Prouver en B une propriété de sécurité d'un système basé sur un micronoyau

présentation: Sophie Coudert
sophie.coudert@telecom-paristech.fr

LabSoC, Telecom-ParisTech

STMicroelectronice, Clearsy, CEA

Rencontre SAFA, 1er Juillet 2008

Objectifs, déroulement et résultats

- Objectifs initiaux:
  - Propriété d'isolation mémoire entre applications
  - Micronoyau: petit $\Rightarrow$ méthodes formelles envisageables
  - Méthode B: type d'application nouvelle $\Rightarrow$ faisabilité ?
- 4 ans: STMicroelectronics, LabSoC, Clearsy, CEA
  - Modèle B du μnoyau. Limites $\Rightarrow$ extensions méthodologiques
  - Enrichissement du μnoyau.
    Développement et modélisation des premiers serveurs.
  - Expression "high level" de la propriété, preuve par raffinement
- De nombreux retours
  - Une méthodologie de preuve crédible
  - D'autres apports: documentation, test
  - Faisabilité raisonnable, quelques points délicats.

Plan de l'exposé

- B: présentation rapide.
- L4 et son modèle B
- Au dessus de L4 (serveurs)
- Modélisation des serveurs et preuve de propriété
- Bilan & Conclusion

B: Machines à états et préservation d'invariants

Machine X

partie statique

Ensembles
Constantes
Propriétés associées

partie dynamique

Variables (état)
Invariant **I**
Initialisation **Init** = ...
Opérations/Events
op1 = ...
op1 = ...
op1 = ...

Propriétés, invariant: 1er ordre.
Initialisation, opérations:
changement d'état conditionné.

- Blogiciel: avec paramètres et résultat
- Event B: sans paramètres

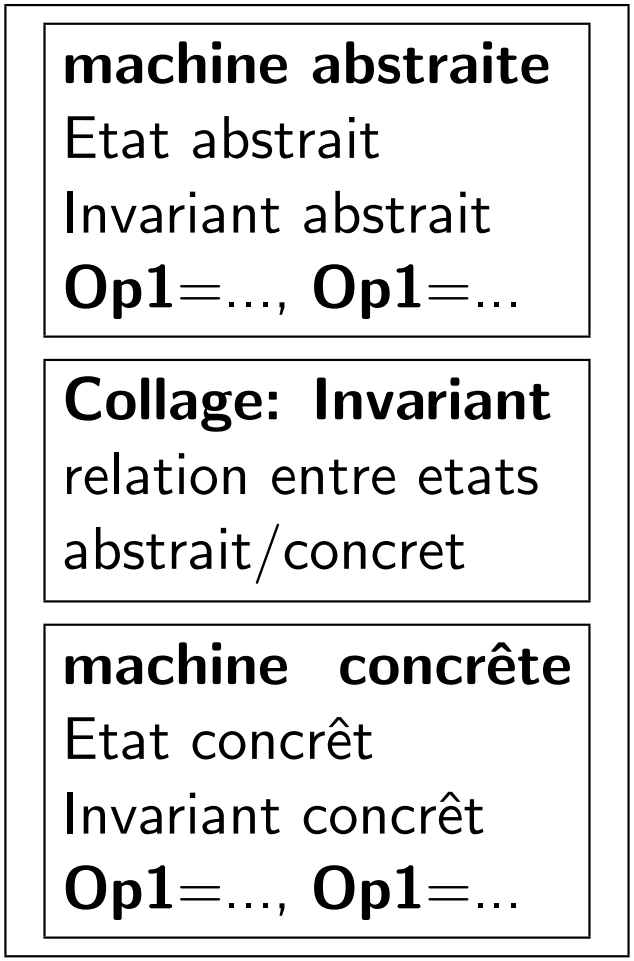
Obligations de preuve:

- [**Init**] **I**: après **Init**,**I** est vrai.
- $\forall$**op**, **I** $\rightarrow$ [**op**]**I**

Sémantique associée:

- suites d'opérations/events
- opérations atomiques

B: Raffinement



Invariant de collage: 1er ordre.
Obligations de preuve (roughly):

- inclusion de traces
- préservation de vivacité

2 optiques de raffinement:

- Blogiciel: Jusqu'au code. Langage évolue. (bien outillé) *Sequencement* traité.
- Event'B: analyse de systèmes. Raffinement du temps. Plus puissant, moins outillé. *Séquencement* manuel.

