Critical Systems Development with UML

Jan Jürjens
(contrib. UMLsec group@TUM)

Software & Systems Engineering
TU Munich, Germany

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

Critical Systems Development

High quality development of critical systems (dependable, security-critical, real-time,...) is difficult.

Many systems developed, deployed, used that do not satisfy their criticality requirements, sometimes with spectacular failures.

Quality vs. cost

Systems on which human life and commercial assets depend need careful development.

Systems operating under possible system failure or attack need to be free from weaknesses.

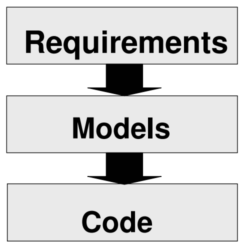
Correctness in conflict with cost.

Thorough methods of system design not used if too expensive.

Model-based Development

Goal: easen transition from human ideas to executed systems.

Increase quality with bounded time-to-market and cost.



Goal: Critical properties by design

Consider critical properties

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

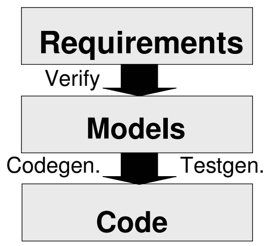
Critical design by model analysis.

Critical implementation by test generation.

Model-based Development

Combined strategy:

- Verify models against requirements
- Generate code from models where reasonable
- Write code and generate test-sequences otherwise.



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 (drawing specifications, simulation, …).

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 (analysis, …).

UML for CSD: Goals

Extensions for critical systems development.

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

The CSDUML profiles

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

Use associated constraints to evaluate specifications and indicate possible weaknesses.

Ensures that UML specification provides desired level of critical requirements.

Link to code via test-sequence generation.

This tutorial

Background knowledge on using UML for critical systems development.

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

Concentrate on safety and security.

Generalize to other application domains.

Roadmap

Prologue

UML

UMLsafe

UMLsec

Model-based Testing

Towards UML 2.0

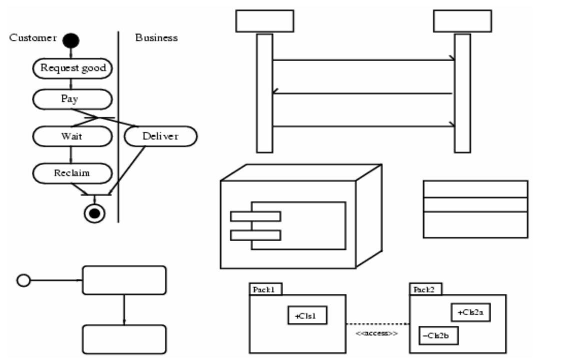
Tools

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

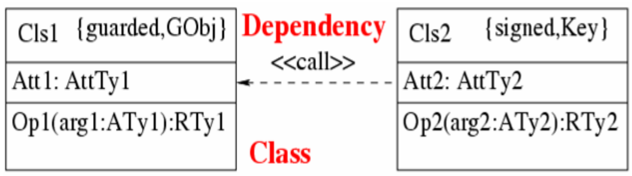


Used fragment of UML

Class diagram: data structure of the system
Statechart diagram: dynamic component behaviour
Activity diagram: flow of control between system components
Sequence diagram: interaction between components by message exchange
Deployment diagram: components in physical environment
Package/Subsystem: collect diagrams for system part

Current: UML 1.5 (released Mar 2003)

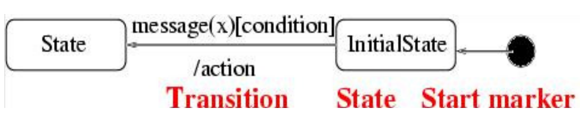
UML run-through: Class diagrams



Class structure of system.

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

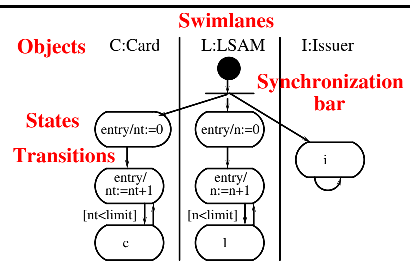
UML run-through: Statecharts



Dynamic behaviour of individual component.

