

Security Modeling with UMLsec

Jan Jürjens

Competence Center for IT Security
Software & Systems Engineering
TU Munich, Germany



juerjens@in.tum.de

<http://www.jurjens.de/jan>



Personal Introduction

Leading the Competence Center for IT-Security
at Software & Systems Engineering,
TU Munich

- Extensive collaboration with industry (BMW, HypoVereinsbank, T-Systems, Munich Re, Deutsche Bank, Siemens, Infineon, ...)
- PhD in Computer Science from Oxford Univ., Masters in Mathematics from Bremen Univ.
- Numerous publications incl. 1 book on the subject



Jan Jürjens, TU Munich: Security Modeling with UMLsec

2

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**.



Jan Jürjens, TU Munich: Security Modeling with UMLsec

3

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..



Jan Jürjens, TU Munich: Security Modeling with UMLsec

4

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**.



Jan Jürjens, TU Munich: Security Modeling with UMLsec

5

Causes II

„Blind“ use of mechanisms:

- 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“ (R. Needham).



Jan Jürjens, TU Munich: Security Modeling with UMLsec

6

Software Engineering & Security

„Penetrate-and-patch“
(aka „banana strategy“):

- insecure
- disruptive



Traditional formal methods: expensive.

- training people
- constructing formal specifications.

Towards Use in Practice

Increase security with bounded investment in **time, costs** (crucial for industry). Idea:



- Extract models from **artefacts** arising in **industrial development** and **use** of security-critical systems (UML models, source code, configuration data).
- Tool-supported **theoretically sound efficient automated security analysis**.

→ **Model-based Security Engineering**

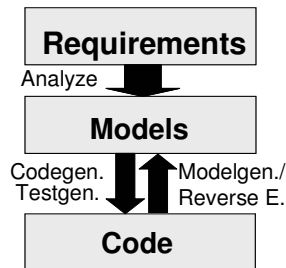
Model-based Security Engineering

Combined strategy:

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

Goal: model-based = source-based.

Idea notation-independent. Here: use UML.



Why UML ?

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

Relatively precisely defined (given the user community).

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

UMLsec: Goals

Extension for **secure systems** development.

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

UMLsec: How

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

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

Ensures that UML specification **provides** desired level of security requirements.

Link to code via round-trip engineering etc.

Roadmap

Prologue
 UMLsec
 Tools
 Security Analysis

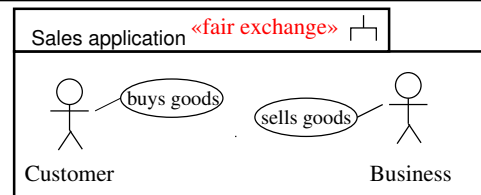
Requirements on UML Extension for Security I

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, ...).

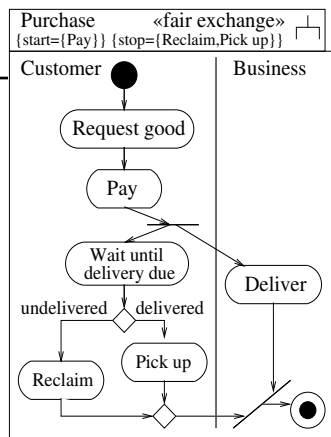
Requirements with Use Case Diagrams



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.).

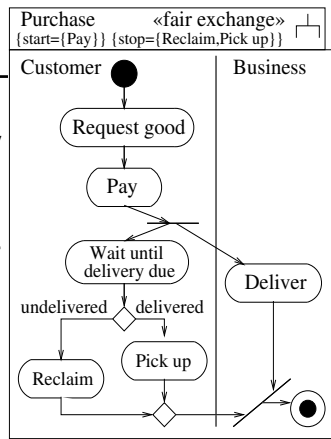


<<fair exchange>>

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», ...

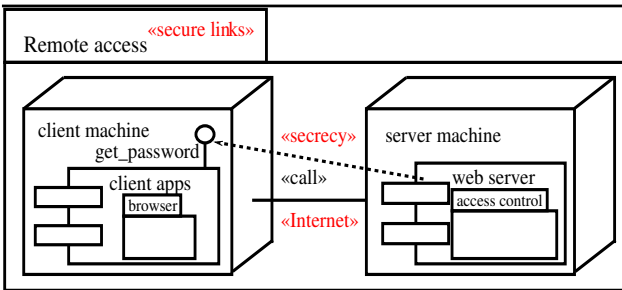
Kinds of communication **links** resp. system **nodes**.

