Ingénierie de la sûreté de fonctionnement

Institut Mines-Télécom

Thomas Robert
thomas.robert@telecom-paristech.fr
Bureau 4D44
Mastère spécialisé – INF722
Version 2019

Structure du cours en général

- 3 contenus de cours :
  - Concepts élémentaires, risque et objectifs de SDF
  - Tolérances aux fautes : données, ou processus (solutions architectures et programmatiques).
  - Evaluation de sûreté de fonctionnement
- Des TPs pour les deux dernières parties + un TD de réflexion + peut être un séminaire industriel (sureté / sécurtié).
- Note finale : 20 % TD rendu en fin de séance, 100 % à l'examen (les deux notés sur 20. Attention le TD sera assez dur 10/20 donnera en gros 2 pts bonus, 20 → 4, arrondi au demi point le plus proche ou inférieur si égalité)

Pré-requis du cours

- Logique (élémentaire) pour les cours et TD tolérance aux fautes : rappel rapide fait en cours
- Programmation en C + fonctionnement de base de la synchronisation par sémaphore (rappelé en début de TP)
- Je prendrai du temps sur les tp analyse quantitative pour faire une mise à niveau si nécessaire en probabilité).

Quelques mots sur l'ingénierie système motivation

- Systèmes logiciels
  - Systèmes d'information (base de clients, comptes bancaires)
  - Systèmes industriels (lignes de production numérique, infrastructure de contrôle pour un processus - e.g. chimique)
  - Systèmes embarqués (satellites, trains, avions)
- Incident/accident : situation aux conséquences négatives
  - Perte d'avantage commercial / perte financière
  - Impact environnemental / mise en danger de la vie des personnes
  - Destruction du système

… comment les maitrise-t-on ? Que faut il faire et à quel coût ?

Risque : définition et traitement

Incident, vraisemblance, et risque

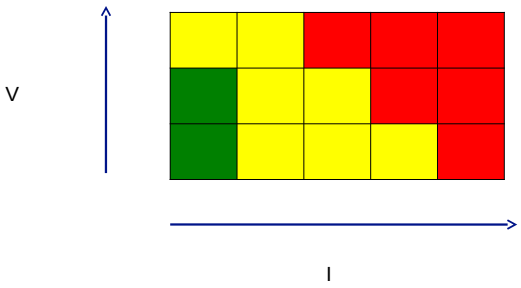
- **Incident = événement redouté portant atteinte aux personnes ou aux biens**
- **Définition générale du risque :**

  Une **estimation** qualitative ou quantitative déterminant **l'importance des différents incidents** par rapport à leur **vraisemblance**

- **Exemple de principe d'estimation qualitative :**
  - Définition de trois classe de risque : faible, moyen, élevé
  - Dérivation du niveau de risque (f,m,e) à partir :
    - Un impact (nature des conséquence de l'incident)
    - Une classe de fréquence : peu fréquent, raisonnablement fréquent, et très fréquent.
- La communauté d'ingénierie friande des formules chocs : R= I*V

Matrice de risque : visualisation de classes d'équivalence (le cas impact/vraissemblance

- **Idée : représenter dans le plan les classes de risques en fonction de l'impact (I) et de la vraisemblance (V)**



Vraisemblance d'un incident ses différentes interprétations

- **Sémantique :**
  - **La vraisemblance = occurrence**

    lien avec une vision fréquentielle / temporelle
  - **Vraisemblance = faisabilité**

    lien avec une vision logique de condition de faisabilité/impossibilité
- **Caractérisation :**
  - **Par système ou population**
  - **Globale ou modulaire**

Vraisemblance d'un incident – exemple

- **Faisabilité VS Occurrence**
  - téléphone portable : l'incident d'explosion de la batterie = faible
  - avion : perte du pilote automatique = perte de 3 calculateurs
- **Caractérisation globale vs multi critères vs modulaire**
  - **La vraisemblance correspond à un probabilité d'occurrence de 12 % dans les 12 premières heures de fonctionnement**
  - **Un robot industriel sur une chaîne de montage, l'incident = blesser un opérateur humain. Critères : présence de l'humain + dysfonctionnement du détecteur de présence ?**
  - **La perte de la fonction autopilot ne peut se produire que lors de la perte de 3 calculateurs (simplification)**

Décomposition et analyse des causes d'un incident

- **Pb : pour analyser l'incident, il faut**
  - **Décomposer la définition de l'incident : séparer la partie « système » de la partie environnement**
  - **Établir les causes possibles.**
- **Exemple : l'ascenseur**

  Une personne est tombé dans la cage d'un ascenseur

  Décomposition : la porte de l'ascenseur s'est ouverte + l'ascenseur est absent + l'utilisateur a franchi le seuil de l'ascenseur

  Nb : la décomposition nécessite souvent d'établir un lien causal entre l'événement incident et des événements plus simples (antérieurs ou simultanés).

L'étude de la chaîne causale : variantes et améliorations

- **Principe : déterminer les étapes considérées comme nécessaires à l'occurrence de l'incident ou le déclenchant**
- **Pb : par quoi commence t on**
- **Cas 1 : la description des incidents est disponible et l'on dispose d'un modèle du système**

  Analyse « arrière » recherche des causes
- **Cas 2 : description des incidents absente**

  Analyse avant = a partir des états possibles étude des déviations d'état/ altération de structure

Détermination de l'impact / Analyse Avant

- **Identification du périmètre d'étude (système + entités pouvant être affectées)**
- **Si décomposition du système**
  - **Identification des sous composants + dépendances (type d'échange, et comportement attendu**
  - **Identification de variation possible de l'attendu**
- **Détermination de l'impact : identification des conséquences du comportement du système (attendu ou non) sur son environnement**
- **Un concept clé : danger**

  Danger (Hazard) = situation ayant le potentiel de causer l'incident

La méthodologie HAZOP : propagation des « déviations »

- **Principe : description modulaire du système + dépendances spécifique au domaine industriel étudié et analyse des états fonctionnel du système et de leur conséquence**
- **Historique : méthodologie conçue pour l'industrie chimique**
  http://people.clarkson.edu/~wwilcox/Design/hazopcas.pdf
- **Les composants : tuyaux, valves, pompes, fours, capteurs …**
- **Les dépendances : circulation de fluide (propagation de pression, température ….)**
- **Principe : considérer toutes les combinaisons possibles de « sorties » ou état des composants**
- **Objectif : éviter le « on ne pensait pas ce scénario possible »**

Traitement du risque

4 actions possibles

- **Refuser la situation à risque (pas toujours possible)**
  Interdire la production / l'usage / ne pas inclure la fonctionnalité
- **Externaliser le risque**
  Trouver un moyen de reporter la responsabilité sur un tiers
- **Atténuer le risque**
  Trouver un moyen de reporter la responsabilité sur un tiers
- **Ignorer/Accepter le risque**
  Trouver un moyen de reporter la responsabilité sur un tiers

Définition d'objectifs de traitement du risque

- **Objectifs possibles :**
  - Identification/documentation du risque
  - Limitation du risque => obligation de réaliser certains traitement en fonction du risque identifié
  - Dans le cas de l'atténuation : exiger une réduction de l'impact ou de la vraisemblance des incidents
  - Dans le cas de l'acceptation : exiger que l'analyse menant à l'acceptation respecte une certaine rigueur.
- **Conséquences :**
  - *Atténuation du risque => contrainte sur la conception du système son architecture ou son exploitation*
  - *Rigueur de l'étude de risque => contrainte sur les processus de développement*

Sureté de fonctionnement : une discipline d'ingénierie

- **PB : comment et qui doit traiter le risque pour un système nécessitant plusieurs expertises**
  - Mécanique
  - Électricité/électronique
  - Informatique
  - Réseau
- **Réponse : expertise séparée transverse + recommandation par métier**

Sûreté de fonctionnement
Définition et Analyse des Termes

- **Objectif :**

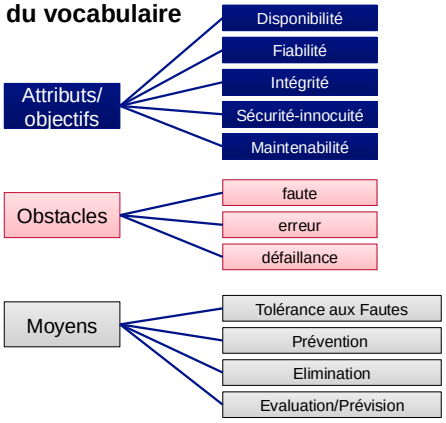
  Obtenir une **confiance justifiée** dans **la capacité du système** à **rendre le service attendu dans des conditions d'usage définies**
- **Ce que cela sous entend**
  - Définir les conditions d'usage sous lesquelles le système doit rendre le service attendu et/ou avoir un impact maitrisé sur
    - Son environnement (utilisateur(s) inclus)
    - Sur lui même
  - Savoir ce que le système doit faire, lier le système ou ses parties à une fonction/responsabilité
  - Définir des objectifs de garanties sur le respect du comportement attendu (qu'il soit sur le service attendu ou sur l'impact du système)

Identification du périmètre

- **système: unité de description permettant de distinguer l'objet d'étude de son environnement**
  - structure ; description des éléments à priori immuables du système (architecture)
  - État : information variable au cours de la vie opérationnelle du système (effectivement représentée en mémoire pour du logiciel)
  - État interne: fraction non observable de l'état du système
  - Interface : fraction de l'état du système partagée avec son environnement (E/S)
- **Environnement : ensemble des éléments ayant un impact sur l'interface du système ou pouvant être impactés par ce dernier**

Sureté de fonctionnement
les nouveaux objectifs, nouvelles contraintes, nouvelles analyses

Décomposition du vocabulaire



Enjeu

- **Service = utilité / obligation**
- **Déviation du comportement attendu => coût**
- **Rôle de l'ingénierie de la SdF == maîtriser ce coût**

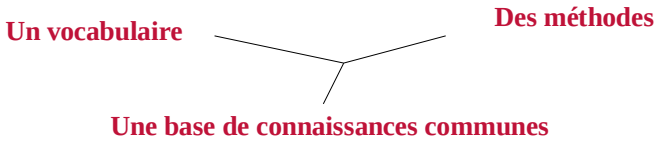
ATTENTION le coût n'a pas toujours une valeur financière !