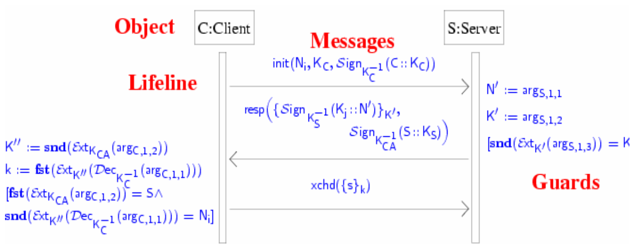
Input events cause state change and output actions.

UML run–through: Activity diagrams



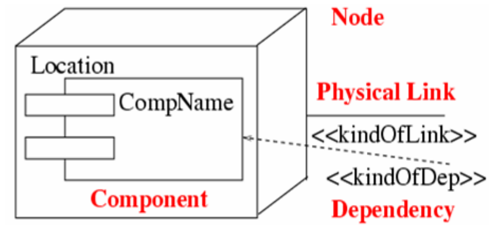
Specify the control flow between components within the system, at higher degree of abstraction than statecharts and sequence diagrams.

UML run-through: Sequence Diagrams



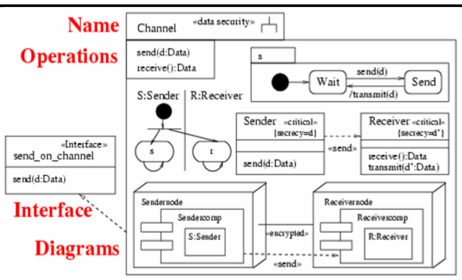
Describe interaction between objects or components via message exchange.

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
UMLsafe

UMLsec
Model-based Testing
Towards UML 2.0
Tools

Safety

Safety-critical systems: five failure condition categories: catastrophic, hazardous, major, minor, no effect.

Corresponding safety levels A - E (DO-178B standards in avionics).

Safety goals: via the maximum allowed failure rate. For high degree of safety, testing not sufficient (1 failure per 100,000 years).

Failures

Exchanged data may be

- delayed (and possibly reordered)
- lost
- corrupted.

Often, failures occur randomly (e.g. hardware).
Failure semantics examples:

- crash/performance: component may crash or exceed time limit, but partially correct.
- value: component may deliver incorrect values.

Fault-tolerance

Redundancy model determines which level of redundancy provided.
Goal: no hazards in presence of single-point failures.

On the following slides, for simplicity:

- focus on safety-critical systems (may in particular may to be reliable)
- focus on fault-tolerance aspects of safety

Embedded Systems

In particular, embedded software increasingly used in safety-critical systems (flexibility):

- Automotive
- Avionics
- Aeronautics
- Robotics, Telemedicine
- …

Our treatment of safety-critical systems also applies to embedded systems.

Failure semantics modelling

For redundancy model *R*, stereotype $s \in$ {≪crash/performance≫, ≪value≫}, have set $\mathrm{Failures}_R(s) \subseteq$ {delay(*t*), loss(*p*), corrupt(*q*)}, with interpretation:

- *t*: expected maximum time delay,
- *p*: probability that value not delivered within *t*,
- *q*: probability that value delivered in time corrupted

(in each case incorporating redundancy).
Or use ≪risk≫ stereotype with {failure} tag.

Example

Suppose redundancy model *R* uses controller with redundancy *3* and the fastest result. Then could take:

- delay(*t*): *t* delay of fastest controller,
- loss(*p*): *p* probability that fastest result not delivered within *t*,
- corrupt(*q*): *q* probability that fastest result is corrupted

(each wrt. the given failure semantics).

≪guarantee≫

Describe guarantees required from communication dependencies resp. system components.

Tags: {goal} with value subset of {immediate(*t*), eventual(*p*), correct(*q*)}, where

- *t*: expected maximum time delay,
- *p*: probability that value is delivered within *t*,
- *q*: probability that value delivered in time not corrupted.

