CAR'08

Vers la vérification formelle du Logiciel de Vol d'un Satellite Autonome

Projet AGATA (ONERA/CNES)

présentation : Michel Lemaître
Office National d'Études et de Recherches Aérospatiales

30 Mai 2008

Le projet AGATA

"Autonomy Generic Architecture : Tests and Applications"

Programme commun ONERA/CNES

Accroître l'autonomie des engins spatiaux pour

- diminuer la dépendance vis-à-vis du sol
- augmenter la réactivité globale du système

Objectif

Objectif

Définir et implémenter l'architecture de contrôle
(= Logiciel de Vol)
d'un satellite autonome de surveillance et d'observation de la Terre

Objectif

Définir et implémenter l'architecture de contrôle
(= Logiciel de Vol)
d'un satellite autonome de surveillance et d'observation de la Terre

Disposer d'un démonstrateur complet, comprenant

- le Logiciel de Vol (LV)
- la simulation des équipements
- la simulation de l'environnement

Objectif

Définir et implémenter l'architecture de contrôle
(= Logiciel de Vol)
d'un satellite autonome de surveillance et d'observation de la Terre

Disposer d'un démonstrateur complet, comprenant

- le Logiciel de Vol (LV)
- la simulation des équipements
- la simulation de l'environnement

Objectif intermédiaire : établir une spécification exécutable du LV pour le démonstrateur en construction au CNES

Objectif

Définir et implémenter l'architecture de contrôle
(= Logiciel de Vol)
d'un satellite autonome de surveillance et d'observation de la Terre

Disposer d'un démonstrateur complet, comprenant

- le Logiciel de Vol (LV)
- la simulation des équipements
- la simulation de l'environnement

Objectif intermédiaire : établir une spécification exécutable du LV pour le démonstrateur en construction au CNES

Servir de point de départ pour des études latérales
(validation/vérification, FDIR, opérateurs sol, ...)

Les choix effectués pour la spécification du LV

Les choix effectués pour la spécification du LV

- formalisme/langage synchrone (Esterel)
  pour le contrôle des parties réactives cycliques

Les choix effectués pour la spécification du LV

- formalisme/langage synchrone (Esterel)
  pour le contrôle des parties réactives cycliques
- compilation Esterel vers Java
  (produit le corps de la boucle de réaction cyclique)

Les choix effectués pour la spécification du LV

- formalisme/langage synchrone (Esterel)
  pour le contrôle des parties réactives cycliques
- compilation Esterel vers Java
  (produit le corps de la boucle de réaction cyclique)
- langage procédural (Java)
  - pour les supports du code réactif
    (types, fonctions et procédures)

Les choix effectués pour la spécification du LV

- formalisme/langage synchrone (Esterel)
  pour le contrôle des parties réactives cycliques
- compilation Esterel vers Java
  (produit le corps de la boucle de réaction cyclique)
- langage procédural (Java)
  - pour les supports du code réactif
    (types, fonctions et procédures)
  - pour les parties asynchrones délibératives

Intérêts d'un langage synchrone

Intérêts d'un langage synchrone

- expressivité et concision

Intérêts d'un langage synchrone

- expressivité et concision
- sémantique parfaitement définie

Intérêts d'un langage synchrone

- expressivité et concision
- sémantique parfaitement définie
- déterminisme

Intérêts d'un langage synchrone

- expressivité et concision
- sémantique parfaitement définie
- déterminisme
- spécification exécutable

Intérêts d'un langage synchrone

- expressivité et concision
- sémantique parfaitement définie
- déterminisme
- spécification exécutable
- possibilités de vérification formelles

La démarche de développement

La démarche de développement

- développement incrémental : “nanoLV”, “**microLV**”, “LV1” ...

