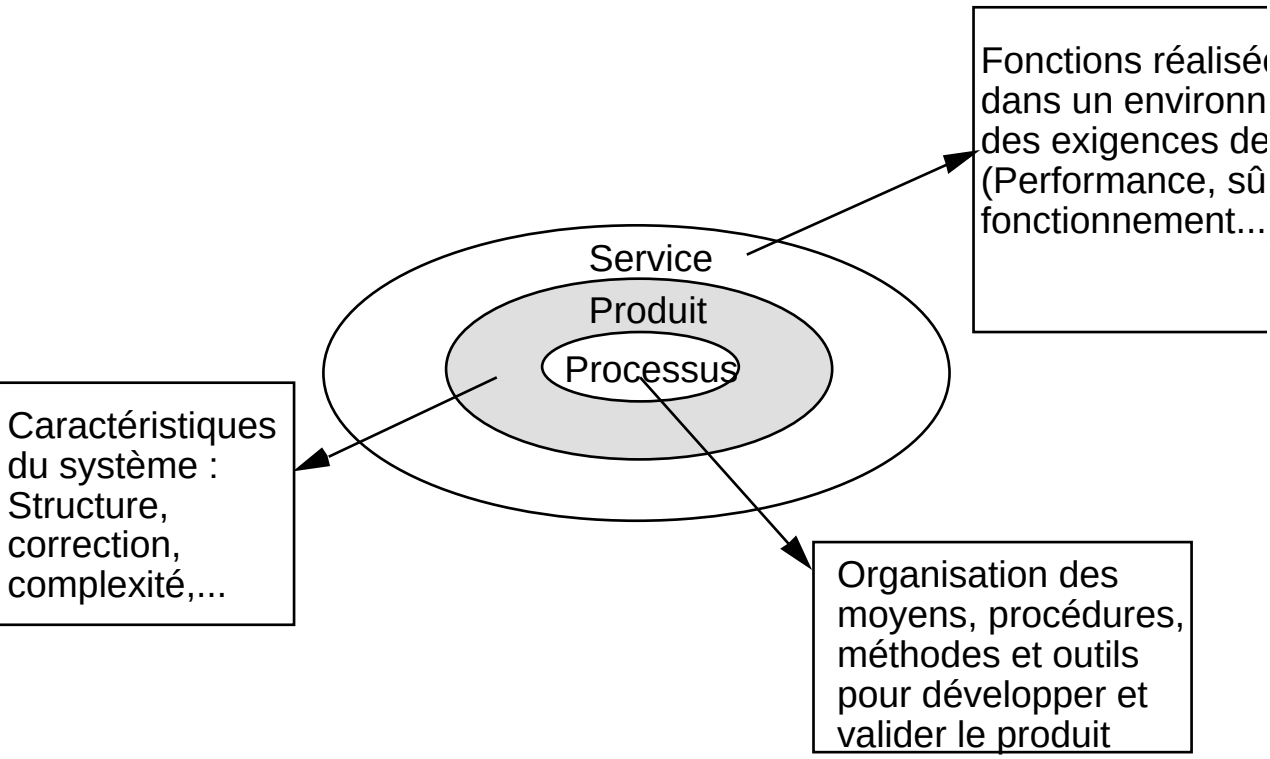
Introduction au Génie Logiciel

Processus de développement
Méthodes et Outils
Gestion de configuration

Véronique Delebarre

Problème du développement logiciel

Le développement du logiciel est un problème d'optimisation



LES CAUSES

- **Coûts financiers**
  - développement
  - maintenance
- **Inadéquation des logiciels aux besoins**
- **Complexité**
- **Sûreté de fonctionnement**

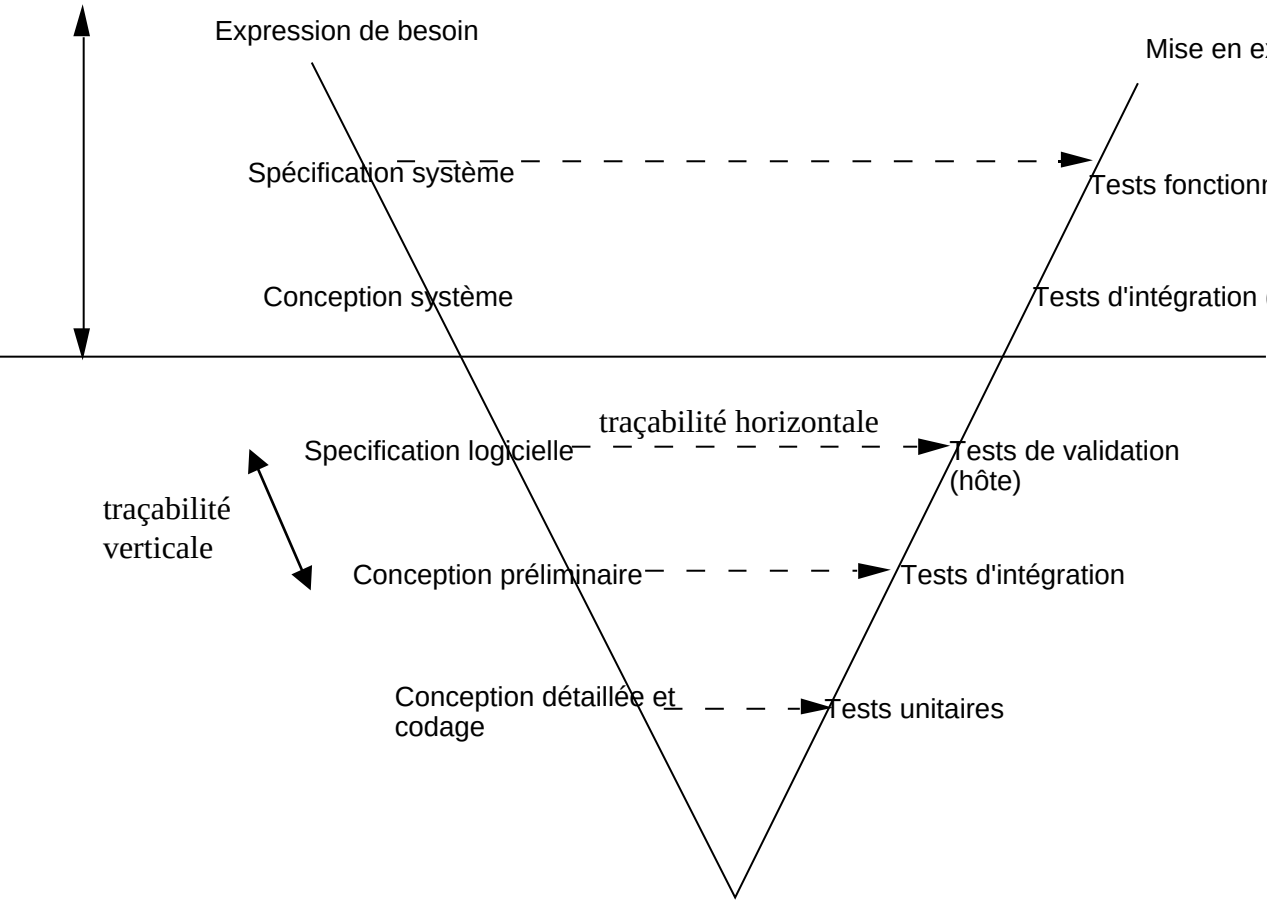
Processus de développement logiciel

- **Processus vu comme une organisation de phases**
  - Séquences et conditions de passage entre phases
  - Cohérence mutuelle entre produits de différentes phases
- **Formalisation des étapes ou phases du processus de développement du logiciel**
  - objectifs
  - entrées
  - sorties (productions)
  - contrôles (conditions de terminaison de la phase)
- **Mise en oeuvre de méthodes et outils support pour la réalisation des phases**

Cycles de vie

- **Différents modèles ont été proposés**
  - prototypage
  - cascade
  - cycle en V
- **Le cycle en V est le cycle qui a été normalisé**
- **Il est largement utilisé, notamment en informatique industrielle et télécoms.**

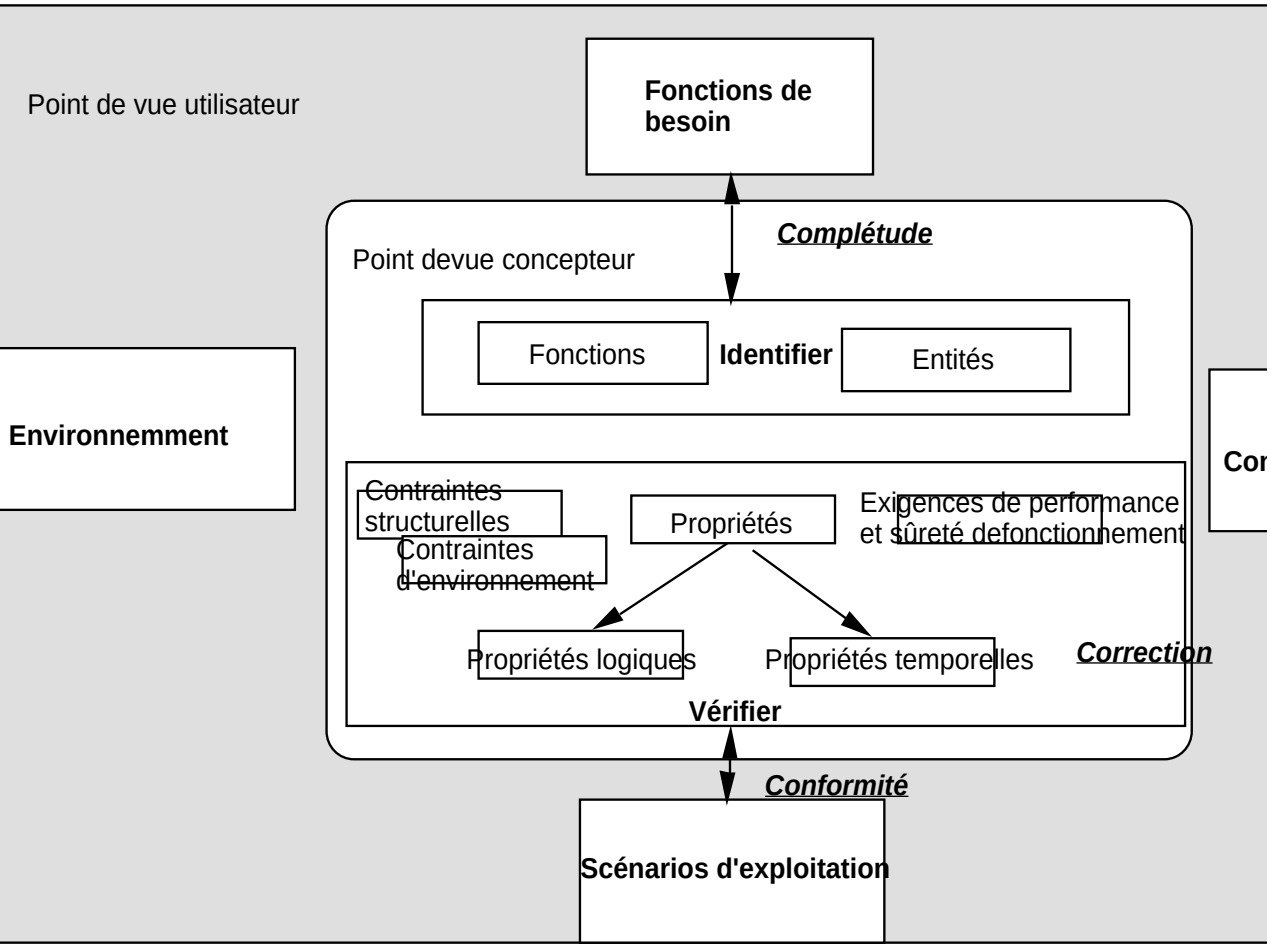
Cycle de Vie logiciel “en V”



Phase de spécification (1)

- **Objectifs :**
  - Décrire la solution, indépendamment de son implantation,
    - Identifier la frontière (les interfaces) du logiciel à développer
    - Identifier les entités du logiciel et leurs interactions avec l'environnement
    - Spécifier le comportement attendu des entités, ainsi que leurs interactions (internes)
  - Valider la solution proposée
    - cohérence interne (cohérence des interfaces, absence d'interblocage, absence d'états puits)
    - vérification de propriétés de comportement
    - vérification de contraintes externes
  - Ecrire le plan de validation
    - Scénarios de test

Phase de spécification (2)



Phase de spécification (3)

- **Documents d'entrée : Spécification de besoin**
  - précise les exigences fonctionnelles et les exigences non fonctionnelles (performance, sûreté de fonctionnement)
  - décrit les contraintes de réalisation, notamment réutilisation de technologies.
- **Documents de sortie**
  - Spécification du logiciel (STBL, DSL, DSBL,...)
    - architecture "fonctionnelle"
    - comportement attendu
  - Document de traçabilité entre spécification du logiciel et exigences
  - Plan de validation
- **A l'issue de cette phase, la spécification est réputée "correcte".**

Que doit contenir le dossier de spécification ?