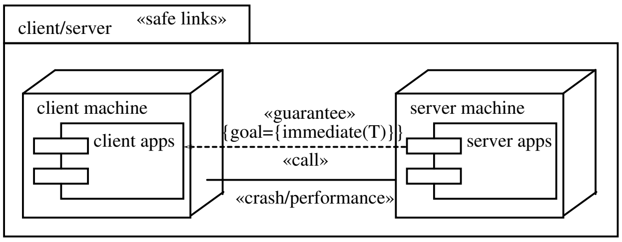
≪safe links≫

Physical layer should meet safety requirements on communication given redundancy model *R*.

Constraint: For dependency *d* stereotyped ≪guarantee≫ and each corresponding communication link *l* with stereotype *s*:

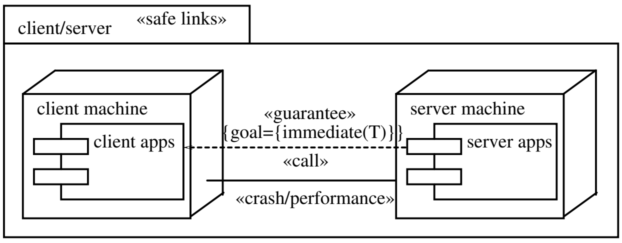
- if {goal} has immediate(*t*) as value then delay(*t'*) $\in$ $\text{Failures}_R(s)$ implies $t' \leq t$,
- if {goal} has eventual(*p*) as value then loss(*p'*) $\in$ $\text{Failures}_R(s)$ implies $p' \leq 1-p$, and
- if {goal} has correct(*q*) as value then corruption(*q'*) $\in$ $\text{Failures}_R(s)$ implies $q' \leq 1-q$.

Example ≪safe links≫



Given redundancy model none, when is ≪safe links≫ fulfilled ?

Example ≪safe links≫



Given redundancy model none, ≪safe links≫ fulfilled iff T $\geq$ expected delay according to Failures_{none}(≪crash/performance≫).

≪safe dependency≫

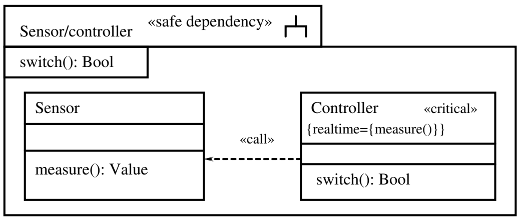
Communication dependencies should respect safety requirements on ≪critical≫ data.

For each safety level {*l*} for ≪critical≫ data, have goals(*l*)$\subseteq${immediate(*t*), eventual(*p*), correct(*q*)}.

Constraint: for each dependency *d* from *C* to *D* stereotyped ≪guarantee≫:

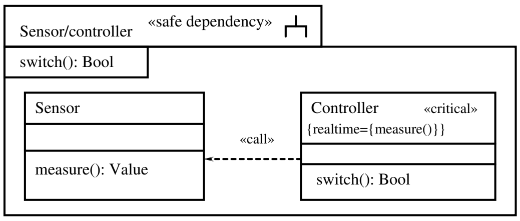
- Goals on data in *D* same as those in *C*.
- Goals on data in *C* that also appears in *D* met by guarantees of *d*.

Example ≪safe dependency≫



Assuming immediate(*t*) $\in$ goals(realtime), ≪safe dependency≫ provided ?

Example ≪safe dependency≫



Assuming immediate(*t*) $\in$ goals(realtime), violates ≪safe dependency≫, since Sensor and dependency do not provide realtime goal immediate(*t*) for measure() required by Controller.

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) Failure model applied as defined above.

Failure models

lq^l_n: messages on link l delayed further n time units.
p^h_n: probability of failure at n^{th} iteration in history h.
For link l stereotyped s where loss(p)$\in$Failures$_R$(s),

- history may give $lq^l_0 := \emptyset$; then append p to $(p^h_n)_{n\in\mathcal{N}}$,
- or no change, then append $1-p$.

