Vérifier le comportement du code d'un système embarqué à partir de son modèle

Anthony Fernandes Pires, Thomas Polacsek, Virginie Wiels et Stephane Duprat (ATOS)

Mercredi 13 novembre 2013

Contexte

Vérification logiciel

- 1969 : 50% du temps de développement : tests des programmes et correction des bugs [Hoare][a]
- 2010 : 40% à 50% des coûts totaux de développement des activités de vérification et validation [Ministère de l'industrie][b]
- jusqu'à 60% : notre retour d'expérience (ATOS)

[a] *An axiomatic basis for computer programming*
[b] *D. Potier. Briques génériques du logiciel embarqué.*

Logiciel embarqué aéronautique

- Nombreuses contraintes de développement en raison du processus de certification (DO-178C)
- Un espoir : l'utilisation de méthodes formelles (DO-333)

Contexte

Un langage d'assertions

- Utilisation du langage de spécification ANSI/ISO C Specification Langage (ACSL) et du logiciel Frama-C.
- Triplet de Hoare:

$$\{P\}prog\{Q\}$$

- Quand P est vrai, Q est vrai après l'exécution de *prog*;
- P est appelée la précondition et Q la postcondition.

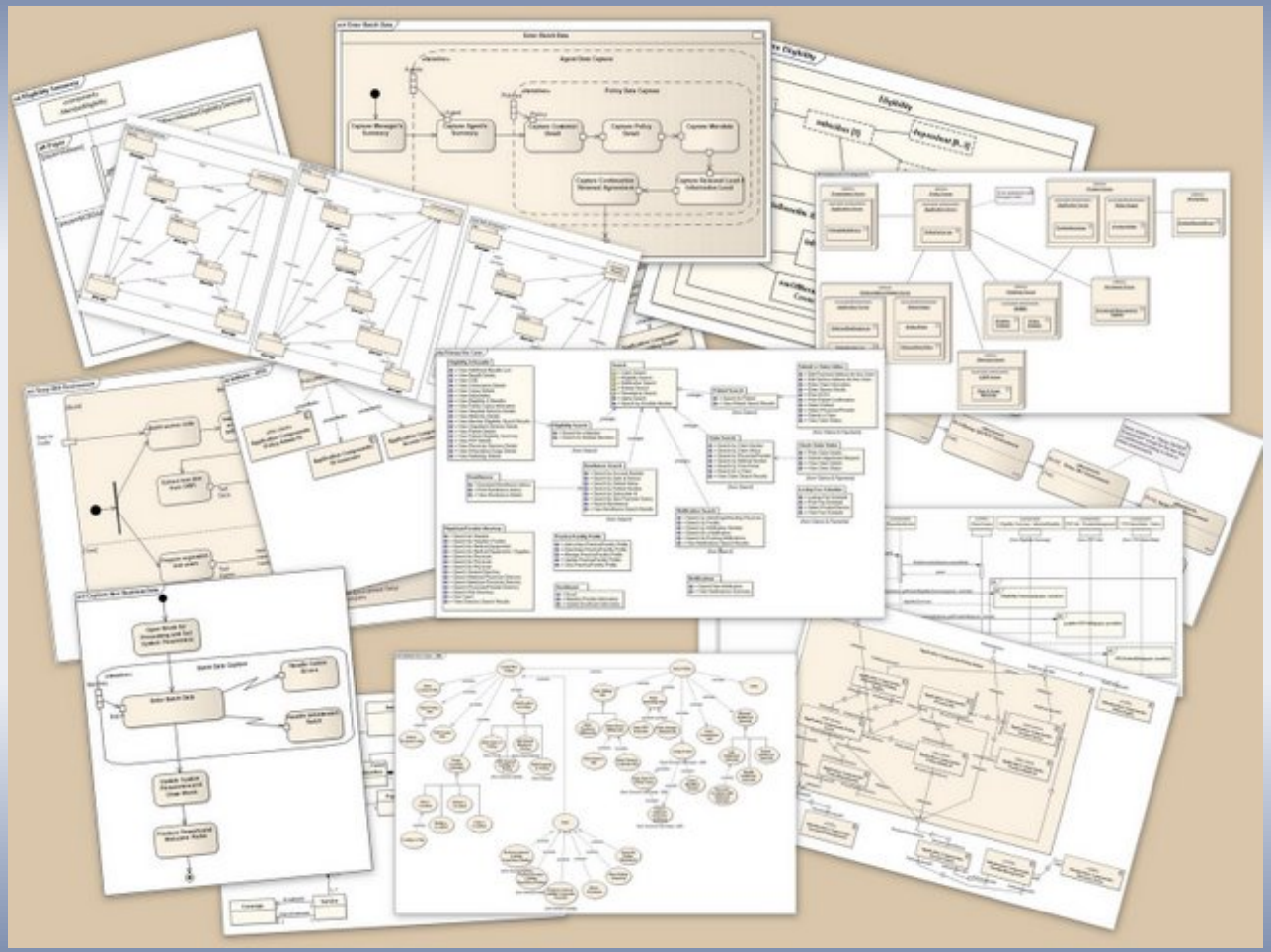
Contexte

ACSL : un exemple

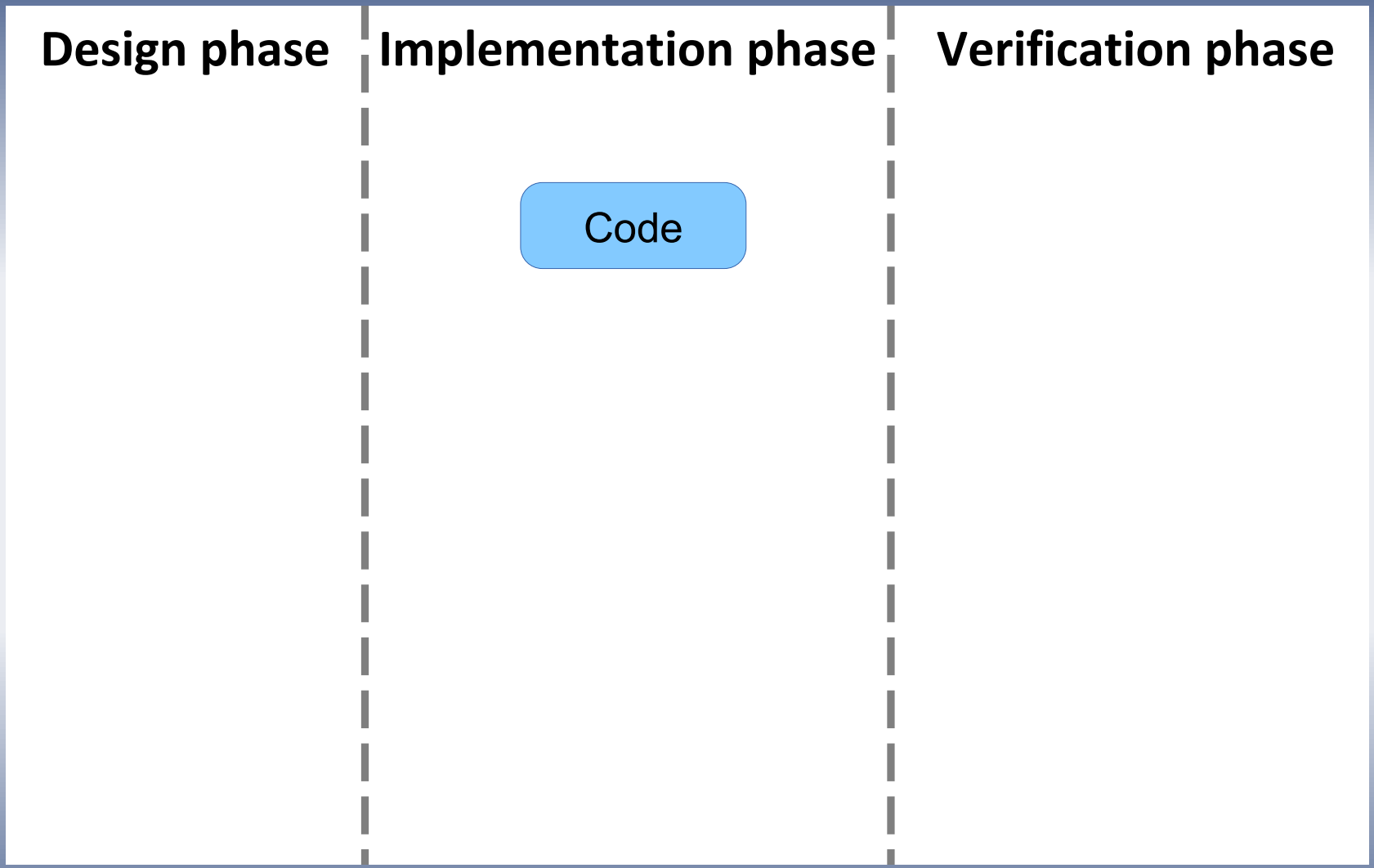
```
/*@ behavior i_greaterThan_j :
          assumes i>j;
          ensures  \result==i;
        behavior j_greaterThan_i :
          assumes j>=i;
          ensures  \result==j;
*/

int max(int i, int j){
        if (i>j)
        return i;
        else
        return j;
}
```