- **Frontière du logiciel (ou système) cible**
  - entités avec lesquelles le système interagit (environnement physiqu humain et informatique)
  - nature des interactions
    - mode nominal, dégradations
  - scénarios d'utilisation
- **Formalisation des objectifs du logiciel ou système cible**
  - entités : composants et fonctions
  - propriétés
    - invariants
    - propriétés événementielles
    - propriétés de sûreté, de vivacité
    - propriétés quantitatives (disponibilité, performance, niveau d'erreurs résiduelles par exemple)

Que doit on vérifier sur la spécification ?

- **Montrer la complétude de la solution**
  - toutes les entrées sorties sont prises en compte
  - toutes les fonctions de besoin sont réalisées
  - tous les scénarios sont réalisables
- **Montrer la correction de la solution**
  - cohérence des interactions (données, messages, événements) interne
  - correction du comportement global
    - absence de blocage, vérification des propriétés de vivacité et de sûreté
    - correction des séquences d'événements
    - correction des variables d'état et des sorties produites

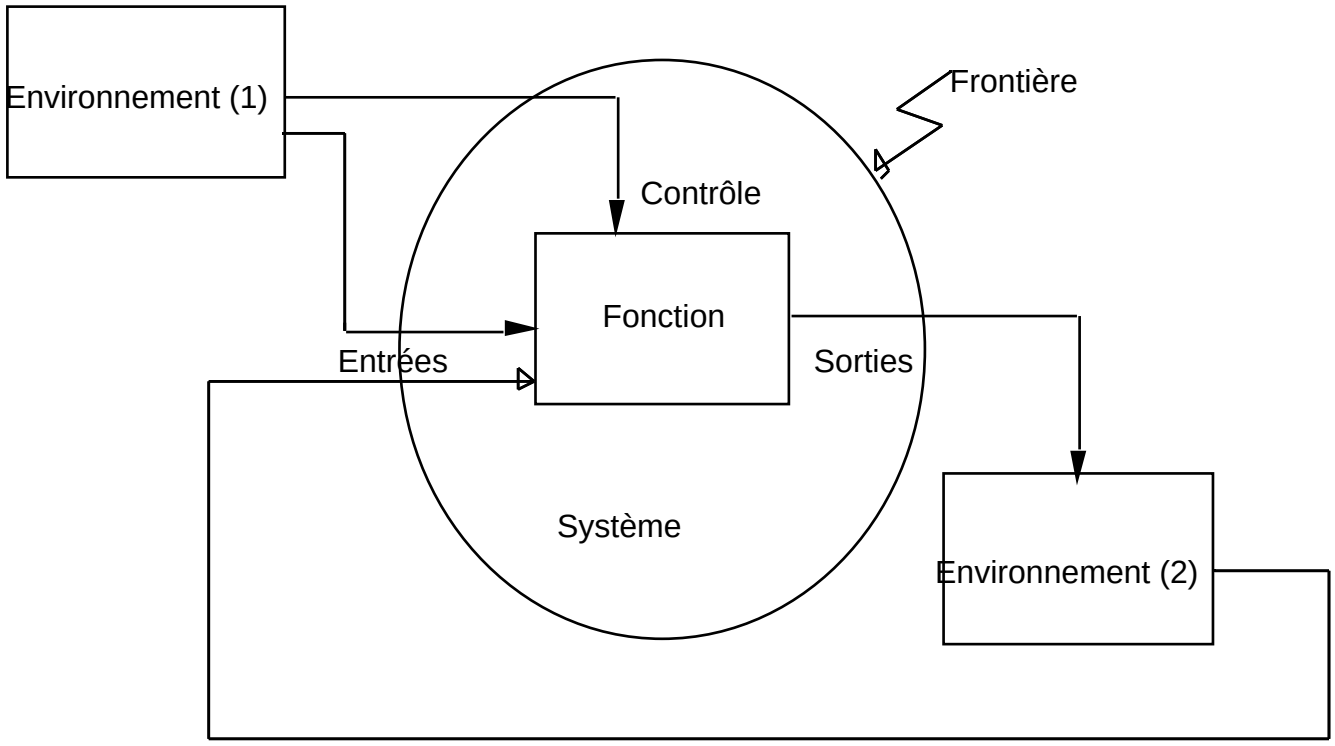
Démarches de spécification

- **Approche fonctionnelle**
  - SA, SADT, SA/RT
  - automates, AEFC, RDP
- **Approche “orienté objet”**
  - OMT
  - UML
- **Approche Flots de données synchrones**
  - Opérateurs, blocs-diagrammes
  - horloge
  - Equations
- **Approche “formelle”**
  - Propriétés

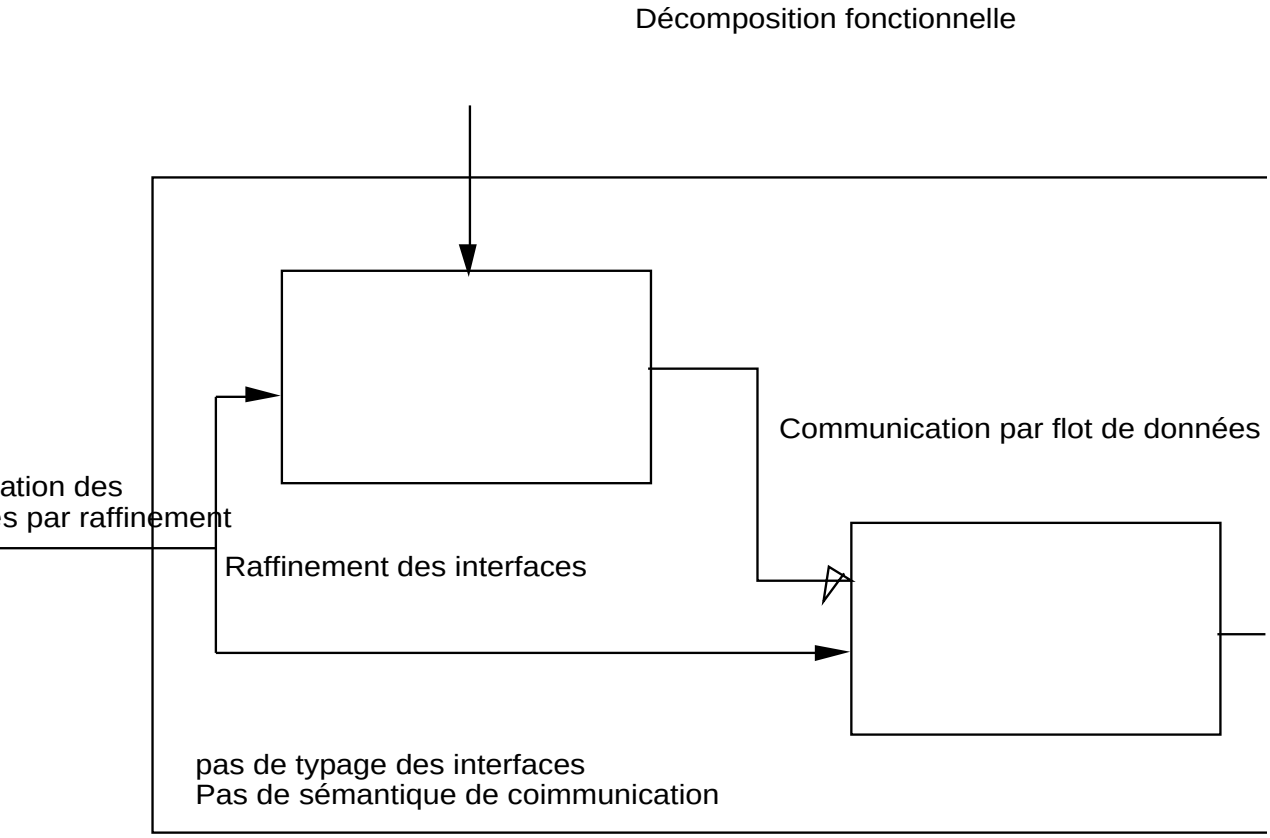
Spécification : approche fonctionnelle

Approche fonctionnelle (1)

1) Le système dans son environnement

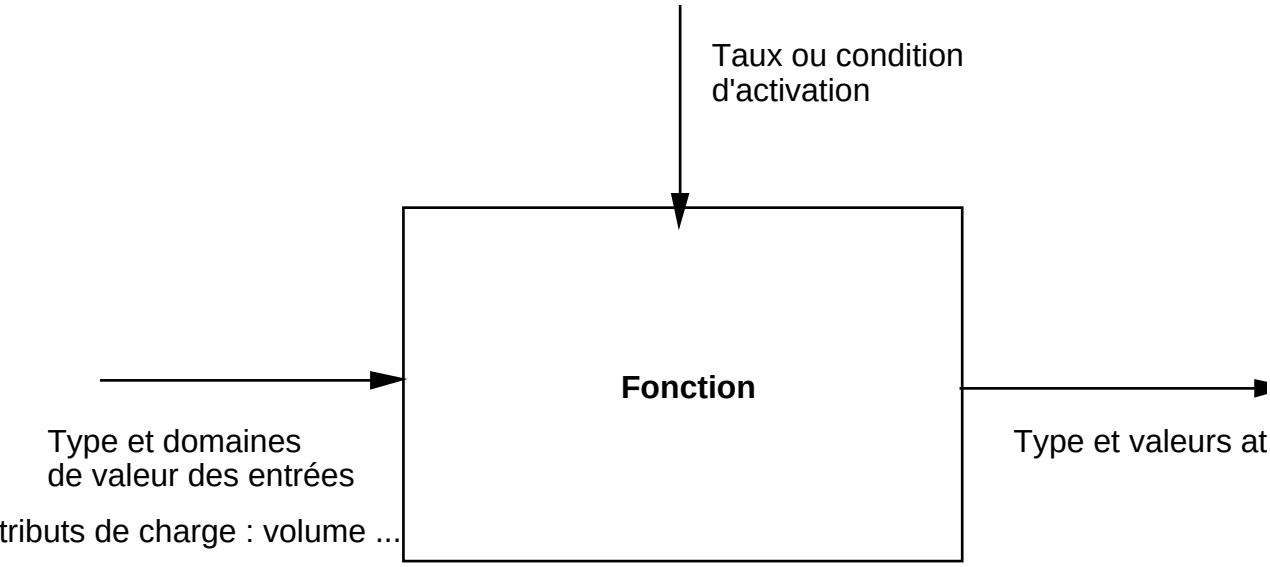


Approche fonctionnelle (2)



Approche fonctionnelle (3)

Enrichissement de la décomposition fonctionnelle par attributs



Pré : ensemble des conditions sur les entrées et les contrôles

Post : ensemble des conditions attendues (entrées/sorties)