PB : où se situe la séparation en qualité et sûreté de fonctionnement ?
=> lié à un seuil de coût par incident…

- **Exemples historiques :**
  - Maîtrise de la qualité de service en téléphonie fixe
  - Maîtrise du comportement d'un ascenseur

Et spécifiquement pour les systèmes logiciels

- **[Avizienis et al'04]**
  "Basic Concepts and Taxonomy of Dependable and Secure Computing "



Vocabulaire de base

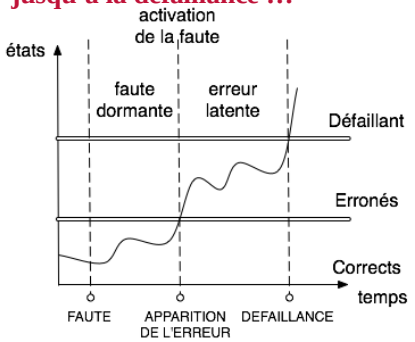
Décrire les entraves – l'alternative

- **Un vocabulaire centré sur le système :**
  - **Défaillance** :
    écart **observable** entre le service attendu et le service rendu
  - **Erreur** :
    Tout ou partie de **l'état interne** du système pouvant causer sa défaillance
  - **Faute** :
    Cause de l'apparition d'une erreur (origine structurelle ou liée à l'état) liée au développement du système ou sa structure

La chaîne faute/erreur/Défaillance

- **Evénements :**
  - Activation
  - Détection
  - Défaillance
- **Etats :**
  - Fautifs (non activés)
  - Corrects (sans faute)
  - Erronés (?)
  - Défaillants (KO)

Pas de détection == erreur dormante jusqu'à la défaillance …

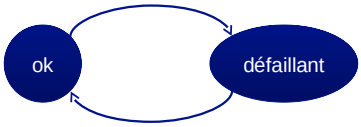


Comportement vs attribut clé

- **2 manière de contraindre la conception d'un système**
  - **Le véhicule doit pouvoir s'arrêter en moins de 6m pour toute vitesse inférieure à 60 km**
  - **Le poids du véhicule est inférieur à 800kg**
- **Une exigence comportementale affecte la fonction du système directement => exigence fonctionnelle**
- **Une exigence sur un attribut du système n'est pas explicitement rattachée à un comportement => exigence non fonctionnelle**
- **Donnez un autre exemple d'exigence fonctionnelle de sûreté de fonctionnement**

Les attributs de Sureté de fonctionnement par l'exemple

- **Idée : caractérisation des attributs par rapport à l'état du système (défaillant / non défaillant) et la nature du risque encouru**
- **Description usuelle de**
  - La disponibilité : être dans l'état OK

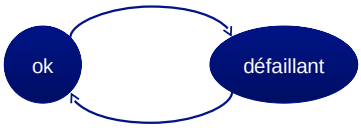


Ou Temps moyen avant transition vers défaillant

$$\frac{Temps(OK)}{Temps(OK)+Temps(défaillant)}$$

Raffinement des attributs

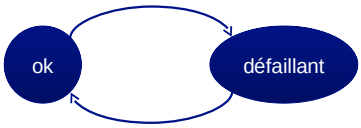
- **Idée : caractérisation des attributs par rapport à l'état du système (défaillant / non défaillant) et la nature du risque encouru**
- **Description usuelle de**
  - La fiabilité : ne pas quitter l'état OK



  - Probabilité ou condition pour transition vers « défaillant »

Raffinement des attributs

- **Idée : caractérisation des attributs par rapport à l'état du système (défaillant / non défaillant) et la nature du risque encouru**
- **Description usuelle de**
  - La maintenabilité : garantir la transition de défaillant vers OK

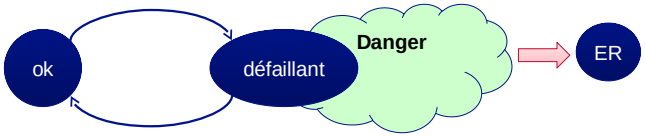


  - Le retour à l'état Ok conditionné par une intervention

=> Analyse du temps moyen jusqu'à intervention, difficulté de l'intervention, coût …

Raffinement des attributs

- **Idée : caractérisation des attributs par rapport à l'état du système (défaillant / non défaillant) et la nature du risque encouru**
- **Description usuelle de**
  - La sécurité innocuité :



  - Identification de la contribution des états défaillants aux états de danger
  - Classification des états défaillant en fonction de la gravité de l'événement redouté (ER) en bout de chaine

le cas de l'intégrité

- **Idée : caractérisation des attributs par rapport à l'état du système (défaillant / non défaillant) et la nature du risque encouru + description du système**
- **Description usuelle de l'intégrité :**
  - **Contra**inte sur la capacité à détecter / corriger des altération des données stockée
  - Mais aussi Contrainte de déterminisme comportemental

L'intégrité des données est fortement lié à la fiabilité d'un dispositif de stockage/communication (i.e. une partie)

Attributs et contraintes de sureté de fonctionnement

- **Attributs :**
  - Disponibilité (availability) : capacité du système à offrir tout ou partie du service
  - Fiabilité (reliability) : capacité du système à effectivement délivrer un service correct (lorsqu'il est disponible)
  - Sécurité-innocuité (safety) maitrise/connaissance des conséquences d'une comportement du système (attendu ou non)
  - Intégrité (integrity) capacité du système à détecter ou empêcher toute altération de sa structure ou de son état en contradiction avec le comportement attendu
  - Maintenabilité (maintainability) capacité du système à faire évoluer sa structure ou son état pour faciliter le retour à un état disponible / fiable…
- **Objectifs (I) : contrainte sur les attributs**

Des méthodes, et des standards …

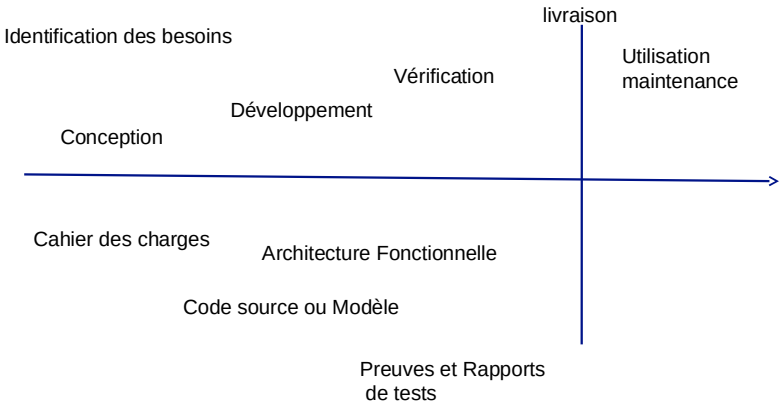
- **La sureté de fonctionnement : un objectif**
  - Impacte tous les niveaux du développement
  - Compromis coût d'application / coût défaillance et conséquences
  - Contrainte réglementaire (ex. nucléaire/chimie lourde) ou bonne pratique
- **Des standards par niches industrielles pour décrire les actions à mener**
  - Ingénierie chimique => premier standard ~198x (OSHA)
  - Ingénierie ferroviaire (IEC 61508)
  - Avionique (ARP4761)
  - Automobile (uniquement depuis ISO 26262)

Quoi et comment ?

Le cycle de développement d'un système cycle de vie

- **Principe d'organisation des différentes phases d'ingénierie**
  - Détermine la nature des activités
  - Détermine l'objectif et les résultats attendus (logiciel, description textuelle …)
  - Détermine l'enchaînement dans le temps de ces étapes
- **Exemple : développement par Cycle en V**
  - Décomposition des activités :
    description des objectifs, du comportement, et de la structure interne du système
  - Planification et identification des dépendances
- **Positionnement différent de chaque activité**

Processus de développement et Ingénierie



Les moyens pour obtenir un niveau voulu de SdF

- **4 Approches complémentaires**
  - **Prévention des fautes :**
    Méthodes destinées à empêcher l'occurrence même des fautes
  - **Élimination des fautes :**
    Méthodes de recherche et de suppression des fautes de conception et développement
  - **Prévision des fautes :**
    Analyse de la fréquence ou du nombre de fautes et de la gravité de leurs conséquences
  - **Tolérance aux fautes :**
    Mécanismes au sein du système destinés à assurer les propriétés de SdF voulues en présence des fautes

Exemple de mise en œuvre

- **Prévenir**
  - Méthode et contraintes de conception permettant d'éviter la présence de fautes dans la conception ou le produit développé

  Ex : Programmez en C, prévention == sans pointeurs, ni tableaux
- **Eliminer**
  - Méthodes de mise au point permettant d'identifier et corriger les éléments structuraux correspondant à des fautes dans le but de les supprimer

  Ex : Programmation en Java, élimination == test Junit et modification du code pour supprimer les sources d'exceptions, idem avec gdb pour le C

Sûreté de fonctionnement
Les moyens…

- **Tolérer**
  - Méthodes de conception permettant de réagir à l'activation d'une faute, la détection d'une erreur, dans l'objectif d'empêcher l'occurrence d'une défaillance ou d'en limiter les conséquences.

  Ex : implémenter le traitement d'exception Java qui redémarre automatiquement l'application
- **Évaluer/Prédire**
  - Méthodes et métriques permettant de quantifier (mesurer) ou qualifier (classer) un système vis à vis des attributs de la sûreté de fonctionnement

  Ex : Émuler un réseau pour identifier les conditions d'engorgement (qualitatif), ou utiliser une distribution de Pareto pour prédire la fréquence d'occurrence de perte de messages sur un réseau sans fil

Le rôle des formalismes et de leur manipulation

- **De quoi a-t-on besoin ?**
  - Description des objectifs du système
  - Description de sa structure et de son comportement
- **Spécification Fonctionnelle**
  - Description des objectifs du système
  - Décomposition du système en plusieurs entités si ses fonctionnalités sont complexes
- **Conception et description d'implémentation**
  - Description détaillée du comportement interne du système et de ses entités assurant la satisfaction des objectifs
  - Description détaillée des ressources (logicielle…matérielle) nécessaire au bon fonctionnement des entités du système

Spécification fonctionnelle:
*** apport d'un modèle architectural**
*** apport de la logique**

De quoi a-t-on besoin ? (bis)

- **Distinguer dans un système qui est responsable**
  - V1 : Un contrôleur de vitesse sur un véhicule a pour fonction de contrôler l'accélération du véhicule à partir d'information sur sa vitesse courante
  - V2 : Le contrôleur de vitesse sur un véhicule à pour fonction de contrôler le couple du moteur à partir de mesures de vitesse rendues par des capteurs, afin d'assurer une vitesse programmée

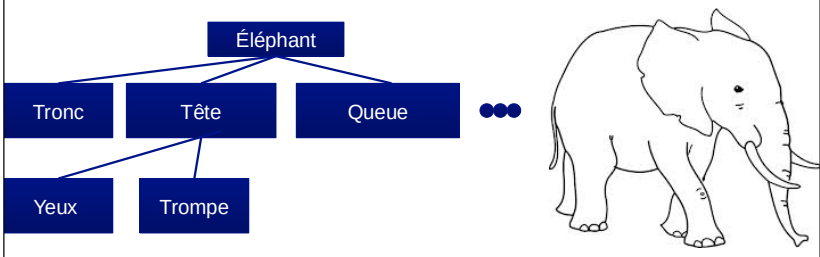
  Si on était tatillon on pourrait dire que dans un des deux cas le contrôleur n'avait pas pour mission d'empêcher une vitesse excessive… (laquelle?)
- **Distinguer ce que l'on veut de ce que l'on veut éviter**
  - Ex : « le train ne doit pas rouler lorsque ses portes sont ouvertes »

Les formes de spécification fonctionnelle

- **Les formats :**
  - Textuelle rédigée
  - graphique annotée
  - Mathématique structurée
- **Qualité de la description :**
  - Rédigée « libre »
  - Structurée
  - Semi-formelle
  - Formelle
- **Leur interprétation :**
  - Exemple1 : spécification d'un déplacement par définition de vecteurs vitesse en 3D => représentation graphique non ambiguë
  - Exemple2 : page de manuel d'une commande shell (texte rédigé)

Décrire le système

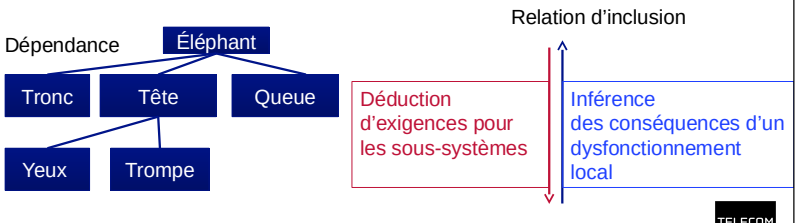
- **Le point de vue « Organique » == structure physique**



Centré sur la structure physique, permet de localiser la manifestation d'une faute à une partie de la structure du système.

