Secure Java Development with UML

Jan Jürjens

Software & Systems Engineering, Informatics, TU Munich
Computing Laboratory, University of Oxford

juerjens@in.tum.de

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

Motivation

Security increasingly important, but difficult to achieve
– and developers often lack background in security.

Can't use security mechanisms "blindly":
Security compromised most often not by
breaking mechanisms, but by circumventing them.

Use formal core of UML (industry standard in OO-modelling)
to enforce correct deployment of security mechanisms.

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.

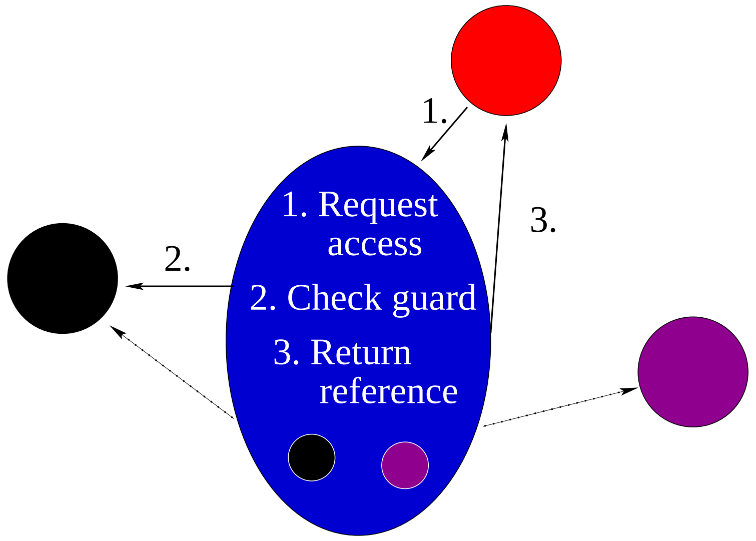
Similarly, need confidentiality.

A SealedObject is an encrypted object.

Guarded Objects

Class java.security.GuardedObject: objects that protect access to other objects

- access controlled by getObject method
- invokes checkGuard method on the java.security.Guard object that is guarding access
- If access allowed: return reference
  Otherwise: throw 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).

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

Solution

Use formal core of Unified Modeling Language (UML) to check dynamical behaviour against expressed security policies on specification-level.

Find correct required security policies in advance (not through "penetrate-and-fix").

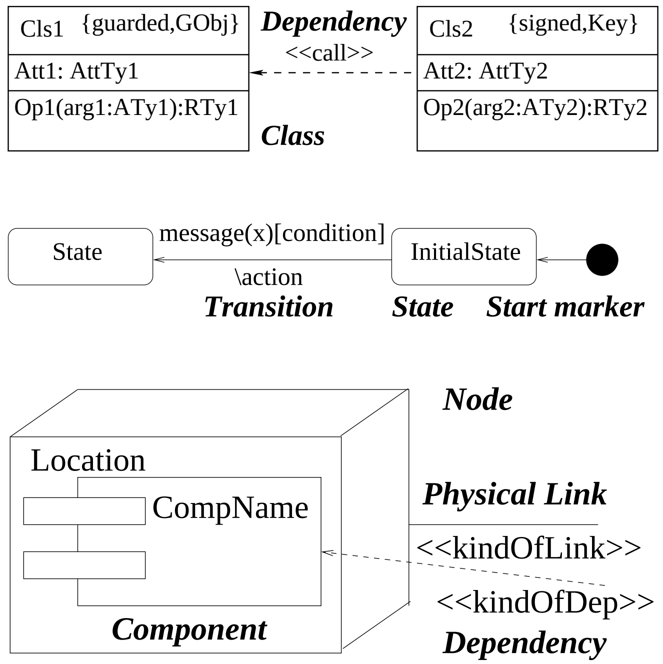
- fewer vulnerabilities
- reduced cost (development, maintenance)

UML

UML: standard modeling language for object-oriented analysis and design.

Different diagrams for different views on system.

Here: simplified fragment: class diagrams, statechart diagrams, deployment diagrams.



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.