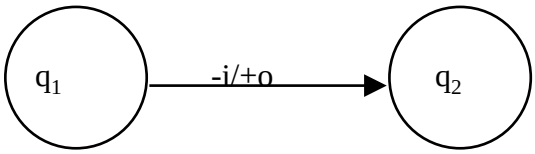
Approche fonctionnelle (4)

- **Description du comportement**
  - associer à chaque fonction la description de son comportement
    - AEF
  - associer aux interfaces entre fonctions une sémantique de communication
    - files de message en entrée et en sortie et communication asynchrone
    - synchrone (rdv)
  - identifier les conditions de déclenchement des fonctions
    - événements
  - construction du comportement global
    - produits d'automates communicants (AEFC)

Modélisation des spécifications par AEFC

- Les AEF constituent un moyen très utilisé de décrire le comportement des systèmes
  - Un AEF peut être défini par le sextuplet <Q, q0, I, O, A, T>
    - Q : ensemble fini d'états
    - q0 : état initial
    - I : ensemble des événements en entrée
    - O : ensemble des événements en sortie
    - A : fonction de génération des événements en sortie QXI-->O
    - T : fonction de transition d'états QXI-->Q
  - les états sont associés à des variables d'états ou à des états des processus qui réalisent les fonctions spécifiées
  - les transitions entre états sont conditionnées par les états (Q) et les événements

AEF

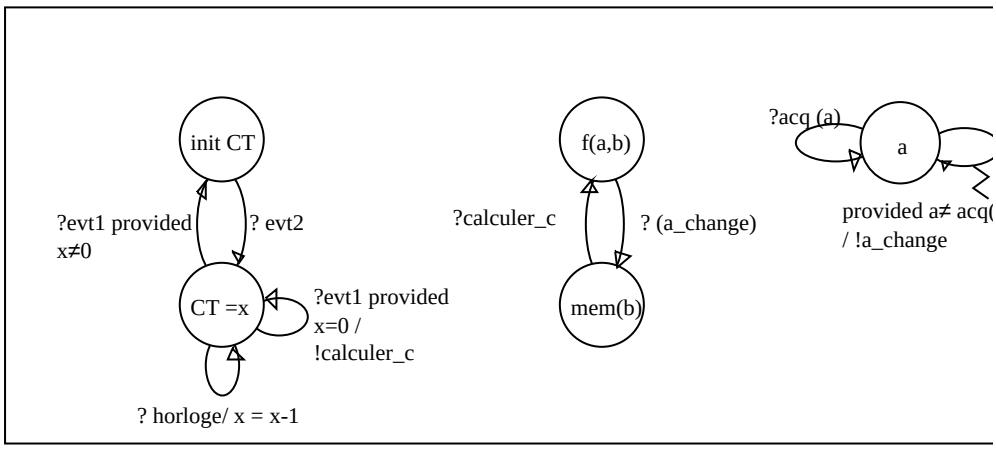


- représentation graphique
- sémantique d’exécution
  - t est tirable si, dans l'état initia q1, on reçoit l'événement i,
  - on tire la transition t, ce qui a pour effet de faire passer le système à l'état q2= T(q1,i)
  - indéterminisme si deux transitions concurrentes sont tirables.
- extensions
  - on peut étendre les automates par des variables locales, qui peuvent conditionner les franchissements transitions. On peut également associer des délais aux transitions

AEF, AEFC

AEFC

- Chaque processus est décrit par un AEF dont les transitions associées aux procédures d'échange entre processus, sont étiquetées par les messages émis ou reçus.
- On construit le graphe des états global par combinaison des états des processus et des files pour les communications de type FIFO ou par fusion sur les transitions, lorsqu'on considère des rendez-vous.
  - "Quand l'événement evt1 se produit (soit t1 l'instant d'occurrence de evt1), si l'événement evt2 ne s'est pas produit depuis x unités de temps alors calculer la sortie c = f(a,b) avec la valeur courante de a et la valeur de b à l'instant où a a changé d'état."



AEF, AEFC

- **de nombreux langages implantent le modèle AEFC**
  - exemple ESTELLE , LDS, PROMELA, LOTOS
  - ...

Modélisation des spécifications par AEFC

- **Comportement local à chaque processus**
  - Chaque module élémentaire est décrit par un AEFC
  - Déclaration de l'espace des états (local)
  - déclaration des variables (locales)
  - les messages sont (en principe) les seuls moyens de partager des données (pas de variables globales)
  - Transitions conditionnées par
    - variables d'état locales
    - réception d'un événement (horloge, message)
  - Les transitions tirables depuis un même état doivent être en exclusi mutuelle (on joue sur les conditions de tir des transitions)

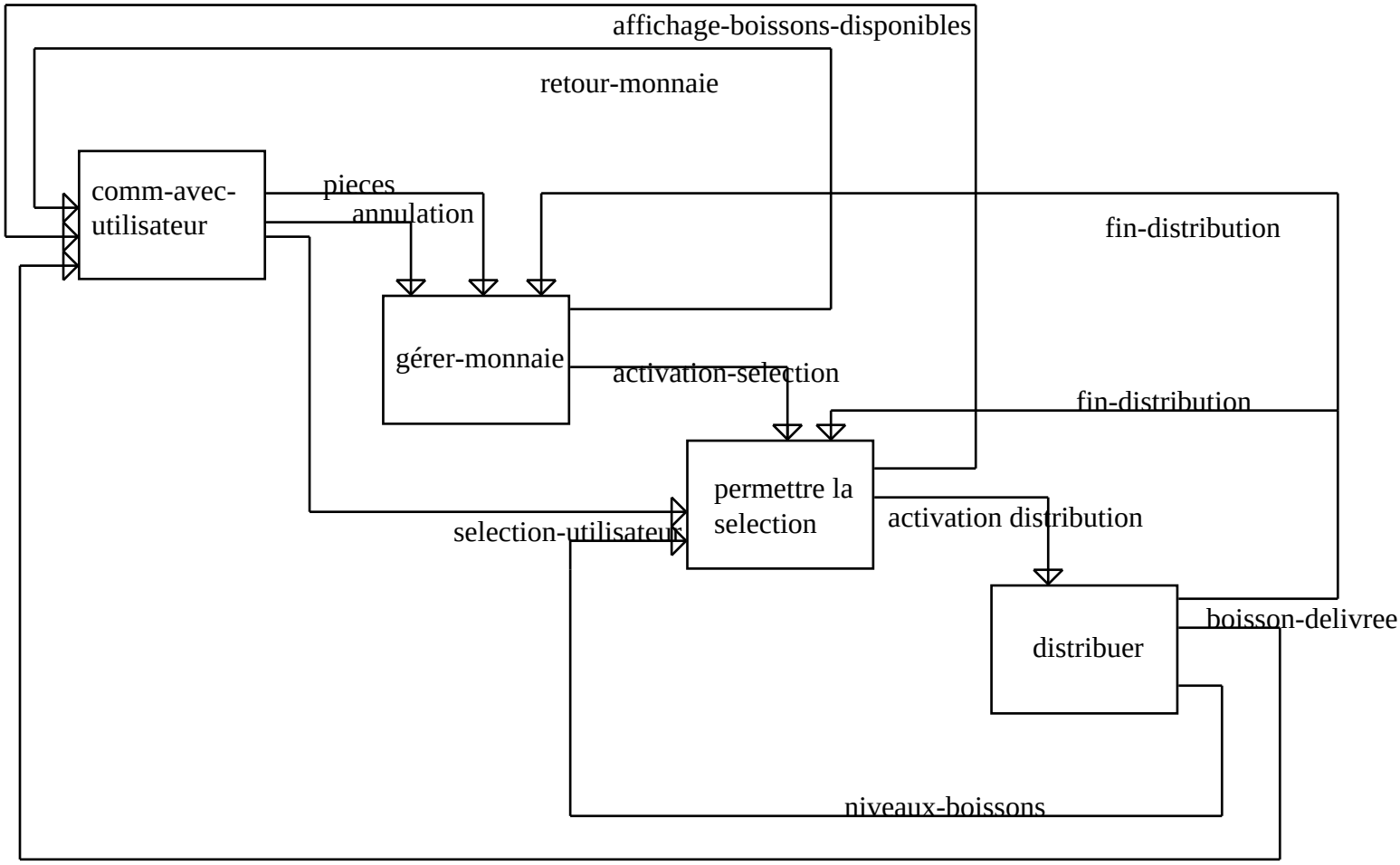
Modélisation des spécifications par AEFC

- **Comportement global**
  - produit des automates
    - tenir compte du mode de communication => produit synchrone si rdv
- **Possibilité de simulation du modèle global ainsi réalisé**
  - scénarios (séquences de transitions)
  - recherche d'états puits,
  - contrôle des états des variables et des séquences

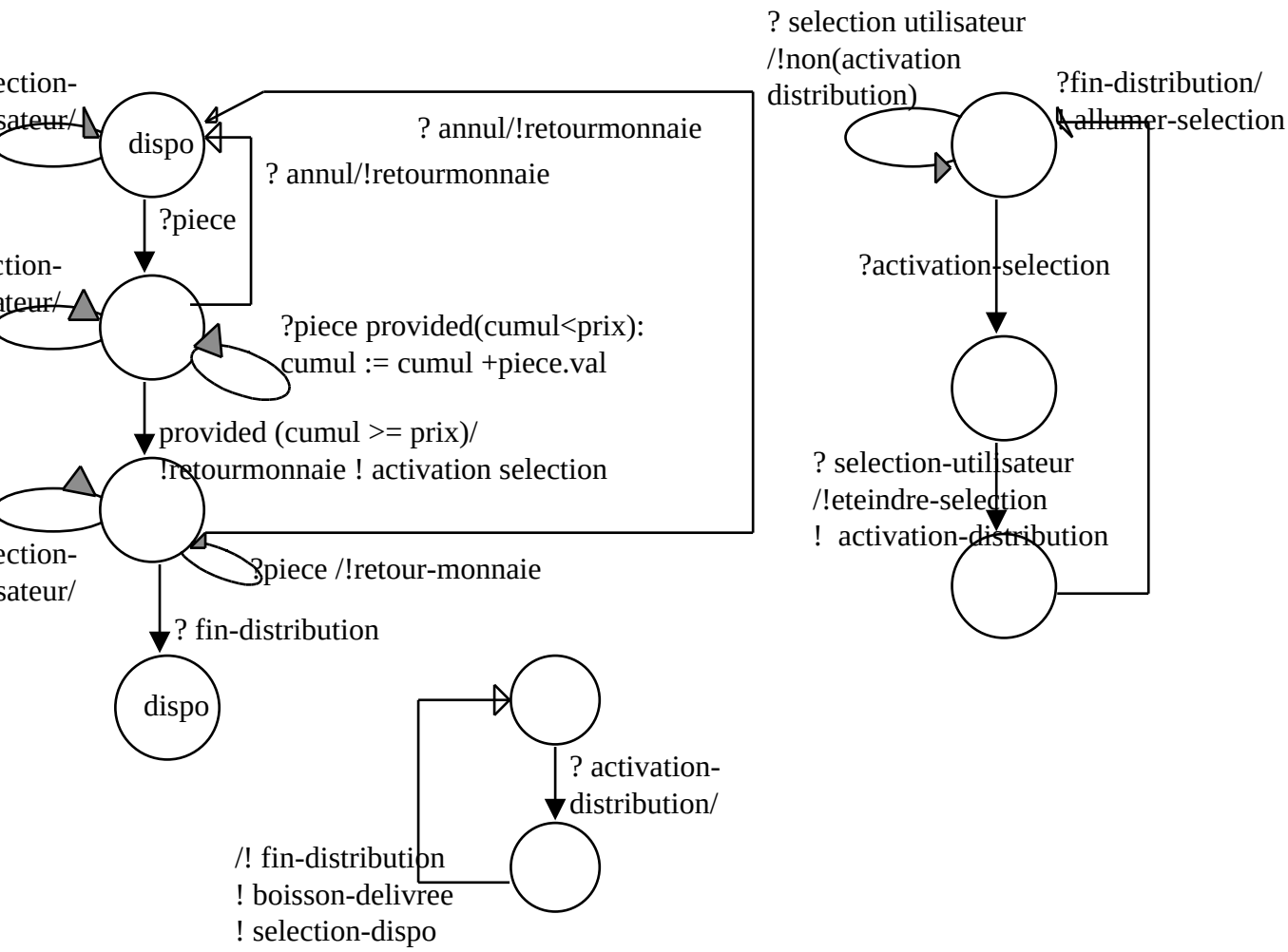
Modélisation de spécifications par AEFC

- **Modèle statique :**
  - ensemble de modules (blocs pour lds) interconnectés par des canau de communication (channels pour estelle, chan pour promela, route pour lds)
  - lien entre module et canaux : point d'interface (en entrée, en sortie)
  - caractérisation du mode de communication : RDV (multiplicité), Fi (taille)
  - identification des messages véhiculés par les canaux
- **Premier niveau de vérification possible**
  - cohérence du modèle
  - pour chaque flot identification de l'emetteur et des récepteurs