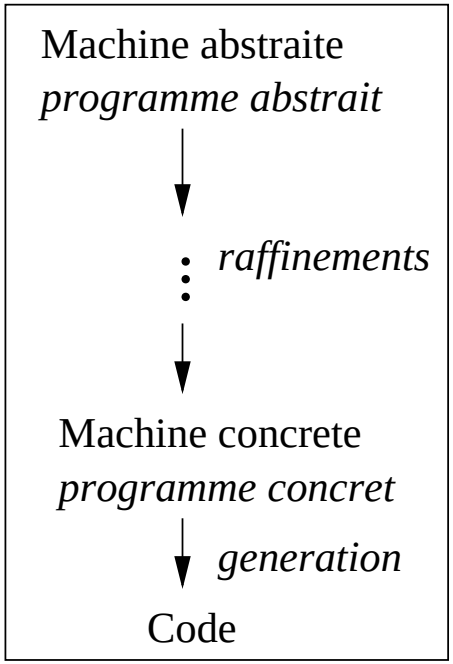
B, miniexemple

```
MACHINE   swap
VARIABLES xx, yy
INVARIANT
 xx : NAT &
 yy : NAT
INITIALISATION
 xx :: NAT ||
 yy :: NAT
OPERATIONS
 echange =
  BEGIN
   xx := yy ||
   yy := xx
  END
END
```

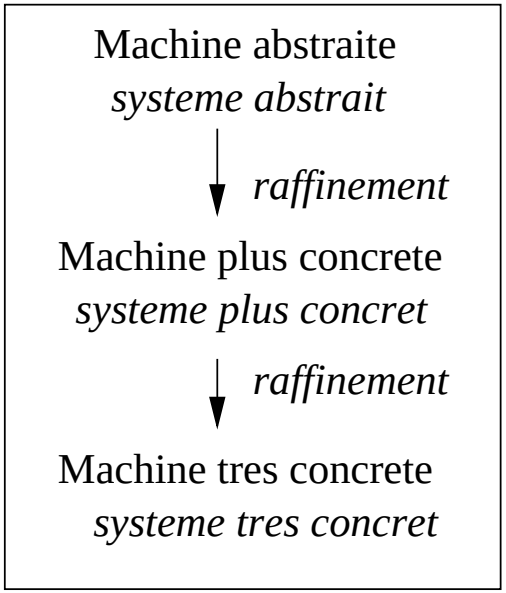
```
REFINEMENT  swapR
REFINES     swap
VARIABLES   xr, yr, zr
INVARIANT
 xr= xx & yr = yy & zr : NAT
INITIALISATION
   xr, yr, zr:= 0, 0, 0
OPERATIONS
 echange =
  BEGIN
   zr := xr;
   xr := yr;
   yr := zr
  END
END
```

Raffinement et garanties

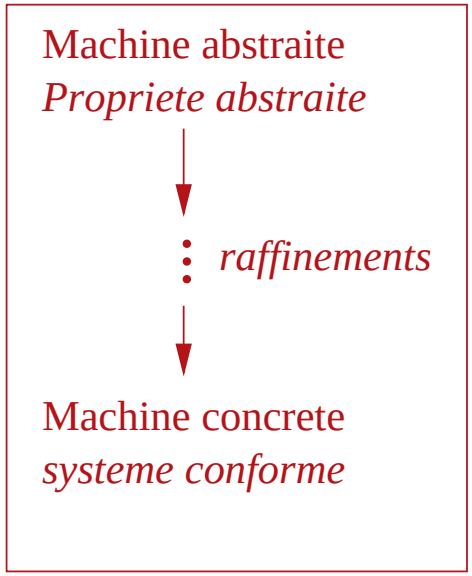
- préservation des invariants abstraits
- respect des comportements abstraits (déterminisation progressive)