La démarche de développement

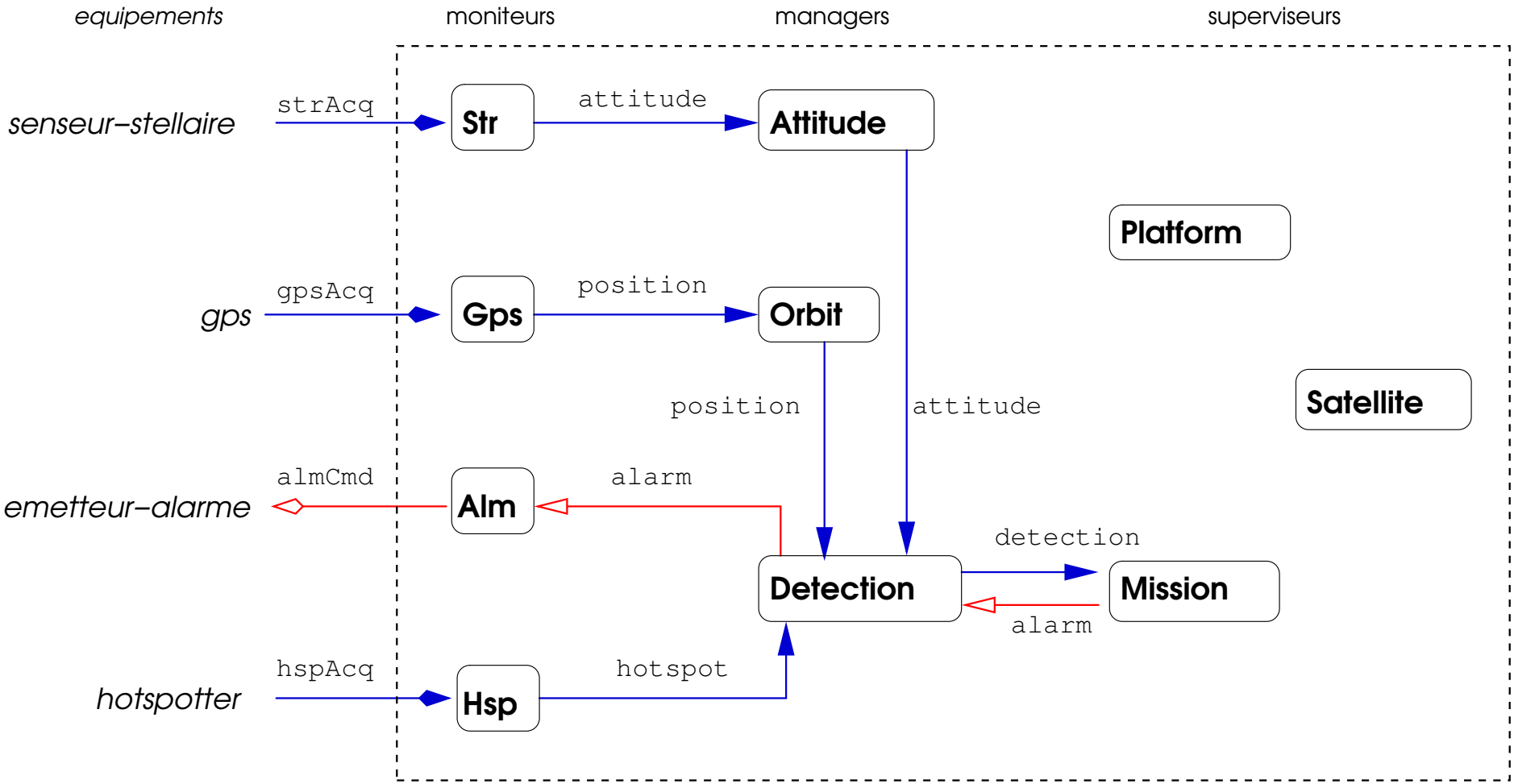
- développement incrémental : “nanoLV”, “**microLV**”, “LV1” ...
- utilisation du simulateur d’équipements “BASILES” développé au CNES

La démarche de développement

- développement incrémental : “nanoLV”, “**microLV**”, “LV1” ...
- utilisation du simulateur d’équipements “BASILES” développé au CNES
- développement conjoint avec le démonstrateur (CNES) du support procédural (types, traitements, bibiothèques ...) (en Java)

Architecture (simplifiée) du “microLV” (état fin 2007)

Mission “HOTSPOT”



Validation/Vérification

Validation/Vérification

Deux grandes voies :

Validation/Vérification

Deux grandes voies :

par simulation :
Très utile (déverminage), mais non complet.