spécification système

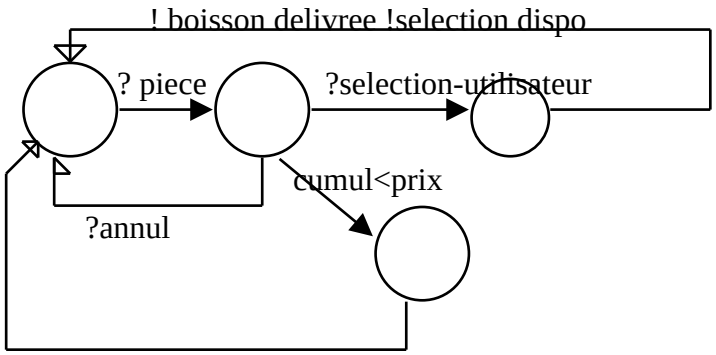


spécification système



quelques propriétés attendues

- **Scenario nominal vis à vis de l'utilisateur uniquement**
  - blocage possible dans la spec actuelle (il manque des événements de délais)
- **Scénarios non couverts**
  - cumul n'atteint pas prix
- **montrer que le nombre de boissons distribuées (occurrences d'activation distribution) est inférieur ou égal au nombre d'activation selection**
- **retour à l'état initial**



Spécification : approche orienté objet

notations UML

Approche Objet

- **Cf cours SIA**
- **Méthode UML met en oeuvre différents types de “diagrammes” qui correspondent aux différentes activités de la spécification**
  - Expression de besoin
    - use cases et diagrammes d’activité (activity diagrams)
  - Identification des entités
    - class diagrams
      - classes
      - associations
      - attributs
      - opérations
      - assertions (design by contract), invariants, pre-conditions, postconditions
      - généralisations
      - classes associations

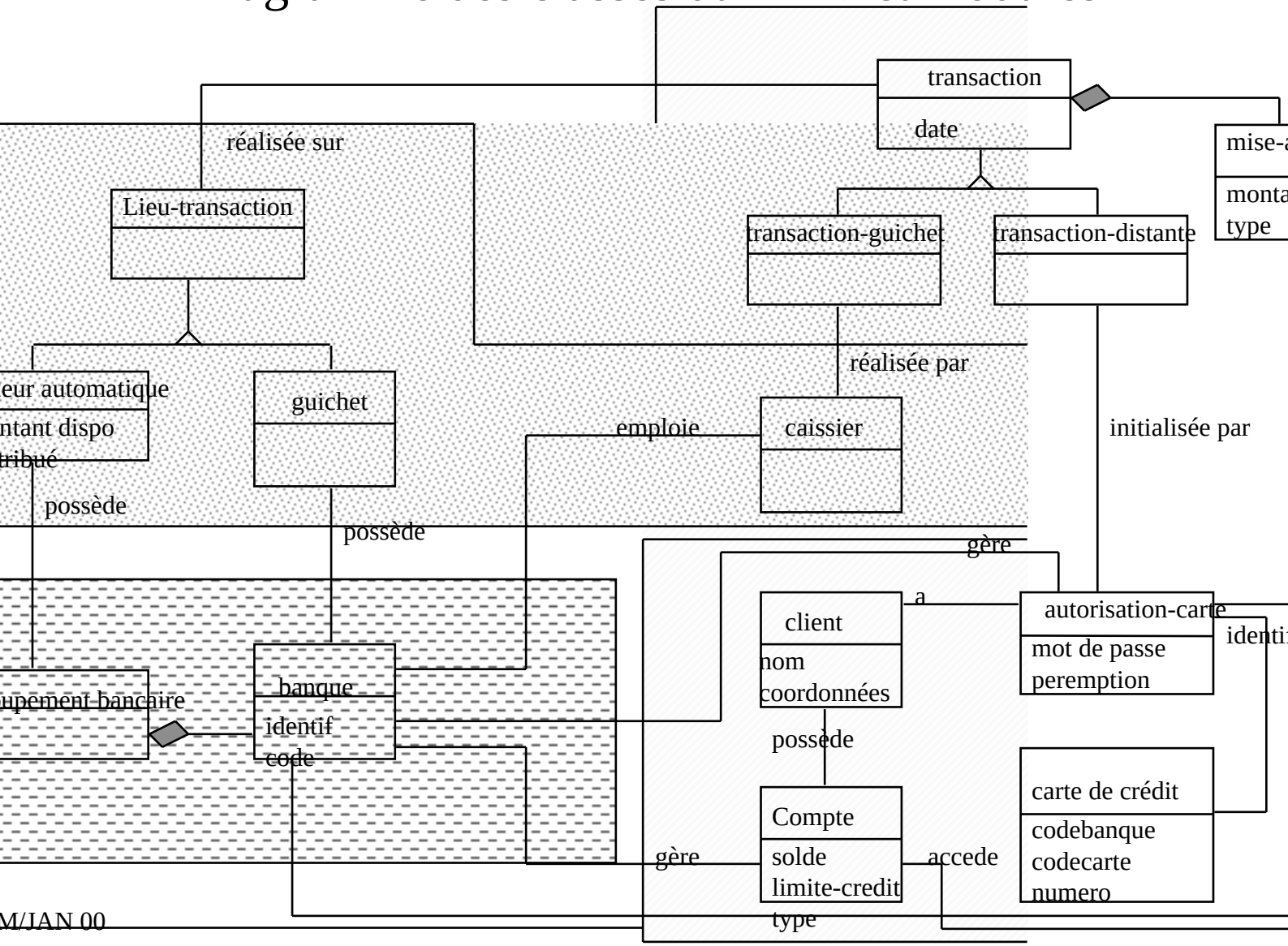
Approche objet

- Description du comportement attendu
  - Interaction Diagrams
    - description des interactions entre “groupes” d’objets : messages échangés entre objets. En général les diagrammes correspondent aux diffétrents use cases.
    - les MSC sont un mode de description couramment employé
- Description du comportement système
  - State Diagrams
    - description des états des objets.
    - dans UML on fait des diagrammes séparés pour les différents objets, en utilisant le formalisme des “state charts” (David et Harel) ; il s’agit d’une description états transitions,
      - etats <=> activités
      - transitions <=> événements
  - problème de description du comportement global?

Présentation des divers “diagrams” au travers d’une étude de cas

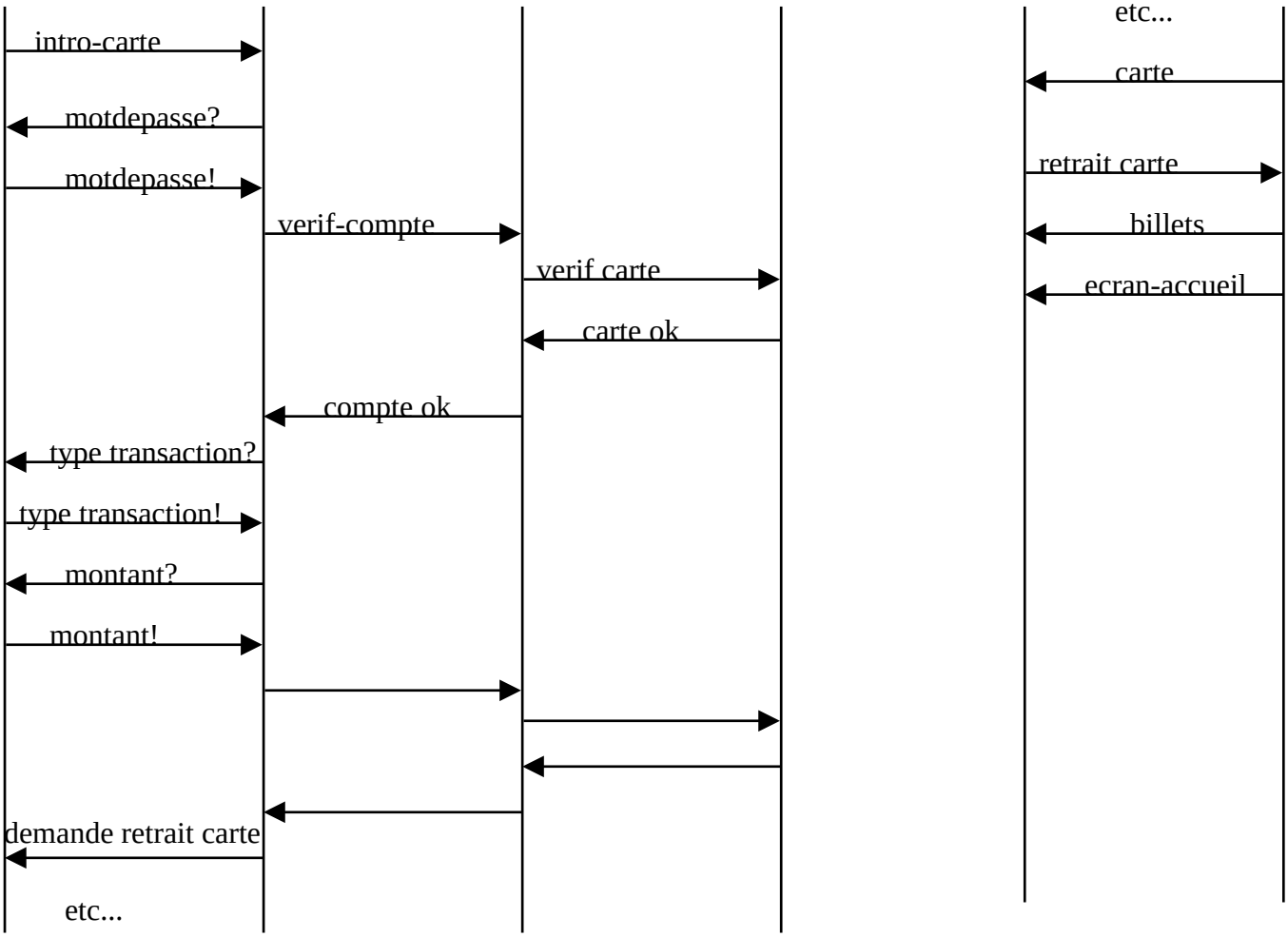
- **Exemple (très utilisé) du DAB**
  - Construction du CD
    - Classes, Attributs et Associations
    - Modules
  - Description du comportement
    - Scénario
      - textuel
      - passage aux diagrammes de séquencements de messages
    - Graphes d’états
      - AEFC (FSM)
      - State Charts
  - Modèle fonctionnel des opérations
    - approche SA

Diagramme des classes du DAB et modules



M/JAN 00

Diagrammes de Séquencement de Messages (MSC)



