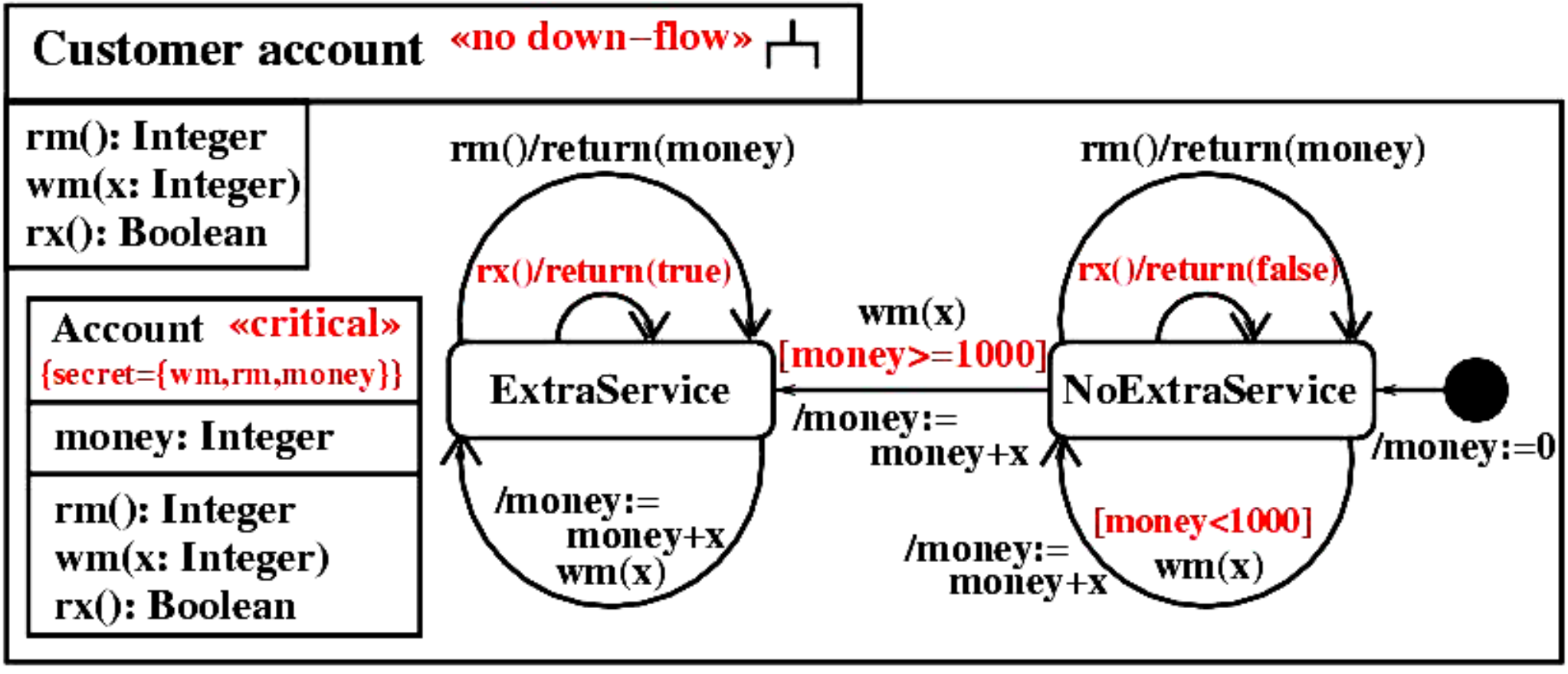
Example <<secure links>>

Given default adversary type, constraint for stereotype <<secure links>> violated:

According to the $Threats_{default}$(Internet) scenario, <<Internet>> link does not provide secrecy against default adversary.



Secure Information Flow



No partial leakage of secrets ?

<<no down–flow>>

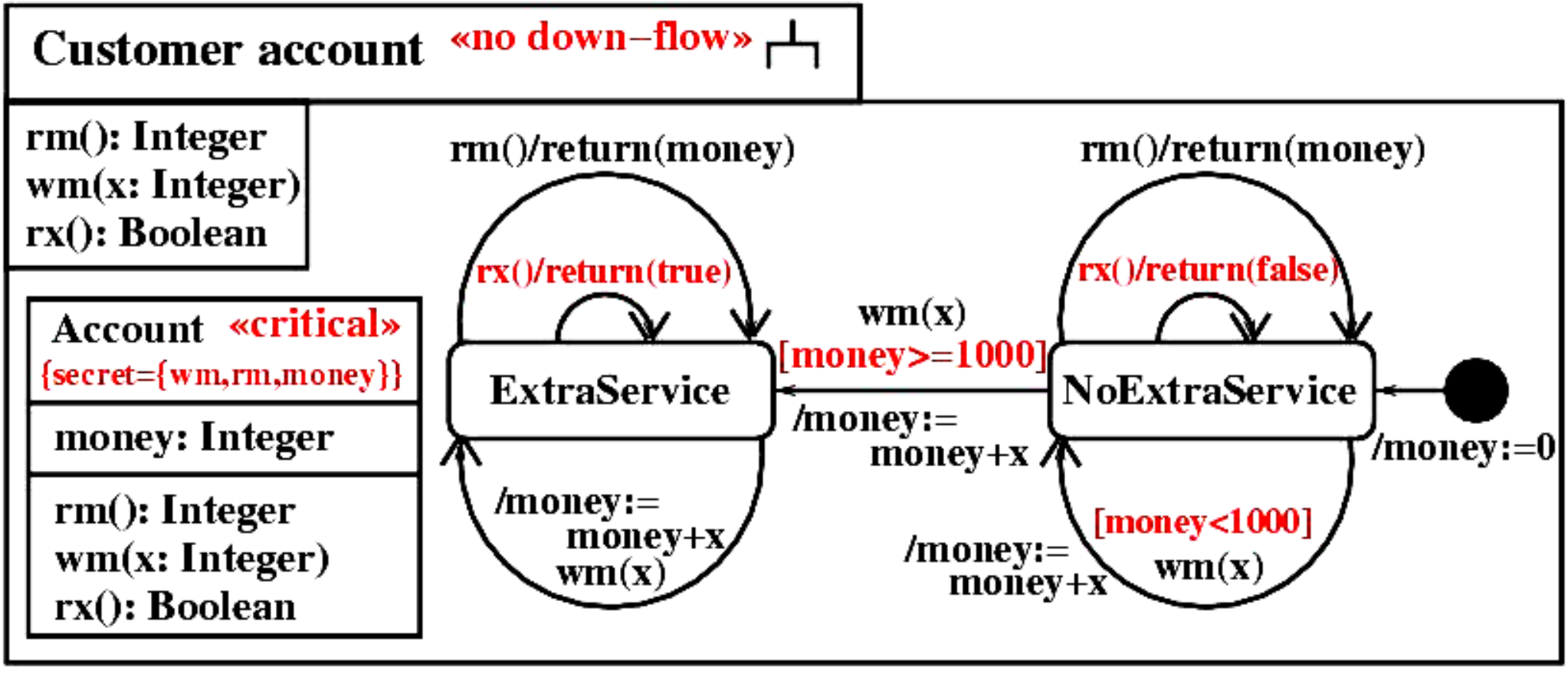
Enforce secure information flow.

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 secret wm() returned by non-secret rx().

LEHRSTUHL 14
SOFTWARE ENGINEERING

Secure Use of Cryptography

«data security»

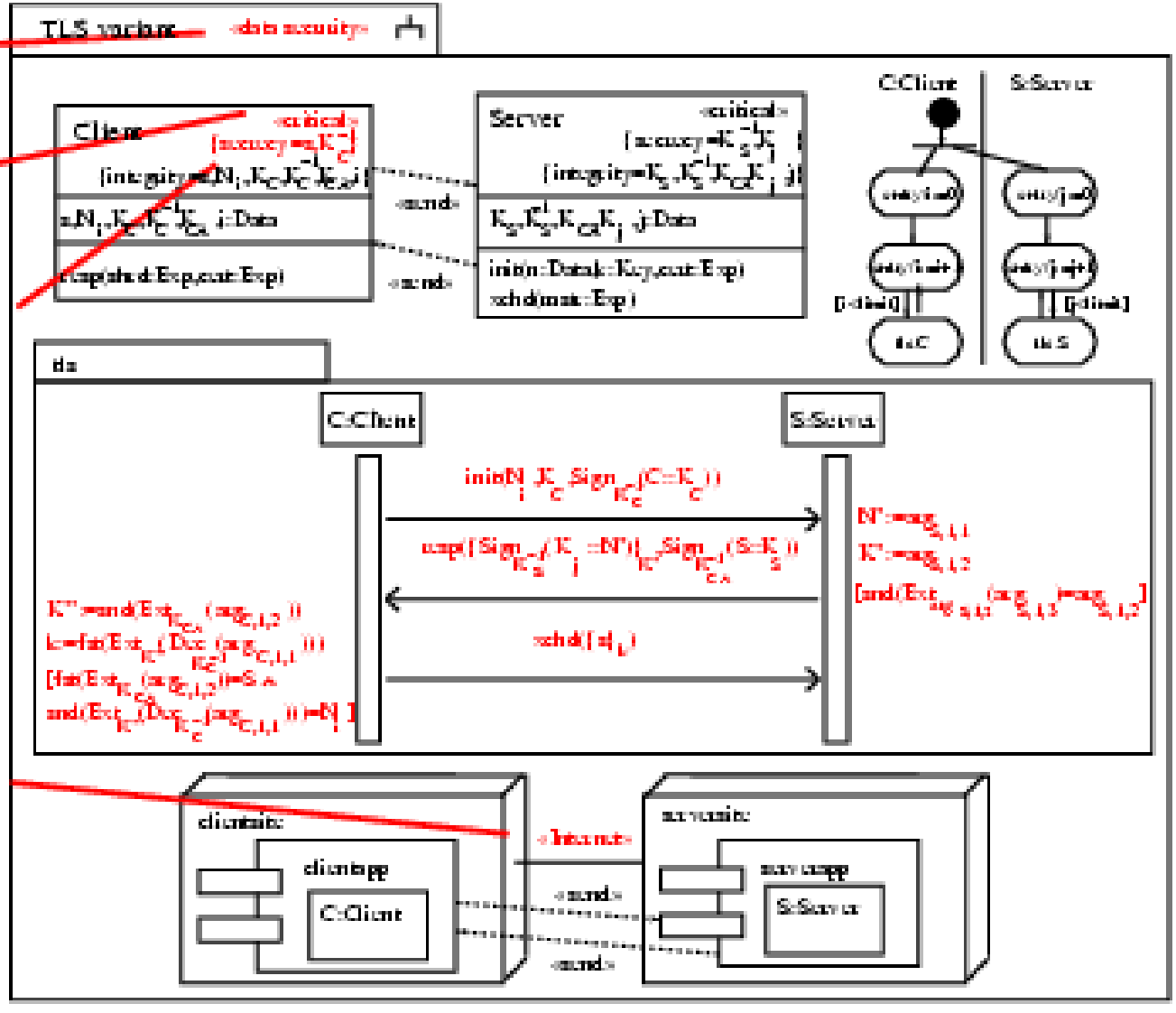
«critical»

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

«Internet»

Variant of TLS (INFOCOM`99).

Cryptoprotocol secure against default adversary ?



<<data security>>

Security requirements of data marked <<critical>> enforced against threat scenario from deployment diagram.

Constraints: Data marked {secrecy}, {integrity}, {authenticity}, {fresh} fulfills respective formalized security requirements.