Developpement logiciel

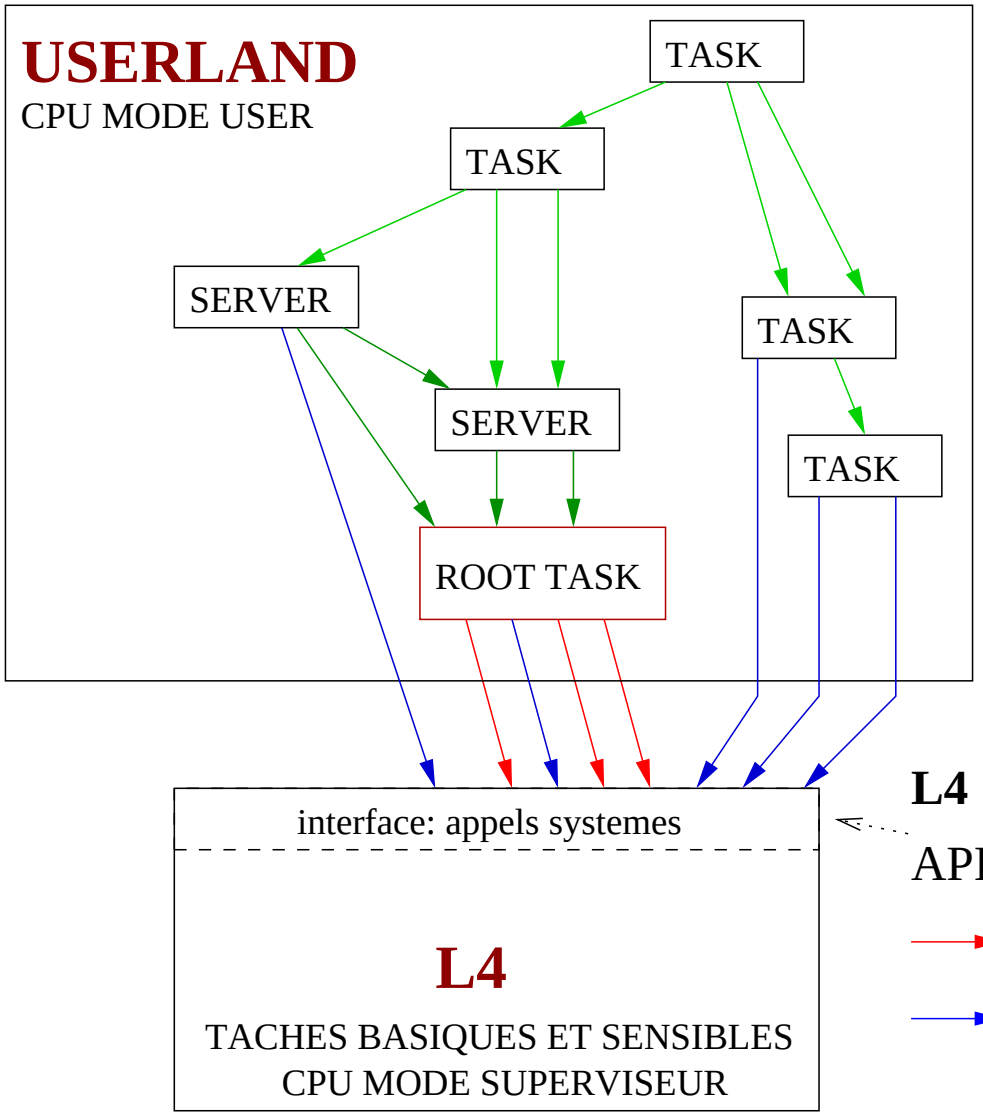


Vues d'un systeme reactif a differentes granularite



Preuve de propriete par raffinement

L4 based system: principe



: *appels de fonctions*

USERLAND:

