Dealing with availability at the architectural level

Jacques Noyé
noye@emn.fr
Obasco EMN - INRIA

Projet DISPO

- Funded by a French programme on security
- ENSTB (Rennes), IRIT (Toulouse), IRISA (Rennes), INRIA Rhône-Alpes (Grenoble)
- EMN : Jean-Claude Royer, Sebastian Pavel

Architecture

- APL: Architecture Programming Language
- Architecture = components + aspects
- Enforce availability through aspect weaving

Availability policy

- Services/ressources (amount of ressources, amount of time) providers and users.
- User get permissions to access the services/ressources.
- Not enough to prevent denial of service : need for provider and user obligations.
- Temporal notions / deontic concepts.

User agreement

- Examples :
  - A resource is released when it is not used any longer.
  - Asking for a resource that is not necessary is forbidden.
  - A resource must be reserved before being used.
- A violation of the agreement may legitimate a denial of service (provider permission).

Formalisation [Cuppens et al.]

- First order logic with equality
- Temporal modalities :
  - `hold-at(p, t)` : formula `p` is true at time `t`
  - `ALWAYS(p)`⇔∀`t, hold-at(p, t)`
- Deontic modalities
  - `O(p)` : `p` is obligatory
  - `F(p)` ⇔ `O(¬p)`: `p` is forbidden
  - `P(p)` ⇔ `¬O(p)`: `p` is permitted

Deontic Modalities

- Axiomes:
  - $O(p \wedge q) \Rightarrow O(p) \wedge O(q)$
  - $\neg(O(p) \wedge O(\neg p))$
- Inference Rule :
  - if $p \Rightarrow q$ is a theorem $O(p) \Rightarrow O(q)$

Availability policy

- R1: S must be able to start task T at most 6 time units after its request
- R2: S must have access to resource R necessary to perform T at most 4 time units after its request
- R2': S must have access to resource R necessary to perform T at most 9 time units after its request
- R4: the maximum realization time of T is 3 time units

What can we do with this ?

- Provided a spec. of the system is available :
  - Analysis of the consistency of the policy
    - (R1, R2, R4) satisfiable
    - (R1, R2', R4) inconsistent
  - Verify availability properties
    - the request of S must be satisfied 9 time units after it has been made
- What about enforcing an availability policy?

Enforcing a security policy

- Security Automata [Schneider2000]:
  - An execution is represented by a (possibly) infinite sequence of actions, states, action/states…
  - A security policy is specified by giving a predicate on set of executions.
  - Enforcement through execution monitoring means that the security policy is a safety property.

- It is then possible to build an AFD to stop the execution if things go wrong.
- Example :

state vars state : {0, 1} initial 0

transitions

pay(C) ∧ state = 0 → state := 1

serve(C) ∧ state = 1 → state := 0

- Cannot deal with liveness properties, eg answering a request in a finite time.
- Answering a request in a bounded time is ok.
- Stopping the execution is not satisfactory! But nothing else is possible.

Architecture

- Architecture = components + aspects
- Component = interface + implementation
- Interface
  - signatures of required and provided services
  - dynamic behaviour described with an STS (Symbolic Transition System)
  - transition = guarded message sending/message execution
  - composition = automaton product

Architecture

- Active and passive sequential (primitive) components
- Synchronous and asynchronous communications
- No component reference through communication channels
- Weaving of the dynamic behaviour within the implementation

(Too) Many open questions

- Notion of time : abstract time (based on number of transitions)
- Restrict availability to security problems ? Use program analysis to predict the future.
- Compositionality issues : what can be modularized, what cannot ?
- Distributed monitoring is difficult.