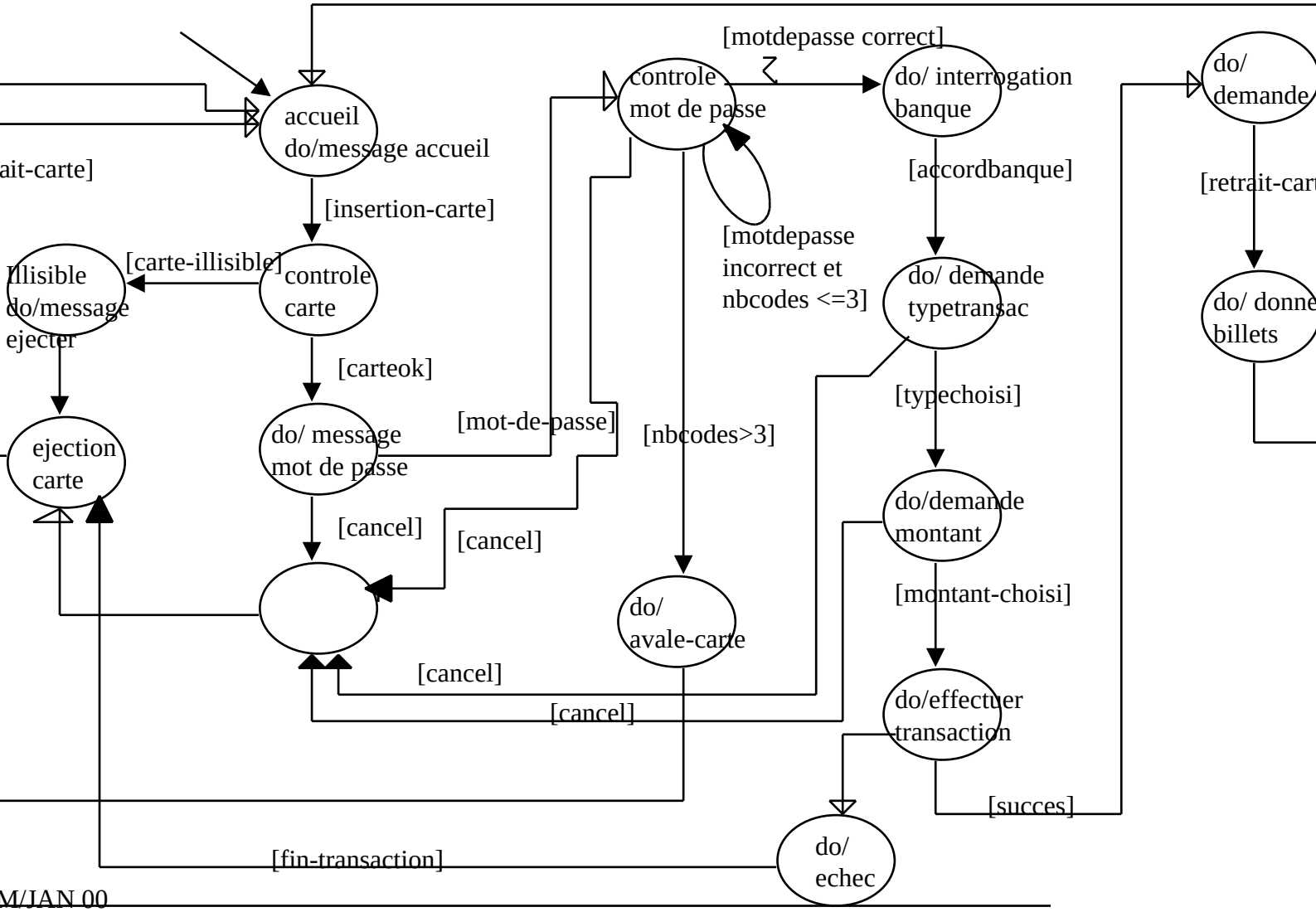
Diagrammes de séquencements messages

- **Enumération des scénarios**
  - cas nominaux
  - prise en compte de tous les comportements possibles
  - prise en compte des pannes des différents éléments
- **l'énumération des scénarios constitue le plan des tests fonctionnels**
- **tâche itérative avec analyse de comportement car pb de complétude de l'énumération**

Diagrammes d’états

- **Décrire le comportement de chacune des entités (classe)**
  - identification de tous les messages échangés entre deux entités (vocabulaire des échanges à identifier)
  - inclure tous les scénarios dans le comportement
- **formalisme système de transition condition action mais pas de sémantique définie des mécanismes de communication et de synchronisation**
- **Faire l’analyse du comportement**
  - comportement global ??????
  - En fait les diagrammes d’états ne sont pas adaptés à la description des coopérations entre objets.

Diagrammes d’états



Diagrammes d’activité

- **Décrire le parallélisme (au sein d’un use case, pour un objet,..)**
- **Il s’agit d’une combinaison (!) d’idées issues de différent langages et des techniques associées**
  - Diagrammes d’événements
  - RDP
  - SDL
- **mais en fait, il s’agit simplement d’une description graphique, qui peut, au mieux, expliciter les traitements qui peuvent être parallélisés . Pour un objet donné, on pourrait ainsi déterminer le taux de multi threading possible.**

Conclusion à la description objet UML

- **Notations graphiques représentant divers points de vue**
  - use case
  - CD
  - diagrammes d'interactions entre classes : MSC ou diagrammes de collaboration(entre instances ?)
  - diagrammes d'états associées à une classe
  - diagrammes d'activité
- **Modèle de développement "fashionable"**
  - gourous : Jacobson, Rumbaugh, Grady Booch,...
  - OMG
- **Projection sur les langages C++, Java**

Validation des spécifications

- **Deux aspects**
  - Validation “interne”
    - cohérence
    - absence d’interblocage
    - propriétés de comportement.
  - Validation vis à vis de l’expression de besoin
    - complétude
    - conformité des comportements spécifiés aux comportements attendus
    - démonstration que les objectifs spécifiés => les objectifs de besoin.

Validation des spécifications

- **Modélisation par graphes d'états du comportement des composants**
  - approche objet : MSC, description des comportements par "State Charts"
  - approche fonctionnelle : description des fonctions par AEFC
    - => importance de la sémantique de communication
  - approche équationnelle : description des opérateurs par des équation ou des fonctions de transfert. Flots de données synchrones
- **Génération du graphe des états et exploration de modèle**
  - simulation interactive
  - simulation exhaustive (difficile)
  - simulation intensive
  - simulation guidée
- **Applicable pour systèmes séquentiels et distribués**

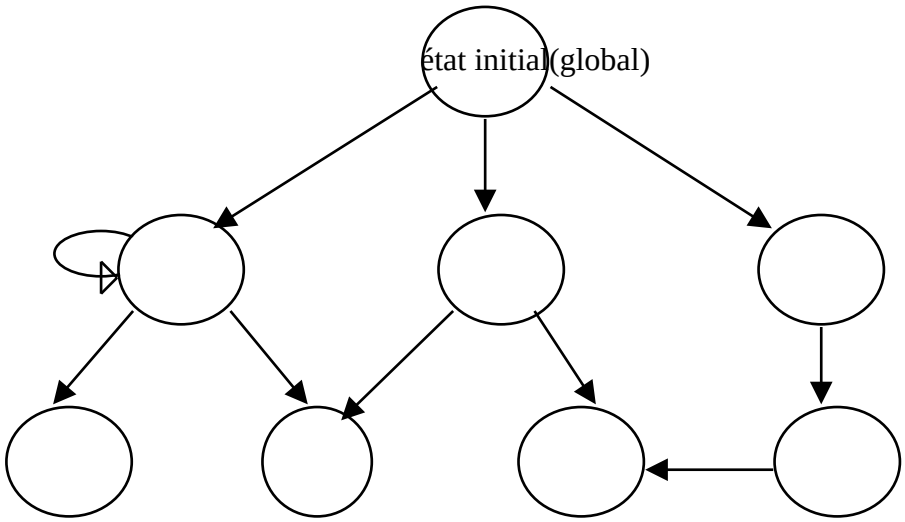
Validation des spécifications

- **Approches basées sur la preuve**
  - Nécessité d'une description dans un langage formel
    - basé sur théorie des ensembles
    - basé sur théorie du contrôle
    - ....
  - Formalisation des propriétés par Invariants et par Pré et Post conditions associées aux événements.
  - Méthode itérative (raffinement)
- **Démonstration**
  - "manuelle"
  - automatique pour certains environnements
- **Difficulté de la preuve par rapport aux approches simulation**

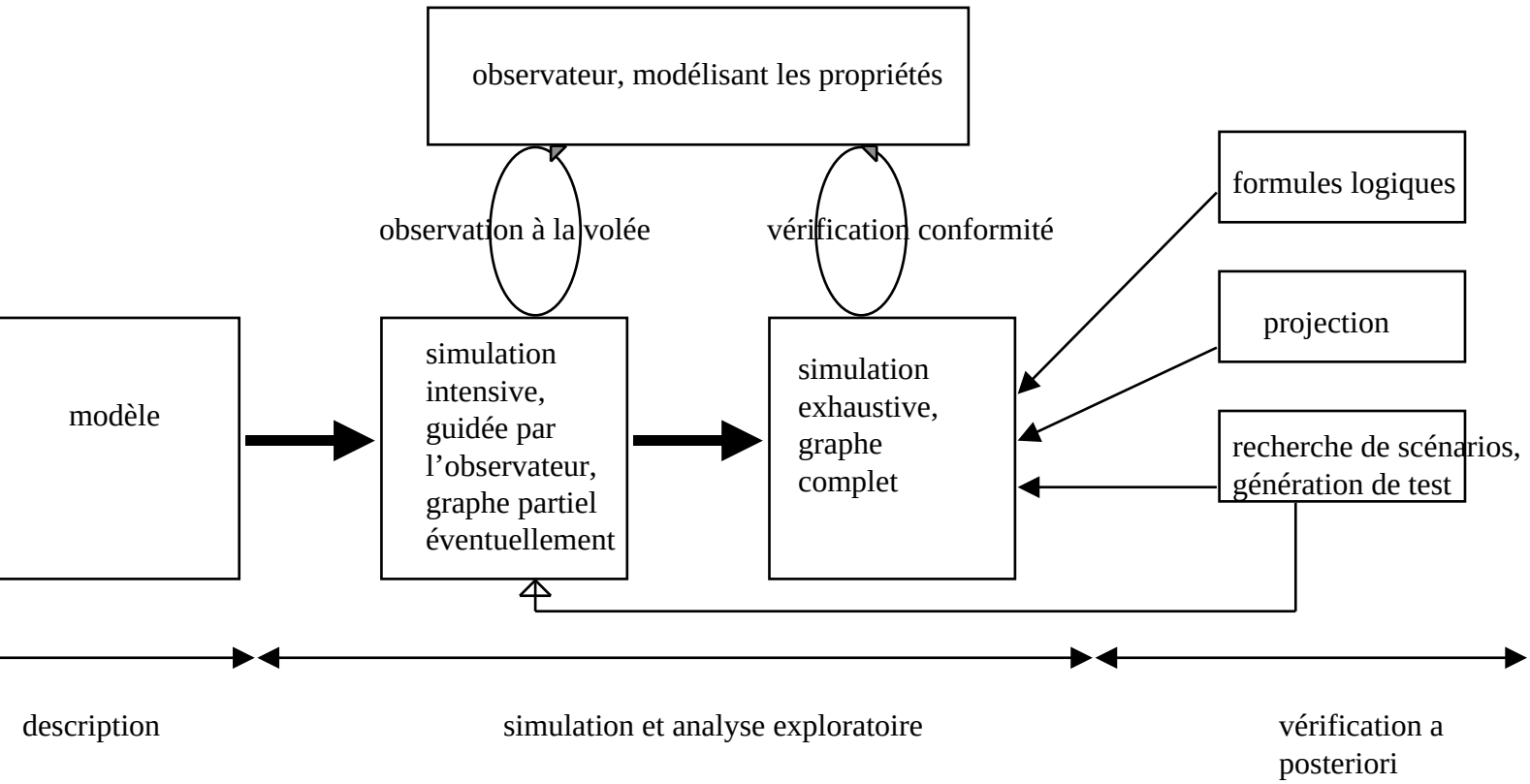
Exploration de graphe d’états

- **La simulation est insuffisante pour vérifier que**
  - tous les comportements possibles du système sont conformes aux spécifications, notamment de sûreté de fonctionnement
  - aucun événement contraire à la spécification (ou du moins à certain propriétés telles que la sécurité par exemple) ne peut se produire.
- **Exploration de graphe d’états**
  - simulation contrainte et guidée par un **observateur**
  - evaluation de propriétés
    - à la volée
    - sur le graphe complet a posteriori
  - construction de graphes “équivalents” par opérateurs sur les graphe projection, réduction