LEHRSTUHL 14
SOFTWARE ENGINEERING

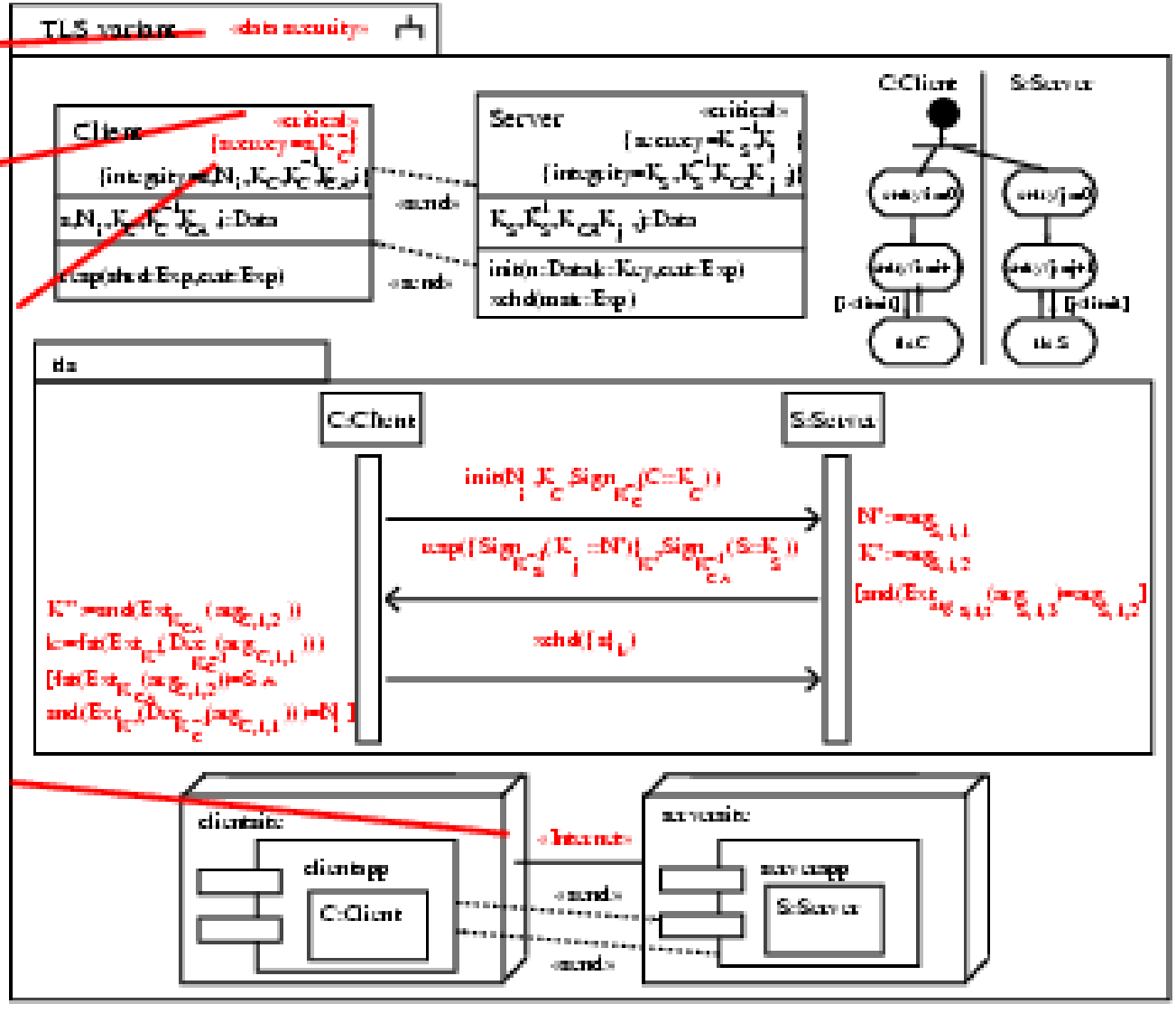
Example <<data security>>

«data security»

«critical»

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

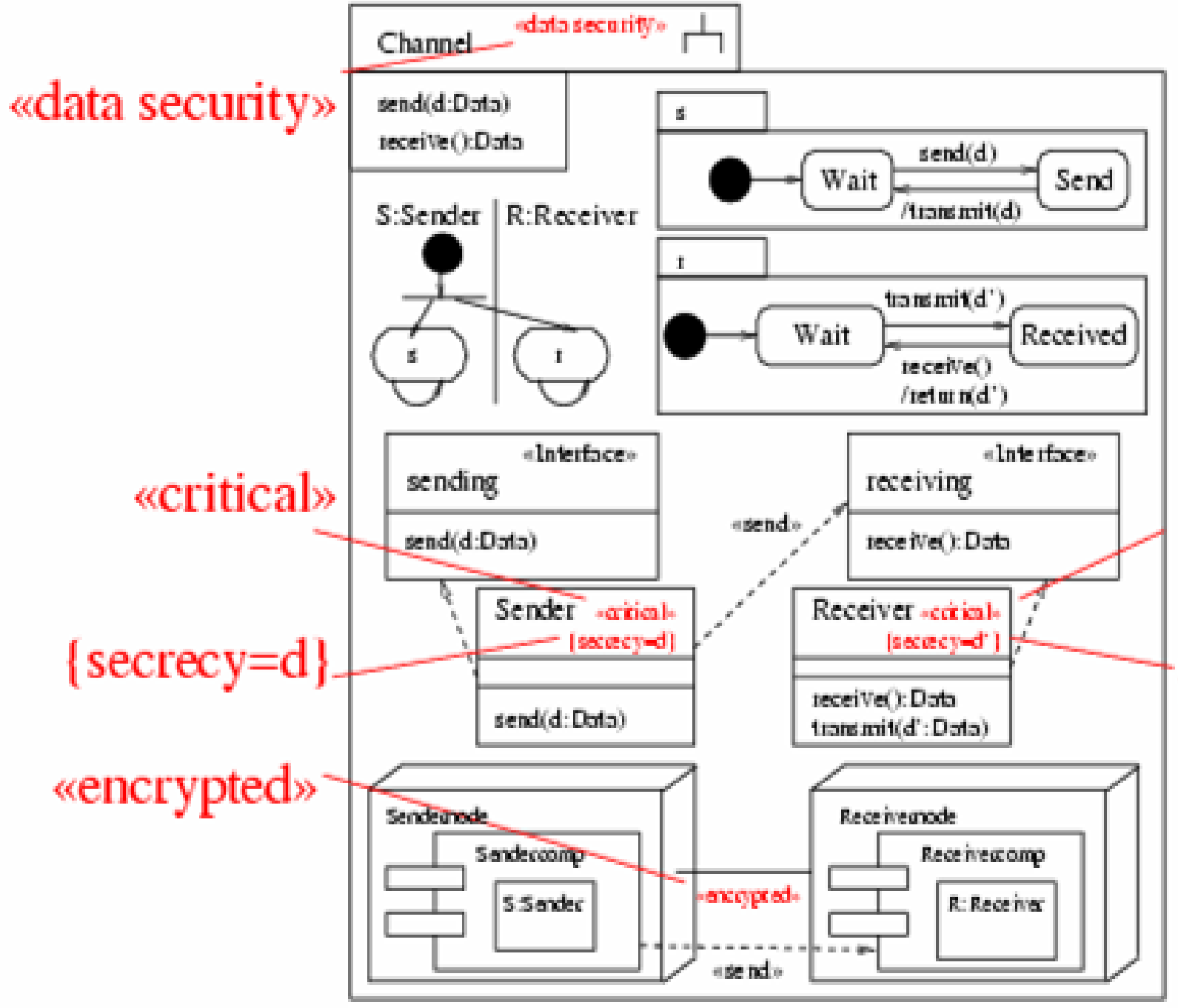
«Internet»



Variant of TLS (INFOCOM`99).

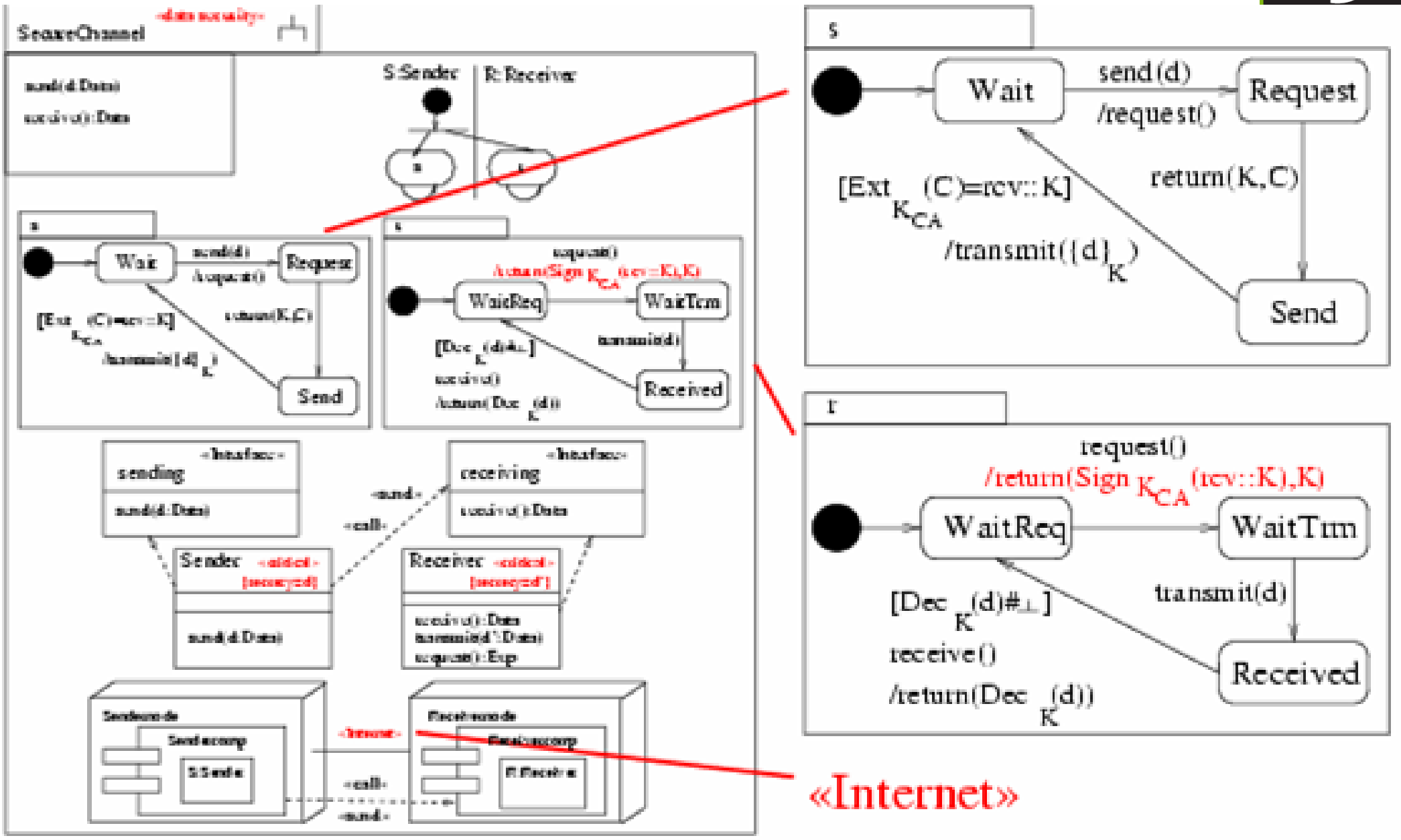
Violates {secrecy} of s against default adversary.

Using Protocols: Secure Channel



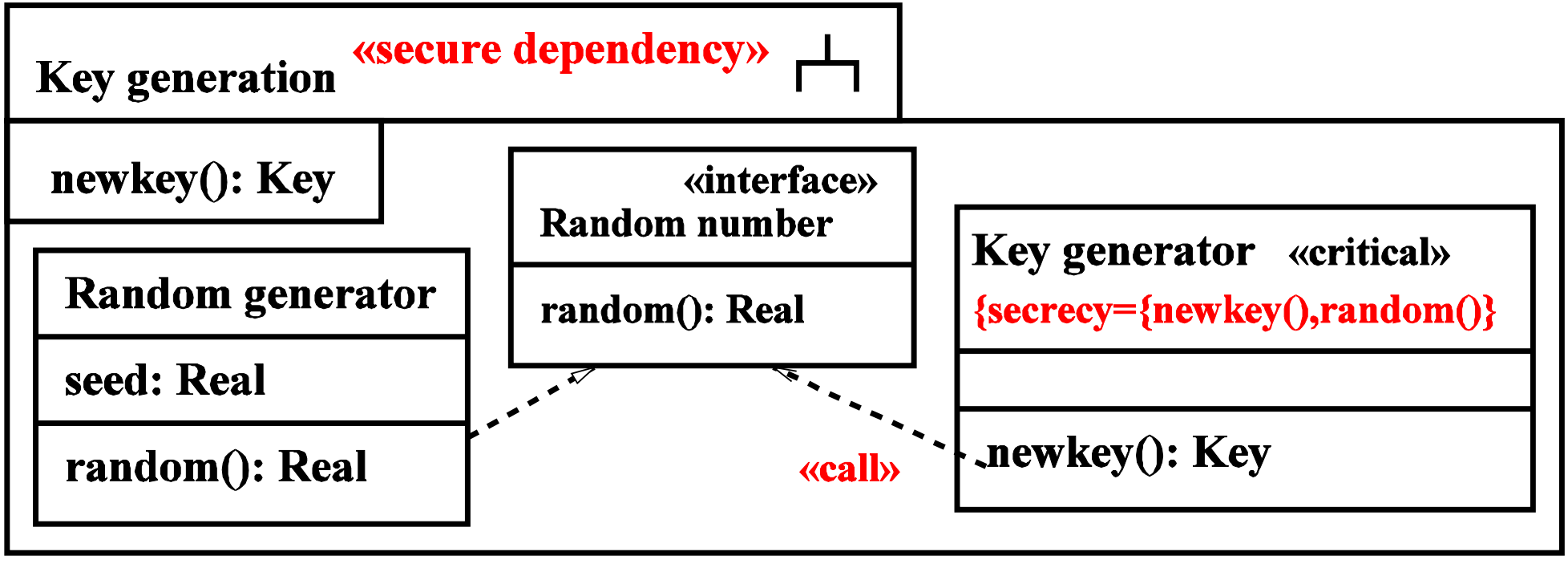
To keep *d* secret, must be sent encrypted.