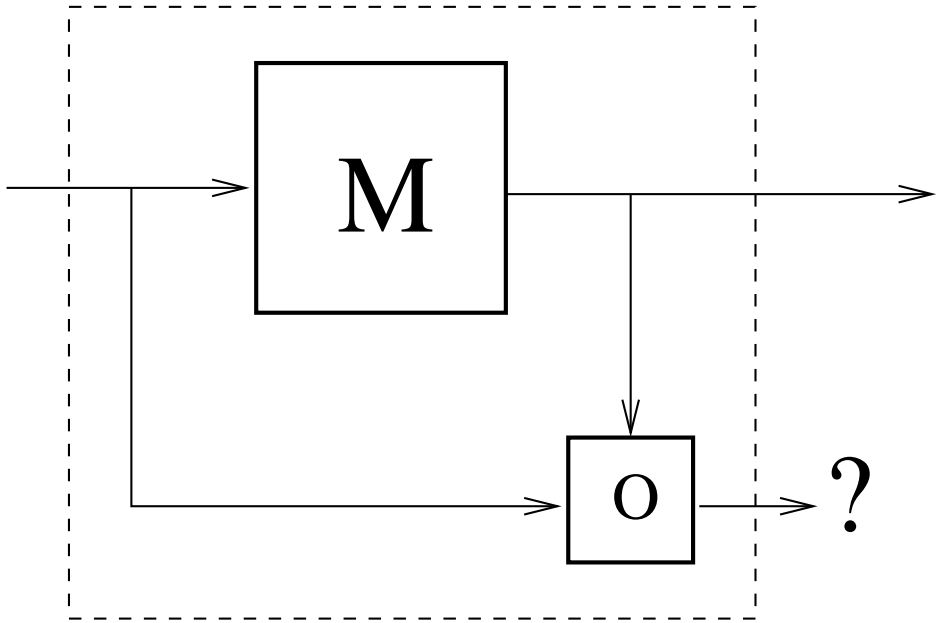
Validation/Vérification

Deux grandes voies :

par simulation :
Très utile (déverminage), mais non complet.

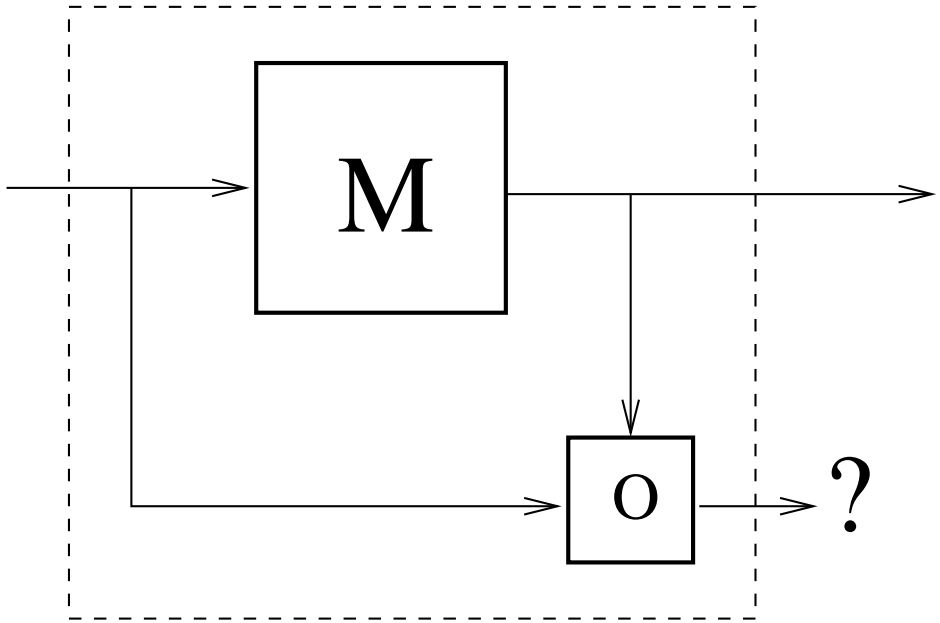
par des peuves formelles :
On peut prouver mathématiquement sur le code même
que certaines propriétés sont ou non satisfaites
sans avoir besoin d'exécuter le code

Validation/Vérification sur langages synchrones : principe des observateurs



Les propriétés à vérifier s'écrivent dans le même langage (ici Esterel)

Validation/Vérification sur langages synchrones : principe des observateurs



Les propriétés à vérifier s'écrivent dans le même langage (ici Esterel)

Des outils sont disponibles
(nous utilisons Xeve : technique de *model-checking*)