Le graphe d’états

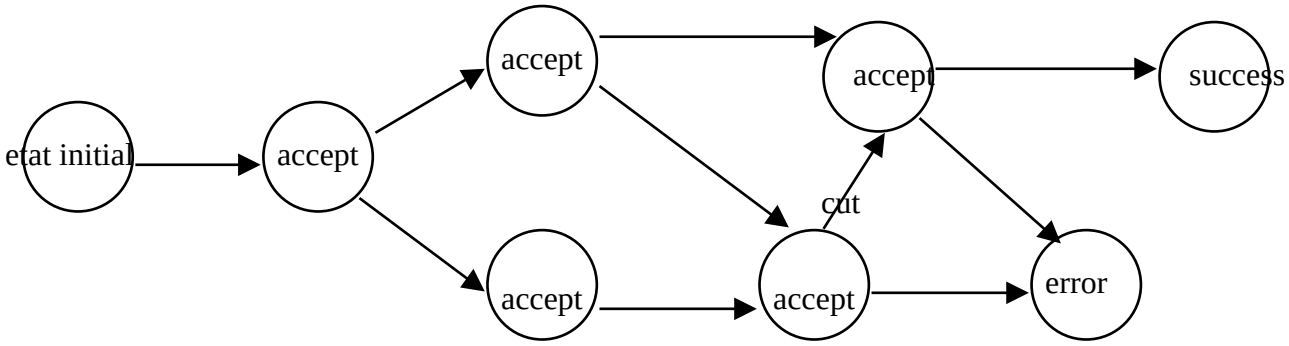


Les différentes techniques d'exploration



Structure d’un observateur

- **un observateur est un AEF, qui a des sondes (points d’observation) sur le modèle (événements, variables internes, etats des modules)**
- **on définit des états accepteurs (qui sont des états de progression ver une situation donnée), des états de succès (propriété cible vérifiée) e des états d’erreur (propriété cible violée)**
- **on peut couper des branches d’exploration du graphe pour faire de observations particulières**

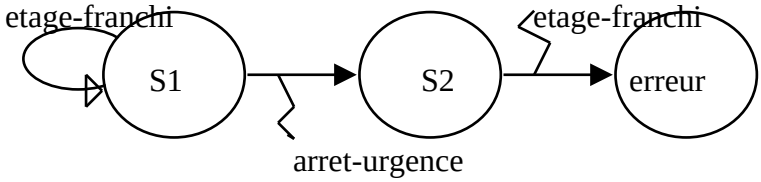


Observateur

- **Evolution de l’observateur :**
  - déterminée par l’observation des événements survenus lors du dernier pas de simulation
  - exécution synchrone du modèle et du (des) observateur(s)
  - A chaque pas, l’observateur évalue si une de ses transitions est tirab et si oui, il va dans l’état successeur adhoc
- **description de scénario attendu par un observateur**
  - init, debut du scenario, evénements observables, états de succès = déroulement complet du scénario, états de progression : états observables qui ne contredisent pas le déroulement du scénario, éta d’erreur correspondant à l’occurrence d’un événement contraire au scénario.
- **très grande simplicité d’utilisation**

exemple d’observateur

- **Propriété cherchée :**
  - après réception du message “arret-urgence”, aucun message étage-franchi n doit être reçu
  - il s’agit d’un invariant de séquence, qui permet de contrôler l’absence de comportement erroné (et de détecter une erreur résiduelle, violant cet invariant).



Evaluation de propriétés logiques

- **Propriétés d'états**
  - il s'agit souvent de conditions statiques de cohérence sur des variables d'états du système ou sur des variables du système et de l'environnement
    - exemple : un train délocalisé a une vitesse nulle :
      - cinematique-non-valide = vrai et contrôler-vitesse.state = vitesse nulle do être toujours vraie
      - on évalue que "non (cinematique-non-valide = vrai et contrôler-vitesse.st = vitesse nulle)" n'est jamais vérifié sur le modèle. Il s'agit ici d'une propriété de **"safety"**
      - à la volée, la simulation s'arrête dès lors qu'on rencontre un état qui vérif "non (cinematique-non-valide = vrai et contrôler-vitesse.state = vitesse nulle)" (c'est l'équivalent d'un état d'erreur sur l'observateur)
      - en exhaustif, on évalue la formule sur tous les états atteints

Propriétés de sûreté et propriétés de vivacité

- **propriété de sûreté**
  - exprime qu'un comportement "mauvais" ne se produit jamais au cours de l'exécution du système
    - exemple : absence de blocage, respect d'exclusion mutuelle ou d'une propriété d'état. il s'agit en général d'exprimer un **invariant du système.**
- **propriété de vivacité**
  - exprime qu'un comportement bon se produit toujours lors de l'exécution d'un système
    - exemple : terminaison, atteinte d'un état de succès (en particulier le retour à l'état initial)

Evaluation de propriétés logiques

- **Propriétés d'atteignabilité**
  - opérateurs portant sur les états
    - recherche des états à partir desquels une transition donnée est tirable (opérateur fireable)
    - recherche des états atteints directement par le tir d'une transition (opérateur after)
  - opérateurs portant sur les arcs
    - recherche des transitions tirables à partir d'un état source donné (opérateur src)
    - recherche des transitions aboutissant à un état cible donné (opérateur t
- **combinaison possible des formules**

Propriétés exprimées à l'aide de formules de logique temporelle

- **Cf cours pour logique temporelle : LTL, CTL, PTL notamment**
  - inevitabilité
    - il s'agit de montrer en général que quelque chose de bon est "inevitabl
      - application à la recherche de propriétés de sûreté
        - inev (appliquer-FU) rend les états dans lesquels seule cette transition est tirab
          Il faut vérifier manuellement que l'ensemble de ces états est complet
          (recouvrement de toutes les alarmes)
  - potentialité
    - extrait les états origines d'une séquence de transition menant à un état cible donné. (dont on vérifie l'atteignabilité)

Liens avec les tests fonctionnels

- **L’analyse des propriétés, et notamment des scénarios couvre :**
  - l’analyse des scénarios vus par l’utilisateur => faire abstraction des transitions internes
  - l’analyse des séquences internes de transitions
- **Tests**
  - boîte noire : on applique les séquences des entrées et on observe les séquences des sorties
  - boîte grise : on connait la structure en module et on observe les séquences de transitions (messages entre module) en plus des sortie perceptibles dans l’environnement
  - boîte blanche : on ajoute les variables d’états internes aux modules, ou encore des séquencements internes

Plan de Validation

- **Identifier pour chaque fonction**
  - objectifs du test
  - hypothèses sur les entrées
  - sorties attendues
  - événements observables
  - oracle
- **Identifier des scénarios “de bout en bout”**
  - objectifs du test
  - situation initiale
  - situation finale
  - séquence(s) d’événements observables
  - oracle

Phase de Conception

Phase de conception

- **Objectifs**
- **Documents d’entrée**
- **Documents de sortie**
- **Activités**

- **Validation de la conception**
  - cohérence et correction
  - conformité avec la spécification
    - traçabilité
    - comportement
    - propriétés

Phase de conception

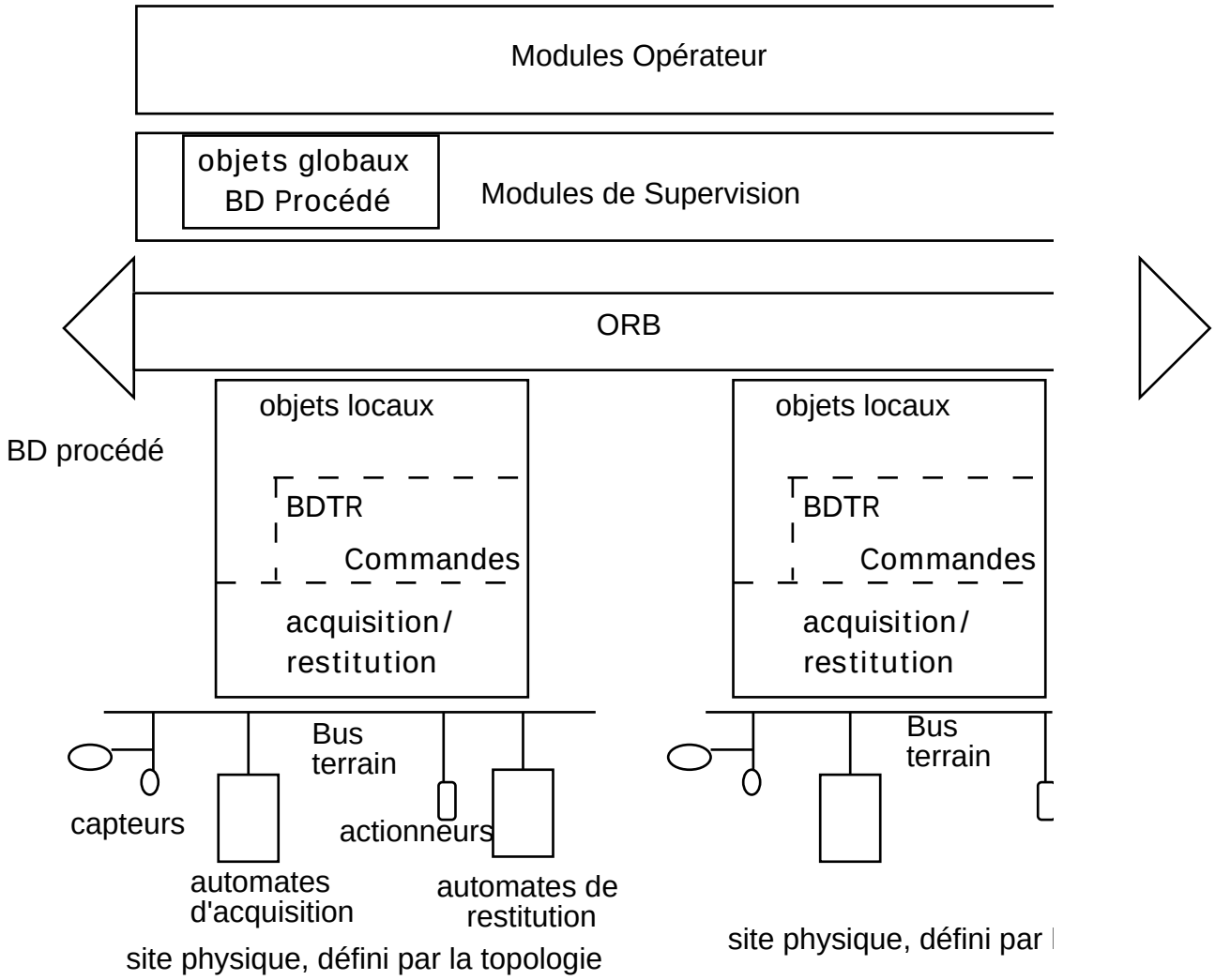
- **Objectifs**
  - Définir l’architecture logicielle
    - répartition des entités (ou des fonctions) identifiées dans la spécificatio sur une architecture matérielle et logicielle
    - “décomposition” ou “raffinement” des entités de la spécification en
      - objets
      - modules
      - constantes
  - Faire et justifier les choix des principes de réalisation (d’implantation)
    - choix d’algorithmes
    - fonctions “ de conception” (prise en compte répartition)
  - Démontrer que la spécification est correctement décrite
    - cohérence entre description spécification et conception
    - comportement
    - propriétés