Secure Channel Pattern: Solution



Exchange key and send encrypted data.

Secure Data Structure



Data structure secure ?

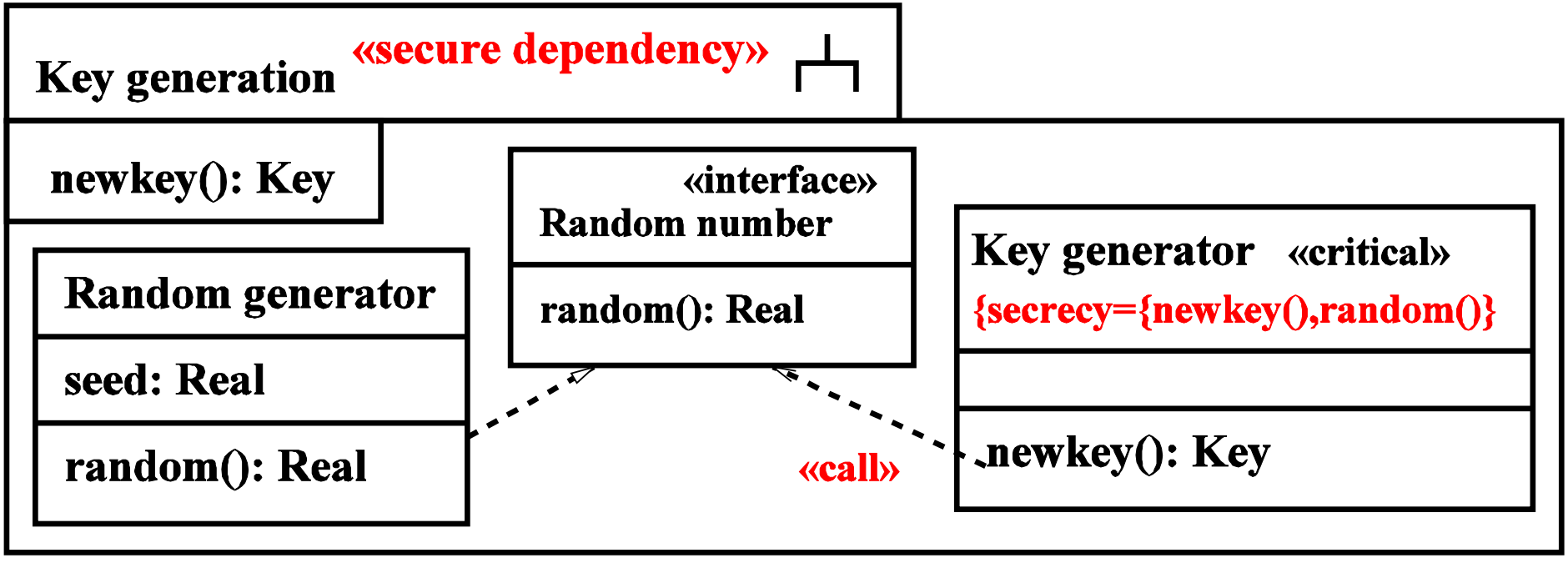
<<secure dependency>>

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.

<<secure dependency>>

existsCallOrSendDependency:
==========

Packages with stereotype secure dependency contain at least one dependency
with stereotype call or send.

```
(Package.allInstances()->select(p |
      (p.oclIsTypeOf(Package)) and
      (p.oclAsType(Package).getAppliedStereotypes().name->includes('secure
dependency'))).ownedElement.getAppliedStereotypes().name->includes('send')) or
(Package.allInstances()->select(p |
      (p.oclIsTypeOf(Package)) and (p.oclAsType(Package).getAppliedStereotypes().name
->includes('secure dependency'))).ownedElement.getAppliedStereotypes().name->includes('call'))
```

<<secure dependency>>

secureDependency:
==========
For each call or send dependency connecting two classes contained in a package that is equipped with the stereotype secure dependency the source and target classes must be equipped with the stereotype critical if the dependency has the stereotype secrecy or integrity.

```
(Package.allInstances()->select(p |
      (p.oclIsTypeOf(Package)) and (p.oclAsType(Package).getAppliedStereotypes().name-
>includes('secure dependency')))).allOwnedElements()->select(d |
            (d.oclIsTypeOf(Dependency)) and
            ((d.oclAsType(Dependency).getAppliedStereotypes().name->includes('call'))
                  or (d.oclAsType(Dependency).getAppliedStereotypes().name->includes('send'))
            ))->forAll(d |
                  (d.oclAsType(Dependency).source->forAll(oclIsTypeOf(Class))) and
                  (d.oclAsType(Dependency).target->forAll(oclIsTypeOf(Class))) and
                  (d.oclAsType(Dependency).getAppliedStereotypes().name->includes('secrecy') and
                   d.oclAsType(Dependency).source.getAppliedStereotypes().name->includes('critical')
and
```

<<secure dependency>>

```
d.oclAsType(Dependency).target.getAppliedStereotypes().name->includes('critical')) or
                (d.oclAsType(Dependency).getAppliedStereotypes().name->includes('integrity') and
                 d.oclAsType(Dependency).source.getAppliedStereotypes().name->includes('critical')
and
                 d.oclAsType(Dependency).target.getAppliedStereotypes().name->includes('critical'))
                )
```

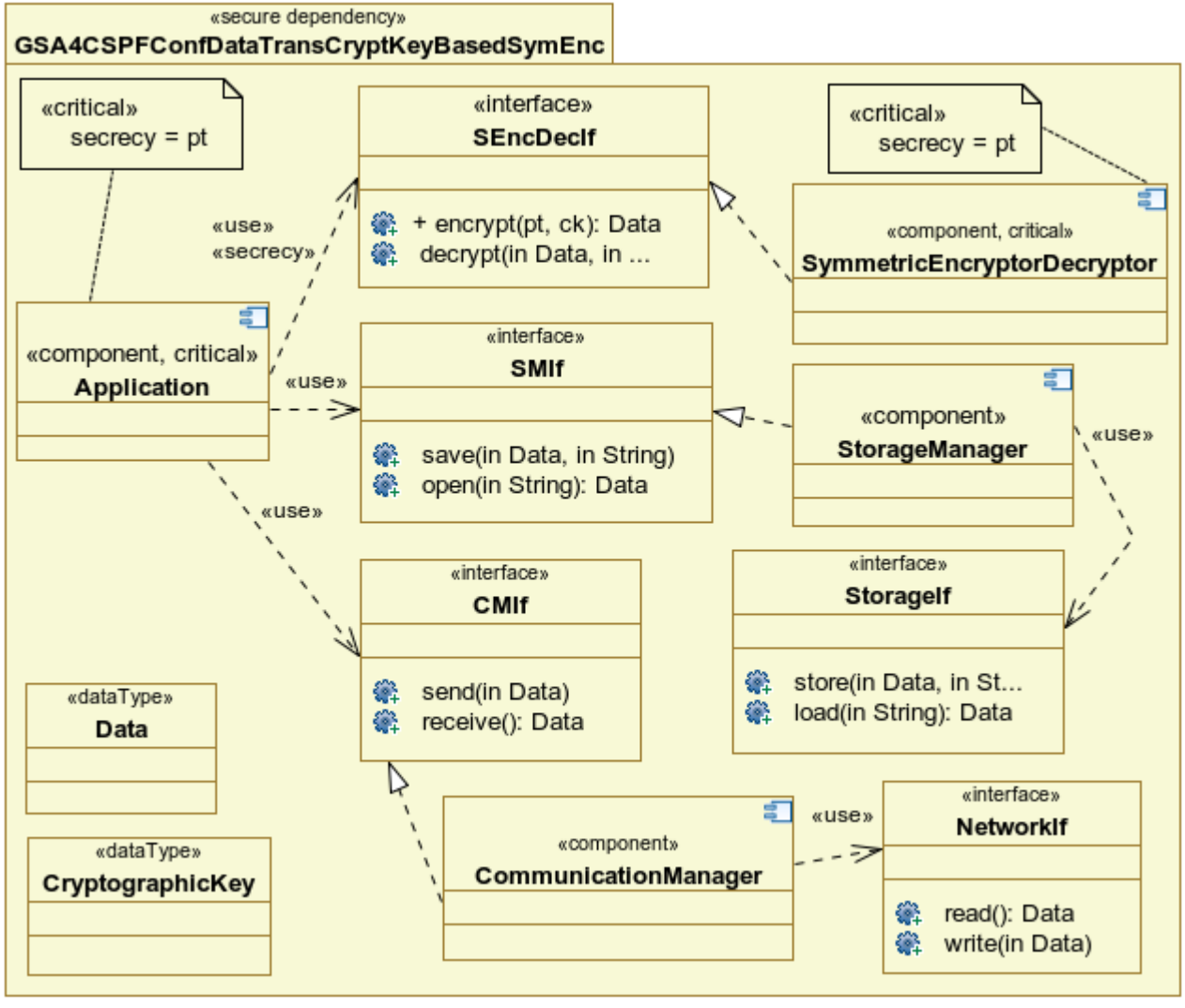
<<secure dependency>>

secrecy_SecrecyTaggedValuesAreEqual:
==========

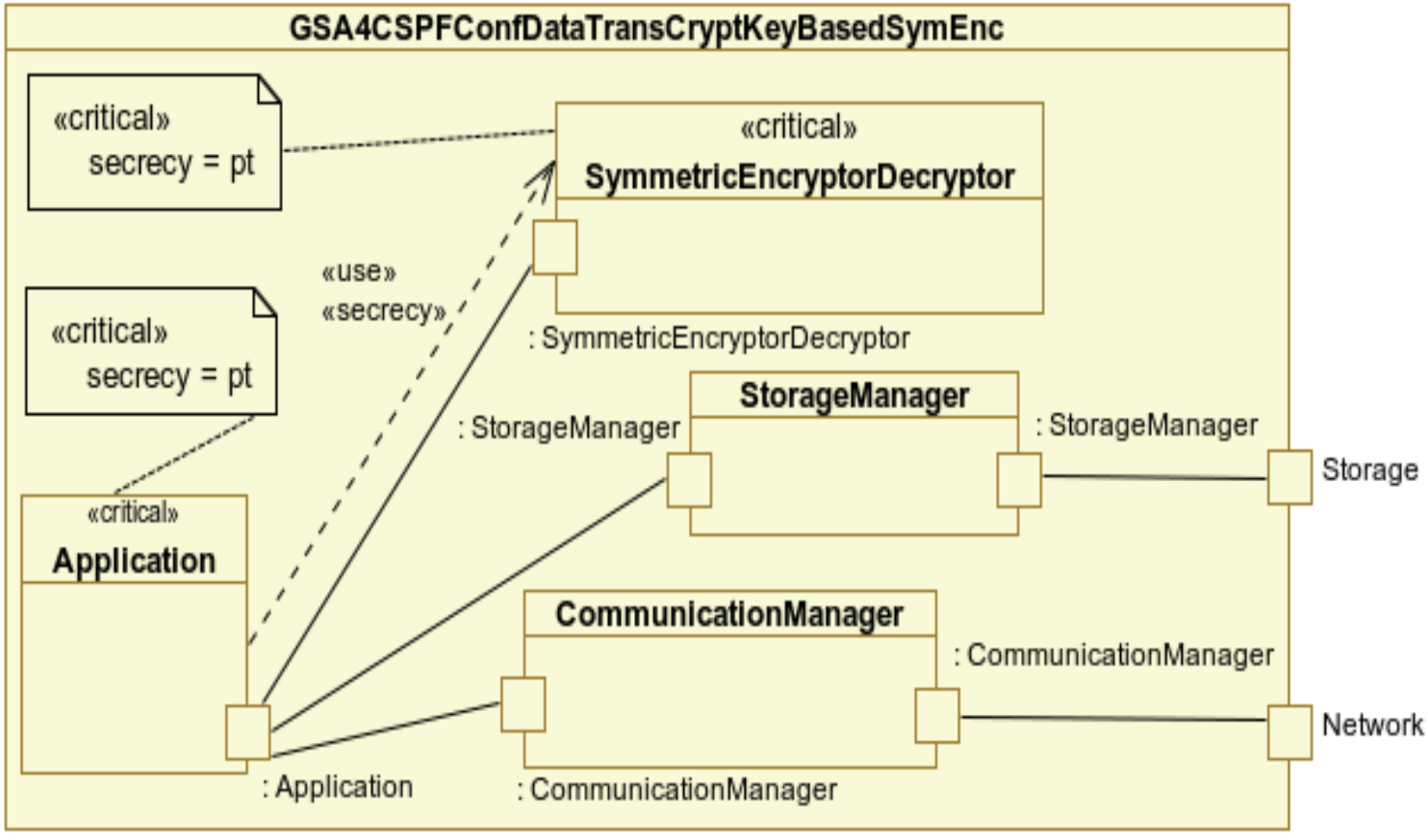
If the dependency has the stereotype secrecy, then the source and target classes must have tags secrecy with equal tagged values.