Validation/Vérification : exemple 1

Propriété d'atteignabilité :
il est possible qu'une alarme soit émise

Validation/Vérification : exemple 1

Propriété d'atteignabilité :
il est possible qu'une alarme soit émise

L'outil de vérification vérifie formellement qu'une alarme peut effectivement être émise : il exhibe une succession d'*inputs* conduisant à l'émission de l'alarme

Validation/Vérification : exemple 2

Propriété de sûreté :

si une alarme est émise, c'est qu'un point chaud vient d'être détecté : `almCmd => hspAcq`

Validation/Vérification : exemple 2

Propriété de sûreté :

si une alarme est émise, c'est qu'un point chaud vient d'être détecté : `almCmd => hspAcq`

on vérifie la négation de la propriété :

```
loop
    present [ almCmd and not hspAcq ] then
        emit VIOLATION_P1
    end ;
    pause
end loop
```

Validation/Vérification : exemple 2

Propriété de sûreté :
si une alarme est émise, c'est qu'un point chaud vient d'être détecté : `almCmd => hspAcq`

on vérifie la négation de la propriété :

```
loop
    present [ almCmd and not hspAcq ] then
        emit VIOLATION_P1
    end ;
    pause
end loop
```

L'outil vérifie formellement que `VIOLATION_P1` ne peut jamais être émis, pour toute exécution possible du code,
ce qui prouve la propriété

Validation/Vérification : exemple 3

Validation/Vérification : exemple 3

Propriété de vivacité bornée :

tout équipement ayant un comportement "anormal"
reçoit un signal `reset` *dans la seconde qui suit*

Validation/Vérification : exemple 3

Propriété de vivacité bornée :

tout équipement ayant un comportement "anormal"
reçoit un signal `reset` *dans la seconde qui suit*

```
loop
    await [ not powered or not working ] ;
    abort
        await second ;
        emit VIOLATION_P2
    when immediate reset
end loop
```

Validation/Vérification : exemple 3

Propriété de vivacité bornée :

tout équipement ayant un comportement "anormal"
reçoit un signal `reset` *dans la seconde qui suit*

```
loop
    await [ not powered or not working ] ;
    abort
        await second ;
        emit VIOLATION_P2
    when immediate reset
end loop
```

On demande à l'outil de vérifier formellement que `VIOLATION_P2` ne peut jamais être émis, pour toute exécution possible du code.

pause

La démonstration échoue : l'outil exhibe une exécution pour laquelle `VIOLATION_P2` est émis : la propriété n'est pas satisfaite !

Validation/Vérification : exemple 3 (suite)

Problèmes :

- l'équipement doit avoir été mis sous tension pour fonctionner (signal `powerOn`)

Validation/Vérification : exemple 3 (suite)

Problèmes :

- l'équipement doit avoir été mis sous tension pour fonctionner (signal `powerOn`)
- avant de re-tester, il faut attendre un nouveau délai, et attendre que l'équipement fonctionne à nouveau.

Validation/Vérification : exemple 3 (fin)

Validation/Vérification : exemple 3 (fin)

La bonne propriété est plus complexe, mais s'exprime bien :

```
await powerOn;
abort
   await 3 second ;
when  [ powered and working ] ;

loop
   await [ not powered or not working ] ;
   abort
      await second ;
      emit VIOLATION_P3
   when immediate reset ;
   abort
      await 3 second ;
   when [ powered and working ] ;
end loop
```

Résumé

Résumé

Version préliminaire d'un Logiciel de Vol de satellite autonome conçu en langage synchrone (Esterel)
$\longrightarrow$ spécification exécutable

Résumé

Version préliminaire d'un Logiciel de Vol de satellite autonome conçu en langage synchrone (Esterel)
$\longrightarrow$ spécification exécutable

Usage effectif d'une méthode et d'un outil de preuve formelle de propriétés d'atteignabilité, de sûreté, de vivacité bornée.

Résumé

Version préliminaire d'un Logiciel de Vol de satellite autonome conçu en langage synchrone (Esterel)
$\longrightarrow$ spécification exécutable

Usage effectif d'une méthode et d'un outil de preuve formelle de propriétés d'atteignabilité, de sûreté, de vivacité bornée.

L'expression des propriétés n'est pas toujours facile, mais faisable grâce à l'expressivité du langage.