Taches utilisateur

1 Tache privilegiee: ROOT (Adress Space)

Serveurs: taches utilisateur

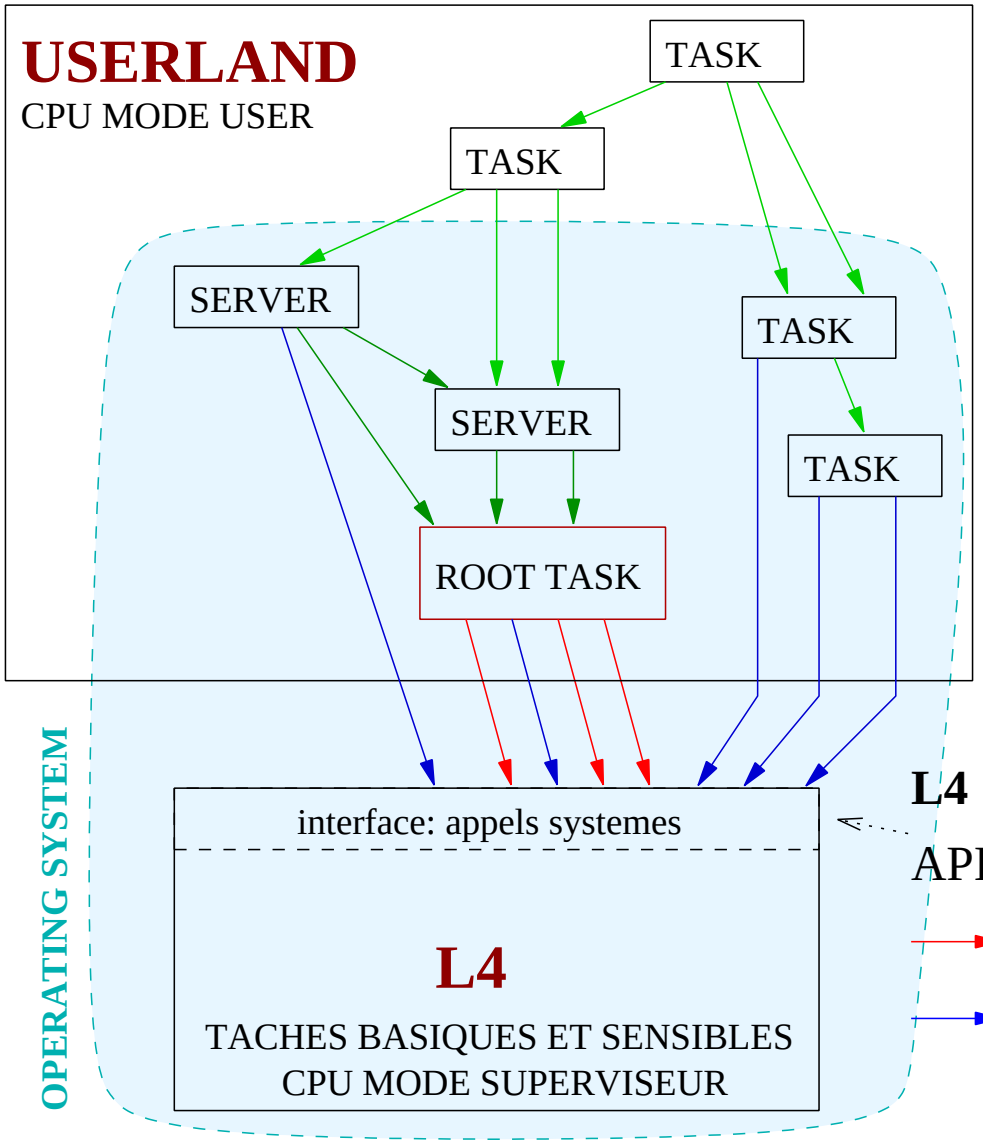
L4

API : specification des appels systeme

Privilegies: tache root

Non privilegies: taches utilisateur

L4 based system: OS



: *appels de fonctions*

USERLAND:

Taches utilisateur

1 Tache privilegiee: ROOT (Adress Space)

Serveurs: taches utilisateur

L4

API : specification des appels systeme

Privilegies: tache root

Non privilegies: taches utilisateur

L4 & Sécurité: extensions

USERLAND
CPU MODE USER

→ : *appels de fonctions*

USERLAND:

