Exams: July 17, August 22, (and probably September)

watch out for
“Systems Programming Lab” in Fall !!!
MODELING DISTRIBUTED SYSTEMS

HERMANN HÄRTIG, DISTRIBUTED OPERATING SYSTEMS, SS2018
MODELS IN GENERAL

- abstract from details
- concentrate on functionality, properties, ... that are considered important for a specific system/application
- use model to analyze, prove, predict, ... system properties and to establish fundamental insights
- models in engineering disciplines very common, increasingly in CS as well
- we'll see many models in “Real-Time Systems” class
THE GENERAL IDEA

Reasoning:
- Common sense
- Formal Verification
- Careful Inspection
- Mathematics

Model → Reasoning → OK?
Reasoning:

- Common sense
- Formal Verification
- Careful Inspection
- Mathematics

“Refinement”:
- Abstraction
## Model Examples in General

<table>
<thead>
<tr>
<th>Model</th>
<th>Objective/Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure Trees</td>
<td>are all failure combinations taken into account</td>
</tr>
<tr>
<td>statics models</td>
<td>does a house eventually fall down</td>
</tr>
<tr>
<td></td>
<td>what kind of vehicles on a bridge</td>
</tr>
<tr>
<td>control laws</td>
<td>stability of controllers</td>
</tr>
<tr>
<td>Ohm’s Law</td>
<td>behavior of circuits</td>
</tr>
</tbody>
</table>
I = V / R
<table>
<thead>
<tr>
<th>Model</th>
<th>Objective/Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing Machine</td>
<td>Decidability</td>
</tr>
<tr>
<td>Amdahl’s Law</td>
<td>Scalability</td>
</tr>
<tr>
<td>Logic</td>
<td>Correctness, Precision, …</td>
</tr>
<tr>
<td>Real-Time “tasks”</td>
<td>can all timing requirements be met</td>
</tr>
<tr>
<td>Byzantine Agreement</td>
<td>Consensus</td>
</tr>
<tr>
<td>Two Army</td>
<td>Consensus</td>
</tr>
</tbody>
</table>
Objective of lecture:
understand the power of models and the need for their careful understanding

Intuition, No proofs
THIS LECTURE’S QUESTIONS

- Q1: Is it possible to build arbitrarily reliable Systems out of unreliable components?
- Q2: Can we achieve consensus in the presence of faults (consensus: all non-faulty components agree on action)?
- Q3: Is there an algorithm to determine for a system with a given setting of access control permissions, whether or not a Subject A can obtain a right on Object B?

2 Models per Question!

All questions/answers/models -> published 1956 - 1982 !!!
Q1: Can we build arbitrarily reliable Systems out of unreliable components?

- How to build reliable systems from less reliable components
- Fault(Error, Failure, Fault, ....) terminology in this lecture synonymously used for “something goes wrong” (more precise definitions and types of faults in SE)
DEFINITIONS

Reliability:
- \( R(t) \): probability for a system to survive time \( t \)

Availability:
- \( A \): fraction of time a system works
INGREDIENTS

- Fault detection and confinement
- Recovery
- Repair
  - Redundancy
    - Information
    - time
    - structural
    - functional
John v. Neumann
Voter: single point of failure

Can we do better
→ distributed solutions?
Parallel-Serial-Systems
(Pfitzmann/Härtig 1982)
Parallel-Serial-Systems
(Pfitzmann/Härtig 1982)
Parallel-Serial-Systems
(Pfitzmann/Härtig 1982)
Parallel-Serial-Systems
(Pfitzmann/Härtig 1982)
Serial-Systems

Each component must work for the whole system to work.
Parallel-Systems

One component must work for the whole system to work.
Each component must fail for the whole system to fail.

\[ R_{\text{whole}} = 1 - \prod_{i=1}^{m} (1 - R_i) \]
Q1/MODEL1: ABSTRACT MODEL

Serial-Parallel-Systems

\[ R_{\text{whole}} = 1 - \prod_{j=1}^{m} \left( 1 - \prod_{i=1}^{n} R_{i,j} \right) \]
Parallel-Serial-Systems