For link l stereotyped s where corruption(q)$\in$Failures$_R$(s),

- history may give $lq^l_0 := \{\blacksquare\}$; then append q,
- or no change; append $1-q$.

For link l stereotyped s with delay(t)$\in$Failures$_R$(s), and $lq^l_0 \neq \emptyset$, history may give $lq^l_n := lq^l_0$ for $n \leq t$; append $1/t$.
Then for each n, $lq^l_n := lq^l_{n+1}$.

≪safe behaviour≫

Ensures that system behavior in presence of failure model provides required safety {goals}:

For any execution trace h, any transmission of a value along a communication dependency stereotyped ≪guarantee≫, the following constraints should hold, given the safety goal:

- eventual(p): With probability at least p, …
- immediate(t): … every value is delivered after at most t time steps.
- correct(q): Probability that a delivered value is corrupted during transmission is at most $1-q$.

≪containment≫

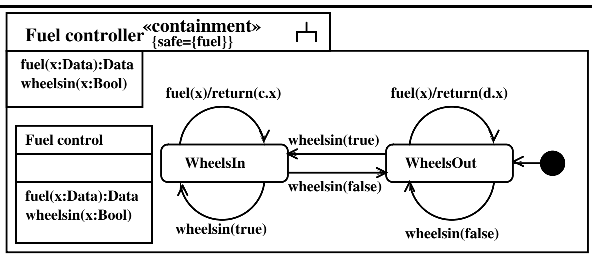
Prevent indirect corruption of data.

Constraint:

Value of any data element d may only be influenced by data whose requirements attached to ≪critical≫ imply those of d.

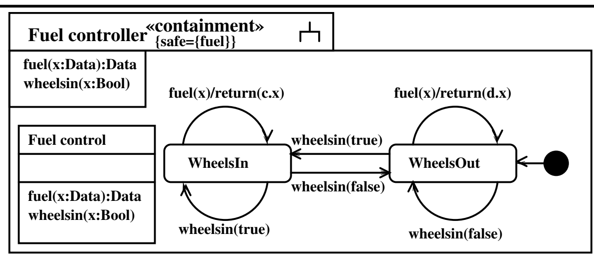
Make precise by referring to execution semantics (view of history associated with safety level).

Example ≪containment≫



Containment satisfied ?

Example ≪containment≫



Violates containment because a {safe} value depends on un{safe} value.

Can check this mechanically.

Other checks

Have other consistency checks such as

- Is the software's response to out-of-range values specified for every input ?
- If input arrives when it shouldn't, is a response specified ?

…and other safety checks from the literature.

Roadmap

A Need for Security

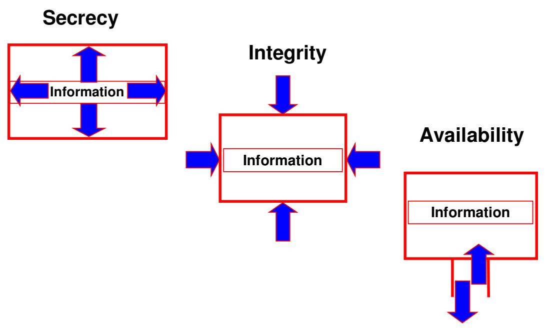
Society and economies rely on computer networks for communication, finance, energy distribution, transportation...

Attacks threaten economical and physical integrity of people and organizations.

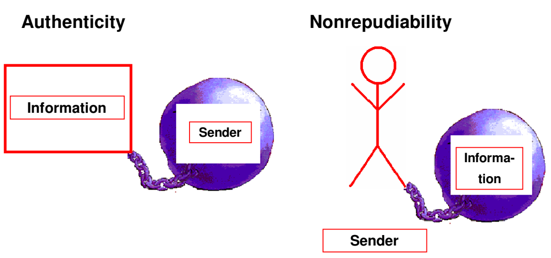
Interconnected systems can be attacked anonymously and from a safe distance.

Networked computers need to be secure.