Taches utilisateur

1 Tache privilegiee: ROOT (Adress Space)

Serveurs: taches utilisateur

Services de securite

interface: appels systemes

extension

L4

TACHES BASIQUES ET SENSIBLES
CPU MODE SUPERVISEUR

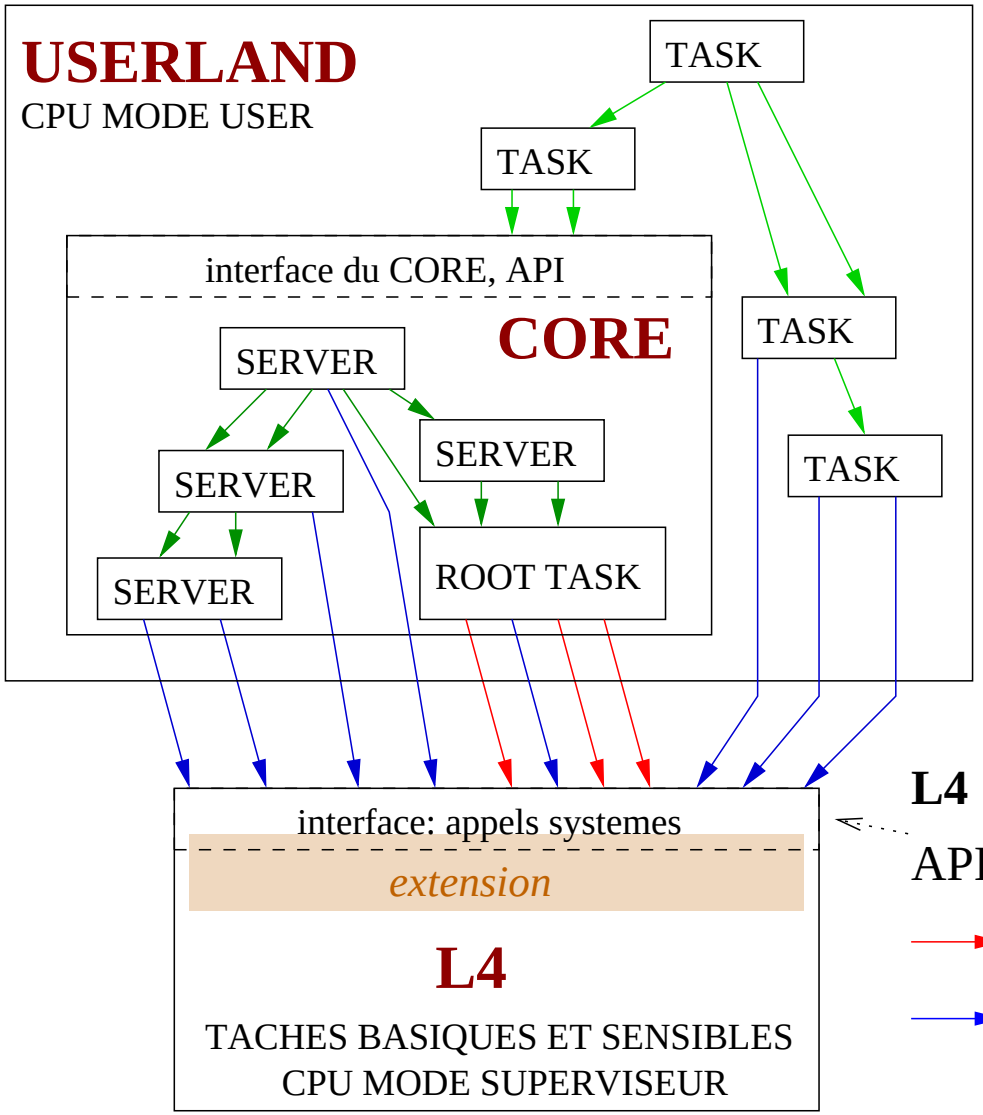
L4 enrichi (securite)

API : specification des appels systeme

→ Privilegies: tache root

→ Non privilegies: taches utilisateur

L4 based system: architecture de sécurité



→ : *appels de fonctions*

USERLAND:

Taches utilisateur

1 Tache privilegiee: ROOT (Adress Space)

Serveurs: taches utilisateur

CORE:

Ensemble de serveurs

Comprend Root (dans 1 serveur)

Interface: API

Services de securite

L4 enrichi (securite)

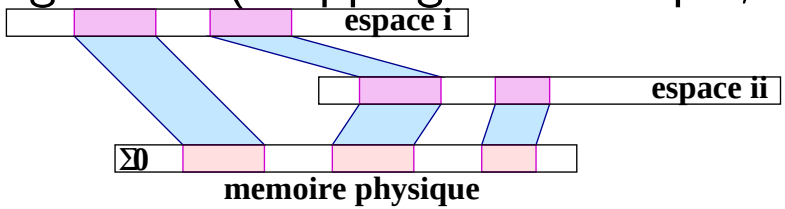
API : specification des appels systeme

→ Privilegies: tache root

→ Non privilegies: taches utilisateur

L4 essentials and API

- **L4 essentials:**
  - Concepts fondamentaux: threads, address spaces, IPC (appel system "Inter Process communication")
  - Système d'espaces d'adressage riche (mapping hierarchique, database complexe)



  - 10 syscall, fausse simplicité: nombreux paramètres. (`threadcontrol, spacecontrol,IPC,...`)
- **L4 API:** (document original)
  - Abstrait les évènements internes.
  - Spécification textuelle de qualité
  - Décrit l'effet de chaque appel pour chaque jeu de paramètres
  - Difficile à aborder (il faut tout lire...)
  - User manual manquant (ébauches).
- Plusieurs implémentations (FIASCO, PISTACCHIO,...)