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

Example Port Definitions Using UMLsec4UML2

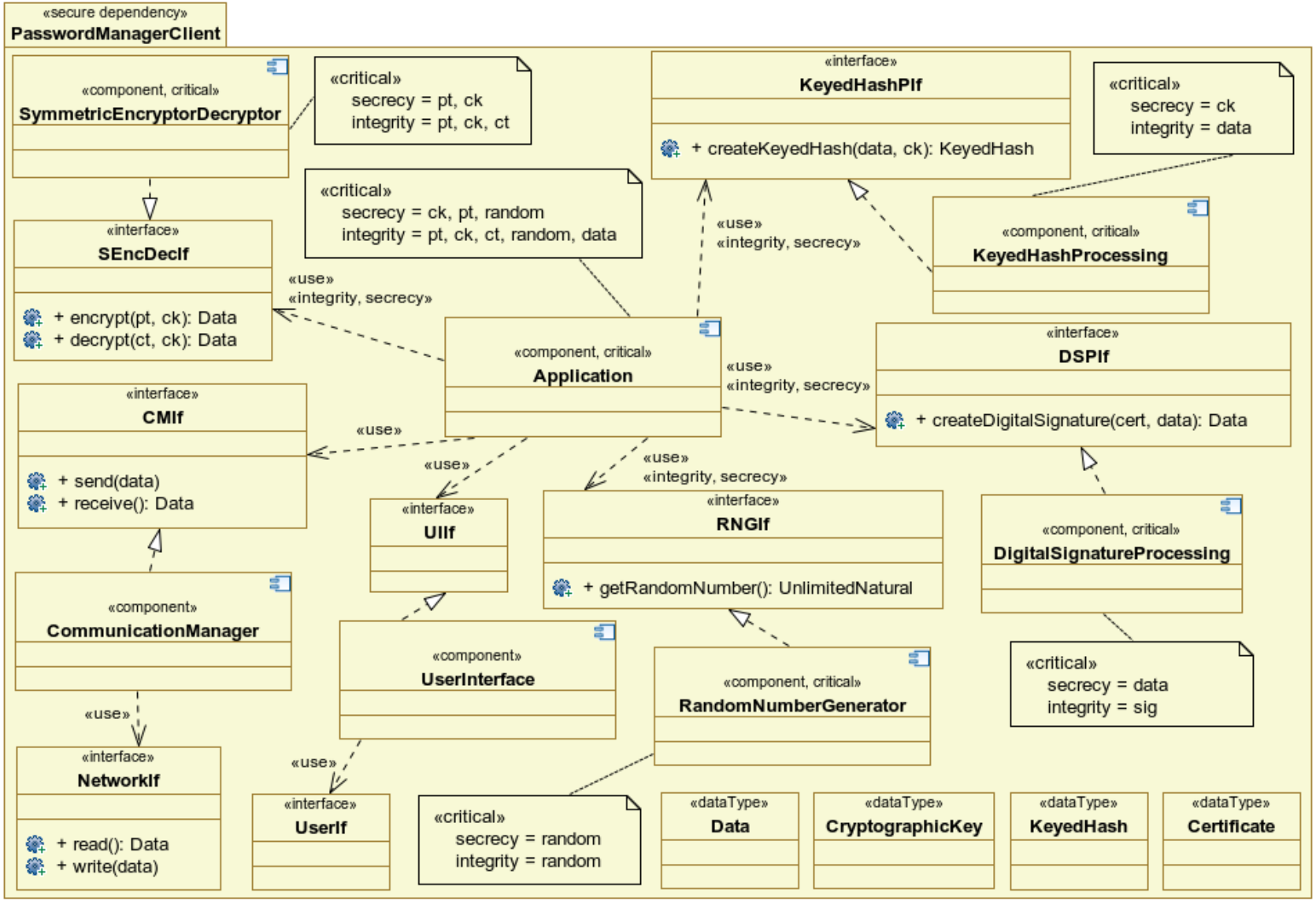


Example Security Architecture Pattern Using UMLsec4UML2



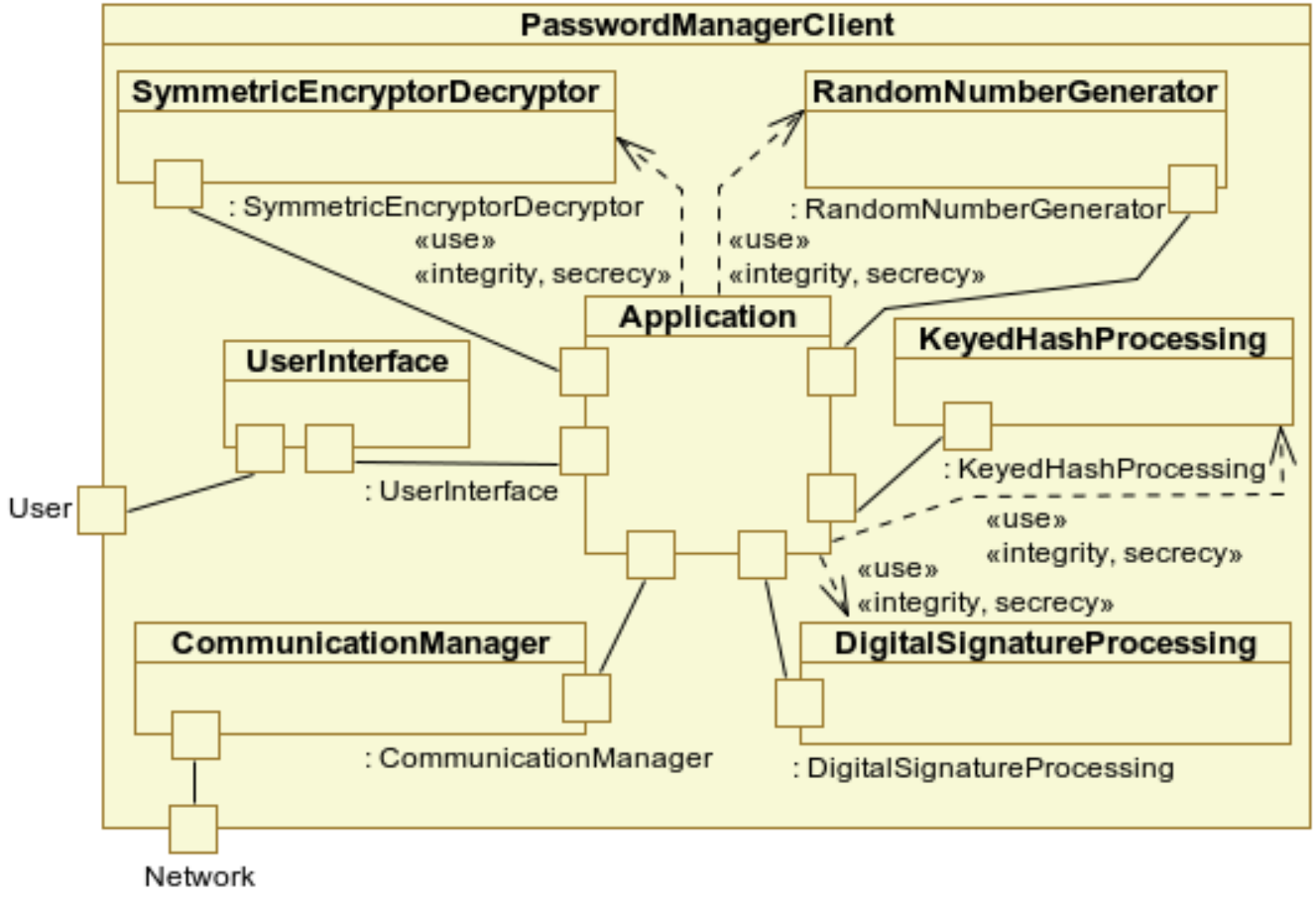
Example Security Architecure „Password Manager“

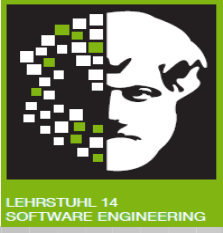
LEHRSTUHL 14
SOFTWARE ENGINEERING



Example Security Architecure „Password Manager“

LEHRSTUHL 14
SOFTWARE ENGINEERING





Java Security Architecture

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.

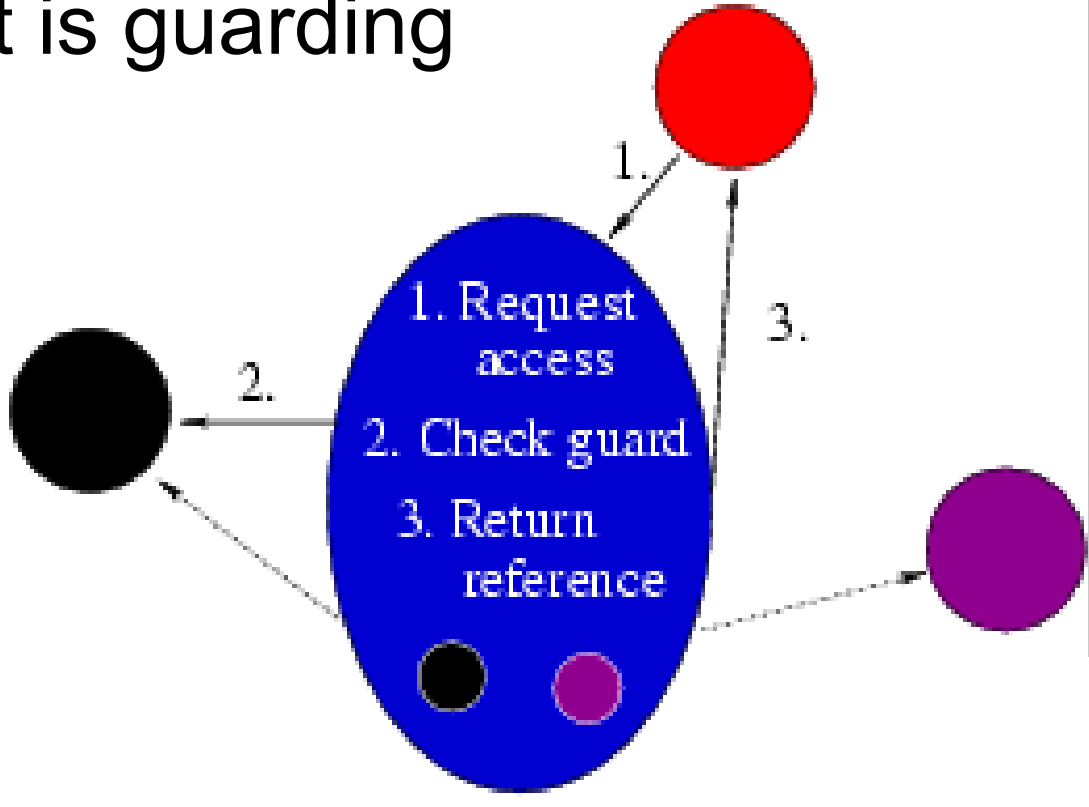
Similarly, need confidentiality.

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

- Permission depends on execution context.
- In particular: multiple threads.
- Thread may cross protection domains.

• doPrivileged() overrides execution context.

• Authentication in presence of adversaries.

• Indirect granting of access with capabilities.

Which run-time objects get permission ?

→ Use UMLsec to find out at design time.