Basic Security Requirements I



Basic Security Requirements II



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

„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“ (Lampson, Needham).

Difficulties

Exploit information spreads quickly.

No feedback on delivered security from customers.

Previous approaches

„Penetrate-and-patch“: unsatisfactory.

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

Traditional formal methods: expensive.

- training people
- constructing formal specifications.

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.

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≫, …

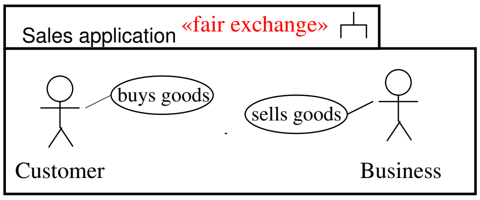
Kinds of communication links resp. system nodes.

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

Default attacker:

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

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.

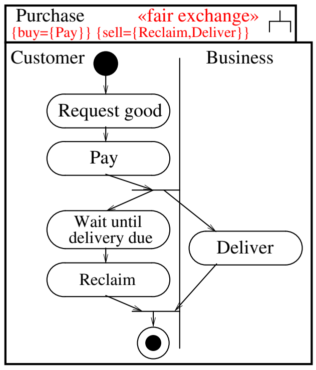
Constraint: after a {buy} state in activity diagram is reached, eventually reach {sell} state.

(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.



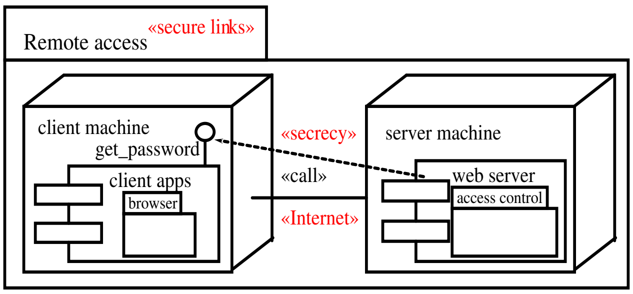
≪secure links≫

Ensures that physical layer meets security requirements on communication.

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

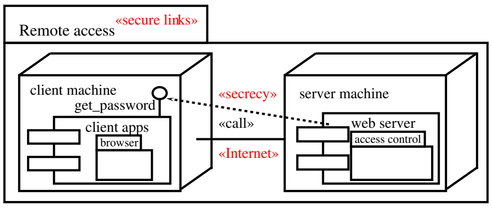
- if *s* = ≪secrecy≫: have read $\notin \text{Threats}_A(t)$.
- if *s* = ≪integrity≫: have insert $\notin \text{Threats}_A(t)$.

Example ≪secure links≫



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

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.

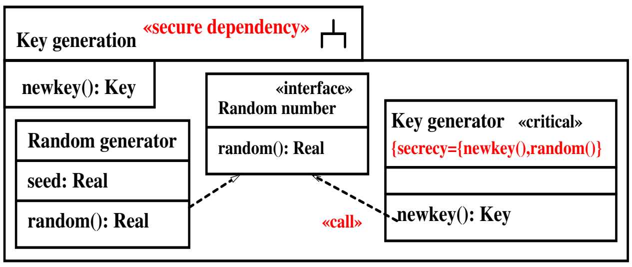
≪secure dependency≫

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}):

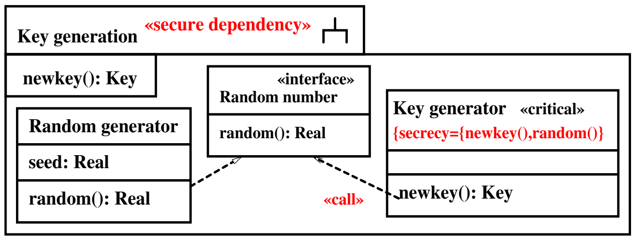
- Msg in *D* is {secrecy} in *C* if and only if also in *D*.
- If msg in *D* is {secrecy} in *C*, dependency stereotyped ≪secrecy≫.