Modèle B de l'API de L4

- Micronoyau monolitique: spécification à plat.
  Modélisation de l'état, partagé par tous les appels.
- B évènementiel:
  - Paramètres supprimés $\longrightarrow$ modification de l'état.
  - Decomposition des appels: $\sim$ 1 évènement par cas (param.)
    En fait: 1 évènement par tâche $\Rightarrow$ "user manual".
- Haut niveau d'abstraction (plus que l'API):
  (pas d'identificateurs,. . .)

Modèle B de L4, exemple

```
L4_ThreadControl_creation_in_a_new_space =
 ANY  in_dest, in_scheduler,...
 WHERE   in_dest              : global &
         in_scheduler         : existing &
         space(current_tcb) : {root_space, sigma0_space} &
         (space_t - allocated_space) /= {} &...
 THEN
  ANY    new_space, tcb, idlocal,...
  WHERE  new_space          : (space_t - allocated_space) &...
  THEN   allocated_space    := allocated_space \/ {new_space} ||...

L4_ThreadControl_creation_active_in_an_existing_space =
  ANY    in_dest, in_scheduler, in_pager, dest_space,...
  WHERE  space(current_tcb) : {root_space, sigma0_space} &
         in_dest              : global &
         in_scheduler         : existing &
  THEN...

L4_ThreadControl_creation_inactive_in_an_existing_space =...
```

Modèle B de L4, bilan

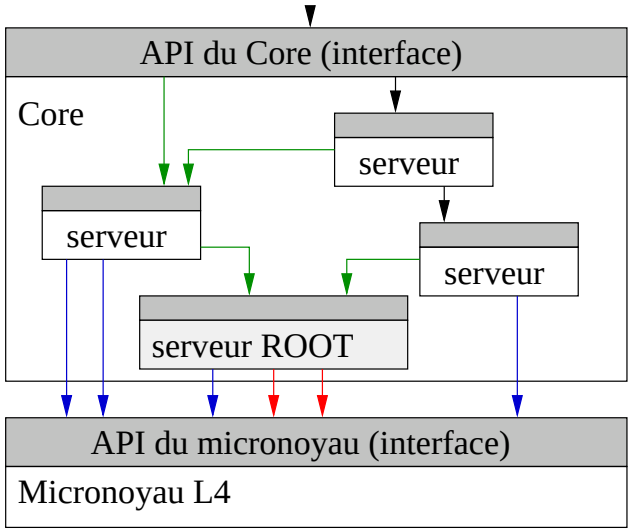
- Faisabilité API L4: OK.
  | stage Master 6 mois, encadré (Clearsy, LabSoC, ST).
  Extension: Plus difficile
  | $2^{\text{ième}}$ stage. Spécification peu mûre & moins d'interactions.
- Abstraction essentielle pour les preuves.
  (Mapping Database, et plus. . .)
- Questions de sécurité: trop "low-level"
  - L4 permissif: programmation "offensive"
    $\Rightarrow$ conditions d'appel essentielles
  - concepts pertinents absents à ce niveau.
- Méthodologie: implémentation existant $\neq$ raffinement
  $\Rightarrow$ complément nécessaire pour compatibilité modèle/code.
  Autres apports et exploitation de la modélisation.

Modèle B de L4, méthodologie et exploitation

- Confronter le modèle au code:
  - Preuve de code (FramaC: ancien CAVEAT, CEA)
    - adaptation de FramaC au C++ de Pistacchio.
    - Approche interprétation abstraite: code fantôme.
      Bien outillée, un peu loin de B.
    - Approche weakest precondition: propriétés logiques.
      Très proches de B. Moins bien outillée.
  - Oracle de Test (Animation B. Collaboration avec codeur)

  Similarités. Abstraction parfois problématique. Raffinement souhaitable.
- Autres apports
  - Préciser la spécification informelle (debug apprécié)
  - Documentation précise, user manual "grain fin".
    génération automatique "langage naturel" à partir d'un dictionnaire

Propriété de sécurité: au dessus de L4



Core: interface pour les applications.