⇒

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

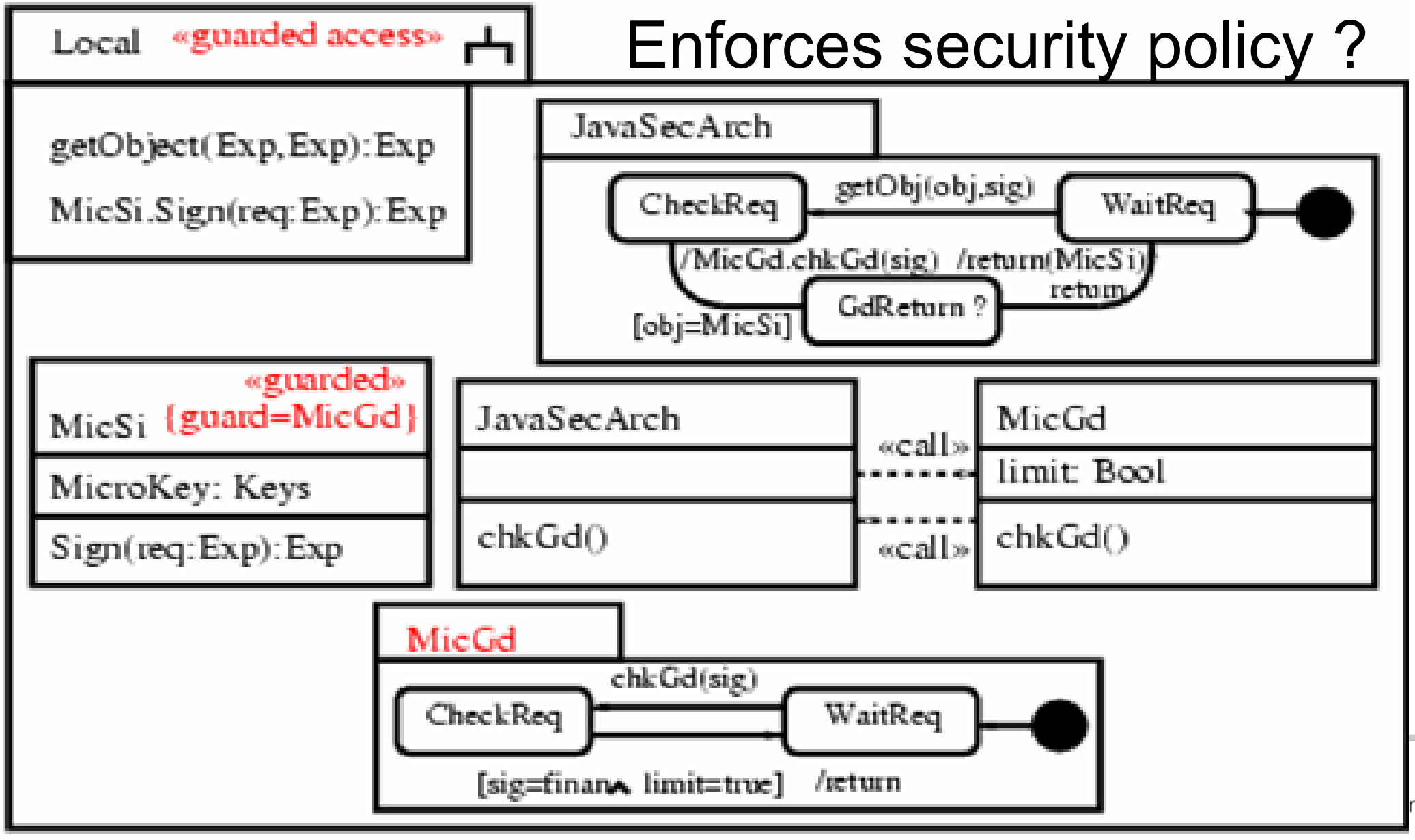
Theorem.

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.

Secure Use of Java Security Arch.



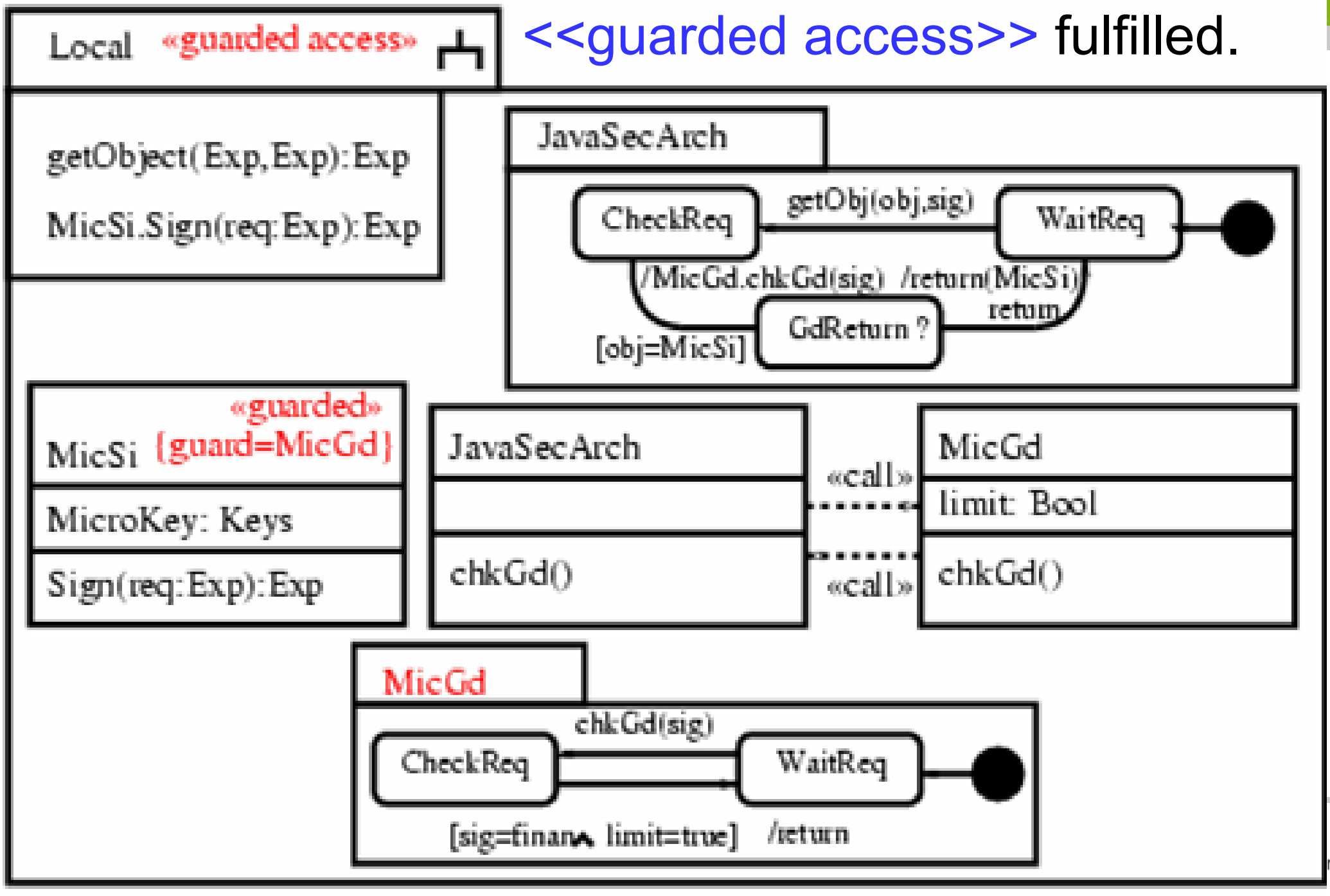
<<guarded access>>

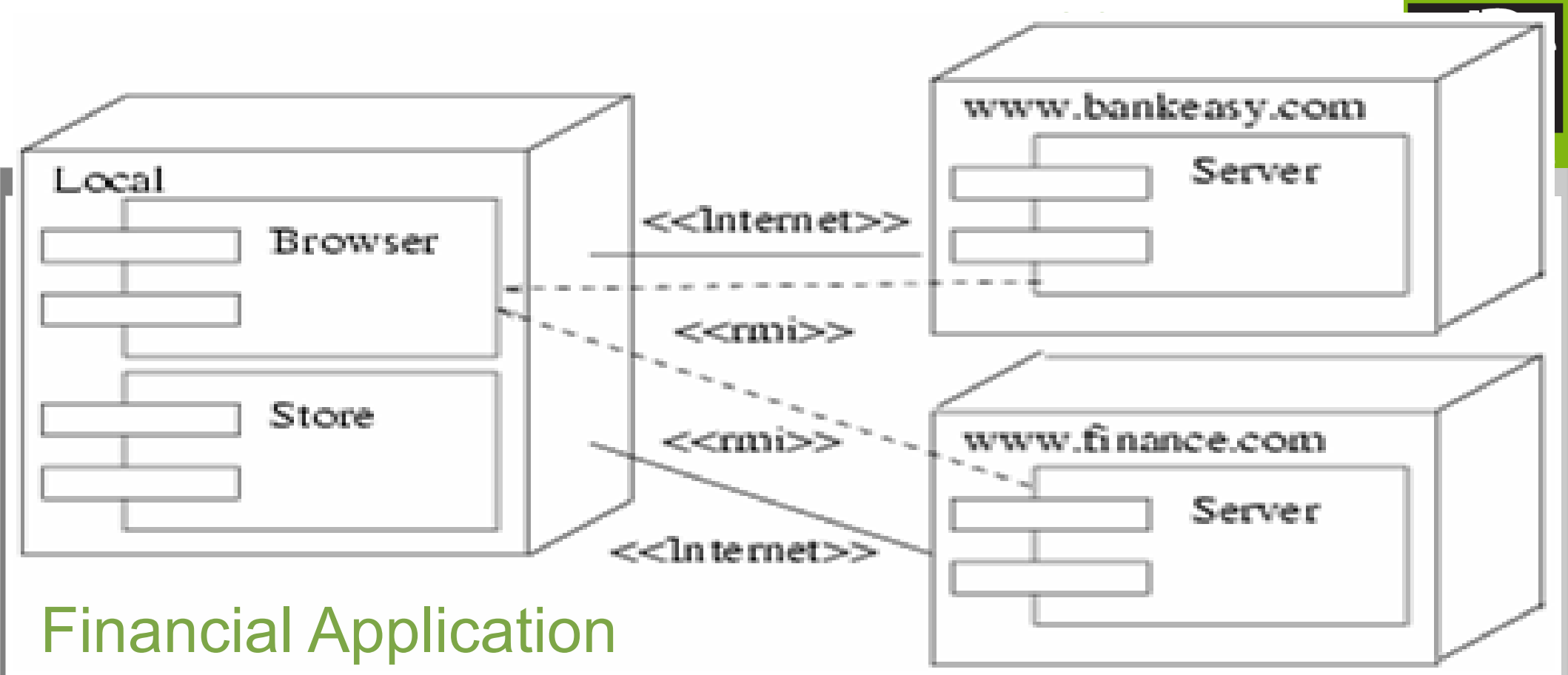
Ensures that in Java, <<guarded>> classes only accessed through {guard} classes.

Constraints:

- References of <<guarded>> objects remain secret.
- Each <<guarded>> class has {guard} class enforcing security policy.

Example <<guarded access>>





Financial Application

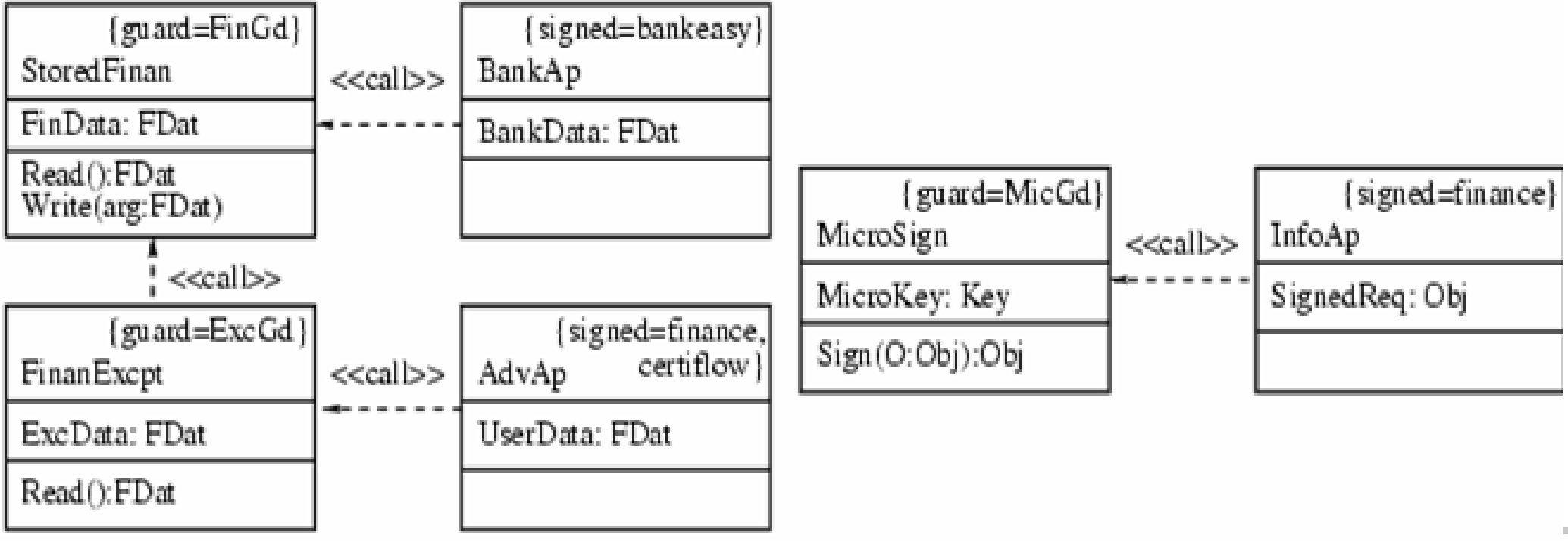
Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain Privileges (step1).