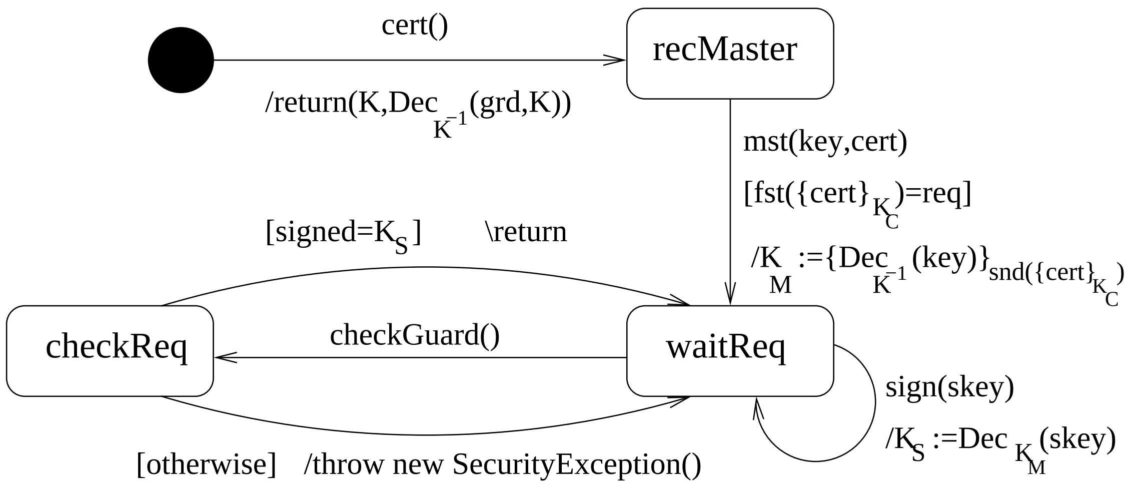
Reasoning

Definition. Statechart preserves secrecy of K if (informally): exists no adversary without prior knowledge of K such that joint execution may eventually output K.

Theorem. Suppose access to resource according to Guard object specifications granted only to objects signed with K. Suppose all components preserve secrecy of K. Then only objects signed with K are granted access.

Relies on formal semantics.

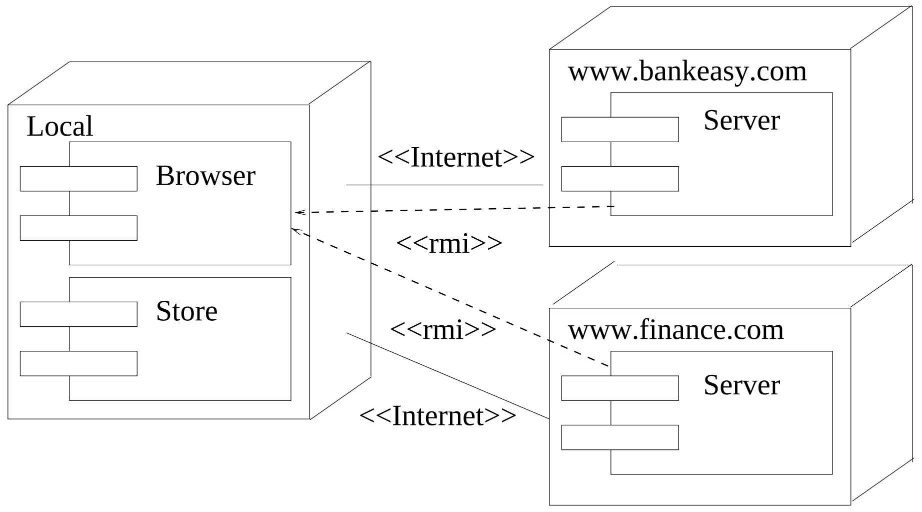
Example: Access control rule


cert()
/return(K,Dec$_{K^{-1}}$(grd,K))
recMaster
mst(key,cert)
[fst({cert}$_{K_C}$)=req]
/K$_M$:={Dec$_{K^{-1}}$(key)}$_{snd(\{cert\}_{K_C})}$
[signed=K$_S$]
\return
checkReq
checkGuard()
waitReq
sign(skey)
/K$_S$:=Dec$_{K_M}$(skey)
[otherwise]
/throw new SecurityException()

May submit keys K_S protected by K_M. Objects signed with K_S granted access to guarded object.
Flaw: adversary A can find out K_M and make grd accept a key K_S chosen by A.

Can exhibit flaws like this when trying to show formally that assumptions of the above theorem are fulfilled.