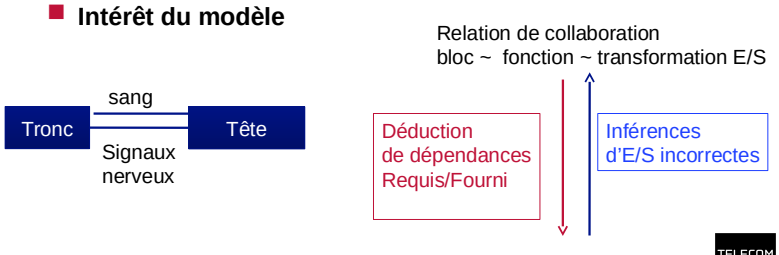
Modèle à composant, Hiérarchie & Interfaces

- **Relations entre « blocs »**
  - **Inclusion** ?
  - **Collaboration** ?
  - Héritage ?
- **Un dessin n'est pas un modèle, modèle = structuration d'une information interprétable**
- **Intérêt du modèle**



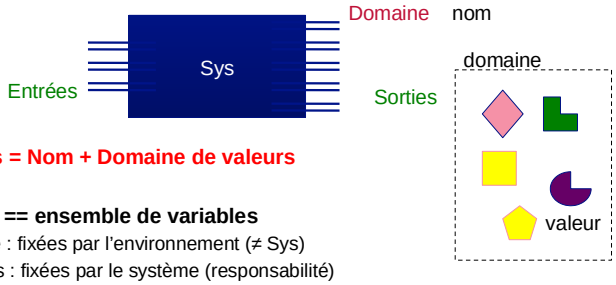
Modèles à composant, Hiérarchie & Interfaces

- **Relations entre « blocs »**
  - **Inclusion** ?
  - **Collaboration** ?
  - Héritage ?
- **Un dessin n'est pas un modèle**
- **Intérêt du modèle**



Et en pratique ….
Modéliser la circulation de données

- **Idée ; l'activité d'un système == circulation de données**



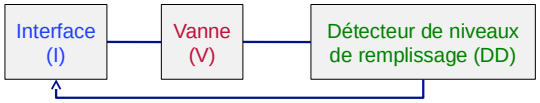
- **Variables = Nom + Domaine de valeurs**
- **Interface == ensemble de variables**
  - Entrée : fixées par l'environnement (≠ Sys)
  - Sorties : fixées par le système (responsabilité)

  (définition améliorée par la suite)
- **Modélisation de l'attendu == relation entrées/sorties**

Exemple : Système de contrôle d'un déversoir

- **Fonction** :

  Contrôle de la tension d'alimentation d'une vanne qui contrôle le délestage d'une réservoir d'eau alimentée par une rivière.
- **Sous-fonctions :**
  - Contrôle ouverture/fermeture de la vanne
  - Détection automatique d'une condition de débordement du réservoir
  - Interface (écrans) pour superviseur humain surveillant/contrôlant le système



Décrire les interactions == E/S par blocs, plus correspondance S-E

Exemple : le modèle fonctionnel
Se donner des variables pour décrire le système

Nom	Composant	Type	Rôle
EV	Interface	{O,F,NC} *(S)*	Affichage état de la vanne
EN	Interface	[0,1] *(S)*	Affichage du Taux de remplissage du réservoir
OU	Interface	{Oui, Non} *(E)*	Commande de l'ouverture d'urgence
F	Interface	{Oui, Non} *(E)*	Commande de la fermeture
Com	Interface	{O,F} (S)	Commande transmise à la vanne depuis l'interface
CIV	Vanne	{O,F} *(E)*	Commande reçue depuis l'interface
DD	Vanne	*{O,N} (E)*	Détection d'un risque de débordement
SV	Vanne	*{O,N} (S)*	Ecoulement en sortie de vanne

A quoi sert un tel modèle ?

- **La décomposition structurelle permet de séparer les responsabilités**
- **L'identification des interactions permet de nommer les données, les artefacts échangés**
- **Il devient donc possible à partir de ces noms de décrire ce que l'on attend**
  - du système (interface avec son environnement)
  - De chacune de ses parties
- **NB: Des langages tels que UML, SysML …. ont pour but de vous donner des termes pour décrire de telles structures (et les interactions entre blocs)**
- **Ensuite, on peut appliquer des méthodes de type Hazop ….**

La vision clé – attribut et modèle architectural

- **Concept de base : la relation binaire**
  - **Nom d'un composant et ses entrée sorties**
  - **Le nom d'une entrée et son type**
  - **Le nom d'une entrée et la sortie à laquelle elle est reliée**
  - **….**
- **Analyse = parcourt des relations binaire**

- **Représentation sous forme de graphe de l'information clés**
- **Les algorithmes d'analyse == logique de parcourt de graphe**
  - **Action sur un nœud**
  - **Action sur les arcs**

L'étape suivante ?

- **Comportement attendu**
  - Ex1: Si le réservoir atteint un niveau de 97% de remplissage alors déclencher l'ouverture de la vanne
  - Ex2 L'interface de contrôle doit afficher en permanence l'état de la vanne
- **Ces descriptions constituent une spécification fonctionnelle du système**
  - Description de ce que doit « faire » chaque bloc sans nécessairement préciser comment
  - Plus le traitement est complexe plus il est nécessaire d'abstraire l'attendu

Description textuelles et textuelles structurée

- **Exemple : fiche man d'une fonction shell**
  - Structuration de la description du fonctionnement de la fonction, de ses paramètres et sorties
  - Champs : Synopsis, Description, Option ….
- **Avantage :**
  - peu ou pas de restriction sur le niveau de détail utilisable (peut contenir des formules mathématiques)
  - Compréhensible par tout le monde (en principe)
  - Concis si l'on fait des hypothèses sur la connaissance partagée (exemple section synopsis d'un fiche de man)
- **Inconvénient :**
  - Risque de contradiction difficile à analyser et détecter
  - Quasi impossible à faire analyser par un ordinateur
  - Risque de description incomplète, ou ambiguë

Les propriétés souhaitées

- **Non ambiguë : les affirmations et formulations utilisées ne doivent pas amener de question d'interprétation du type**

« le terme ''dès que" signifie-t-il juste à près ou en même temps ? »

- **Vérifiable : il est doit être possible de dériver, de chaque affirmation, une méthode pour en vérifier la réalisation dans l'implémentation du système.**
- **Consistante : si la spécification est complexe et constituée de plusieurs affirmations, ces dernières ne doivent pas être en contradiction.**

Formalisation par description logique

- **Principe :**
  - Formule logique == séquence structurée de mots correspondants :
    - À des affirmation (vraie ou fausses)
    - A des combinaisons de ces affirmations (et, ou, si … alors …)
  - Un raisonnement mathématisé pour calculer pour chaque formule les conditions dans lesquelles la formule est vraie
- **Les logiques que nous allons voir :**
  - Propositionnelle, premier ordre
  - Temporelle

La syntaxe

- **Formule logique == séquence de termes « typés »**
- **Les constantes : vrai, faux. Une variable booléennes == un nom dont la valeur appartient à {vrai, faux}**
- **L'ensemble des variables booléennes *V ={x1,x2,…., xn}***
- **Notion d'opérateur vs fonction :**
  - Un opérateur est une fonction qui a 1 ou 2 paramètres !!
  - Notation : (x1 op x2) équivalent à op(x1,x2)
  - Intérêt : permet souvent des notations plus compactes
- **Opérateurs logiques :**
  - Et : 2 paramètres noté ∧
  - Ou : 2 paramètres noté ∨
  - Implique : 2 paramètres noté =>
  - Négation : 1 paramètre noté not

Interprétation d'une formule

- **Affectation de valeur aux variables booléennes : valuation, ou fonction de vérité : f: x in {x1,…xn} -> {V,F}**
- **Interprétation d'une formule : calcul de la valeur résultant de l'évaluation des opérations**
- **Tables d'évaluations : (a,b deux variables et F,V constantes faux, vrai, a V signifie a vaut V.**

a∧b	a=V	a=F
b V	V	F
b F	F	F

a∨b	a=V	a=F
b V	V	V
b F	V	F

a=>b	a=V	a=F
b V	V	V
b F	F	V

not a	
a V	F
a F	V

Modèle logique et formule

- **Rappel : un modèle simplification de la réalité**
- **Un modèle logique associé à un ensemble de variables booléennes, {x1,…,xn}, est un ensemble de valuations**
- **Exemple modélisation d'un problème de répartition de de deux balles dans 3 sacs :**
- **Variables : bxsy est vrai si il y a x balles dans le sac y**
- **Considérons le vecteur μ=(b0s1,b1s1,b2s1,b0s2,b1s2,b2s2,b0s3,b1s3,b2s3)**
- **Un modèle correct de répartition est l'ensemble des valuations qui assurent qu'il y a au plus 2 balles dans l'ensemble des sacs …**
- **μ=(F,F,F,F,V,F,F,V,F) fait partie de ce modèle**
- **μ=(F,F,F,F,F,F,F,F,F) n'en fait pas partie.**

Les prédicats: des fonctions particulières

- **Un prédicat : fonction associant un booléen à n-uplet de paramètres p1, …pn de types respectifs T1,… Tn**
- **Le nombre de paramètres d'un prédicat est appelé l'arité du prédicat**
- **Opérateur de comparaison : <,>,==**
  - Des prédicats déguisés :
  - < : int,int -> {V,F}
  - > : réel, réel -> {V,F}
  - ….
- **Tous les opérateurs ne sont pas des prédicats : +, *,/…**
- **Une formule contenant des prédicats verra ses valuations définies sur le types des paramètres des prédicats**

Exemple

- **Variables typées : x : entier, y : booléen, z: ensemble d'entiers**
- **Opérateur : |r| retourne le cardinal de r (si r est un ensemble**
- **Formule :**
  1) x<4,
  2) y=> |z|<1
  3) (y>2)=>x

Formules logiques avec prédicats

- **Exemple : F = (x<1) => y ≠ 0**
  - Problème on ne connaît pas le type de x et y
  - x et y sont dit des variables **libre,** elle peuvent prendre n'importe quelle valeur la formule a un sens différent pour les entiers, les réels…
- **La quantification : détermine l'ensemble de valeur du type de la variable pour lesquelles la formule doit être vraie**
  - $F = \forall x \in X.\ p(x)$: signifie que F est vraie si pour tout x dans X le prédicat p(x) est vrai
  - $F = \exists x \in X.\ p(x)$: signifie que F est vraie si il existe un x dans X tel que le prédicat p(x) est vrai

Satisfiabilité et Validation d'une formule

- **La satisfiabilité et la validation d'une formule F sont des propriétés qui s'exprime par rapport à un ensemble de valuations X pour un ensemble variables V.**
- **Une formule est dite satisfiable dans X si il existe au moins une valuation f des variables de V dans X telle que F est évaluaée à vrai pour la valuation f**
- **Une formule est dite valide dans X si pour toute valuation de X, alors la formule F est évaluaée à vrai.**
- **Si X représente l'ensemble de toutes les valuations possibles de V alors,**
  - on dit que F est une tautologie si elle est valide dans X (c'est tout le temps vrai)
  - On dit que F est une contradiction si F n'est pas satisfiable dans X.