(Pfitzmann/Härtig 1982)
Fault Model

„Computer-Bus-Connector“ can fail such that Computer and/or Bus also fail

=› conceptual separation of components into

  Computer, Bus: can fail per se

  CC: Computer-Connector fault also breaks the Computer

  BC: Bus-Connector fault also breaks Bus
Q1/MODEL1: CONCRETE MODEL

1 Buses

1 Computers

Bus 1

Computer 1

BC 1,1

CC 1,1

1 Buses

1 Computers
Q1/MODEL1: CONCRETE MODEL

1 Buses
2 Computers

Computer 1
CC1,1
BC 1,1
BC 2,1

Computer 2
CC2,1

Bus 1
Q1/MODEL1: CONCRETE MODEL

1 Buses
N Computers
Q1/MODEL1: CONCRETE MODEL

M Buses

1 Computers

Bus 1

Computer 1

Bus m

BC 1,1

CC 1,1

CC 1,2

BC 1,m

CC 1,m
Q1/MODEL1: CONCRETE MODEL

M Buses
N Computers

Computer 1
CC1,1
CC1,2
CC1,m

Computer 2
CC2,1
CC2,2
CC2,m

Computer n
CCn,1
CCn,2
CCn,m

Bus 1
BC1,1
BC2,1
BCn,1

Bus m
BC1,m
BC2,m
BCn,m
Q1/MODEL1: CONCRETE MODEL FOR N,M

\[ R_{\text{whole}}(n, m) = \left( 1 - \left( 1 - R_{\text{Bus}} \cdot R_{BC}^n \right)^m \right) \cdot \left( 1 - \left( 1 - R_{\text{Computer}} \cdot R_{CC}^m \right)^n \right) \]

then: \( R_{CC}, R_{BC} < 1 \), \( \lim_{n, m \to \infty} R(n, m) \)
System built of Synapses (John von Neumann, 1956)

Computation and Fault Model:
- Synapses deliver "0" or "1"
- Synapses deliver with $R > 0.5$:
  - with probability $R$ correct result
  - with $(1-R)$ wrong result

Then we can build systems that deliver correct result for any (arbitrarily high) probability $R$
Q2: Can we achieve consensus in the presence of faults all non-faulty components agree on action?

- all correctly working units agree on result/action
- agreement non trivial (based on exchange of messages)
Q2/MODEL 1: "2 ARMY PROBLEM"

- p,q processes
  - communicate using messages
  - messages can get lost
  - no upper time for message delivery known
  - do not crash, do not cheat

- p,q to agree on action (e.g. attack, retreat, ...)

- how many messages needed?

- first mentioned: Jim Gray 1978
Q2/MODEL 1: “2 ARMY PROBLEM”

Result: there is no protocol with finite messages

Prove by contradiction:

- assume there are finite protocols (mp--> q, mq --> p)*
- choose the shortest protocol MP,
- last message MX: mp --> q or mq --> p
- MX can get lost
- => must not be relied upon => can be omitted
- => MP not the shortest protocol.
- => no finite protocol
n processes, f traitors, n-f loyals

- communicate by reliable and timely messages (synchronous messages)
- traitors lie, also cheat on forwarding messages
- try to confuse loyals
Goal:

- loyals try to agree on non-trivial action (attack, retreat)
- non-trivial more specific:
  - one process is commander
  - if commander is loyal and gives an order, loyals follow the order otherwise loyals agree on arbitrary action
Q2/Model 2: “Byzantine Agreement”

3 Processes: 1 traitor, 2 loyals
Q2/Model 2: “Byzantine Agreement”

3 Processes: 1 traitor, 2 loyals

=> 3 processes not sufficient to tolerate 1 traitor
Q2/MODEL 2: “BYZANTINE AGREEMENT”

4 Processes

- Commander
- Lieutenant 1: He said: attack
- Lieutenant: He said: attack
- Lieutenant: He said: retreat

4 Processes: Commander, Lieutenant 1, Lieutenant, Lieutenant.
4 Processes

all lieutenant receive x, y, z => can decide

**General result:** 3 f + 1 processes needed to tolerate f traitors
Q3: Is there an algorithm to determine for a system with a given setting of access control permissions, whether or not a Subject A can obtain a right on Object B?
Q3: Is there an algorithm to determine for a system with a given setting of access control permissions, whether or not a Subject A can obtain a right on Object B?

Given a System of Entities (“Objects”) acting as Subjects and/or Objects

- with clearly-defined limited access rights among themselves
- can we achieve clearly-defined Security Objectives?
Definition and Example of “higher-Level” Security Policies (Security Policy Models) (Bell La Padula, Chinese Wall)

- Mechanisms to express/set clearly-defined access rights: Access Control Matrix, ACL, and Capabilities
- Q3 “formalized” in 2 Models: “ACM-based” & “Take Grant”
- Decidable?
- No proofs (in 2018)
THE GENERAL APPROACH

“Reasoning”:
- Common sense
- Formal Verification
- Careful Inspection
- Mathematics

“Refinement”:
- Abstraction
- Implementation

TU Dresden: Hermann Härtig, Marcus Völp
Modeling Distributed Systems, SS 2018

43
“Reasoning”:
- Common sense
- Formal Verification
- Careful Inspection
- Mathematics
- “Common Criteria Assurance”

“Refinement”:
- Abstraction
- Implementation
Definition: Policy

Examples:
Higher-Level Policies (very short):
- Bell La Padula
- Chinese Wall
Operating Sys. Mechanisms:
- Access Control List
- Capabilities

... Explain Q3 and formalize per model!