For adversary type A , stereotype s , have set $\text{Threats}_A(s) \in \{\text{delete, read, insert, access}\}$ 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 ?

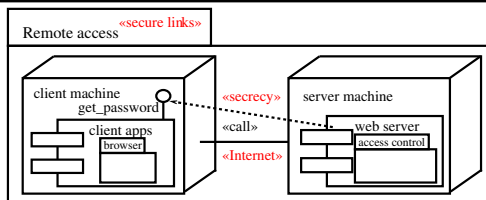
«secure links»

Ensures that physical layer meets security requirements on **communication**.

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

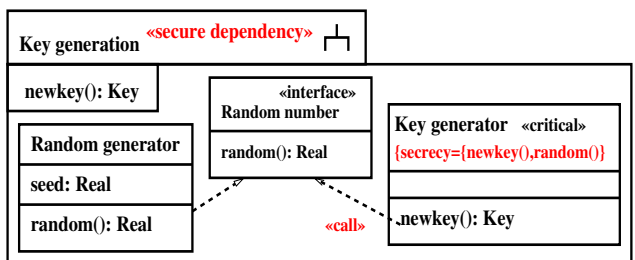
- if $s = \text{«secrecy»}$: have $\text{read} \notin \text{Threats}_A(t)$.
- if $s = \text{«integrity»}$: have $\text{insert} \notin \text{Threats}_A(t)$.

Example «secure links»



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

Secure Data Structure



Data structure secure ?

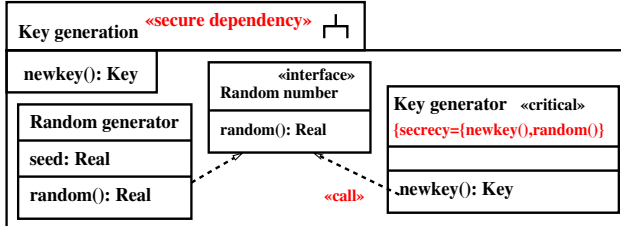
«secure dependency»

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**}):

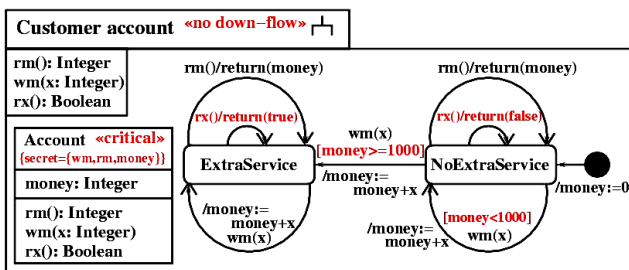
- Msg in *D* is {**secrecy**} in *C* if and only if also in *D*.
- If msg in *D* is {**secrecy**} in *C*, dependency stereotyped «**secrecy**».

Example «secure dependency»



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

Secure Information Flow



No partial leakage of secrets ?

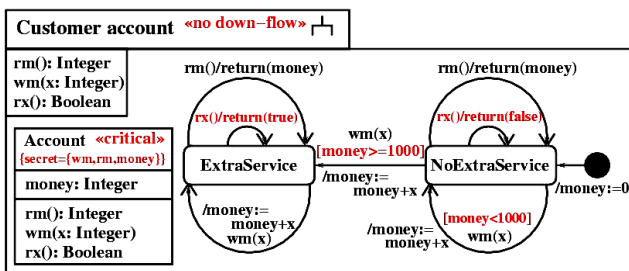
«no down-flow»

Enforce secure information flow. Constraint:

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

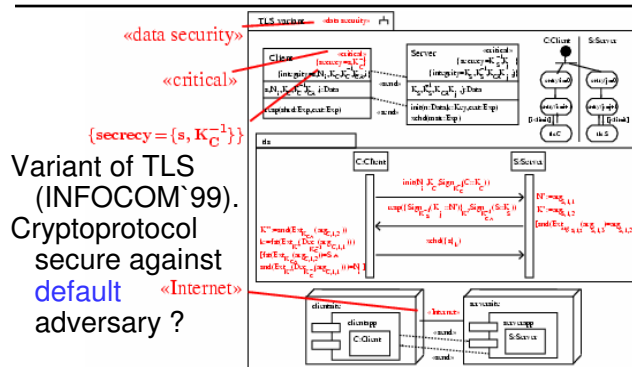
Formalize by referring to formal behavioural semantics.