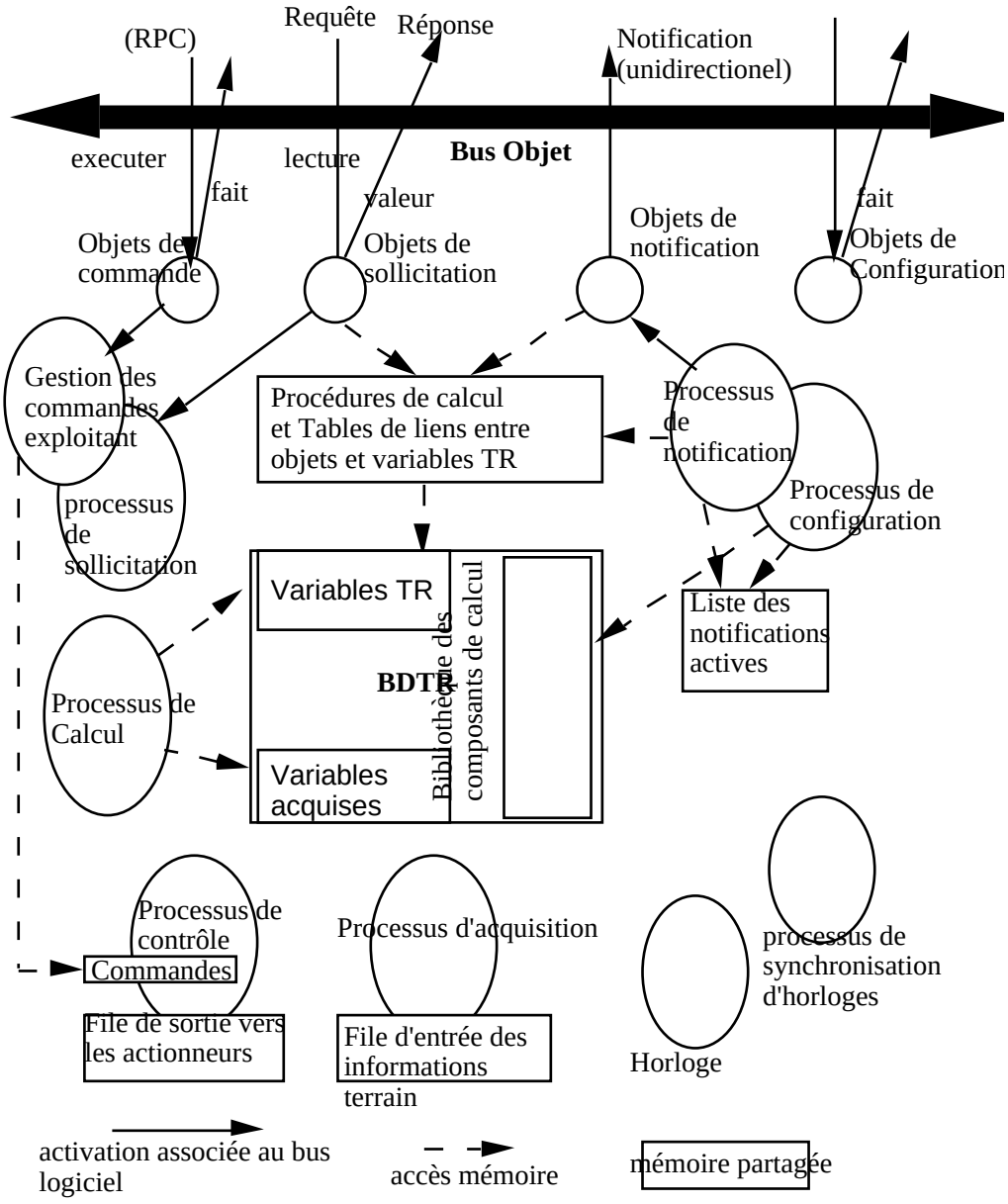
Phase de conception

- **Documents d’entrée**
  - Document de spécification
    - entités
    - description des comportements locaux
    - description des comportements globaux
- **Documents de sortie**
  - Documents de Conception Générale (ou Préliminaire)
    - description des modules
    - description des interfaces entre modules
    - traçabilité avec la spécification
  - Plan de test d’intégration
  - Document de conception détaillée
  - Plan de test unitaire

Phase de conception

- **Activités**
  - Répartition des entités de spécification sur architecture matérielle (distribution)
    - blocs
    - processus
  - spécification des protocoles entre entités réparties ou des contrainte de coopération
  - identification des mécanismes support des communications entre objets ou des interfaces fonctionnelles
    - mémoire partagée
    - message
    - RPC
    - autre mécanisme
  - Vérification de la conception (blocs, processus)


Modules Opérateur
objets globaux
BD Procédé
Modules de Supervision
ORB
BD procédé
objets locaux
BDTR
Commandes
acquisition/
restitution
objets locaux
BDTR
Commandes
acquisition/
restitution
Bus
terrain
capteurs
actionneurs
automates
d'acquisition
automates de
restitution
Bus
terrain
site physique, défini par la topologie
site physique, défini par


(RPC)
Requête
Réponse
Notification
(unidirectionel)
Bus Objet
executer
fait
lecture
valeur
fait
Objets de
commande
Objets de
sollicitation
Objets de
notification
Objets de
Configuration
Gestion des
commandes
exploitant
processus
de
sollicitation
Procédures de calcul
et Tables de liens entre
objets et variables TR
Processus
de
notification
Processus de
configuration
Variables TR
BDTR
Variables
acquises
Bibliothèque des
composants de calcul
Liste des
notifications
actives
Processus de
Calcul
Processus de
contrôle
Commandes
File de sortie vers
les actionneurs
Processus d'acquisition
File d'entrée des
informations
terrain
Horloge
processus de
synchronisation
d'horloges
activation associée au bus
logiciel
accès mémoire
mémoire partagée

Phase de conception

- **Activités (suite)**
  - Choix et spécification (au niveau considéré) des structures de données
  - Choix des principes de réalisation des méthodes ou fonctions (compte tenu du mode de spécification)
  - Vérification de la conformité des choix de conception aux propriété de comportement des objets ou aux propriétés des fonctions implantées.
    - justification textuelle
    - dérivation de code
    - raffinement et preuve

Phase de conception

- **Validation des choix de conception**
  - Au niveau Blocs et Processus
    - Faisabilité
    - Performance
    - Comportement : ordonnancement, absence de blocage, cohérence des données partagées,
    - Robustesse aux pannes
  - Au niveau Données, Structures de Données et Algorithmes
    - Description formalisée
    - Analyse/Relecture critique
    - Raffinement et démonstration

Phase de conception

- **Formalisation, notations ???**
  - Architecture et parallélisme
    - Modèles SDL : blocs, canaux, processus, services
    - Modèles UML
  - Projection sur une architecture physique
    - caractérisation des canaux
    - diagrammes de déploiement
  - Corps des procédures ou méthodes
    - pseudocode
    - SA

Phase de conception

- **Elaboration des plans de test d'intégration**
  - Le plan de test prend en compte la description des interfaces entre modules et la nature des modules
  - Description de la stratégie d'intégration
    - Intégration Module i et Module j
      - conditions d'entrée (devant être initialement vérifiées) sur chacun des modules
      - interactions entre les modules
        - nominal
        - dégradé
      - tests préconisés
        - conditions de mise en oeuvre
        - résultat attendu

Conception détaillée codage

Passage du modèle au code ?

Projection des concepts du modèle sur un langage de programmation

- en général le modèle statique contient tout ou partie de la tructure déclarative :
  - en approche fonctionnelle, il faut projeter le dictionnaire des données (interfaces des fonctions) sur des variables et les fonctions sur des procédures. Variables et procédures sont réparties sur les divers processus de la conception. Il s'agit d'un processus manuel.
  - certains langages de spécification, dérivés directement des AEFC permettent de décrire des types, des blocs, des processus, des signaux o messages échangés entre processus, des procédures.
    - exemple de tels langages : SDL, pour les problèmes "event driven", les protocoles, les systèmes asynchrones,...
    - des langages tels que LUSTRE, SIGNAL, ESTEREL pour les problèmes "time triggered"

Passage du modèle au code ?

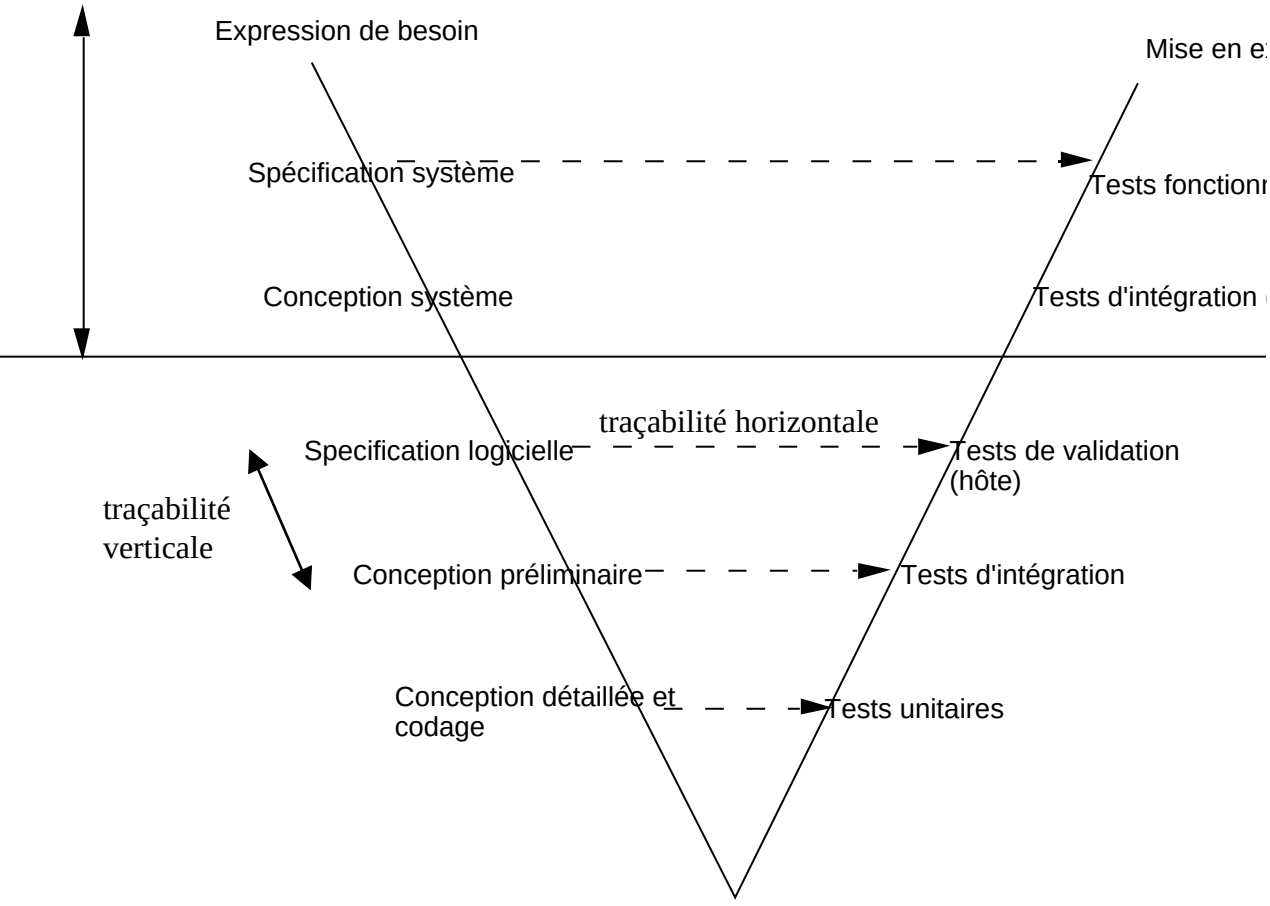
- **projection du modèle sur les langages de programmatio (suite)**
  - passage du modèle statique à la structure déclarative (suite)
    - passage du modèle objet (type UML) sur les langages objets (C++, Jav : manuel ou dérivation du code (par exemple génération des classes à partir des Class diagrams)
      - cas particulier des objets persistants (SGBD objet)
    - passage du modèle objet à un langage de programmation classique (exemple ADA ou C)
      - traduction manuelle de tous les éléments d'un Class diagram en structures de données
    - dans tous les cas de figure, nécessité d'implanter "manuellement" les associations et aggrégations

Passage du modèle au code ?