- 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

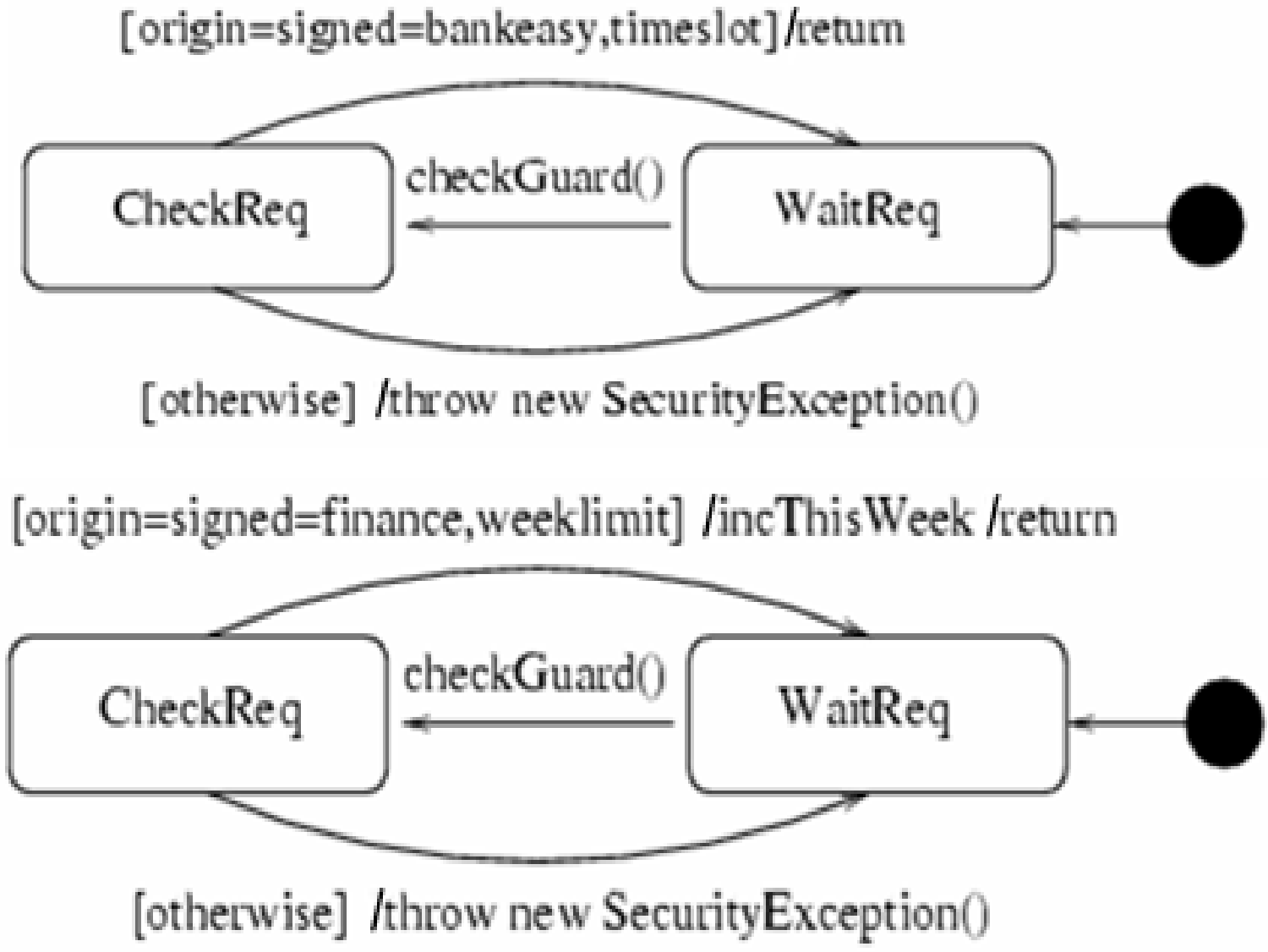
Sign and seal objects sent over Internet for integrity and confidentiality.

GuardedObjects control access.



Financial App: 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 vs. functionality (step 4). E.g.:

Proposition. If applet should get access to micropayment key according to policy, access granted.

Mobile objects sufficiently protected (step 5) - objects sent over Internet signed and sealed.

CORBA Access Control

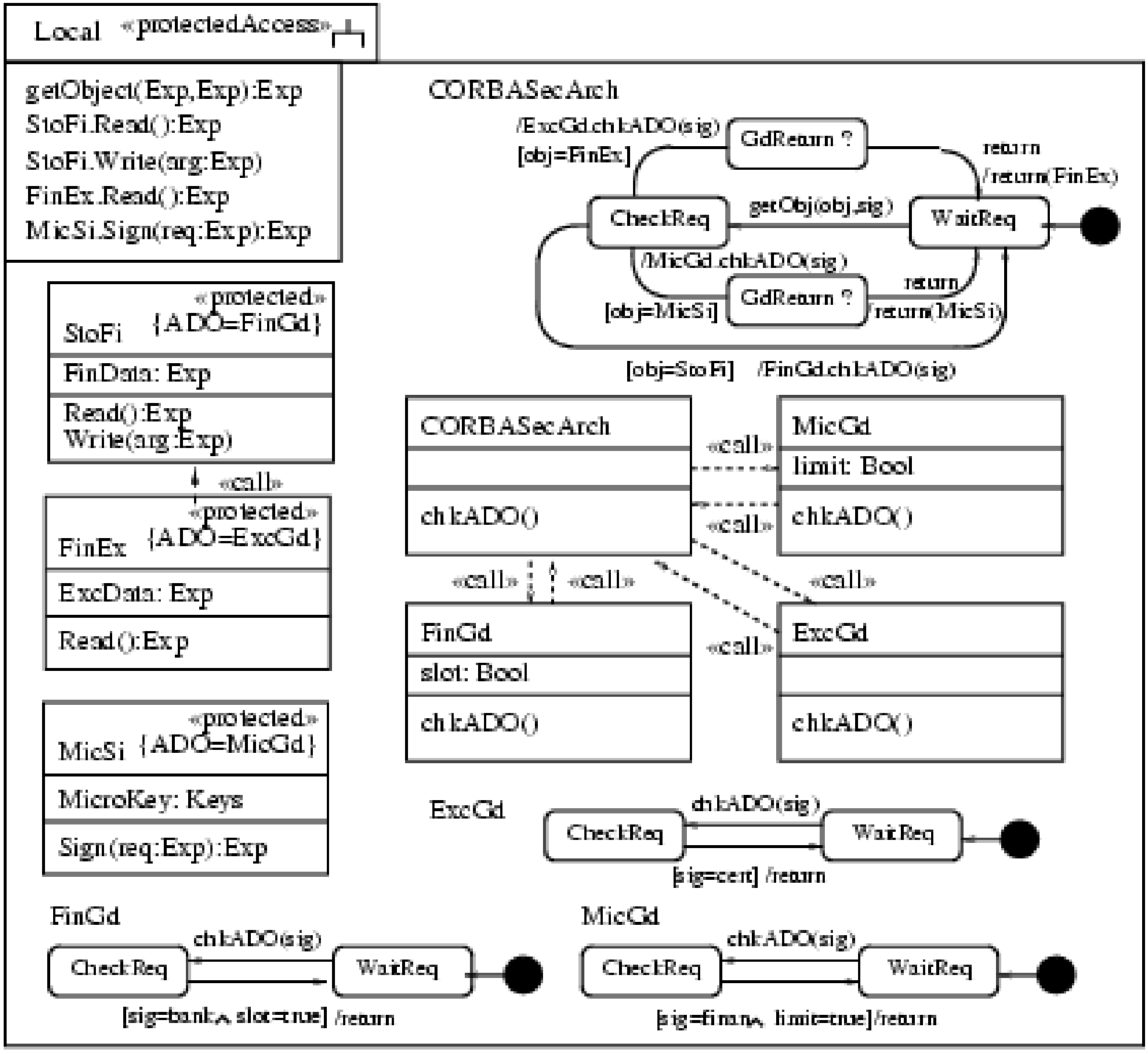
Object invocation access policy controls access of a client to a certain object via a certain method.

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.

CORBA Access Control with UMLsec



UMLsec Intro: Overview

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.

Some Applications

[ACSAC05,ICSE07]

Analyzed designs / implementations / configurations for

- biometry, smart-card or RFID based identification
- authentication (crypto protocols)
- authorization (user permissions, e.g. SAP systems)

Analyzed security policies, e.g. for privacy regulations.

Allianz

Deutsche Bank

HypoVereinsbank

CEPS™

BMW Group

msg systems

Bundesministerium für Bildung und Forschung

Bundesministerium für Wirtschaft und Technologie

O2

infineon

Münchener Rück Munich Re Group

fakultät für informatik

Intranet Information System

MetaSearch Engine: Personalized search in company intranet (including password protected).

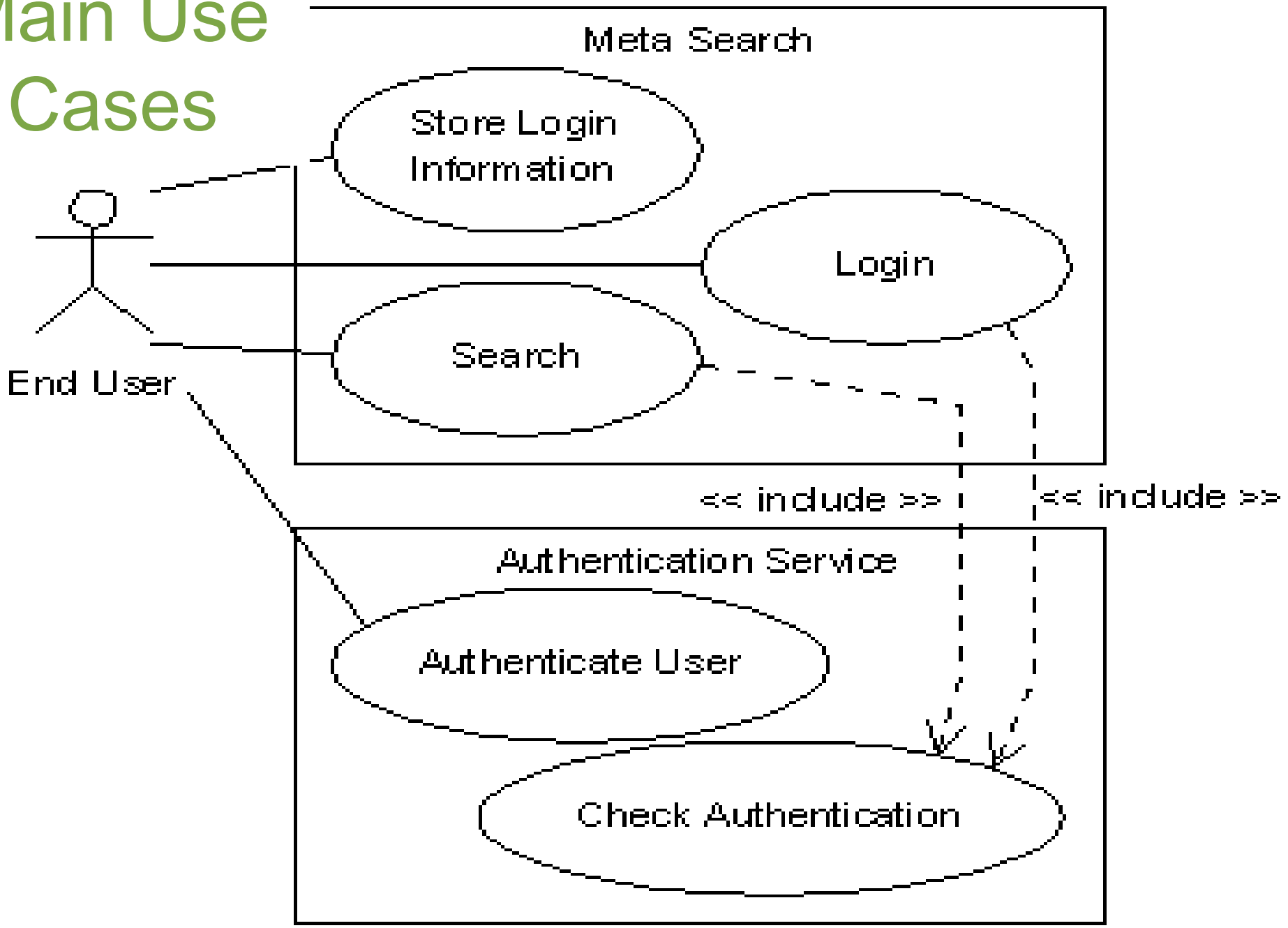
Some documents highly security-critical.

More than 1,000 potential users, index 280,000 documents, allow 20,000 queries per day.