Example ≪secure dependency≫



≪secure dependency≫ provided ?

Example ≪secure dependency≫



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

≪no down–flow≫

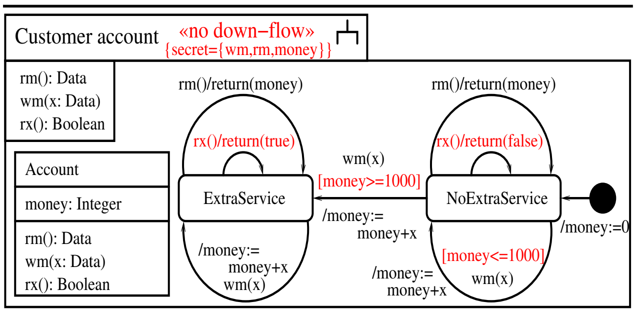
Enforce secure information flow.

Constraint:

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

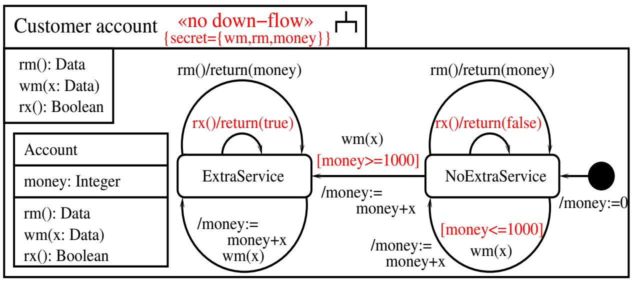
Formalize by referring to formal behavioural semantics.

Example ≪no down-flow≫



≪no down–flow≫ provided ?

Example ≪no down-flow≫



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

≪data security≫

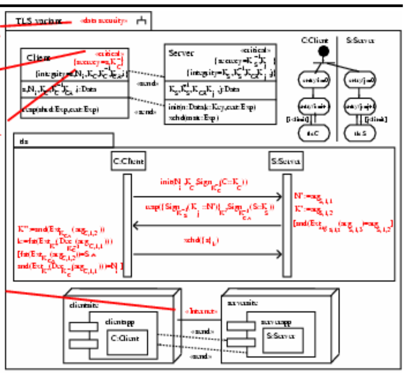
Security requirements of data marked ≪critical≫ enforced against threat scenario from deployment diagram.

Constraints:

Secrecy of {secrecy} data preserved.

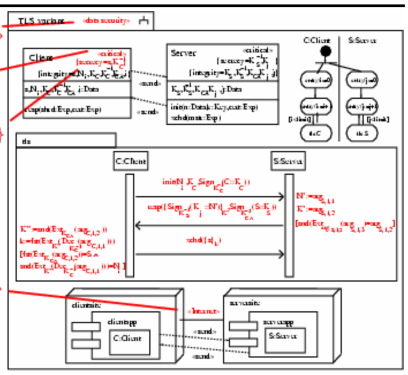
Integrity of {integrity} data preserved.

Example ≪data security≫



Variant of TLS (INFOCOM`99). ≪data security≫ against default adversary provided ?

Example ≪data security≫



Variant of TLS (INFOCOM`99). Violates {secrecy} of s against default adversary.

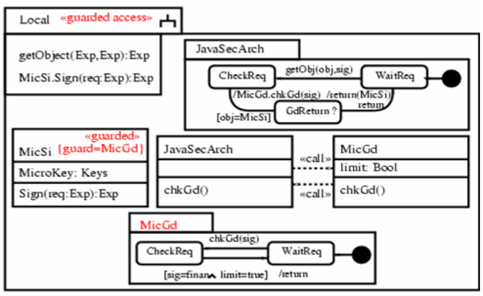
≪guarded access≫

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.

Concepts covered by UMLsec

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.

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.

Expressions

Exp: term algebra generated by Var ∪ Keys ∪ Data and

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

by factoring out the equations $Dec_{K^{-1}}(\{E\}_k) = E$ and $Ext_K(Sign_{K^{-1}}(E)) = E$ (for K∈ Keys).