Example «no down-flow»



«no down-flow» violated: partial information on input of secret wm() returned by non-secret rx().

Secure Use of Cryptography

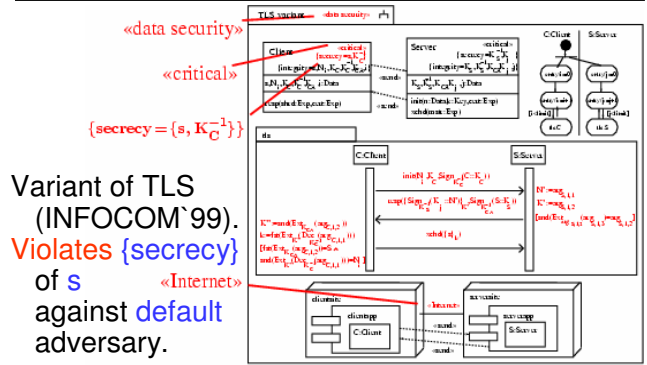


«data security»

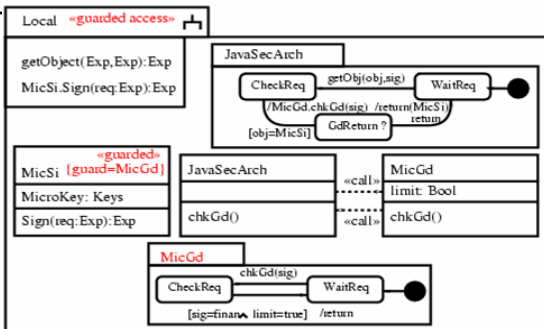
Security requirements of data marked «critical» enforced against threat scenario from deployment diagram.

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

Example «data security»



Secure Use of Java Security Arch.



Enforces overall security policy ?

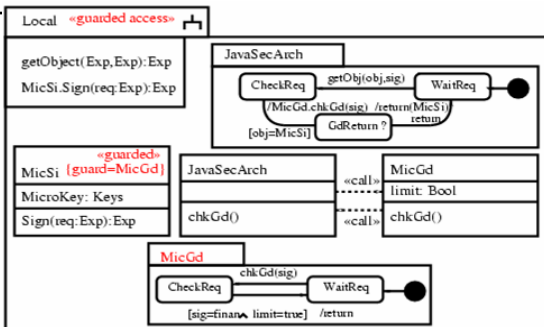
«guarded access»

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»



«guarded access» fulfilled.

Does UMLsec Meet Requirements ?

Security requirements: «secrecy»,...

Threat scenarios: Use Threats_{adv}(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
UMLsec
Tools
Security Analysis

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: How

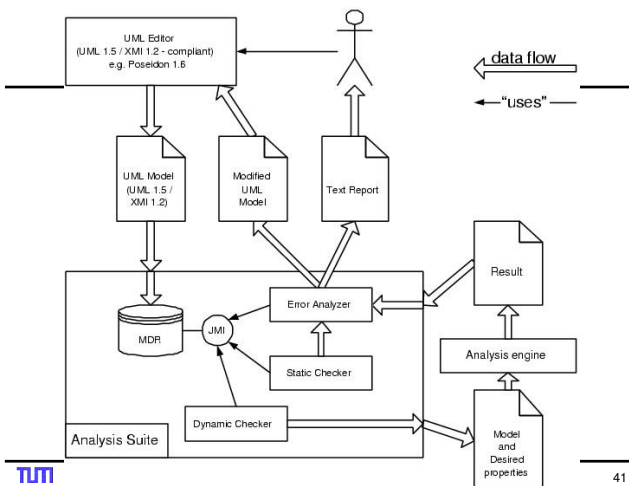
Diagrams in **context** (using subsystems).
Model **actions** and internal **activities** explicitly.

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).

CSDUML Framework

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://www4.in.tum.de/~umlsec>. Upload UML model (as .xmi file) on website. Analyse model for included critical requirements. Download report and UML model with highlighted weaknesses.



Roadmap

Prologue
UMLsec
Tools
Security Analysis

Security Analysis

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

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

Cryptography

Keys are **symbols**, crypto-algorithms are **abstract** operations.