Utilisation de la logique pour caractériser les états défaillants

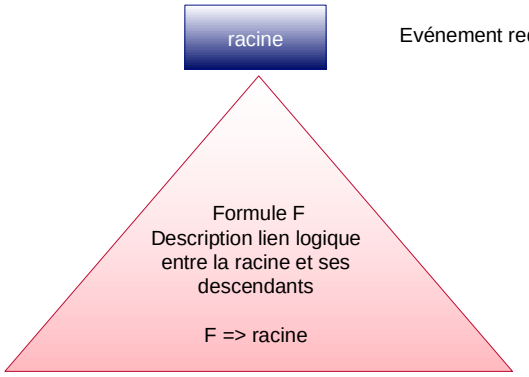
- **Logique : mathématique == langage**
- **identifier les événements redoutés, et leurs causes possibles par des formules**
  - Etat binaire des composants du système) ok/ko
  - Des formules pour décrire les lien causes=>conséquences
  - Des méthodes d'analyse de modèles d'architectures pour construire ces formules pour + de couverture (discriminer les bons états)
- **Utilisation de prédicats pour identifier les événements redoutés : (hors du périmètre du cours)**

Arbres de Fautes

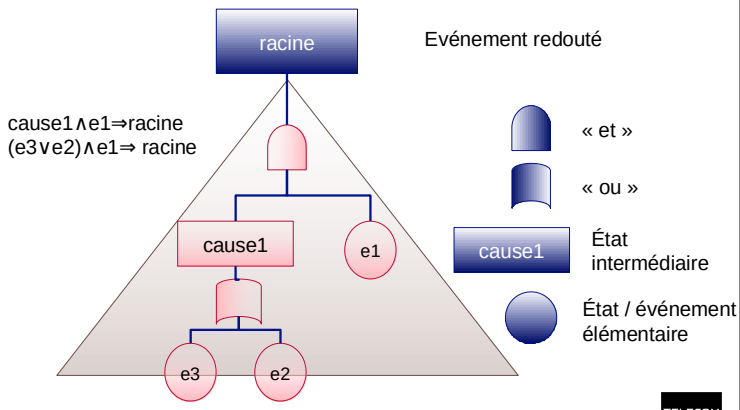
- **Idée : se concentrer sur la relation de causalité**
- **Pré-requis : connaître les événements redoutés d'intérêt**
- **Principe :**
  - Définir l'événement racine de l'arbre (doit être un événement dont on cherche à déterminer les causes)
  - Identifier de manière exhaustive les causes logiques de l'événement
  - Relier ces causes par un connecteur définissant leurs interactions
    - Et
    - Ou
    - Ou exclusif
    - …

Un arbre de faute = formule logique sur l'état du système avec identification des variables/formules déterminant l'état complet

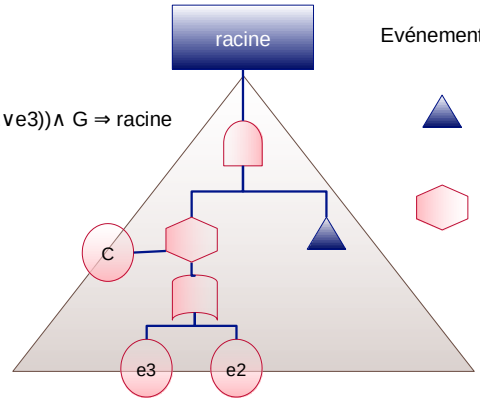
Syntaxe & sémantique : arbres de fautes



Syntaxe & sémantique (basique) : arbres de fautes



Syntaxe & sémantique (avancé) : arbres de fautes



Evénement redouté

Référence à un arbre de faute

Porte conditionnelle : si C est faux => faux sinon l'entrée est propagée

Analyse qualitative de l'arbre

- **La fonction de structure : SF (x_1,…., x_n)**
  déduite de l'arbre telle que SF(x_1,…., x_n) ⇒ racine
- **Fonction de l'état des feuilles de l'arbre**
- **Hypothèse :**
  - ordre partiel sur les valuations de X=(x_1,…., x_n) dans B^n (vecteur de booléens)
  - Règle 1 : Vrai > faux
  - Règle 2 : Soit x et y deux valuations de X

    $x < y \Leftrightarrow \forall i \in \{1,..,n\}, \exists j, (xi < yi \vee xi = yi) \wedge xj < yj$

- **Exemples : (vrai, faux , vrai) >(faux, faux, vrai)**
  mais (vrai, faux, vrai) et (faux, vrai, faux) non comparables

Coupure minimale d'un arbre

- **Un état amenant à rendre vrai l'événement redouté (ER) tel que tout événement feuille retiré empêche l'événement de se produire => explication minimale de l'ER**
- **Formellement :**
  X0 est une coupure minimale de SF(X) si quelque soit Y<X SF(Y) est faux, et SF(X) est vrai.

Analyse quantitative sur l'arbre calcul de vraisemblance

- **Idée : associer une probabilité aux feuilles**
- **Pr(racine)= Pr(SF(X)).**
- **Comment calculer Pr(SF(X)) ?**
- **Il existe une règle de transformation**
  Pr(SF(X))= G(Pr(x1),…, Pr(xn)) si les feuilles correspondent à des états indépendants.
- **Mise en œuvre : « et » □ *, « ou » □ +, not(x) □ 1-Pr(x)**
- **Exercice calculer la probabilité de l'exemple « basique » transparent**

Conclusion

- **Ce qu'il faut retenir :**
  - **Définition risque/ incident/ faute / erreur défaillance**
  - **4 grandes approches pour arriver à avoir un système sûr de fonctionnement (prévention/élimination/tolérance et évaluation).**
  - **Les 5 attributs de la SdF + différence exigence fonctionnelle ou non**
  - **Arbre de fautes (et rappels de logique)**
  -
- **La suite : la tolérance aux fautes.**

Quelques mots sur l'ingénierie système motivation (suite)

- **Problèmes :**
  - Que doit contenir la description du service attendu ?
  - Comment identifie-t-on un dysfonctionnement dans les phases de conception (a priori) ?
  - Comment détermine-t-on ses conséquences, ses causes ?
- **Ingénierie Système et sûreté de fonctionnement :**
  - Description du système (objectif et implémentation)
    - Analyse de la cohérence de la description
    - Étude de sa faisabilité
  - Prise en compte d'exigences spécifiques à la sûreté de fonctionnement issues des décision sur l'objectif de limitation du risque (à un moment, il faut décider si l'on contrôle la vraisemblance ou si on altère les conséquences d'un incident).
  - Guider le processus de développement, de validation, mais aussi décrire la vie opérationnelle du système.

Une parenthèse sur le rôle des probabilités

- **Probabilité : caractérisation d'un phénomène aléatoire**
  - Moyen de représenter une incertitude maitrisée
  - Moyen de capturer une caractéristique d'une population
- **Variable discrète : probabilité** $P(x=V)$
- **Variable continue : distribution de probabilité (Probability density function)**

Tolérance aux Fautes des Systèmes Informatiques

Thomas ROBERT
2019

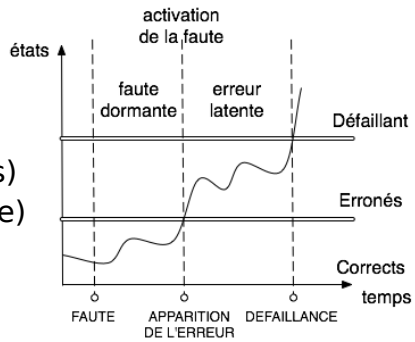
Plan du cours

- Tolérance aux fautes sur les données
- Tolérance aux fautes des traitements version «logicielle »
- Tolérance aux fautes des traitements version « matérielle »

Rappel : La chaîne faute/erreur/Défaillance

- Evénements :
  - Activation
  - Détection
  - Défaillance
- Etats :
  - Fautifs (non activés)
  - Corrects (sans faute)
  - Erronés (?)
  - Défaillants (KO)

Pas de détection == erreur dormante jusqu'à la défaillance …



Le modèle de faute I

- Une faute == cause adjugée ou effective d'une erreur et donc d'une défaillance

modèle de faute=caractériser la faute

- Quelques exemples :
  - **Corruption de la mémoire physique** -> les valeurs peuvent être modifiées aléatoirement — environnement
  - **Faute de développement**, pointeur mal initialisé -> donnée incorrecte / boucles infinies — Production
  - **Défaillance du matériel ou sous-système** -> arrêt complet du système, comportement aléatoire — Interaction

Le modèle de faute II

- Capturer leurs caractéristiques :
  - Phase de création ou activation
    - Modèle d'activation : déterministe / aléatoire
    - Phase de création : développement/opération
  - Positionnement % système (logicielle/matérielle) (interne externe)
  - Source (humaine / « naturelle »)
  - Persistance (permanente/transitoire)

=> Permet de déterminer la stratégie adaptée / permet de créer les modèles d'évaluation ....

Structure hiérarchique et systèmes de systèmes

- La structure d'un système :
  - Hiérarchie
  - Dépendances matériel / logiciel
  - Description des interactions aux interfaces
- Principe de propagation :

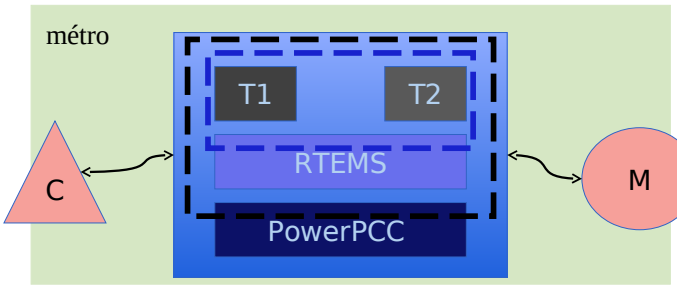
  La défaillance d'une partie d'un système peut devenir une faute pour le reste du système
- La définition d'une architecture aide à raisonner (un des intérêts des ADL)

Défaillances, fautes et propagation des erreurs

- Fautes : interactions (propagées à travers l'interface)
- Propagation des erreurs : capteur (C) -> régulateur (R) -> moteur (M)
- Erreur dans C -> défaillance de C -> faute pour R -> erreur dans R

On peut « entrer » dans R…

Propagation : Logiciel/Matériel



- Décomposition du boitier en éléments relativement indépendants
- La défaillance d'une tâche peut altérer le fonctionnement de RTEMS déclenchant une altération du matériel (effacement du bios

⇒Propagation interne au boitier du logiciel vers le matériel

C'est très [illegible] raisonner par type de fautes, pas à pas.