Models:
- based on Access Control Matrix
Security Policy
A security policy $P$ is a statement that partitions the states $S$ of a system into a set of authorized (or secure) states (e.g., $\Sigma_{\text{sec}} := \{ \sigma \in \Sigma \mid P(\sigma) \}$) and a set of unauthorized (or non-secure) states.

Secure System
A secure system is a system that starts in an authorized state and that cannot enter an unauthorized state (i.e., $\Sigma_{\text{reachable}} \subseteq \Sigma_{\text{sec}}$)

Reference: Matt Bishop: Computer Security Art and Science

ref MB: page 95
Definitions:

Information or data I is **confidential**

- with respect to a set of entities X if no member of X can obtain information about I.

Information I or data is **integer** if (2 definitions in textbooks)

- (1) it is current, correct and complete
- (2) it is either is current, correct, and complete or it is
Model for Confidentiality

Secrecy Levels:
- Classification (documents)
- Clearance (persons)
- The higher the level the more sensitive the data
- totally ordered
EXAMPLES BLP (TANENBAUM)

- categories: NATO, Nuclear
- levels/clearance: top secret, secret, confidential, unclassified
- document: Nato, secret
- person clearance: read
  - secret, Nato -> allowed
  - secret, Nuclear -> not allowed
  - confidential, Nato -> not allowed
Confidentiality & Integrity

- Subjects
- Objects: pieces of information of a company
- CD: Company Data Sets
  objects related to single company
- COI: Conflict of Interest class
  data sets of competing companies
- Sanitized Objects
  version of object that does not contain critical information

Ref MB: Chapter 7.1
CHINESE WALL, EXAMPLE

COI

CD

Objects

Sanitized O

Subject
CHINESE WALL, RULES

PR(S): set of Objects previously read by S

S can read O, if any of the following holds

- first-time read
- ∀ O, O’ ∈ PR(S) => COI(O) ≠ COI(O’)
- O is a sanitized Object
CHINESE WALL, EXAMPLE

COI
CD
Objects
Sanitized O
Subject

VW → BMW
D

intel → ARM

write request
read request

read request
write request

PR
PR(S): set of Objects read by S

S can write O, if

- “S can read O”
- ∀ unsanitized O’, “S can read O’” \(\Rightarrow CD(O) = CD(O')\)
CHINESE WALL, EXAMPLE

COI → VW → BMW → intel → ARM

CD → VW → BMW → intel → ARM

Objects

Sanitized O

Subject

write request

pr

read request
Operating Sys. Mechanisms:
- Access Control List
- Capabilities

Explain Q3 and formalize per model!

Models:
- *based on Access Control Matrix*
- "take grant" model
**Subjects:** S  
**Objects:** O  
**Entities:** $E = S \cup O$  
**Rights:** \{read, write, own,...\}  
**Matrix:** $S \times E \times R$

Simple ACM Operations:
- create subject / object
- destroy subject / object
- enter / delete R into cell (s,o)

<table>
<thead>
<tr>
<th></th>
<th>O1</th>
<th>O2</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S1</strong></td>
<td>r,w,own</td>
<td>r,w</td>
<td>r,w,own</td>
<td>--</td>
<td>r,w</td>
</tr>
<tr>
<td><strong>S2</strong></td>
<td>r,w</td>
<td>r,w,own</td>
<td>--</td>
<td>r,w,own</td>
<td>r</td>
</tr>
<tr>
<td><strong>S3</strong></td>
<td>r,w</td>
<td>r</td>
<td>w</td>
<td>--</td>
<td>r,w,own</td>
</tr>
</tbody>
</table>
### ACM

- **Access Control List (ACL)**

- **Capabilities**

---

<table>
<thead>
<tr>
<th></th>
<th>O1</th>
<th>O2</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>r,w,own</td>
<td>r,w</td>
<td>r,w,own</td>
<td></td>
<td>r,w</td>
</tr>
<tr>
<td>S2</td>
<td>r,w</td>
<td>r,w,own</td>
<td></td>
<td>r,w,own</td>
<td>r</td>
</tr>
<tr>
<td>S3</td>
<td>r,w</td>
<td>r</td>
<td></td>
<td>w</td>
<td>r,w,own</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th></th>
<th>O1</th>
<th>O2</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>r,w,own</td>
<td>r,w</td>
<td>r,w,own</td>
<td></td>
<td>r,w</td>
</tr>
<tr>
<td>S2</td>
<td>r,w</td>
<td>r,w,own</td>
<td></td>
<td>r,w,own</td>
<td>r</td>
</tr>
<tr>
<td>S3</td>
<td>r,w</td>
<td>r</td>
<td></td>
<td>w</td>
<td>r,w,own</td>
</tr>
</tbody>
</table>