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

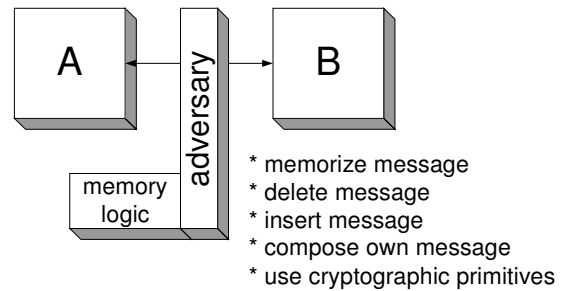
Expressions

Exp: term algebra generated by $\text{Var} \cup \text{Keys} \cup \text{Data}$ and

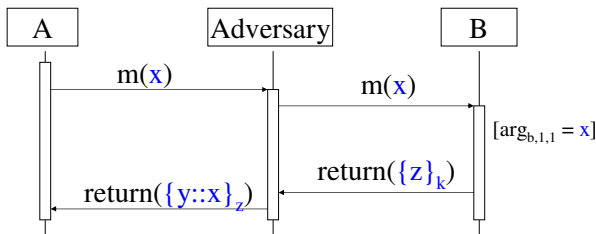
- $_ :: _$ (concatenation) and empty expression \mathcal{E} ,
- $\{ _ \} _$ (encryption)
- $\text{Dec}(_)$ (decryption)
- $\text{Sign}(_)$ (signing)
- $\text{Ext}__()$ (extracting from signature)
- $\text{Hash}(_)$ (hashing)

by factoring out the equations $\text{Dec}_{K^{-1}}(\{E\}_k) = E$ and $\text{Ext}_K(\text{Sign}_{K^{-1}}(E)) = E$ (for $K \in \text{Keys}$).

Adversary Model

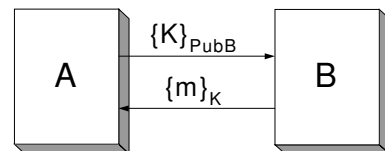


Adversary: Simulation



Adversary knowledge: $\boxed{k^{-1}, y, x, \{z\}_k, z}$ • $\forall e, k. \text{Dec}_{k^{-1}}(\{e\}_k) = e$

Example: Hybrid Scheme



Secrecy of m **not preserved** against attacker who can **delete** and **insert** messages.

Security of m **preserved** against attacker who can **listen**, but not alter messages.

Security Analysis in First-order Logic

Idea: **approximate** set of possible **data values** flowing through system **from above**.

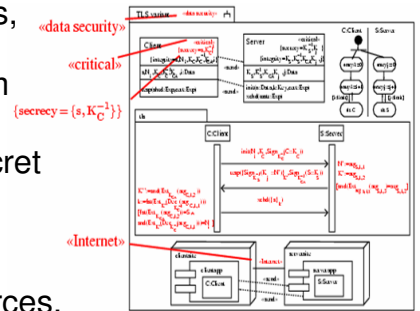
Predicate *knows(E)* meaning that the adversary may get to know *E* during the execution of the protocol.

For any secret *s*, check whether can derive *knows(s)* using automated theorem prover.

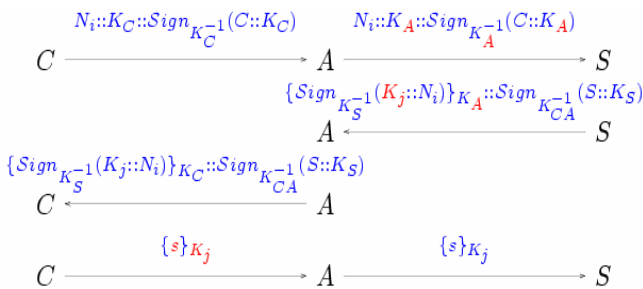
Example: Proposed Variant of TLS (SSL)

Apostolopoulos, Peris, Saha; IEEE Infocom 1999.

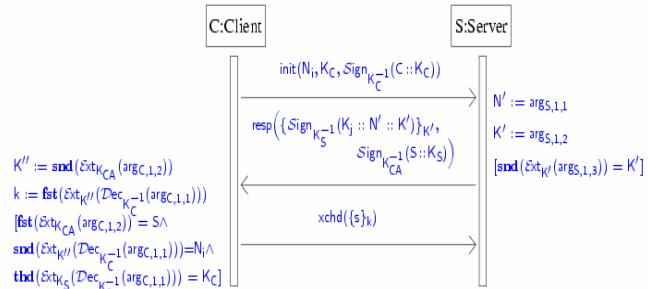
Goal: send secret protected by session key using fewer server resources.