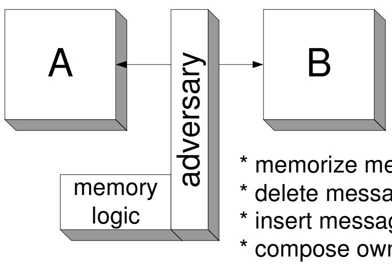
Abstract adversary

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

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

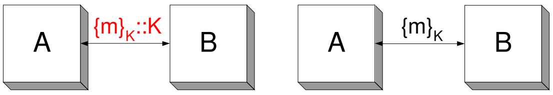
M kept secret by *S* if *M* never output in clear (Dolev, Yao 1982).

Abstract adversary

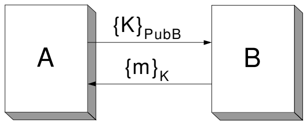


* memorize message
* delete message
* insert message
* compose own message
* use cryptographic primitives

Example: secrecy



Example: secrecy



- Security of *m* is not preserved against an attacker who can delete and insert messages
- Security of *m* is preserved against an attacker who can listen, but not alter the link

Example: secrecy

Component sending $\{m\}_K$::$K \in$ 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.

➔

Abstract adversary (alternative)

Define: Suppose K_A^{n+1} is the Exp-subalgebra generated by K_A^n and the expressions received after *n+1*st iteration of the protocol.

Theorem.
S keeps secrecy of *M* against attackers of type *A* if there is no *n* with $M \in K_A^n$.

Roadmap

Prologue
UML
UMLsafe

UMLsec
Model-based Testing
Towards UML 2.0
Tools

Tool-support: Test-generation

Two complementary strategies:

- Conformance testing
- Testing for criticality requirements

Conformance testing

Classical approach in model-based test-generation (much literature).

Can be superfluous when using code-generation [except to check your code-generator, but probably once and for all]

Works independently of criticality requirements.

Conformance testing: Problems

- Complete test-coverage usually infeasible. Need to somehow select test-cases.
- Can only test code against what is contained in the behavioral model. Usually, model is more abstract than code. So may have „blind spots“ in the code.

For both reasons, may miss critical test-cases.

Criticality testing

Shortcoming of classical model-based test-generation (conformance testing) motivates „criticality testing“ (e.g., papers by Jürjens, Wimmel at PSI'01, ASE'01, ICFEM'02).

Goal: model-based test-generation adequate for (security-, safety-) critical systems.

Criticality testing: Strategies

Strategies:

- Ensure test-case selection from behavioral models does not miss critical cases: Select according to information on criticality („internal“ criticality testing).
- Test code against possible environment interaction generated from external parts of the model (e.g. deployment diagram with information on physical environment).

Internal Criticality Testing

Need behavioral semantics of used specification language (precise enough to be understood by a tool).

Here: semantics for simplified fragment of UML in „pseudo-code“ (ASMs).

Select test-cases according to criticality annotations in the class diagrams.

Test-cases: critical selections of intended behavior of the system.

External Criticality Testing

Generate test-sequences representing the environment behaviour from the criticality information in the deployment diagrams.

Roadmap

Prologue
UML
UMLsafe

UMLsec
Model-based Testing
Towards UML 2.0
Tools

Some new concepts in UML 2.0

UML extended with concepts from UML RT (Selic, Rumbaugh 1998).

Focus on software architecture.

New: capsules, ports, connectors.

Capsules, ports, connectors

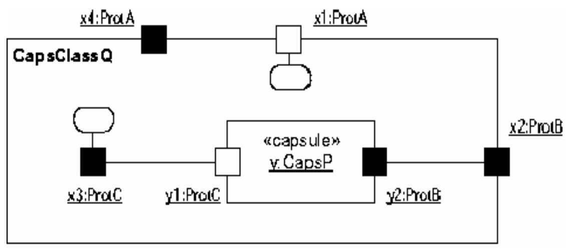
Capsules: architectural objects interacting through signal-based boundary objects (ports).