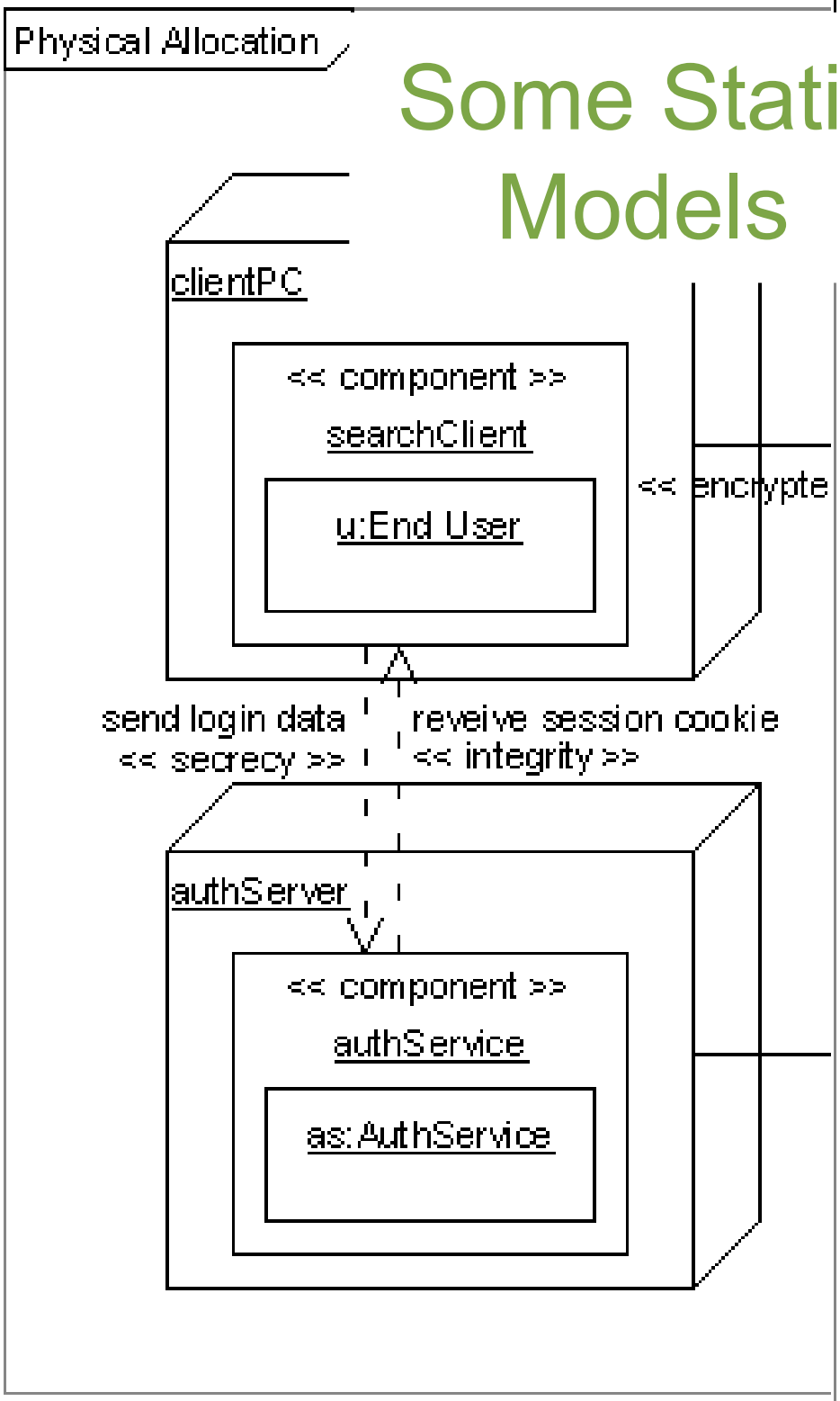
Seamlessly integrated in enterprise-wide security reference architecture. Provides security services to applications, including user authentication, role-based access control, global single-sign-on and hook-up of new security apps.

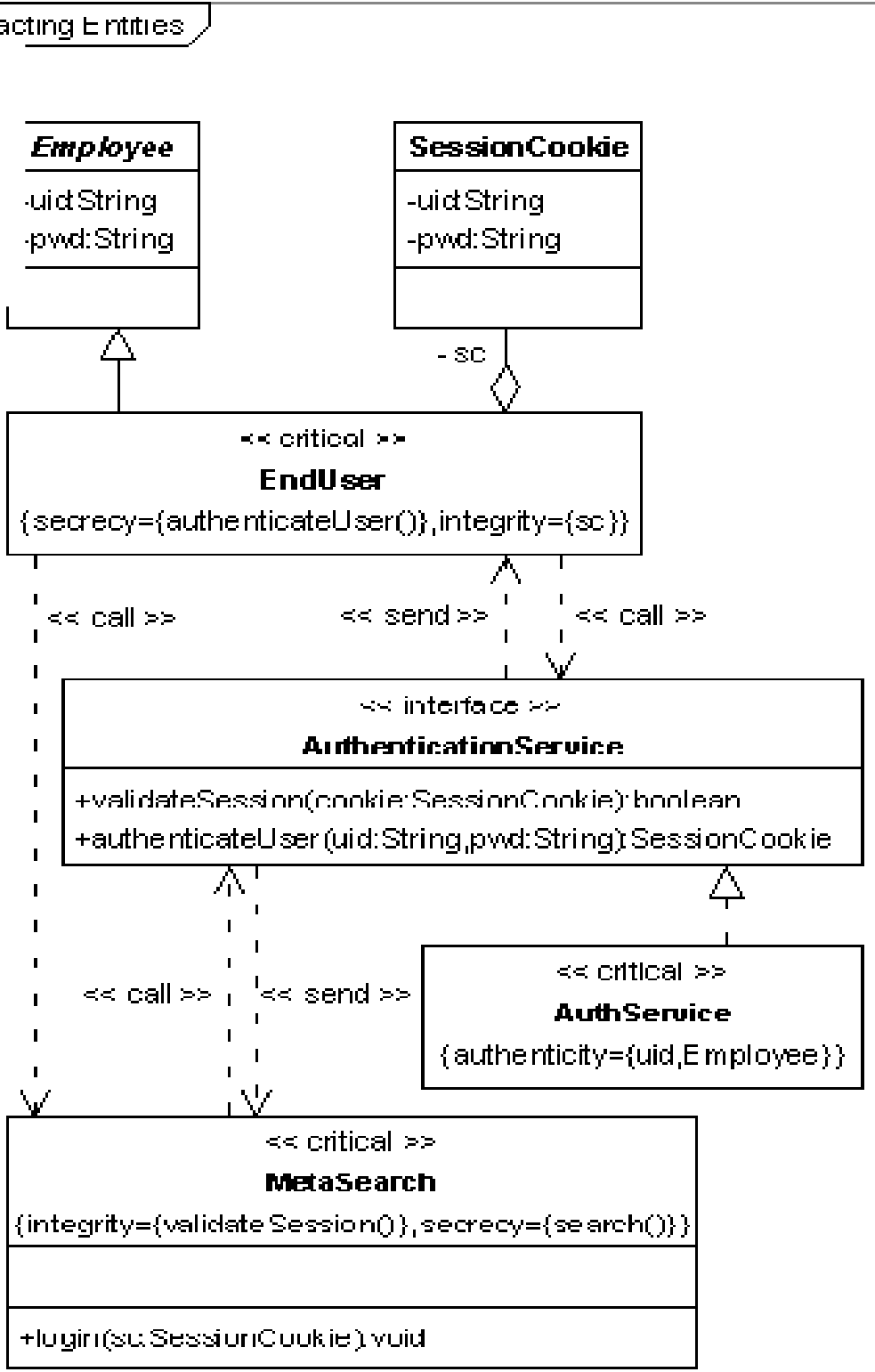
Use Case Overview

Main Use Cases



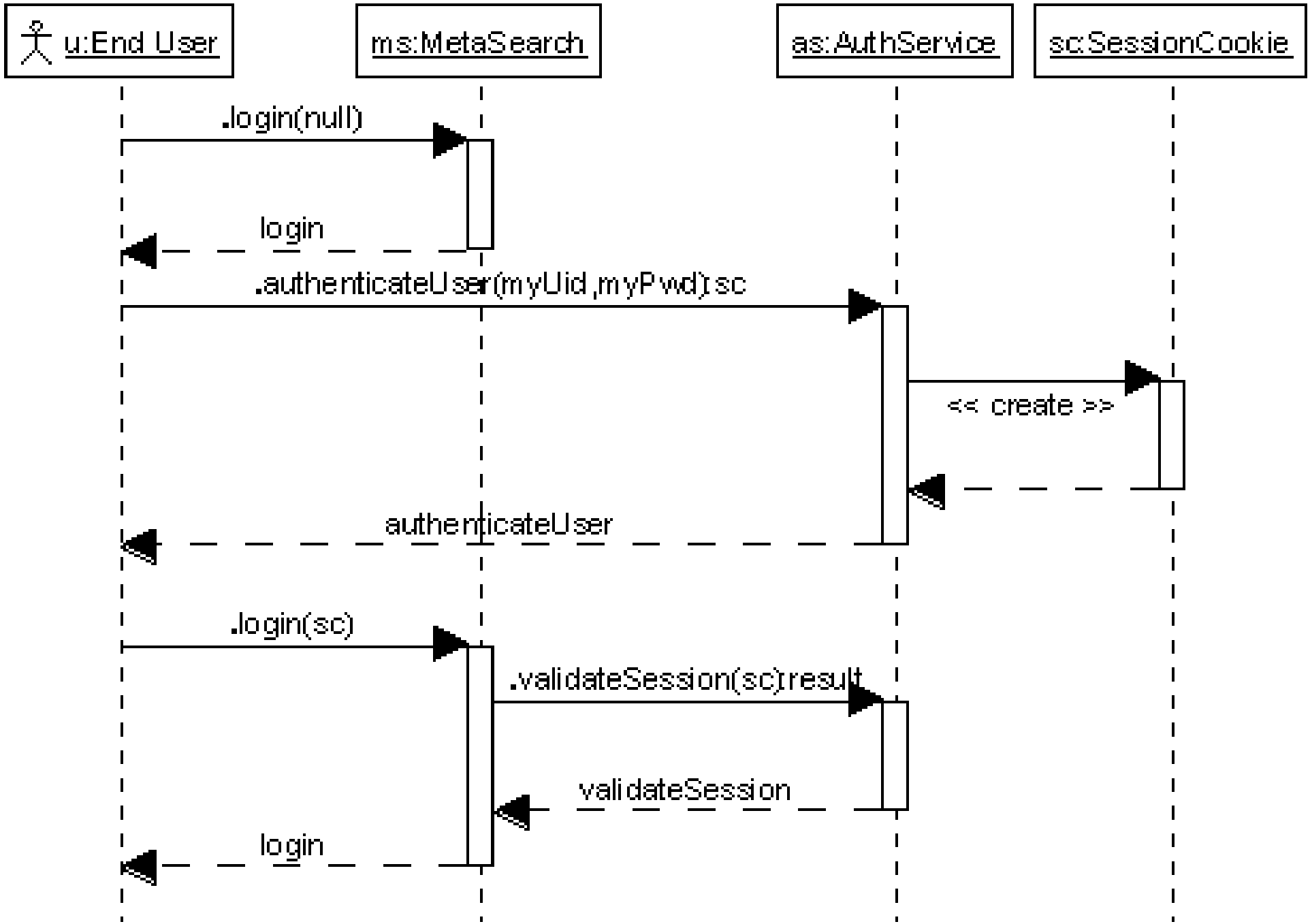
Some Static Models





Information Flow

A Dynamic Model



Security Analysis of Metasearch

Applied above approach to security-critical core of Metasearch (single sign-on mechanism).

To assess effectiveness, performed iterative approach. Start with model with several flaws. Subsequently found by the tools.

Note 100% security proof is impossible for principled reasons. Goal is optimal cost effectiveness for finding weaknesses in highly security-critical system parts.

The security properties that were considered were found to be enforced (details in paper).

Some Insights

- Model-based development with notations such as UML does incur effort.
  - That effort seems manageable when applied to core critical parts of the system.
  - It seems to be justifiable in case of high assurance needs (e.g. in security).
  - We believe it to compare favorably with traditional assurance methods offering a similar degree of trustworthiness.
- UMLsec seems to be well-suited for the domain of distributed information systems.

Possible Improvements

Experiences indicated potential improvements:

- extend notation further (e.g. to cover specialized domains such as mobility)
- provide tool support for these extensions
- Improve intuitiveness of parts of the existing tool support

Need further case-studies to investigate recent work on relating models to code [ASE05,06].

Challenge

Advanced tool support. For example:

- consistency checks
- mechanical analysis of complicated requirements on model level (bindings to model-checkers, constraint solvers, automated theorem provers, …)
- code generation
- test-sequence generation
- configuration data analysis against UML.

Tool-support: Concepts

Meaning of diagrams stated informally in (OMG 2003).

Ambiguities problem for

- tool support
  - establishing behavioral properties (e.g. safety, security)

Need precise semantics for used part of UML, especially to ensure critical 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 model arising from threat scenario in deployment diagram.

Use Abstract State Machines (pseudo-code).

Execution Semantics

Behavioral interpretation of a UML subsystem:

(1) Takes input events.

(2) Events distributed from input and link queues between subcomponents to intended recipients where they are processed.

(3) Output distributed to link or output queues.

(4) Apply adversary model.