Le principe de la TaF

- Empêcher les défaillances :
  1. Éviter l'activation des fautes (différent de l'élimination au développement)
  2. Éviter qu'une erreur n'entraîne une défaillance (tolérance aux fautes)

- 1) Traitement des fautes,
  2) Traitement des erreurs
- Comment faire 2) ?
  - Détection & Rétablissement
  - Masquage (ou compensation)
- Cela va avoir coût

Principes fondamentaux

- Avoir des composants aux défaillances maitrisées
- Gérer la propagation des défaillances internes
- Mise en œuvre :
  - Connaître l'état du système
  - Savoir traiter / maitriser un état erroné
  - Comprendre l'origine des erreurs pour la maintenance

Zone de confinement des erreurs

- **Définition**
  périmètre/interface d'un système muni de mécanismes de protection empêchant une erreur de se propager sans être au moins détecté et signalée
  (ie de contaminer l'environnement du système)
- En pratique :
  - Description architecturale définissant la décomposition du système en sous-systèmes dépendant les uns des autres
  - Description des défaillances possibles associées au système et chaque sous-système.
  - modèle de fautes (apparition des erreurs) et de leur propagation
- Mise en œuvre=contrôle aux interfaces, modifications architecturales / comportementales internes

Traitement des Fautes

- **Détection (diagnostic)** : déterminer la cause première d'une défaillance. (ex. régulateur)
- **Isolation** : relier la plus petite partie du système à la faute cause de la défaillance
  Détermination d'une zone de confinement
- **Reconfiguration** :
  altération de la structure et de la logique du système pour inhiber la faute

Traitement des Erreurs

- **Détection**
  - Test de vraisemblance : erreur si Observé ≠ Attendu
  - Exécution multiple + Comparaison, erreur si V1≠V2
- **Recouvrement :** Démarche de correction de l'erreur par
  - Redéfinition de l'état ou reconstruction
  - Compensation de l'état erroné (cf redondance)
- **Utilisation de la détection**
  - Signalement/journalisation (exception Java)
  - Synchronisation d'action de Recouvrement

Tolérance aux fautes sur les données

Quel est le problème

- Donnée stockée hiérarchiquement % différents niveaux d'abstraction
  - Fichier
  - Bloc de k octets
  - Dispositif de stockage de masse

Solution orientée par le besoin de disponibilité + modèle de faute

Le cas traité

- Séquence de bits de taille fixe
  faute = modification aléatoire d'un bit de la séquence
- Séquence de bits de taille fixe

faute = modification de n bit contigues

- Solutions vues : codage par bloc Hamming + principe de dissémination

Abstraction du codage

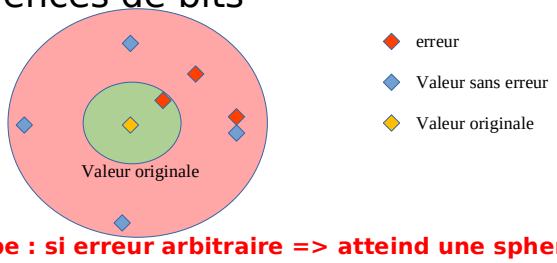
- Codage (binaire) : représentation sous forme de séquence de bits d'une information

D (information) => mot de code C tel que C =G(D) une séquence de n bits, n > m (G fonction génératrice)

- Soit V un ensemble de valeurs distinctes de taille 2^m => m bits au minimum pour son codage

Détection vs Correction I

- 1 une information codée = 1 point
- 1 distance pour comparer des séquences de bits



Principe : si erreur arbitraire => atteind une sphere autour de la valeur originale
Pb : si la sphère contient un mot de code pas de détection

Détection vs Correction II

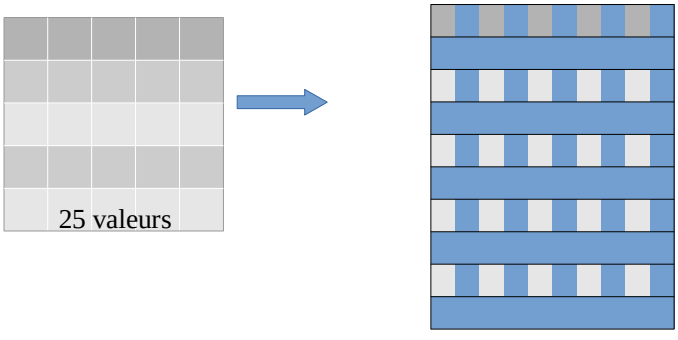
- Distance de Hamming ($d_H(v,v')$) = nombre de bits distincts entre deux séquences de tailles identiques.

$d_H(0101,1100)=2$

- Distance d'un codage : minimum de la distance de hamming sur l'ensemble des mots de code obtenus par G (fonction génératrice). (noté dmin(G)).
- Détection de k erreurs si dmin(G)> k (néc.)
- Correction si dmin(G) > floor((k-1)/2) (suf.)

Codage pour la détection

- Visuellement



Principe le cas linéaire (mathématisé – opt.)

Vecteur de donnée (ex 4 bits)
X=(0100),
codage par G matrice
G.X= (0010110) =,Mc
Mc’ : mot potentiellement fauté
Décodage par matrice H
H.Mc’= 0 => pas de faute
H.Mc’ ≠ 0 => faute → reconstruction

Alternatives – la localité

- Idée : les altérations sont peu fréquentes mais groupées > taille d'une donnée élémentaire
- Codage de x dans V : x1, …. , xn
- **Pb si u > n bits modifiés ?**
- Créer K vecteurs de taille l ≤ n tels que V_i contient un sous ensemble des « coordonnées » de x (mélangé avec d'autres messages peut être)

- Propriété : il suffit de disposer de suffisamment de copies intègres pour retrouver x.

Mise en pratique

- Soit un Message avec 45 bits utiles, et 19 bit de redondance (supposons le code parfait: $d_H(C1, C2) \geq 19$
- Avant émission on accumule 8 Messages C1, C2 … C8 de façon à tranférer le premier octet de C1, puis C2 … jusqu'à C8 (puis on passe au second octet)
- Combien de bit altérés successif tolère t on ? (nb. 64Bits= 8 octets).

Tolérance aux fautes logicielle

Génie logiciel et TaF

- Les activités:
  - Identification / classification des défaillances
  - Conception / mise en œuvre des zones de confinement
  - Vérification des comportements/performances
- Les moyens
  - Des modèles de fautes (hypothèses de travail)
  - Des designs patterns (architectures)
  - Des modèles de performances (chaînes de markov)

TaF logicielle ≠ logiciel pour la TaF

- TaF logicielle => zone de confinement au niveau LOGICIEL (ce doit être justifié)
- **Pré-requis** :
  connaître les modes de défaillance du logiciel et leur propagation au matériel
- **Exemple au tableau sur une application Java hébergée sous Unix**
  - **OutOfBoundException**
  - **NullPointerException …**

Confinement logiciel quelle zone (de confinement) ?

- Défaillances concernées :
  tout ce qui ne compromet pas le matériel mais altère l'exécution
  - Valeurs incorrecte sur les interfaces des fonctions
  - Comportement aberrant de l'ordonnancement
  - Corruption de l'intégrités des donnée de programme
- Exemple : process Unix & sigsev
  - 1 process unix
  - 1 accès illicite à la mémoire du noyau (pointeur null)
  - Détection par la MMU (hw), et signalement au processus
  - Le processus se termine en indiquant au système d'exploitation (env du process) la cause de l'arrêt (SIGSEV)

Confinement par fonction

Idée : une fonction d'une bibliothèque = ZCE

Pb : comment signaler / confiner les erreurs.

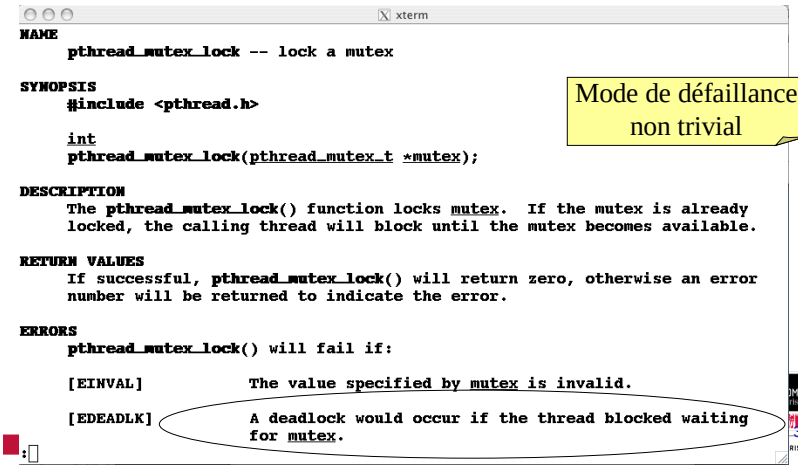
- Pré-requis: taxonomie des erreurs
- Mise en œuvre :
  - encodage de l'état fonctionnel du composant dans la valeur de retour des fonctions
  - altération de la signature
- Pb : surcharge du type de retour ou altération de sa signature
- Exemple : code retour en C

Confinement par blocs

- Les exceptions :
  - Pré-requis :
    - encapsulation des traitements séquentiel dans des « blocs » (fonctions, accolades),
    - arrêt et déroutement de l'exécution d'un bloc sur réception d'un événement
    - taxonomie des erreurs + support pour le déroutement d'exécution
  - Composants : blocs → méthodes, fonctions, boucles
  - Mise en œuvre :
    - Utilisation du typage : 1 type d'exception=1 classe d'erreur
    - Définition d'une politique de propagation des signalements en l'absence de traitement explicite

Pb : souvent très mal utilisé malgré la puissance.

Décryptons une page de man



Le principe de l'enveloppe

- **Soit T1 f(Tp1,...Tpn) une fonction pouvant défaillir à cause de fautes d'interaction (mauvais paramètres).**
- **Pb : f ne peut signaler son état de fonctionnement….**
- **On crée ft_t**
  - **Type de retour status de fonctionnement**
  - **Utilisation d'une référence (en valeur) pour la valeur de sortie de f**
  - **Test des paramètre avant appel à f**
  - **Test de la valeur retournée par f**

L'enveloppe dans le détail

- **Soit T1 f(Tp1,...Tpn) une fonction pouvant défaillir à cause de fautes d'interaction (mauvais paramètres).**
- **Solution (en C, mais transposable)**

fonction ft :

  - **Tsig ft(Tp1,….,Tpn,T1*)**
  - **Tsig type utilisé pour l'état de fonctionnement**
  - **T1 * adresse vers emplacement du résultat**
  - **Lors d'un appel à ft, ft appelle f.**
  - **Valeur de retour de f copiée à l'adresse désignée par le dernier paramètre**

Confinement par moniteur externe

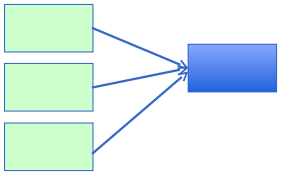
- Watchdog et interruption logicielle:
  - Défaillance constaté par l'absence de progrès dans l'exécution d'un programme
  - Causes possibles : boucle infinie, blocage sur un verrou pris et jamais restitué, récurrence trop longue ...
  - Mise en œuvre : alarme, plus mécanisme de raz à des points critiques du code.

Recouvrement des erreurs 2 visions complémentaires

Corriger avant de poursuivre (on corrige l'erreur)

Recouvrement

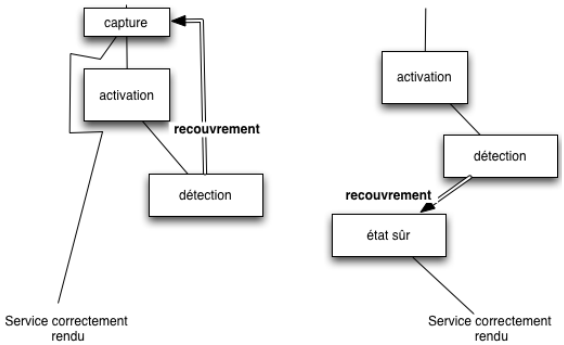
Choisir pour poursuivre (pas de correction nécessairement)



Masquage

Détection et recouvrement

- L'exécution du système est une séquence d'états
- **Détecter** la 1ere transition vers un état erroné
- **Reprendre** l'exécution depuis un état « correct »



Détection et recouvrement

Problème où trouver l'état correct ? :

- Dans le passé :
  - Hyp : il existe un mécanisme de capture d'états qui mémorise de manière répétée un état correct « récent »
  - Restauration sur détection d'erreur du dernier état sauvé
- Dans une liste d'états prédéfinis :
  - Hyp : Identification, a priori, d'états de poursuite d'exécution sûrs en fonction de l'erreur
  - Forçage d'une transition vers l'état de poursuite d'exécution correspondant à l'erreur détectée

Réflexions sur l'usage du recouvrement

- Dans l'approche « arrière », échec si :
  - la **faute est toujours active** et cause systématiquement l'erreur
  - la **latence de détection** est si grande que **l'état sauvegardé contient déjà une faute dormante ou que l'erreur s'est déjà propagé à l'extérieur du composant**
- Dans l'approche « avant » echec si
  - Les états de poursuite sûr sont erronés (mauvaise évaluation du lien erreur – état sûr).
  - La faute sous-jacente et toujours active et cause à nouveau l'erreur
- Comparaison sur une faute causée par un bug :
  - Le recouvrement arrière a de fortes de chances de rater si le bug est persistent
  - L'activation des états sûrs permet d'appeler un autre code.