- **Implantation de la dynamique**
  - traduction des diagrammes d'états (FSM ou state charts) en structur de contrôle (squelette)
  - projection de la communication inter-processus sur un langage et u cible
    - Pas de problème pour les langages orientés parallélisme et répartition contrôle (ESTELLE, LOTOS, CSP,CCS, SDL, ADA) Dans ce cas on peut faire une projection automatisée => génération de code, sous réserve que le code source ainsi généré puisse être ensuite produit sur machine cible considérée
    - Pour les langages objet qui supportent les notions de parallélisme (Jav dérivation manuelle mais systématique
    - Pour les langages objet ne supportant pas ces concepts nécessité d'utiliser des bibliothèques système => dérivation plus "artistique"

Techniques de test, Validation du logiciel

Situation dans le cycle de vie du logiciel



Rôle du test

- **Elimination des erreurs résiduelles**
  - logiciel => erreurs de conception par opposition aux défaillances catalectiques
- **Réduction des risques liés à l'utilisation du logiciel**
  - orientation des tests vis à vis des événements redoutés (sécurité, indisponibilité)
- **erreur -> défaut -> défaillance**
  - logiciel correct : pas de défaut ("0-défaut")
  - logiciel fiable : pas de défaillance (notion d'observabilité de la défaillance, c'est à dire que le défaut traverse la frontière logiciel - environnement)
  - Le test participe à la démonstration de la correction d'un logiciel, a moins vis à vis de certaines propriétés. Il s'agit donc d'une démarch SYSTEMATIQUE

Le test dans un schéma de démonstration

- **Un certain nombre d'activités complémentaires**
  - Sur la branche descendante
    - Utilisation de méthodes de conception et de développement
    - Méthodes de spécification formelle et de production de code par raffinements successifs
  - Sur le code source
    - Inspection de code,
    - Relecture critique de code
    - Analyse statique
  - Sur la branche d'intégration
    - recette des modules
    - validation hôte
    - validation cible

Tests dans le cycle de vie

- **Hierarchisation issue de la décomposition descendante**
  - tests système correspondent au test du logiciel sur cible et dans un environnement simulé (baies de test) puis réel (essais)
  - tests de validation tests du logiciel sur hôte souvent (parfois sur cib également). on vérifie le comportement pour l'intégralité des fonctionnalités identifiées dans la spécification de besoin
  - tests d'intégration
    - intégration de modules jusqu'au niveau fonctionnel
    - stratégie d'intégration
  - tests de composants logiciels
  - tests unitaires

Tests dans le cycle de vie

- **Prise en compte des itérations successives dans le cycle d développement**
  - tests de non régression : vérifier que les modifications n’ont pas altéré le logiciel par rapport à sa version précédente. Il peut y avoir des tests de non régression à divers niveaux en fonction de la localisation et du niveau de la modification.

Quelques problèmes du test

- **Plan de tests**
  - comment déterminer les “jeux” de test pertinents ?
    - couverture des instructions
    - couverture des branches
    - couverture des CDD
    - couverture des domaines d’entrée
    - couverture des domaines de sortie
  - comment déterminer les oracles des tests ?
    - calcul du résultat attendu
    - propriétés
  - comment décider de l’arrêt des tests ?
    - objectif de couverture structurelle
    - objectif de couverture des entrées et des sorties
    - Nombre d’erreurs résiduelles estimé

Quelques problèmes du test

- **Réalisation des tests**
  - Ecriture de tests opérationnels
  - observation des variables et des résultats
  - Contrôle de l’effort de test
    - progression de la couverture des tests
      - instructions, branches
      - domaines des entrées et des sorties
    - Suivi de la croissance de fiabilité
    - Test aléatoire, injection de faute

analyse quantitative des bugs

Répartition par phase d’apparition

- spécifications + fonctionnalités : 25%
- structuration : 25%
  - séquencement
  - traitements
- données : 23%
- Codage : 10%
- Intégration : 10%

répartition spatiale des erreurs résiduelles

- points durs, parties oubliées et peu testées...

mesure de l’effet

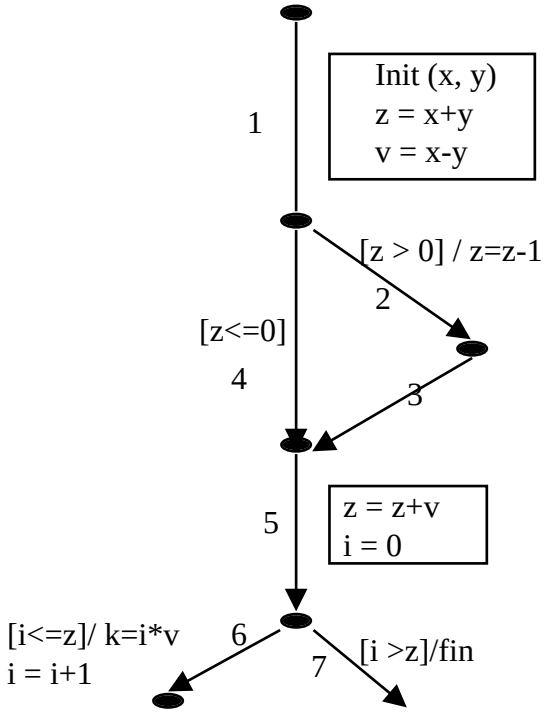
- fréquence d’apparition,
- coût de correction
- màj
- conséquences

Test structurel basé sur les chemins: flots de données et graphe de contrôle

Test structurel et graphe de contrôle

- **Modèle sous jacent :**
  - le programme est vu comme un graphe
    - les noeuds décrivent les points de décision
    - les arcs décrivent des branches et donc généralement des séquences d'instructions.
- **objectif :**
  - le test des chemins cherche à couvrir toutes les instructions et toutes les décisions
    - Couverture de chacune des instructions des blocs séquentiels
    - Parcours de tous les noeuds de décisions et de chacune des branches (CDD)

exemple



test structurel et sélection des entrées

- **détermination des séquences permettant de parcourir toutes les branches**
  - à chaque point de décision, déterminer les conditions à atteindre po couvrir chacune des branches.
    - le calcul de la valeur des entrées pour positionner le test sur un chemin donné peut se faire par exécution manuelle ou symbolique.
      - exemple trouver x et y de manière à couvrir les différents chemins :
        - 1234567
        - 14567
        - 123567
        - 1457
        - 12357
      - pour le point de décision 1la branche 14 est atteinte pour (x,y) / x+y <=0 => y < -x avec x>=0. par exemple x=1 et y =-1.....

test structurel et sélection des entrées

- **hypothèse sur les classes d'équivalence**
  - si une variable prend ses valeurs dans un domaine borné [inf, sup], si on fait l'hypothèse que le programme (ou un bloc donné) se comporte de manière identique sur cet intervalle, on testera les valeurs inf, sup, éventuellement un point milieu pour vérifier la conformité du comportement au résultat attendu.
- **prise en compte des valeurs hors domaine de définition**
  - vérifier qu'une telle valeur ne provoque pas de comportement erron
- **attention aux variables qui peuvent prendre des valeurs hors domaine suite à des opérations**
  - => risque d'erreurs d'exécution telles que débordements de tableau (un pointeur est incrémenté et dépasse les bornes du tableau), ou encore les divisions par 0 ou les débordements,....

test structurel et structures itératives ou récursives

- **Trouver le nombre min de chemins pour couvrir différentes structures :**
  - boucles simples
    - combien de tours ??
  - boucles concaténées
  - boucles imbriquées
    - trouver les combinaisons limites (inf d'une boucle, inf de la seconde, puis inf et sup, ...)

contrôle de la couverture de test structurel

- **Détection de code mort**
  - il s'agit de modules non accessibles dans une configuration opérationnelle des paramètres
  - mauvaise conception
- **Contrôle du taux de couverture obtenu**
  - couverture des instructions
  - couverture des chemins de écision à décision
- **Contrôle des résultats et du chemin suivi.**
- **Complexité et taux de couverture**
  - limiter la complexité du programme si on veut atteindre un taux de couverture structurel de 100% sur les chemins
    - limiter le nombre cyclomatique (v(g)<10 par exemple)

test structurel et flots de données

- **Le test des chemins ne rend pas toujours compte de l’évolution des données**
  - on peut orienter la selection des chemins par les flots de données, e
    suivant les différents modes d’utilisation d’une variable donnée.

Test d’intégration

test d’intégration

- **Objectif**
  - vérifier la cohérence des interfaces entre modules
  - vérifier les propriétés de comportement au niveau des appels.
- **Modèle sous jacent**
  - le graphe d’appel de l’application
    - architecture : modules et liens structurels entre composants
    - comportement : relations d’appel ; il s’agit d’un graphe qui explicite le passages de paramètres.
- **Hypothèse**
  - les modules unitaires à intégrer sont supposés testés avec un niveau suffisant de couverture pour chaque module.

test d’intégration

- **Stratégie d’intégration**
  - supposée définie dans le Plan de test d’intégration, réalisé en phase de conception générale
  - dépend de la structure de l’application
    - test de chaque module ou composant avec les modules qui interagissent avec lui, indépendamment
      - éventuellement utiliser des bouchons qui simulent les interactions.
    - puis intégration des modules sous forme de big bang, ou fonction à fonction.
  - l’intégration peut se faire selon une logique “fonctionnelle”
    - intégrer les modules fonction par fonction

test d’intégration

- **Contrôle de l’effort de test**
  - couverture des chemins d’appel
  - couverture des fonctionnalités
- **difficultés spécifiques de l’intégration**
  - mémorisation de variables ou “décalages” temporels entre modules
    - tester tous les cas de figure et vérifier les propriétés de cohérence globale peut être très difficile
      - exemple des protocoles de communication
      - modèles à files d’attente
- **la conception de l’application doit être la plus simple possible**
  - exemple des modèles FDS pour les applications critiques
  - RDV plutôt que files d’attente
  - ....

Tests d’intégration

- **Stratégie d’intégration pour les méthodes objet**

Test fonctionnel

Test fonctionnel

- **Objectif**
  - vérifier la conformité du comportement du logiciel à sa spécificatio
    - test de bout en bout, en boîte noire ou grise (car un test fonctionnel peu être l'aboutissement d'un test d'intégration!!)
- **Modèle sous jacent : le modèle de spécification!**
  - Flots de données de bout en bout
  - Scénarios fonctionnels et chemins formalisés par des observateurs pour les spécifications basées sur les AEF
  - Diagrammes de séquencement de messages pour les spécifications SDL, UML,..

Test fonctionnel

- **Stratégie de test**
  - couverture fonctionnelle
    - couvrir les scénarios identifiés dans le modèle de spécification
    - couvrir les domaines d'entrée pertinents de l'environnement (frontière identifiée)
    - couvrir les comportements aux limites de l'environnement (et des opérateurs)
  - architecture de test
    - sur machine hôte
    - sur machine cible
    - environnement simulé ou environnement réel
  - difficulté de l'observation en environnement réparti
    - cohérence globale
    - reproductibilité des résultats

Test fonctionnel

- **Contrôle de l’effort de test**
  - croissance de fiabilité
  - épuisement de l’imagination des testeurs et des utilisateurs
  - épuisement du budget
- **Difficultés**
  - l’oracle
    - propriétés de logique temporelle
      - formules
      - observateur
  - conditions initiales d’observation d’un scénario dans l’environnement