Man-in-the-Middle Attack

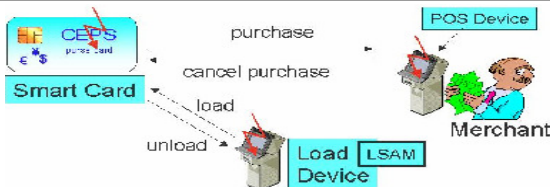


The Fix



e-Setheo: *knows(s)* not derivable. Thus secure.

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**).



Load Protocol

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

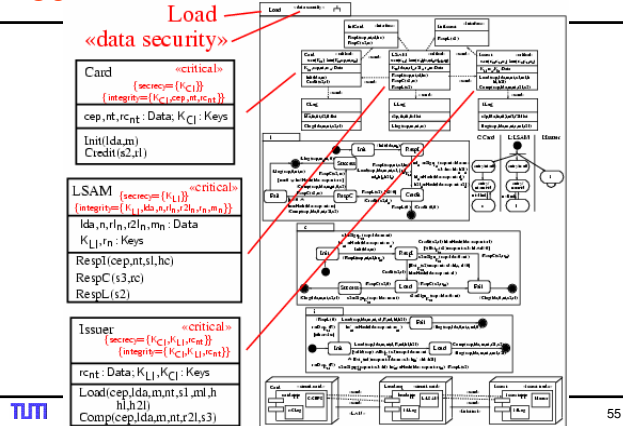
Load value onto card using cash at **load device**.

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



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.

Load Acquirer Security

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

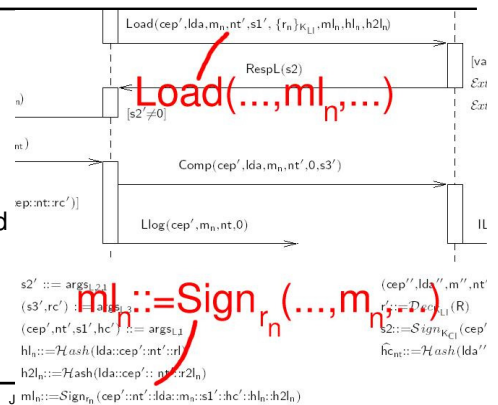
Then after execution either of following hold:

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

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

Flaw

ml_n : „Proof“ for bank that load machine received money. But: r_n shared between bank and load machine.



Some Further Applications

- Biometric Authentication System
- Analysis of multi-layer security protocol for web application of German bank
- Analysis of SAP access control configurations for German bank
- Telematic automobile emergency application of German car company
- Electronic signature architecture of German insurance company
- Electronic purse for Oktoberfest

Beyond Specification Analysis

Model-based test generation.

Configuration analysis.

- Analyze permission data using Prolog (e.g. SAP R/3)
- Analyze firewall configurations using model-checkers

Source-code analysis (C).

Ongoing Work, Open Problems

Ongoing work on most of the above issues:

- Security properties: E.g. composability
- Crypto verification: crypto-specific equations
- Tools: E.g. Extensibility for self-defined stereotypes
- Source-code analysis: extract Dolev-Yao model
- Application domains: Mobility

Conclusions

Model-based Security Engineering using UMLsec:

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

Resources

Jan Jürjens, Secure Systems Development with UML, Springer 2004

PhD Course: May 2005,
Carlos III Univ. Madrid

Workshops: WITS05@POPL05, CSDUML05,
WITS06@ETAPS06, ...