Port: object implementing interface of capsule. Associated with a protocol defining flow of information.

Connector: abstract signal-based communication channels between ports.

Functionality of capsule realized by associated state machine.

Example



From Selic, Rumbaugh 1998.

Roadmap

Prologue
UML
UMLsafe

UMLsec
Model-based Testing
Towards UML 2.0
Tools

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/safe: include adversary/failure model arising from threat scenario in deployment diagram.

Use Abstract State Machines (pseudo-code).

Tool-supported analysis

Choose drawing tool for UML specifications

Analyze specifications via XMI (XML Metadata Interchange)

skip compar.

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

MDR: MetaData Repository, Netbeans library (www.netbeans.org)

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.

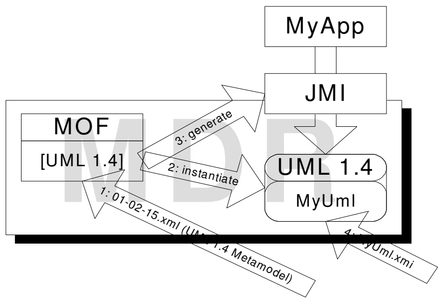
MDR Standards

- 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

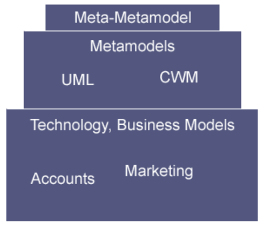
MDR Services

- Load and Store a MOF Metamodel (XMI format)
- Instantiate and Populate a Metamodel (XMI format)
- Generate a JMI (Java Metadata Interface) Definition for a Metamodel
- Access a Metamodel Instance

UML Processing



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

Framework for CSDUML tools: viki

Implements functionality
- MDR wrapper
- File handling
- Properties management
- Tool management

Exposes interfaces
- IVikiFramework
- IMdrWrapper
- IAppSettings

viki Tool

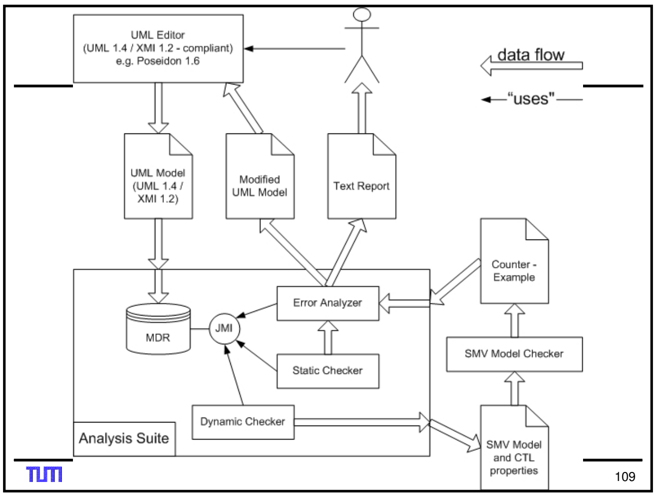
- Works in GUI and/or Text mode
- Implements interfaces
  - IVikiToolCommandLine
    - Text output only
  - IVikiToolGui
    - Output to JPanel + menu, buttons, etc
- Exposes set of commands
  - Automatically imported by the framework

Implementing tools

Exposes a set of commands.

Has its internal state (preserved between command calls).

Every single command is not interactive (read user input only at the beginning).



Tool

Currently implementing web-interface (see http://www4.in.tum.de/~umlsec and demo after this presentation).

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

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.

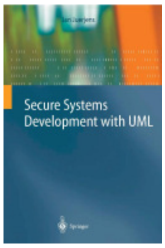
Some resources

Book: Jan Jürjens, Secure Systems Development with UML, Springer-Verlag, 2004

Follow-on Tutorials: Oct: LADC (Sao Paulo); Nov: WWW/Internet (Algarve), FMOODS/DAIS (Paris), ICSTEST-E (Bilbao) …

Special SoSyM issue on Critical Systems Development with UML

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



More information (slides, tool 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 !

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