Raffinement:

- appel Core $\longrightarrow$ suite d'appels serveurs.
- appel serveur $\longrightarrow$ suite d'appels serveurs/L4.

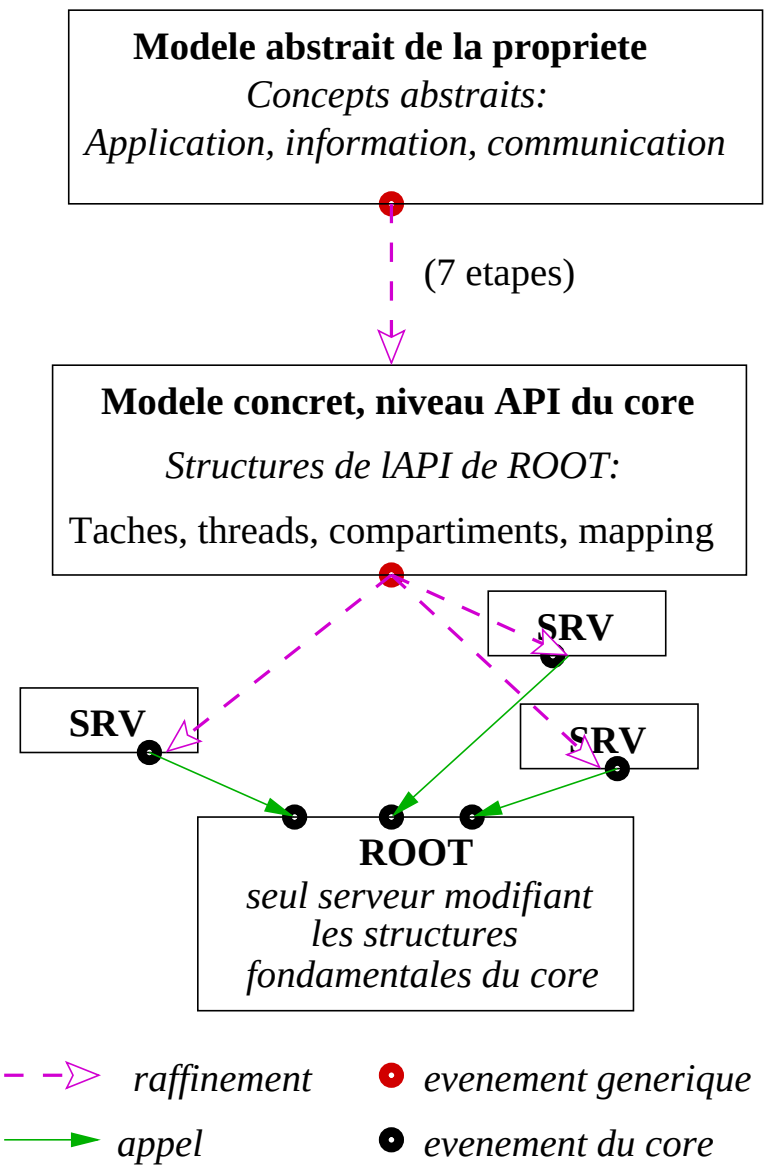
ROOT: tâches sensibles (mémoire,root l4).

Serveurs maison: petits, états locaux.

Difficulté pour B: interleaving et atomicité des évènements.

- B Logiciel: séquencement outillé mais supposé ininterruptible. Solution crédible si l'interleaving est non critique (à montrer).
- B évènementiel: séquencement (manuel) interruptible. Découpage subtile des évènements pour l'atomicité. Gère correctement l'interleaving mais. . . trop laborieux.

Preuve de sécurité: Méthodologie



Modèles pour preuve: éviter l'inutile.

- Prouver sur des modèles abstraits
- Ne raffiner que ce qui concerne la propriété recherchée.

Modèle de la propriété:

- Appartenance de l'information aux applications et communications
- 1 évènement générique pouvant modifier tout aspect de l'état

Relation au Core:

- l'évènement générique est raffiné vers les concepts de ROOT.
- Tout evènement de ROOT (appel possible par un serveur) raffine l'évènement générique.