---

*ref MB: chapter 2.2*
Define Protection Mechanisms of an Operating System in terms of primitive ACM operations

only the defined mechanism provided by the OS can be used
“Leakage”:
an access right is placed into S/O that has not been there before
it does not matter whether or not that is allowed

Is leakage decidable?
Examples for OS-Mechanisms defined by ACM-Operations:

UNIX create file (S1,F)
create object
enter own into A(S1,F)
enter read into A(S1,F)
enter write into A(S1,F)

ref MB: chapter 2.2
Examples for OS-Mechanisms defined by ACM-Operations:

UNIX chmod -w (S2,F)
   if own ∈ A(caller,F)
   then delete w in A(S2,F)

Q3:
Given an OS with a ACM-based description of protection mechanisms is “Leakage” decidable for any R in A(x,y) ?

ref MB: chapter 2.2
Decidable

- no subjects/objects can be created

or

- only one primitive ACM operation per OS-Mechanism

by exhaustive search!

Q3 in general:

- undecidable (proof: reduction to Turing machine)
“Capabilities”
an intuitive example
- files: a privileged process
- Photo: an untrusted process
- Photo brings a small initial set of “capabilities” on installation
- needs permission to edit a specific photo P

privilegierter Prozess: files
request P

files: asks usr for permission
creates a capability for P

“grants” capability to Photo
Directed Graph:

Subjects: ●
Objects: ○
Either S or O: ×

x has capability with set of rights \( \alpha \) on y:

\[
\begin{array}{c}
\bullet \quad \text{x} \\
\alpha \quad \longrightarrow \\
\quad \text{y} \\
\end{array}
\]

\[ x \text{ has cap with set of rights } \tau \text{ that includes } t \]

\[
\begin{array}{c}
\bullet \quad \text{x} \\
\text{t} \quad \longrightarrow \\
\quad \text{y} \\
\end{array}
\]

\[ g \text{ grant right} \]

\[ x \text{ has cap with set of rights } \gamma \text{ that includes } g \]

\[
\begin{array}{c}
\bullet \quad \text{x} \\
g \quad \longrightarrow \\
\quad \text{y} \\
\end{array}
\]
Q3/ 2: TAKE GRANT RULES

Rules:

**take rule** \((\alpha \subseteq \beta)\)

a takes \((\alpha \text{ to } y)\) from \(z\)

\[
\begin{array}{cccc}
\bullet & t & \blacklozenge & \beta \\
x & z & y \\
\end{array}
\quad \vdash 
\begin{array}{cccc}
\bullet & t & \beta & \blacklozenge \\
x & z & y \\
\end{array}
\]

**grant rule** \((\alpha \subseteq \beta)\)

\(z\) grants \((\alpha \text{ to } y)\) to \(x\)

\[
\begin{array}{cccc}
\bullet & g & \blacklozenge & \beta \\
x & z & y \\
\end{array}
\quad \vdash 
\begin{array}{cccc}
\bullet & g & \beta & \blacklozenge \\
x & z & y \\
\end{array}
\]

ref MB: chapter 3.3
Rules:

create rule

\[ x \text{ create (} \alpha \text{ to new vertex)} \; y \]

remove rule

\[ x \text{ removes (} \alpha \text{ to)} \; y \]
CanShare(α, x, y, G₀):

there exists a sequence of G₀ ... Gₙ with \( G₀ \vdash^* Gₙ \)

and there is an edge in Gₙ:

\[ x \xrightarrow{\alpha} y \]
take rule ($\alpha \subseteq \beta$)

a takes ($\alpha$ to y) from z

grant rule ($\alpha \subseteq \beta$)

z grants ($\alpha$ to y) to x

ref MB: chapter 3.3
create rule

z takes (g to v) from x

z grants (α to y) to v

ref MB: chapter 3.3
**Q3/M2: FORMALIZED**

CanShare(\(\alpha, x, y, G_0\)):

there exists a sequence of \(G_0 \ldots G_n\) with \(G_0 \vdash^* G_n\)

and there is an edge: \(\xymatrix{ x \ar[r]^-\alpha & y }\)

CanShare decidable in linear time!
- three questions, 2 models per question, different answers !!!
- modeling is powerful
- need to look extremely carefully into understanding models !!!

Q1/M2: John v. Neuman, PROBABILISTIC LOGICS AND THE SYNTHESIS OF RELIABLE. ORGANISMS FROM UNRELIABLE COMPONENTS.

Q2: most textbooks on distributed systems