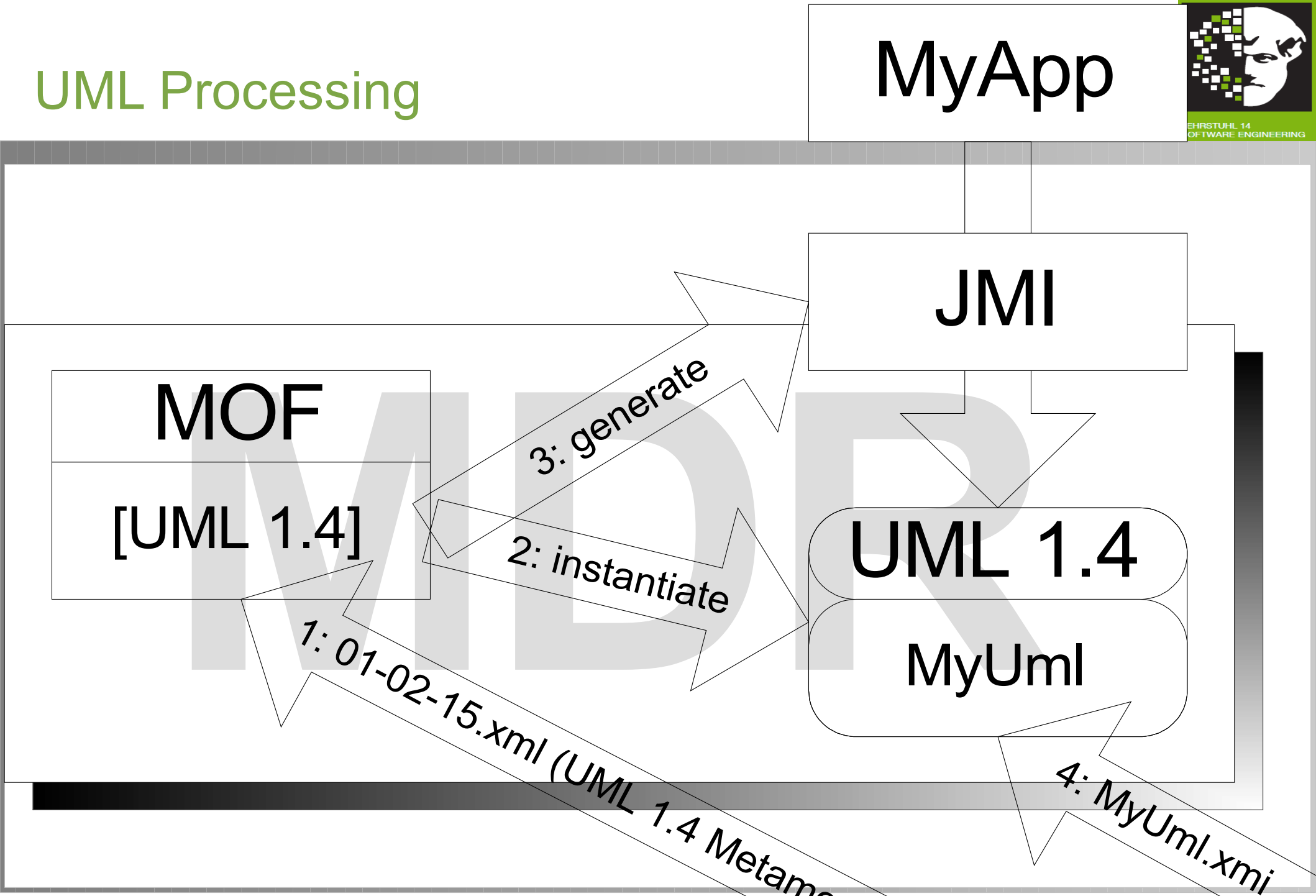
Tool-support: Pragmatics

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

Goal: sophisticated analysis. Solution:

- Draw UML models with editor.
- Save UML models as XMI (XML dialect).
- Connect to verification tools (automated theorem prover, model-checker …), e.g. using XMI Data Binding.

UML Processing



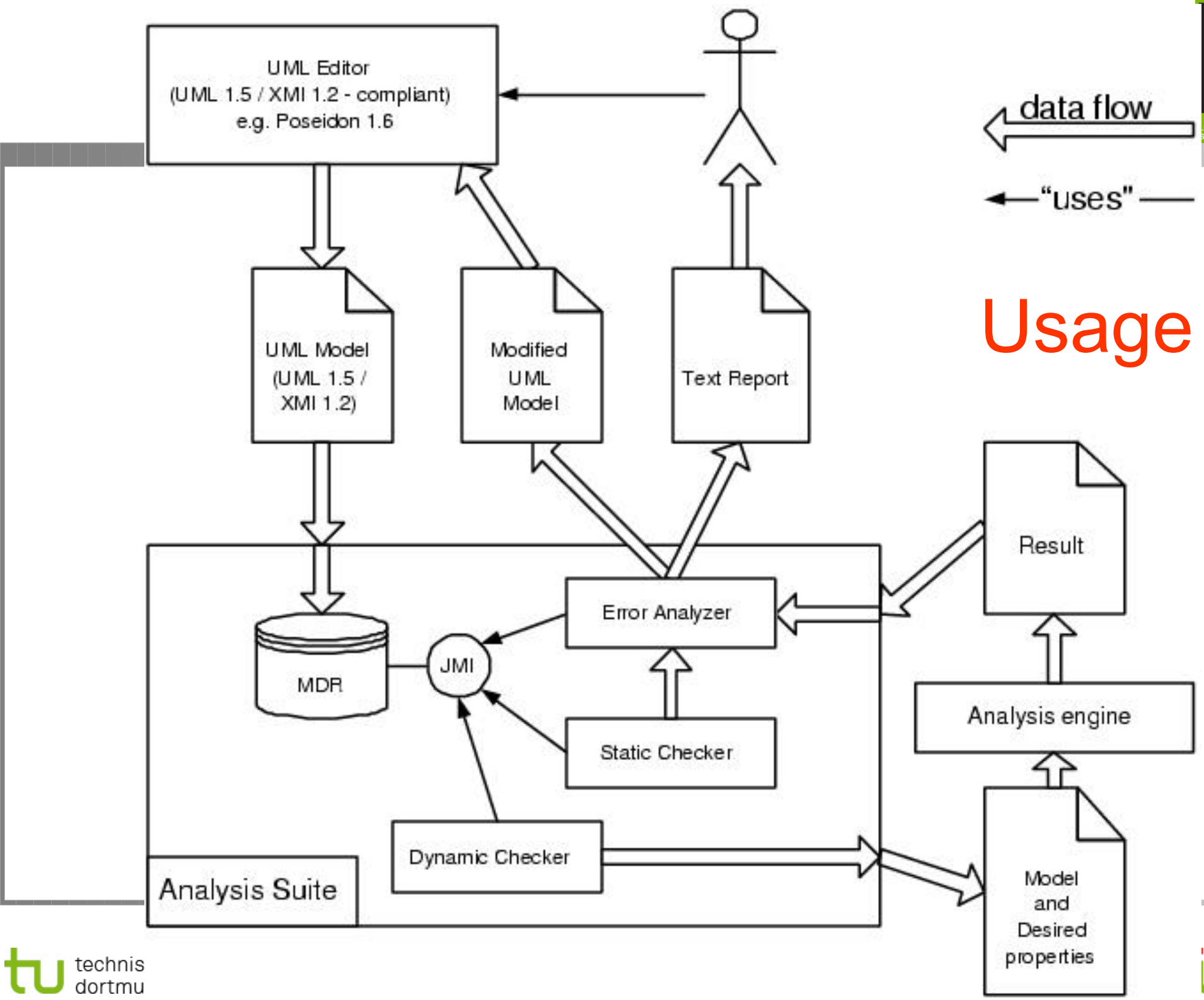
CSDUML Framework: Features

Framework for analysis plug-ins to access UML models on conceptual level over various UI's.

Exposes a set of commands. Has internal state (preserved between command calls).

Framework and analysis tools accessible and available at http://www-jj.cs.tu-dortmund.de/jj/umlsectool .

Upload UML model (as .xmi file) on website. Analyse model for included critical requirements. Download report and UML model with highlighted weaknesses.


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
Usage
Result
Error Analyzer
MDR
JMI
Analysis engine
Static Checker
Dynamic Checker
Analysis Suite
Model
and
Desired
properties

Vision

- Simple independent tools
- Media-independent
- Easy to use
  - Simple developer interface
- Easy to maintain
  - Simple architecture

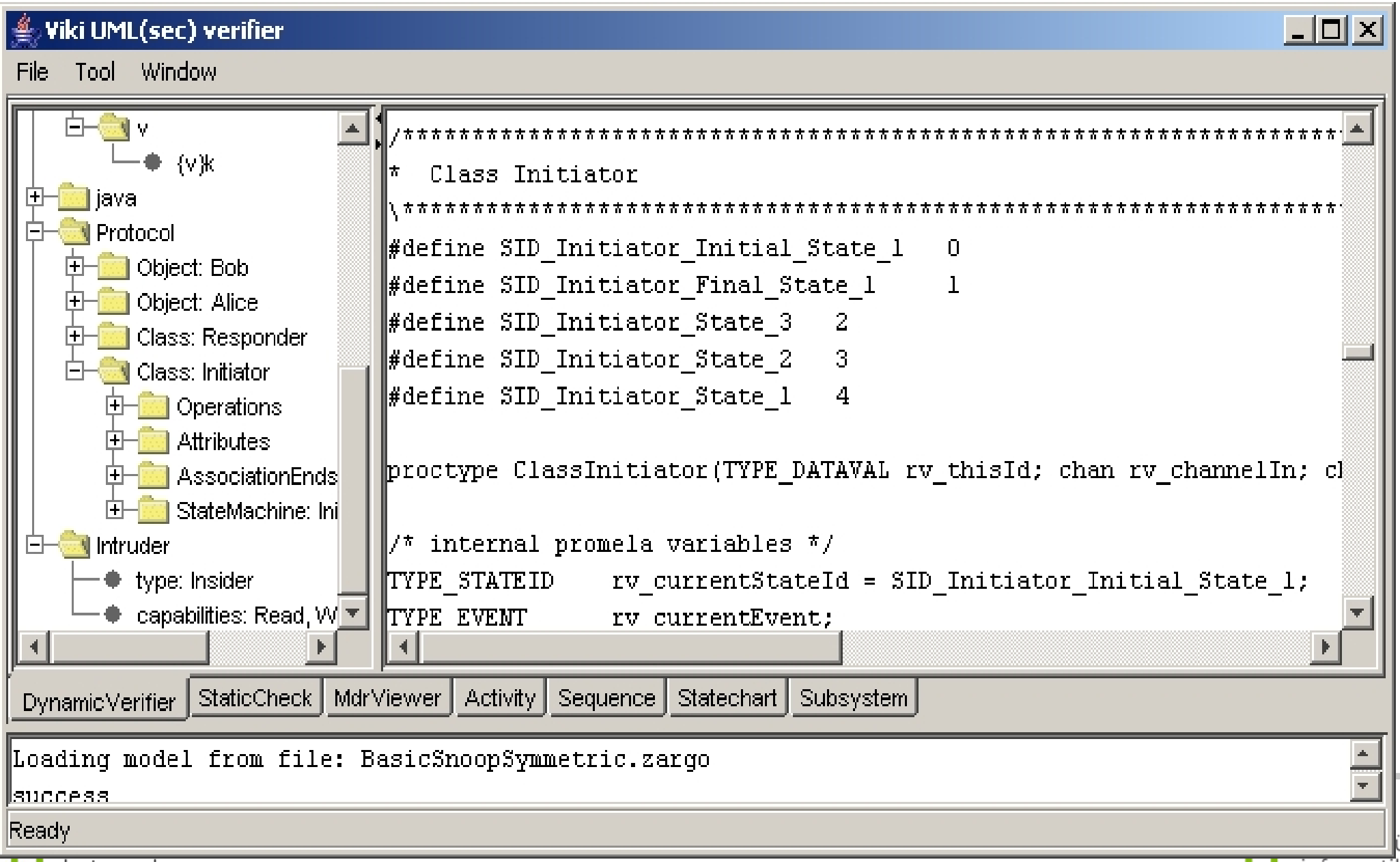
Concept

- Set of plug-in tools
  - Tool exposes predefined interfaces
  - Tool can use framework interfaces
- Tool implements a set of commands
  - Each command has parameters
- Framework = common code
  - UML model management
  - Other services

Implementing Tools

- Define the set of commands
  - have parameters
- Tool State preserved between commands
- Commands are not interactive
  - receive parameters
  - execute
  - deliver output

GUI


Viki UML(sec) verifier
File
Tool
Window
v
{v}k
java
Protocol
Object: Bob
Object: Alice
Class: Responder
Class: Initiator
Operations
Attributes
AssociationEnds
StateMachine: Ini
Intruder
type: Insider
capabilities: Read, W
/***
* Class Initiator
****
#define SID_Initiator_Initial_State_1   0
#define SID_Initiator_Final_State_1   1
#define SID_Initiator_State_3   2
#define SID_Initiator_State_2   3
#define SID_Initiator_State_1   4
proctype ClassInitiator(TYPE_DATAVAL rv_thisId; chan rv_channelIn; cl
/* internal promela variables */
TYPE_STATEID   rv_currentStateId = SID_Initiator_Initial_State_1;
TYPE_EVENT   rv_currentEvent;
DynamicVerifier
StaticCheck
MdrViewer
Activity
Sequence
Statechart
Subsystem
Loading model from file: BasicSnoopSymmetric.zargo
success
Ready