Résumé

Version préliminaire d'un Logiciel de Vol de satellite autonome conçu en langage synchrone (Esterel)
$\longrightarrow$ spécification exécutable

Usage effectif d'une méthode et d'un outil de preuve formelle de propriétés d'atteignabilité, de sûreté, de vivacité bornée.

L'expression des propriétés n'est pas toujours facile, mais faisable grâce à l'expressivité du langage.

Les propriétés sont limitées à des formules "logiques" (non numériques).

Résumé

Version préliminaire d'un Logiciel de Vol de satellite autonome conçu en langage synchrone (Esterel)
$\longrightarrow$ spécification exécutable

Usage effectif d'une méthode et d'un outil de preuve formelle de propriétés d'atteignabilité, de sûreté, de vivacité bornée.

L'expression des propriétés n'est pas toujours facile, mais faisable grâce à l'expressivité du langage.

Les propriétés sont limitées à des formules "logiques" (non numériques).

Les preuves sont automatiques.

Résumé

Version préliminaire d'un Logiciel de Vol de satellite autonome conçu en langage synchrone (Esterel)
$\longrightarrow$ spécification exécutable

Usage effectif d'une méthode et d'un outil de preuve formelle de propriétés d'atteignabilité, de sûreté, de vivacité bornée.

L'expression des propriétés n'est pas toujours facile, mais faisable grâce à l'expressivité du langage.

Les propriétés sont limitées à des formules "logiques" (non numériques).

Les preuves sont automatiques.

Ce processus de preuve renforce la confiance dans la spécification.

MERCI DE VOTRE ATTENTION

Démonstrations possibles :

- de l'outil de preuve
- du Logiciel de Vol (version plus avancée mars 2008).

Esterel en une planche

Esterel en une planche

Hypothèse synchrone : les événements sont ordonnés totalement sur des instants discrets ; plusieurs sont possibles au même instant ; les réactions sont “instantanées”

Esterel en une planche

Hypothèse synchrone : les événements sont ordonnés totalement sur des instants discrets ; plusieurs sont possibles au même instant ; les réactions sont “instantanées”

Structures de contrôle :

- signaux émis et reçus en *broadcasting* instantané
- séquentialité
- parallélisme
- interruptions (préemption, suspension)
- synchronisation possible avec des tâches externes asynchrones

Esterel en une planche

Hypothèse synchrone : les événements sont ordonnés totalement sur des instants discrets ; plusieurs sont possibles au même instant ; les réactions sont "instantanées"

Structures de contrôle :

- signaux émis et reçus en *broadcasting* instantané
- séquentialité
- parallélisme
- interruptions (préemption, suspension)
- synchronisation possible avec des tâches externes asynchrones

Compilation :

- détecte et interdit tous les cycles de causalité instantanée
- vers C, Ada, Java, Bliff, ...
- très efficace : threads logiques applatis en un seul thread physique

Esterel en une planche

Hypothèse synchrone : les événements sont ordonnés totalement sur des instants discrets ; plusieurs sont possibles au même instant ; les réactions sont "instantanées"

Structures de contrôle :

- signaux émis et reçus en *broadcasting* instantané
- séquentialité
- parallélisme
- interruptions (préemption, suspension)
- synchronisation possible avec des tâches externes asynchrones

Compilation :

- détecte et interdit tous les cycles de causalité instantanée
- vers C, Ada, Java, Bliff, ...
- très efficace : threads logiques applatis en un seul thread physique

En pratique l'implantation doit valider l'hypothèse synchrone : toute réaction doit tenir en temps réel dans l'intervalle temporel qui lui est dévolu (plusieurs implantations possibles)

quelques publis AGATA

G. Beaumet, G. Verfaillie.
Planification continue pour la conduite d'un satellite agile autonome.
JFPDA 2006

G. Verfaillie, M.-C. Charmeau.
A generic modular architecture for the control of an autonomous spacecraft.
IWPSS 2006

M. Lemaître, G. Verfaillie.
Interaction between reactive and deliberative tasks for on-line decision-making.
ICAPS 2007 Workshop on "Planning and Plan Execution for Real-world Systems"

En savoir plus sur le projet AGATA

Marie-Claire.Charmeau@cnes.fr
Eric.Bensana@onera.fr
Michel.Lemaitre@onera.fr
Gerard.Verfaillie@onera.fr