Masquage et exécutions multiples

- Rappel : masquage possible car plusieurs instances de processus réalisant la fonction (e.g. le calcul)
- 2 visions : Vrai et faux parallélisme
  - Faux parallélisme == N instances exécutées en séquence
  - Vrai parallélisme == N instances exécutées en parallèle avec diffusion des données et récupération du résulat.

Intérêt de la redondance

- Pb: comment s'assurer après recouvrement que l'erreur ne revienne pas …
- Raisonnement alternatif « l'indépendance » :

  Il est peu très peu probable qu'une même faute s'active sur deux exécutions indépendantes d'une même fonction
- La redondance ~ création de données, de composants, de « résultats indépendants »

Intérêt de la redondance

- La redondance permet de masquer les erreurs
  - Vote-élection / reconstruction / moyenne
    - Consensus
    - Code correcteurs d'erreur
    - Capteur de pression consolidés
- Problème : comment caractérise-t-on l'indépendance ?
  - Physique : réplication et séparation (COM-MON)
  - Processus : développement diversifié (NVP

Design pattern

- Architecture flexible pour la SdF
- Approches :
  - Programmatiques (syntaxe et enchainement)
  - Architecturales (modèles de flux et indépendance)
- On peut souvent combiner les deux…

N-version programming (process)

- L'idée est la même :
  avoir N versions indépendantes d'un même système :
  - 1 seule spécification
  - N équipes indépendantes de développement (lieu, formation, hiérarchie …)
  - ⇒ Une faute a peu de chances de s'activer de la même manière dans 2 versions distinctes

La transformée de fourrier discrète :

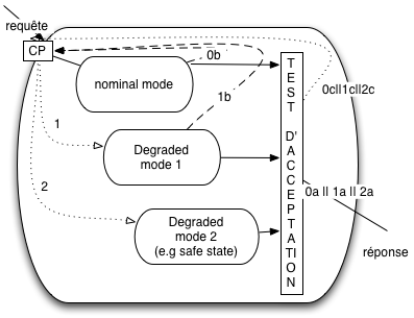
⇒ 2 algorithmes pour la calculer avec une sensibilité différente aux erreurs numériques

Recovery Blocks [Randall'95]

- Exemple d'intégration de NVP dans une application
- Un RB est un composant implémentant un service à la demande (client/server)
- La fonction est implémentée de N manières distinctes A1…AN
- Chaque version peut elle-même être un RB
- Il existe un test d'acceptation que doit pouvoir passer chaque alternative en l'absence d'erreur

Un exemple d'intégration de NVP : les Recovery Blocks

- A l'exécution :
  - Chaque requête génère la capture d'un point de reprise
  - La requête est transmise au premier alternat.
  - Si erreur ou échec du test, alors rétablir l'état du système et passer au bloc suivant
  - Dimension temporelle (watchdogs, timeouts)



CP : checkpoint; 1a,1b,1c chemins d'exécution possibles

Capture de contexte … et en vrai ?

- Principe :
  - Déterminer l'information représentative de l'état d'un système (variable, pile, ports de communication …)
  - Déterminer une méthode pour capturer un état cohérent de ces données
  - L'information enregistrée doit être suffisante pour recommencer l'exécution du système depuis cet état
- Obstacles
  - Usage de ressources systèmes et **effets de bords** (réservations de ressources, communications en cours)
  - **Implémentations concurrentes** du système
  - Quand doit-on capturer l'état ?

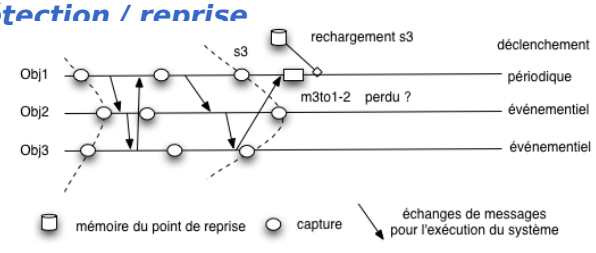
La vision centralisée

- Capture d'état == capturer l'état d'une machine, d'un processus
- L'approche totale : recopier l'état de l'application et de sa plateforme d'exécution
  - Connaître l'OS (process / E-S / verrous)
  - Connaître le matériel (configuration des périphériques …)
- L'approche sémantique : identifier des états dans lesquels l'information utile est très faible
  - Connaître l'application à 100%

Le cas réparti et l'état inconsistant

- Principe : cas de figure désavantageux pour la capture d'états concurrents

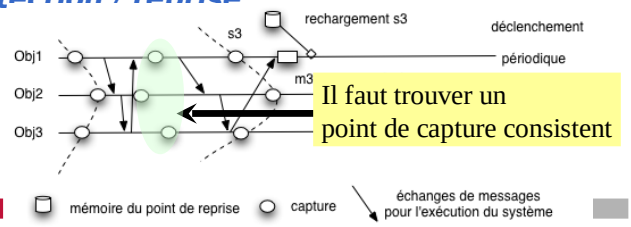
Le délai de transmission des messages fait qu'un site croit avoir transmis une donnée alors qu'un autre l'a ignorée à la suite d'une détection / reprise



Le cas réparti et la cause de l'effet domino

- Principe : cas de figure désavantageux pour la capture d'états concurrents

Le délai de transmission des messages fait qu'un site croit avoir transmis une donnée alors qu'un autre l'a ignorée à la suite d'une détection / reprise



Tolérance aux fautes matérielle

Intérêt de la redondance matérielle

- La redondance logicielle sans redondance matérielle suppose que le matériel est isolé du logiciel -> cela reste difficile à prouver
- La redondance matérielle permet de rendre un système informatique tolérant aux défaillances du logiciel sans isolation particulière
- Le système est modélisé comme système distribué : 1 ensemble de calculateurs communicant par message

Modèle de fautes dans un système distribué

- 4 gabarits classiques de fautes
  - Silence (perte définitive du service)
    Arrêt du matériel/blocage de l'OS
  - Omission (perte occasionnelle du service)
    lien réseau peu fiable ou timing très mauvais
  - Temporelle (mauvais timing)
    mauvaise estimation de WCET
  - Byzantine (service totalement incontrôlé)
    modélisation classique pour la sécurité
- Haut niveau d'abstraction
  système=réseau

1 faute == défaillance d'un site de calcul

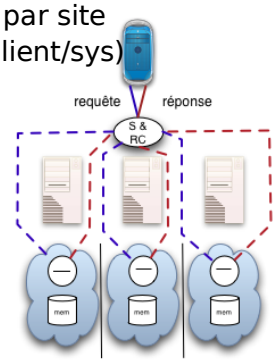
Stratégies de réplication

- Principe :
  - Déployer de manière concurrente plusieurs fois la même fonction
  - Utiliser un contrôleur pour
    - Piloter l'exécution de ces répliques
    - Assurer la transmission du résultat
- 3 gabarits (motifs de conception) :
  - Réplication active
  - Réplication passive
  - Réplication semi-active

Comment et avec quelles ressources ?

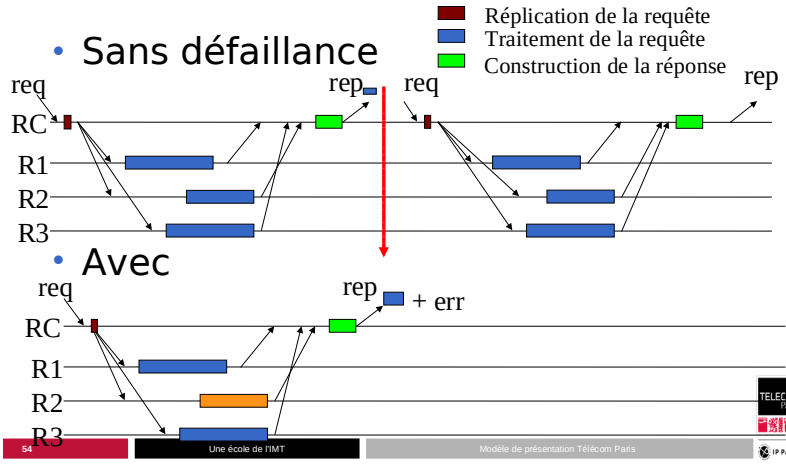
Réplication Active

- N calculateur => 1 application par site
- 1 contrôleur pour interpréter (client/sys)
- Décision: vote
- Communications : diffusion des requête + agrément sur le résultat
- Déroulement
  1) Envoyer la requête à tous
  2) Chaque site exécute son service
  3) Construction de la réponse par vote majoritaire

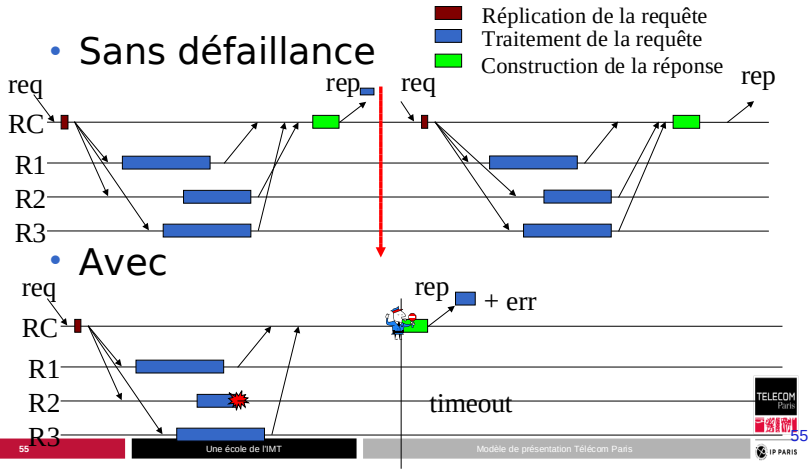


S & RC : vote et contrôle des répliques

Communication sans et avec défaillance



Communications sans et avec défaillance



Caractéristiques de la réplication active

- Plusieurs répliques actives en même temps Il faut définir la condition de disponibilité :

Quand produit t on une valeur ?

- Cas n°1 : 2 réponses identiques parmi 3 doivent être reçues
- Cas n°2 : dès que l'on reçoit une valeur on la transfère si c'est la seule reçue, sinon on applique n°1.
- Cas n°1 dispo = fiabilité, Cas n°2 c'est différent ! (i.e. on préfère produire un résultat même si il peut être faux.