Aperçu: Propriété, modèle abstrait

- 4 objets: informations, les applications, les liens de communication, politique de sécurité.
- Principes:
  - Une information est possédée par une unique application
  - Les liens de communication entre les applications sont autorisés par la politique de sécurité

  (implicite: les informations détenues par une application sont accessibles aux applications en communication avec elle)
- Un événement `modify_infos_and_communications` permettant
  - la création/suppression d'informations, d'applications, de liens de communications
  - le transfert d'informations par changement d'application propriétaire

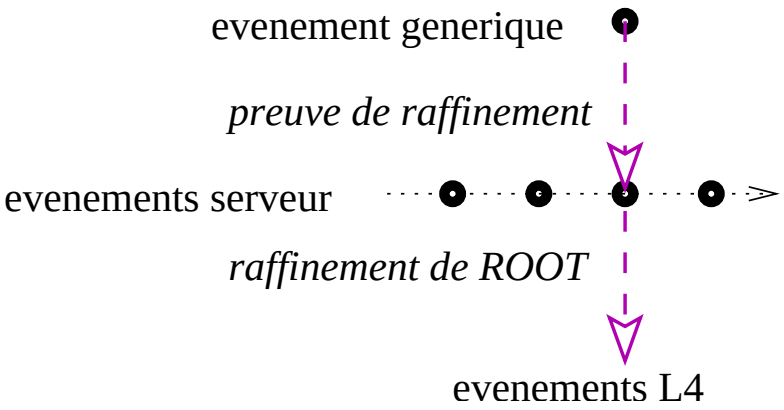
Aperçu: Propriété, modèle abstrait

```
INVARIANT ...
  Applications <: APPLICATIONS &
  Infos        <: INFORMATIONS &
  Communication : Applications <-> Applications &
  Communication~ <: Communication &
  SecurityPolicy : APPLICATIONS <-> APPLICATIONS &
 /* SECURITE */
  InfoOwner     : Infos --> Applications &
  Communication <: SecurityPolicy
EVENTS ...
  modify_infos_and_communications =
    ANY applications, infos, infoowner, communication,...
    WHERE
      applications <: APPLICATIONS &
      communication~ <: communication & ...
    THEN
      Applications  := applications  ||
      Communication := communication || ...
  END
```

Aperçu: Propriété, modèle concret

```
REFINEMENT
VARIABLES ...
  /* Application remplacee par compartiment ID */
  CompID,          /* Ensemble des compartiments ID */
  SecurityPolicyMappingID,
  Tasks,           /* Ensemble des taches actives */
  TaskCompID       /* Lien application/taches */
  Threads,...      /* ensemble des threads actifs */
INVARIANT
  CapOwner : Caps --> Threads & ...
EVENTS
  modify_infos_and_communications =
  ANY compids, memphys,...
  WHERE ...
  THEN
    CompID      := compids ||
    MemPhys     := memphys ||...
  END
```

Autres étapes: Résumé



Serveur: evènements séquencés
Exemple: creation de thread (extrait)

- vérifier le quota de l'application
- puis appel à ROOT requiérant de la mémoire

Appel à ROOT raffiné vers L4

Problèmes d'interleaving:

- Hypotèses sur les serveurs (traitemement des requêtes les unes après les autres)
- Interactions entre serveurs: variables propres à chacun.
  Seule concurrence: ROOT, qui appele L4.
- ROOT: concurrence potentiellement dangereuse.
  $\Rightarrow$ vérifier que non pour pouvoir utiliser B Logiciel.
  $\Rightarrow$ Sinon, savoir faire technique: copie locale de variables à synchroniser et découpage subtile des évènements. Complexe.

Conclusion

- Etude de faisabilité intéressante et concluante:
  - Modélisation B et usage divers:
    "manpower" raisonnable pour un apport intéressant
    (debuggage, précision, tests, documentation,...)
  - Une méthodologie de preuve complête.
    - Orientée par la propriéte $\Rightarrow$ une abstraction abordable.
    - exercée de bout en bout (B) pour la création de thread
  - Une perspective pour la confrontation modèle/Code
  - Queques difficultés:
    - confrontation code/modèle (test, preuve de code): différence de niveau d'abstraction.
      $\Rightarrow$ 1 étape de raffinement (dur).
      Problèmes d'actualité ailleurs (représentation de la mémoire).
    - Atomicité des évènements et interleaving. Faisable si l'interleaving n'a pas d'impact (B Logiciel), complexe sinon (limites des outils B évènementiel actuels)
  - Premier système de ce type traité en B.