More information (papers, slides, tool etc.):
<http://www.umlsec.org>
(user Participant, password lwasthere)

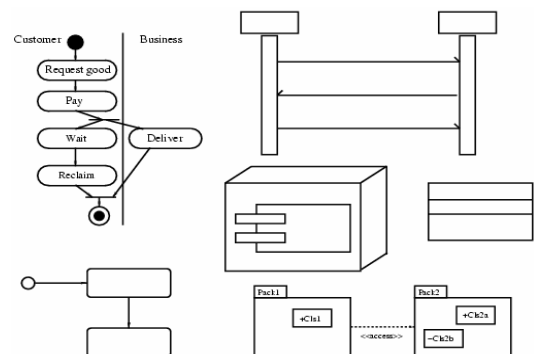


UML

Unified Modeling Language (UML):

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

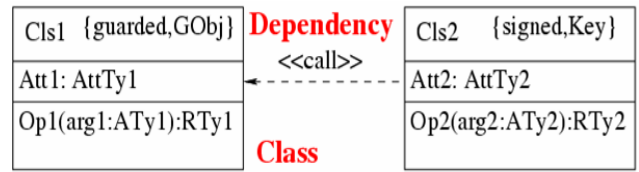
A Glimpse at UML



Used Fragment of UML

- Use case diagram:** discuss requirements of the system
 - Class diagram:** data structure of the system
 - Statechart diagram:** dynamic component behaviour
 - Activity diagram:** flow of control between components
 - Sequence diagram:** interaction by message exchange
 - Deployment diagram:** physical environment
 - Package/Subsystem:** collect diagrams for system part
- Current: UML 2.0 (released 2004)

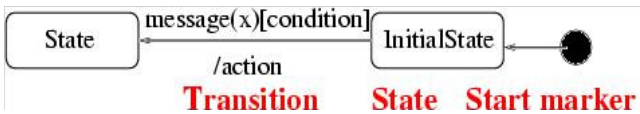
UML Run-through: Class diagram



Class structure of system.

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

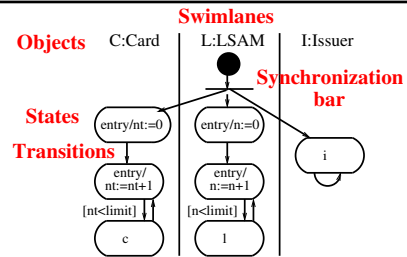
UML run-through: Statecharts



Dynamic behaviour of individual component.

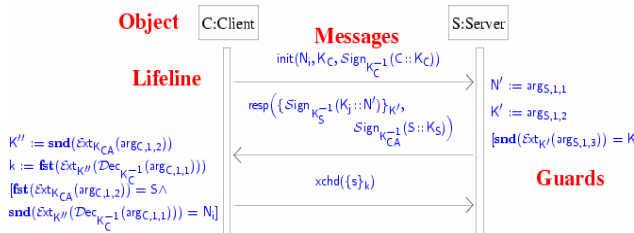
Input events cause state change and output actions.

UML Run-through: Activity Diagram



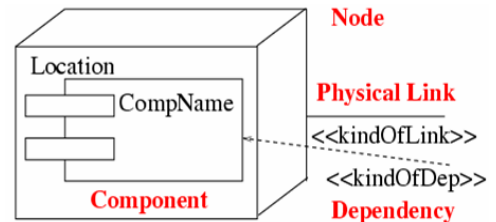
Specify the control flow between components within the system, at higher degree of abstraction than statecharts and sequence diagrams.

UML Run-through: Sequence Diagram



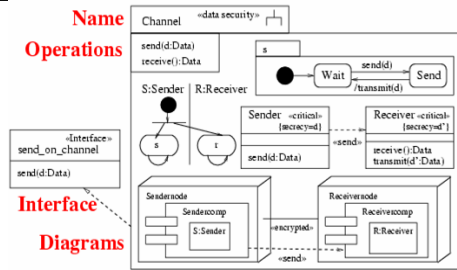
Describe interaction between objects or components via message exchange.

UML Run-through: Deployment Diagram



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

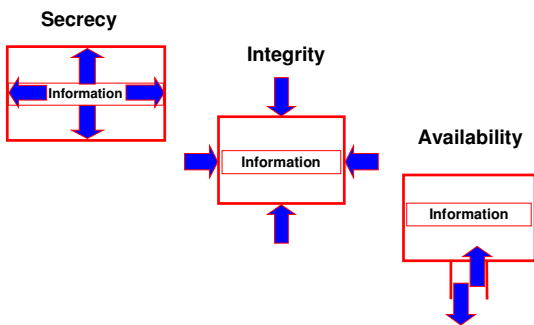
Stereotype: **specialize** model element using `<<label>>`.

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

Constraint: **refine** semantics of stereotyped element.

Profile: **gather** above information.

Basic Security Requirements I



Basic Security Requirements II