Caractéristiques de la réplication active

Modèle de faute toléré pour N répliques :

N/2-1 fautes Byzantines, N-1 crash

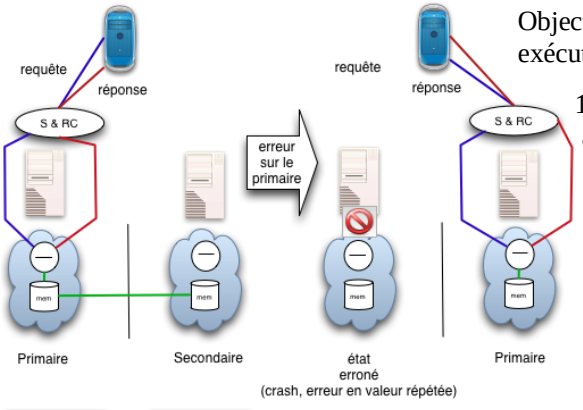
Détection réussie si au moins 1 correct

Pour qu'une faute entraîne une erreur non détectée, elle doit s'activer sur :

le voteur ou sur plus de N/2 répliques

Δt(régime normal / rétablissement) ~ nul

Réplication Passive



Objectif : éviter les exécutions concurrentes

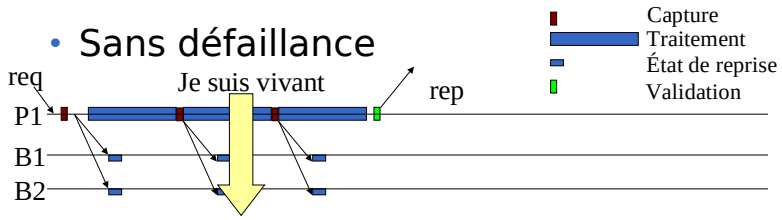
1 seule exécution à la fois

Communication : Transfert d'un état

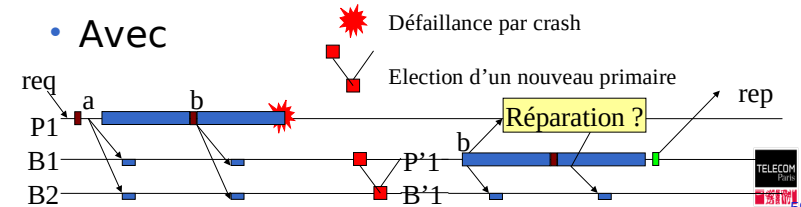
Le primaire doit capturer son état pour l'envoyer au secondaire

Communications et opérations significatives

- Sans défaillance



- Avec



Caractéristiques de la réplication passive

- 1 seul site exécute le service, jusqu'à la détection d'erreur (défaillance du primaire)
- Après détection d'erreur, basculement vers une réplique (nouveau primaire)
  - Identification du crash par « battements de cœur »
  - Élection d'un nouveau primaire
- Hypothèse forte : le primaire n'a qu'un seul mode de défaillance, le crash.
- Sans réparation la fiabilité service est identique à priori à la disponibilité service

Analyse : Caractériser l'état de fonctionnement

- Chaque réplique à un état de fonctionnement
  - Ok / byzantin
  - Ok / wrong result
  - Ok / crash
- On peut définir : disponibilité / fiabilité % répliques
- Il doit être possible de caractériser l'état de l'architecture (dispo / fiabilité) par une formule

Caractériser l'état un exemple

- Triplication (3 répliques) active
  - Cas 1 : 1 mode de défaillance - byzantin
  - Cas 2 : 1 mode de défaillance – crash
- Soit N le nombre de répliques défaillantes
- Caractériser dans chaque cas : l'état de disponibilité puis fiabilité

Hypothèse:

- le système est considéré disponible tant qu'il produit un résultat.
- Le voteur ne réalise pas de filtrage.

Modes de fonctionnement dégradés

- Modes dégradés :
  État du système tel qu'une partie du système est dysfonctionnelle sans pour autant compromettre l'intégralité du service
- Mode fail-safe: mode dégradé n'assurant aucun service mais garantissant la sûreté des biens et personnes

Conclusion

- Tolérance aux fautes = domaine établi
  - Des motifs de conception utilisés mais à adapter à chaque application
  - Pas de dogme mais une prise de conscience
    - La Sûreté de fonctionnement est difficile à obtenir
    - Il y a nécessairement des compromis à faire mais « les bons »
- Importance des zones de confinement
  - Savoir si les défaillances sont détectées/signalées
  - Lister les modes de défaillances connus, indiquer la cause si externe

Acronymes

- SdF : sûreté de fonctionnement
- TaF : Tolérance aux Fautes
- TMR, NMR : triple/ N - moduar replication
- RB : recovery blocks ou Rollback...
- ND : non - déterminisme (ou non déterministe)
- SR : stratégies de réplication

Références

Concepts de la SDF :

- PA Lee, T Anderson, JC Laprie, A Avizienis, ... - 1990 - Springer-Verlag New York, Inc. Secaucus, NJ, USA
- A. Avizienis; J. Laprie; B. Randell & C.E. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing“, IEEE Transaction Dependable Sec. Computing 2004, 1, 11-33
- J.C. Laprie, “Guide de la sûreté de fonctionnement (2° Ed.)“, ed. Lavoisier, 330p

Techniques de mise en place de la TaF

- B. Randel and J. Xu, "The Evolution of the Recovery Block Concept," Software Fault Tolerance, M.R. Lyu, ed., John Wiley & Sons, New York, 1995, chapter 1
- Elnozahy, E. N., Alvisi, L., Wang, Y., and Johnson, D. B. 2002. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. 34, 3 (Sep. 2002), 375-408. DOI= http://doi.acm.org/10.1145/568522.568525
- [Cristian91] F. Cristian, "Understanding fault-tolerant distributed systems", Communications of the ACM, 34(2), February 1991
- [KD+89] Kopetz, Damm, Koza, Mulazzani, Schwabl, Senft, Zainlinger. "Distr[redacted]ime systems: the Mars approach", IEEE Micro, pp. 25-40, February 1989

Références

- Xavier Défago and André Schiper, “Semi-passive replication and lazy consensus“, Journal of Parallel and Distributed Computing, 64(12):1380–1398, December 2004.
- Koo, R. and Toueg, S. Checkpointing and rollback-recovery for distributed systems. In Proceedings of 1986 ACM Fall Joint Computer Conference (Dallas, Texas, United States). IEEE Computer Society Press, Los Alamitos, CA, 1150-1158, 1986.
- Powell, D. 1994. “Distributed fault tolerance—lessons learnt from Delta-4“. In Papers of the Workshop on Hardware and Software Architectures For Fault Tolerance : Experiences and Perspectives: Experiences and Perspectives, M. Banâtre and P. A. Lee, Eds. Springer-Verlag, London, 199-217.

Algorithmique distribuée et prise de décision :

- Lamport, Leslie; Marshall Pease and Robert Shostak, "Reaching Agreement in the Presence of Faults". Journal of the ACM 27 (2): 228--234, April 1980
- Xavier Défago and André Schiper, “Semi-passive replication and lazy consensus“, Journal of Parallel and Distributed Computing, 64(12):1380–1398, December 2004.
- Chandra, T. D. and Toueg, S. 1996. Unreliable failure detectors for reliable distributed systems. J. ACM 43, 2 (Mar. 1996), 225-267.

Conception de stratégies de réplication :

- Schneider, F. B. 1990. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. 22, 4 (Dec. 1990), 299-319.

TELECOM ParisTech

Une école de l'IMT

Analyse quantitative et prévision des fautes

Institut Mines-Télécom

Contexte

- **1 système avec potentiellement des mécanismes de tolérance aux fautes**
- **Ce que vous savez déjà faire normalement**

  Identifier des limites dans le type de fautes et leur nombre telles que le système reste fiable, ou peut revenir dans un état fonctionnel.
- **Ce que l'on voit aujourd'hui :**
  - Modéliser l'incertitude sur l'occurrence des fautes via des probabilités
  - Modéliser le cycle de vie complet du système via un processus stochastique
  - Utiliser des modèles « sur étagère » pour faciliter certaines analyses (Chaînes de Markov, et PTA)

Rappel de probabilité

- **Une variable aléatoire est définie pour un domaine D, la variable en elle-même est une fonction de certaines parties de D vers un espace probabilisé.**
- **Si votre variable a un nombre fini de réalisations, la fonction attribuant une probabilité à chacune est la distribution de masse**
- **Si votre variable a un nombre infini de réalisations, on parle de distribution de probabilité**
- **Deux lois sont cruciales de notre point de vue : le théorème de Bayes, et la somme des événements disjoints.**

Occurrence d'une défaillance

- **Ce que l'on va modéliser :**
  - Système S, défaillance f
  - 2 cas :
    - Le système peut défaillir de manière asynchrone à l'usage
    - Le système ne peut défaillir que lorsqu'il est sollicité
  - 2 caractérisations
    - Probabilité de défaillir par demande
    - Distribution des dates de défaillances

Occurrence de défaillance à la demande

- **Modélisation triviale : Variable aléatoire booléenne indépendante de tout autre phénomène**
  - Caractérisation : 1 paramètre pour la loi de Bernouilli
  - Fiabilité = 1-p
  - Défaillance = p
- **Modélisation plus avancée : exploiter la structure de sorte que l'occurrence d'une défaillance est caractérisée par pred (X1,…..Xn) où pred est un prédicat et X1,….Xn sont des variables aléatoires.**

Occurrence de défaillance « asynchrone »

- **Principe : X variable aléatoire réelle (positive) telle que X caractérise la date d'occurrence de la prochaine défaillance à partir de la date 0.**
- **Rappel : X peut être vue comme la caractérisation d'un processus d'arrivée. Ainsi, on peut définir une séquence X1,….Xn décrivant les différentes dates successives de défaillance (si cela fait sens)**
- **Le cas sympathique de la loi exponentielle**
  - Modélise des temps d'arrivée « sans mémoire »
  - Pas besoin de se souvenir de la date 0
  - 1 seule paramètre pour la caractérisation.

Rappel sur la loi exponentielle

- **Pdf :**

$f(t)=1/\lambda * \exp(-t/\lambda)$

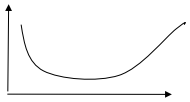
- **Espérance :**

$1/\lambda$

Idée : la probabilité qu'une défaillance se produise instantanément (ie entre t et t+h pour h tendant vers 0, c'est constante égale à λ).

Le cas des loi à mémoire : Weibull

- **Pb : faire l'hypothèse d'un taux constant s'applique assez mal au matériel**



- **Taux d'occurrence variable → courbe en baignoire**
- **Weibull : variation du taux de défaillance contrôlée par 3 paramètres… je n'en dirai pas plus car ces modèles sont propres aux matériel…**

Prise en compte de mode de fonctionnement / du recouvrement de l'état défaillant

- **Problème :**

un système tolérant aux fautes possède différents états de fonctionnement

→ **différentes défaillances**

→ **Différents taux de défaillance**

Solution

Modélisation par processus stochastique (vision séquentielle)

Modélisation par vecteur d'états (vision hiérarchique)

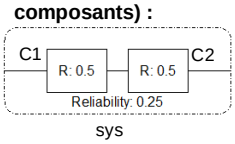
Modélisation mixte : vecteur de processus stochastiques

Propositional random formula et reliability block diagram

- **Question : peut on se donner un modèle graphique proche de l'architecture pour modéliser ces situations**
- **Idée : proposer des structures hiérarchisables et composables + une méthode d'évaluation de fiabilité**
- **Solution Reliability Block Diagram (RBD)**
- **http://pages.mtu.edu/~pjbonamy/rbdtool.html**
- **Principe :**
  - 1 block = 1 système avec un état binaire : fiable / non fiable
  - Une loi de probabilité pour la défaillance (Bernouilli paramètre p)
  - Des motifs composites de blocs : série, parallèle, K out of N….

Motifs de base, sémantique et évaluation la série

- **Le bloc série (2 à n composants) :**



sys

- **Signification :**

Le système requiert C1,C2 pour être fiable

- **Formalisation**

si fx état booléen / vrai => x fiable

fsys = fc1 & fc2

- **Interprétation numérique**

R(C1)= P(fc1), R(C2)=P(fc2)

R(Sys) = P(fc1)*P(fc2)

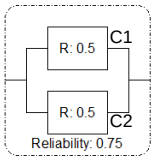
- **Hypothèse sous jacente**

Indépendance des défaillances de C1 vs C2

- **Pb possible : un même composant ne peut apparaître à deux endroits**

Motifs de base, sémantique et évaluation la composition parallèle

- **Le bloc parallèle (2 à n composants possibles):**



sys

- **Signification :**

Le système requiert C1 ou C2 pour être fiable

- **Formalisation**

si fx état booléen / vrai => x fiable

fsys = fc1 ou fc2

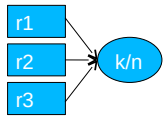
- **Interprétation numérique**

R(C1)= P(fc1), R(C2)=P(fc2)

R(Sys) = P(fc1)+P(fc2)-P(fc1)*P(fc2)

Motifs de base, sémantique et évaluation le K parmi N

- **Le bloc série :**



- **Signification :**

Le système requiert
X={Cj| Cj fiabl} / Card(X) ≥ k

- **Formalisation**

(peu d'intérêt)

Remarque : il y a une théorie sous jacente pour automatiser les calculs

Chaque structure est associée à une formule logique à base de booléens !