Example: Financial Application

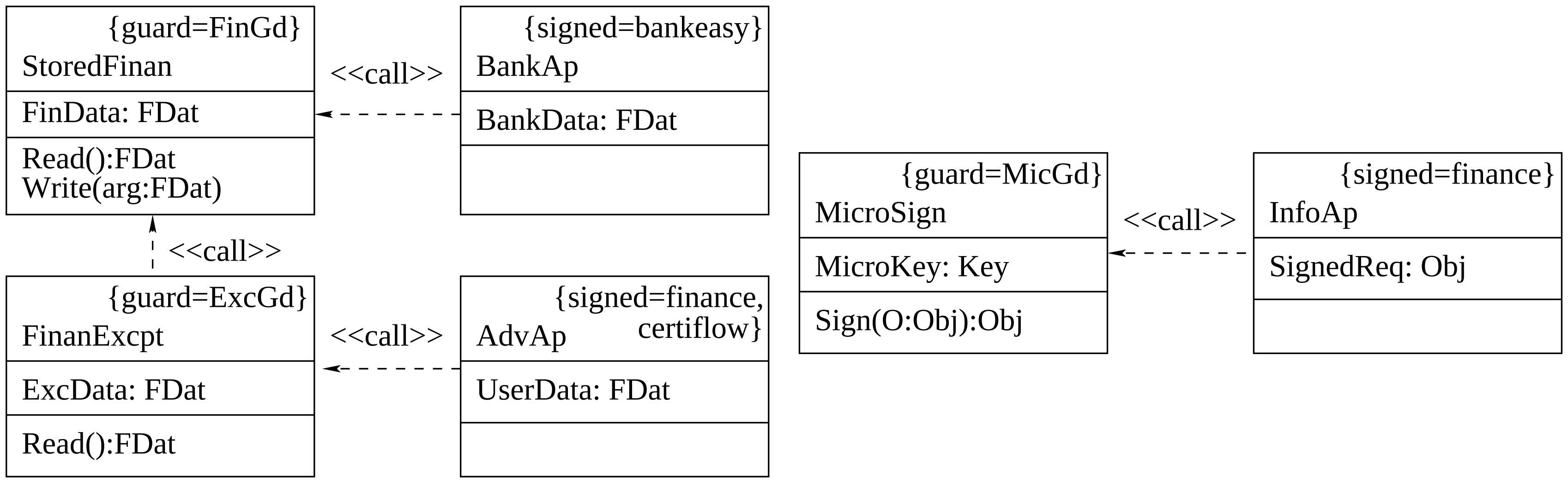


Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain privileges (step 1).

- 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.

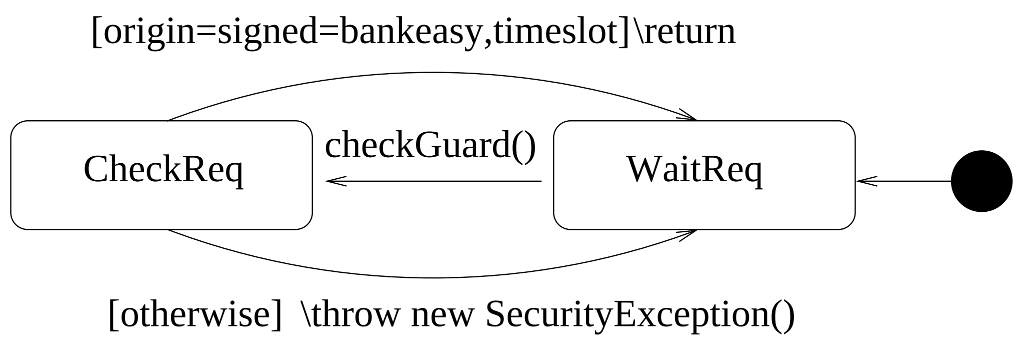
Financial Application: Class diagram

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

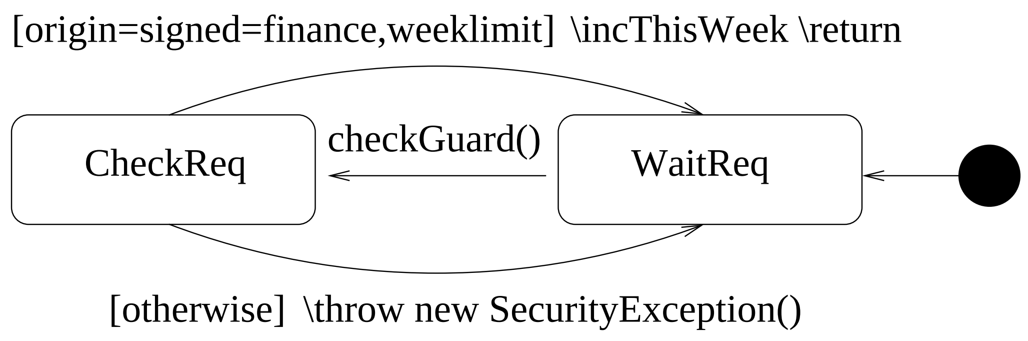


Financial Application: Guard objects (step 2)

timeslot true between 1pm and 2pm.


[origin=signed=bankeasy,timeslot]\return
CheckReq
checkGuard()
WaitReq
[otherwise] \throw new SecurityException()

weeklimit true until access granted five times; incThisWeek increments counter.


[origin=signed=finance,weeklimit] \incThisWeek \return
CheckReq
checkGuard()
WaitReq
[otherwise] \throw new SecurityException()

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.

Proofs. Use formal semantics and above theorem.

Mobile objects sufficiently protected (step 5), since objects sent over Internet are signed and sealed.

Related Work

Li Gong: "Inside Java 2 Platform Security", 1999

Kassab et al. 98, Wallach et al. 98, Karjoth 00:
formal reference model for Java 2 access control.

Hauswirth et al. 00: higher-level abstractions
for Java security policies

UMLsec (FASE '01), applications to electronic purses
(IFIP SEC 01), CORBA (EPS 2002)

Conclusion

Use formal core of UML to specify and reason about access control in Java-based systems.

- Provides support for correct use of relatively complicated mechanisms.
- Relatively feasible: developers may already know UML; UML allows high level of abstraction.

More general approach for secure systems design using UML.

Future Work

- Extend to doPrivileged, JAAS etc.
- Tool support (verification, testing).
- Automatic generation of minimal permission sets.