La fiabilité d'une architecture == une formule logique

Formalisation logique pure du RBD

- **Une architecture avec des noms …**
  - Chaque composant élémentaire se voit affecter un état booléen
  - Les compositions série / parallèle / k parmi N définissent des formules logiques à partie des formules des blocs ou structures contenues
  - Les variables d'état booléen sont vues comme des variables aléatoires 2 à 2 indépendantes
- **Avantages**
  - Utilisation de noms => un même composant peut être utilisé à deux endroits, ce n'est plus grave !
  - Indépendance des états élémentaires => règles de calculs simples si la formule possède une bonne propriété : forme normale de disjonction en conjonction disjointes
  - ⇒ Cela fournit un cadre complet, possibilité d'utiliser des variables continues pour chaque état (date de passage de vrai à faux) le modèle devient « temporisé »).

Utilisation en direct de RBD pour fiabilité réplication passive / active à 3 répliques

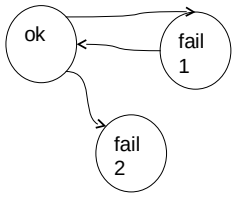
Modélisation fine des modes de défaillance

- **Problèmes :**
  - que faire pour modéliser un mode de défaillance réparable ?
  - Comment modéliser l'éventualité de deux modes de défaillance distincts à partir d'un même état ?
- **Solutions :**
  - Modéliser les changements d'états explicitement
  - Modèles pour l'analyse : Chaînes de Markov (discrètes et continues), et peut être les probabilistic timed automata

Rappel : automate et système à transition

Modèle Automate à état fini

- **Q : ensemble des états**
- **Δ : ensemble des transitions (q,q') (changements d'états possibles)**
- **I : états initiaux**
- **L : étiquetage de Δ par des noms /des symboles**



Chaine de Markov en temps discret (pour un domaine fini)

Idée : l'état de votre système est aléatoire car
il y a une incertitude quantifiable à l'initialisation
il y a une incertitude lors des changements d'états

Chaine de Markov discrète = processus stochastique : (Xi), i dans N tel que

- **chaque Xi est une variable aléatoire sur D.**
- **Xi représente l'état du processus à la date discrète i**
- **$P(X_i=v_i \mid X_{i-1}=v_{i-1}, X_{i-2}=v_{i-2},\ldots X_0=v_0)= P(X_i=v_i \mid X_{i-1}=v_{i-1})$**

Interprétation : L'état i ne dépend que de l'état i-1, et de la probabilité conditionnelle $P(X_i=v_i \mid X_{i-1}=v_{i-1})$

Chaine de Markov en temps discret (pour un domaine fini)

Vocabulaire et notations

$P(X_i=v_i \mid X_{i-1}=v_{i-1})$ est la probabilité de transition de vi-1 vers vi

La définition des probabilités de transition et de la distribution de X0 caractérise totalement la chaîne

Représentation graphique : système à transition étiqueté par les probabilités de transitions

Modélisation d'un cas pratique

3 états possibles : ok failed1, failed2 tel que

V_{i-1}	V_i	CPD
ok	ok	0.8
ok	failed1	0.15
ok	failed2	0.05
Failed1	ok	0.8
Failed1	Failed1	0.2
Failed2	Failed2	1

A chaque transition

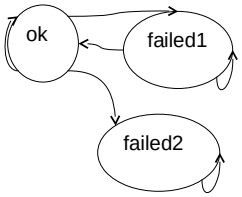
- **Soit le système est « utilisé » avec pour effet un comportement normal, ou une défaillance fail1, ou une défaillance fail2.**
- **Soit, une réparation est tentée, avec pour effet un retour à l'état ok, ou un échec.**
- **Soit le système est dans l'état failed2 et reste définitivement défaillant**

Modélisation d'un cas pratique

3 états possibles : ok failed1, failed2 tel que

A chaque transition

- **Soit le système est « utilisé » avec pour effet un comportement normal, ou une défaillance fail1, ou une défaillance fail2.**
- **Soit, une réparation est tentée, avec pour effet un retour à l'état ok, ou un échec.**
- **Soit le système est dans l'état failed2 et reste définitivement défaillant**



Ce que l'on peut analyser

Probabilité stationnaire = vecteur π de probabilité

Tel que $\frac{Card\left(\left\{X_j = v_i \mid j \leq n\right\}\right)}{n}$ ¿

~ temps passé en moyenne dans l'état v_i pour chaque v_i

Probabilité transitoire = probabilité que Xi vaille v pour i = une date précise, ou i dans un intervalle

Intérêt (exemple) : calcul la probabilité d'être fiable à la date i.

Evaluation de fiabilité

Fiabilité à la demande = probabilité de transition d'un état fiable vers un état fiable …..

Que peut on étudier :

- **Probabilité de rester fiable pendent N instants**
- **Probabilité de ne pas subir un type particulier de défaillance sur un intervalle I d'instants.**

Comment l'exprimer ? C'est une propriété d'invariance…. On formalisera plus tard.

Evaluation de la disponibilité

Disponibilité ~ ratio de temps passé dans un état où l'on n'est pas dans un état défaillant sur le temps total.

⇒ **Probabilité stationnaire fournie une mesure de la disponibilité « en moyenne à l'infini »**

⇒ **Probabilité transitoire sur un intervalle autre mesure de disponibilité (i.e. probabilité d'être dans un état fonctionnel à un quelconque moment d'un intervalle donné).**

PB = les transitions modélisent un temps logique (un nombre d'action), il faut que cela ait du

Chaîne de markov en temps continu

Idée : dynamique en temps continue entre état discret =

Une séquence de paire (état discret, durée de séjour)

Chaine de Markov continue = processus stochastique : (Xi,Ti), i dans N tel que

- **(X_i), i dans N est une chaîne discrète de markov**
- **(T_i), i dans N est une suite de variables aléatoires continues**
- **On définit X(t) comme le X_k tel que t est dans $[T_k, T_{k+1}]$**
- **La chaîne est caractérisée par**

$P(X(t+h)=v_i \mid X(t)=v_j) = q_{ji} h + o(h)$

Principe : q_{ji} constante désignant le taux de transition de la valeur v_j vers v_i

Conséquence et interprétation

Hypothèses de travail:

- **Taux de transition constant (chaine dite stationnaire)**
- **Domaine de valeur fini**

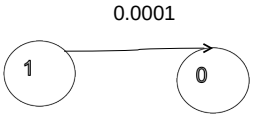
Conséquences :

- **Temps de séjour dans l'état v_i suit une loi exponentielle de paramètre : $\sum_{j \neq i} qij$**
- **le taux de sortie est la somme des taux individuels de transition**

Exemple

Système composé de 1 composant avec 2 états de fonctionnement : 1 (tout va bien) et 0 (défaillance par crash).

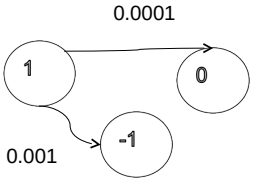
Le taux de transition de l'état 1 vers 0 est supposé constant et égal à 0.0001 h^{-1}



Exemple suite

Système composé de 1 composant avec 2 états de fonctionnement : 1 (tout va bien) et 0 (défaillance par crash), -1défaillance en valeur

Le taux de transition de l'état 1 vers 0 est supposé constant et égal à 0.0001 h^{-1}, le taux de transition de l'état 1 vers -1 est de 0.001 h^{-1}



le taux de sortie de 1 est de 0.0001+0.001

Ce qu'il faut savoir

Analyse symbolique possible de la probabilité transitoire mais potentiellement fastidieuse (résolution pour chaine stationnaire connue)

Possibilité d'automatiser l'analyse via des outils (non symbolique)

Limitations : les comportements sont non bornés dans le temps.

Modélisation de systèmes complexes par réseau de chaines

Idée : 1 chaîne 1 état dans [0,N].

⇒ P chaînes P états dans [0,N] évoluant en parallèle

Utilité : étude d'un système structuré complexe.

Pb : comment modéliser les effets de type « propagation de faute ? »

Outil théorique : réseau d'automate / chaînes de markov synchronisées.

Systèmes à transitions et produit synchronisés

Soit A1 et A2 deux systèmes à transitions définis respectivement par

- l'ensemble d'état Q1, (Q2)
- l'ensemble de transition d'état Δ1, (Δ2) sous ensemble de Q1x Q1, (Q2 x Q2)
- L'ensemble d'étiquette L1, (L2) tel que chaque transition est étiquetée par un élément de cet ensemble

Le produit synchronisé A1 x A2 est un système à transition dont l'ensemble des états est Q1 xQ2, dont les étiquettes sont dans (L1 U L2) et dont les transition sont définies comme suit …..

Systèmes à transitions et produit synchronisés (suite)

Pour chaque transition (q, p) de Δ1 étiqueté l tel que l est dans L1 / L2, alors pour tout état q' de Q2, la transition ((q,q'), (p,q')) est dans l'ensemble des transitions de A1 x A2.

Pour chaque transition (q, p) de Δ2 étiqueté l tel que l est dans L2 / L1, alors pour tout état q' de Q1, la transition ((q',q), (q',p)) est dans l'ensemble des transitions de A1 x A2.

Pour toute étiquette dans l'intersection de L1 L2 ((q',q), (p',p)) est dans l'ensemble de transitions de A1 x A2 ssi (q',p') est dans Δ1, et (q,p) est dans Δ2, et toutes les deux sont étiquetés l

Logique temporelle et calcul de probabilité

Idée : on peut évaluer la probabilité pour une chaîne

- De passer par une séquence d'états
- D'être dans un état à une date précise
- D'être dans un ensemble d'état à une date précise
- ....

On peut évaluer la probabilité de n'importe quel événement qui est une contrainte sur « l'exécution de la chaîne »

Logique temporelle = contrainte sur la séquence d'états franchie (LTL), ou sur les transitions possibles en chaque états (CTL)

Logique temporelle et calcul de probabilité

Syntaxe :

Des contraintes instantanées : logique propositionnelle ou prédicat (sans quantification)

Des contraintes sur l'enchaînement d'état (non bornées):

G φ : φ doit être vraie en chaque instant futur

F φ : φ doit être vraie en au moins un instant du futur

Φ1 U Φ2 : Φ2 doit être vraie au moins un instant dans le futur et Φ1 doit être vrai jusqu'à ce que Φ2 le soit.

Chaque opérateur peut être borné dans le temps via l'ajout d'un interval I après l'opérateur

Logique temporelle et calcul de probabilité

Une formule temporelle peut ensuite être soumise à un opérateur de probabilité : P

P=? [Φ] permet de demander le calcul de la probabilité que Φ soit vraie pour une chaine donnée

P(<|<=) c [Φ] permet de déclencher la vérification de la formule déclarant que Φ est vraie avec une probabilité inférieure strictement (ou égale) à c pour une chaîne donnée

Outil Prism

Un outil pour modéliser les chaines discrètes et continues

Un langage pour décrire des objectifs d'analyse stationnaires / transitoires

Une méthodologie pour analyser les résultats

Passage à la pratique