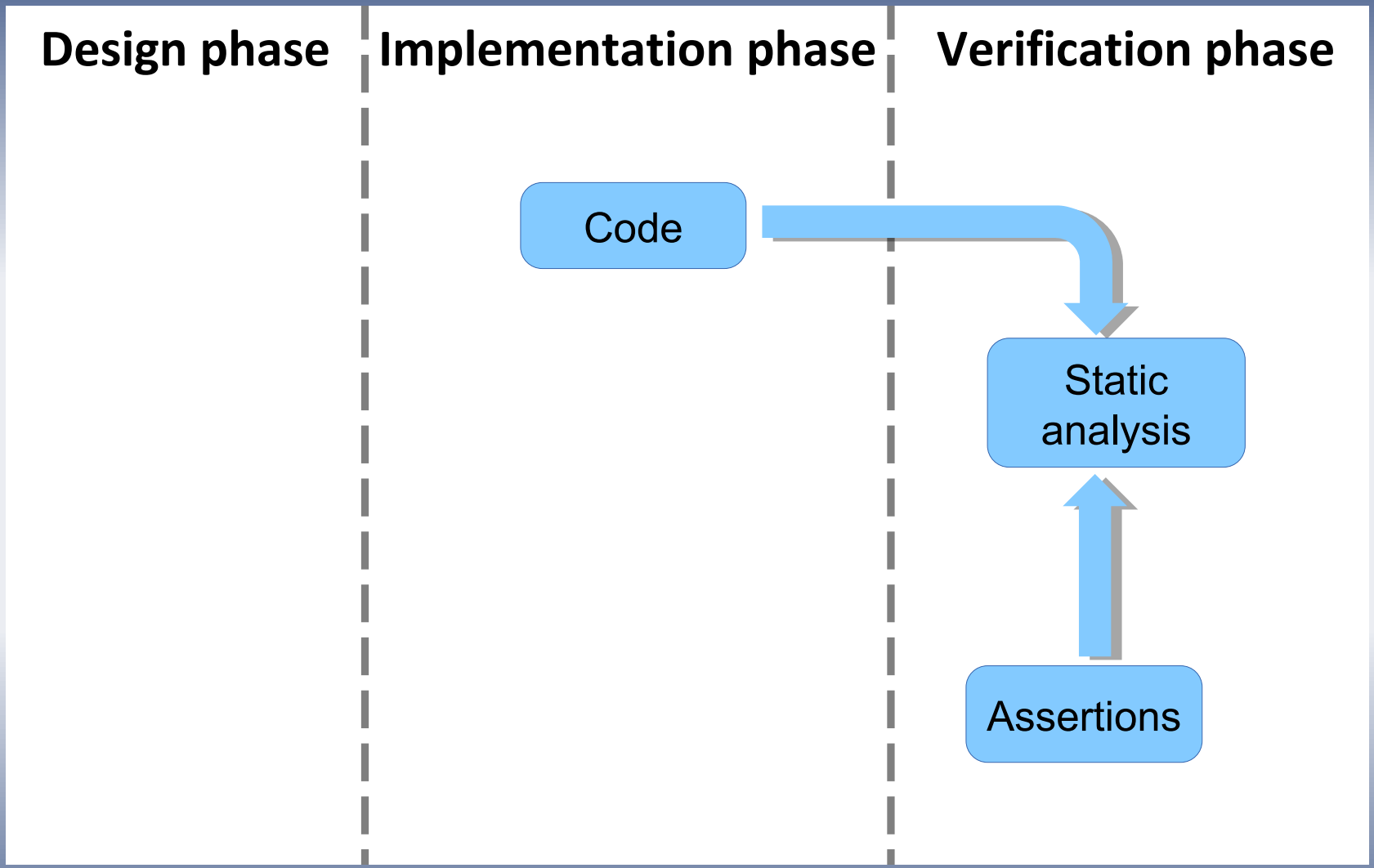
Contexte : UML/SysML



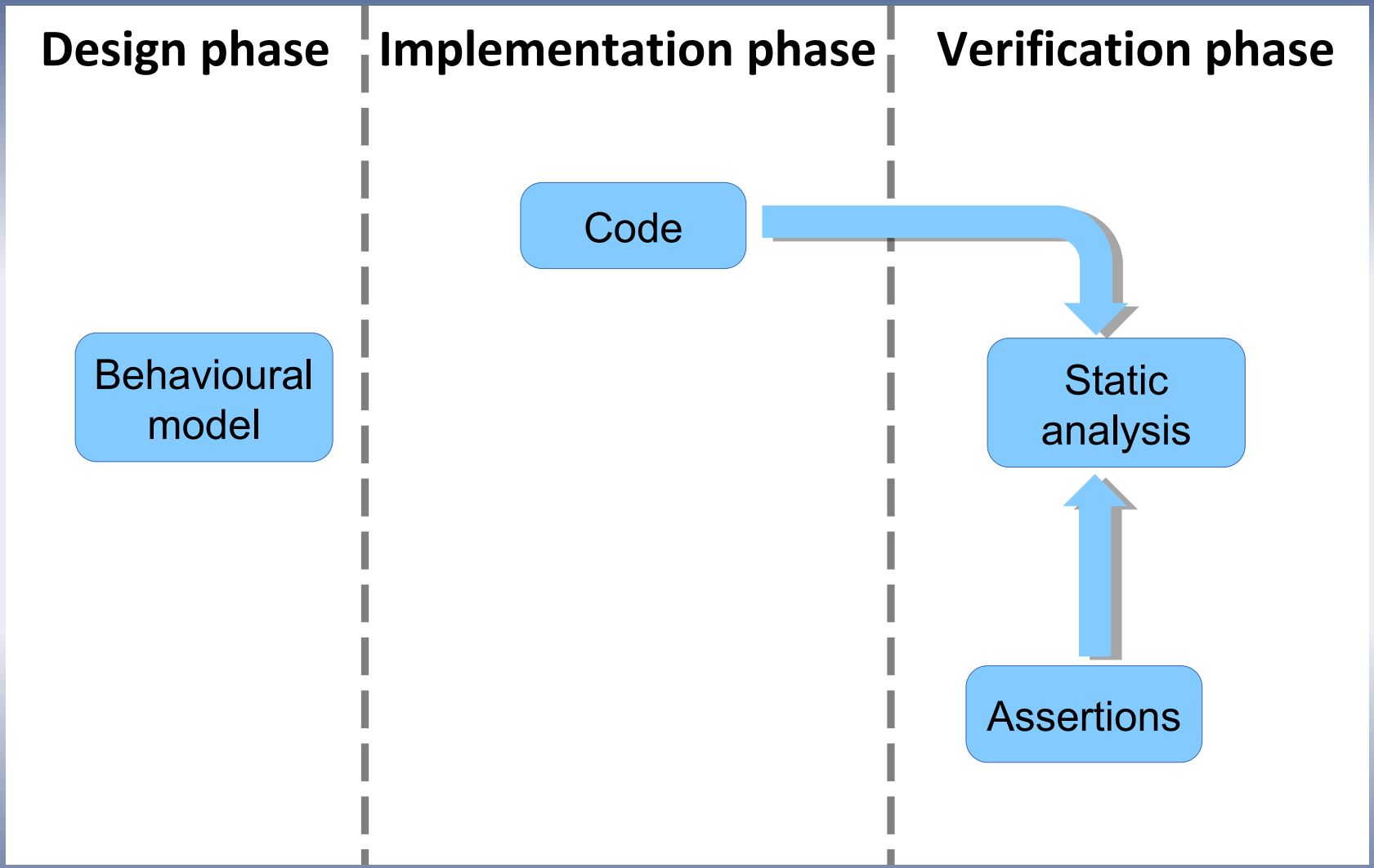
Objectifs


Design phase
Implementation phase
Verification phase
Code

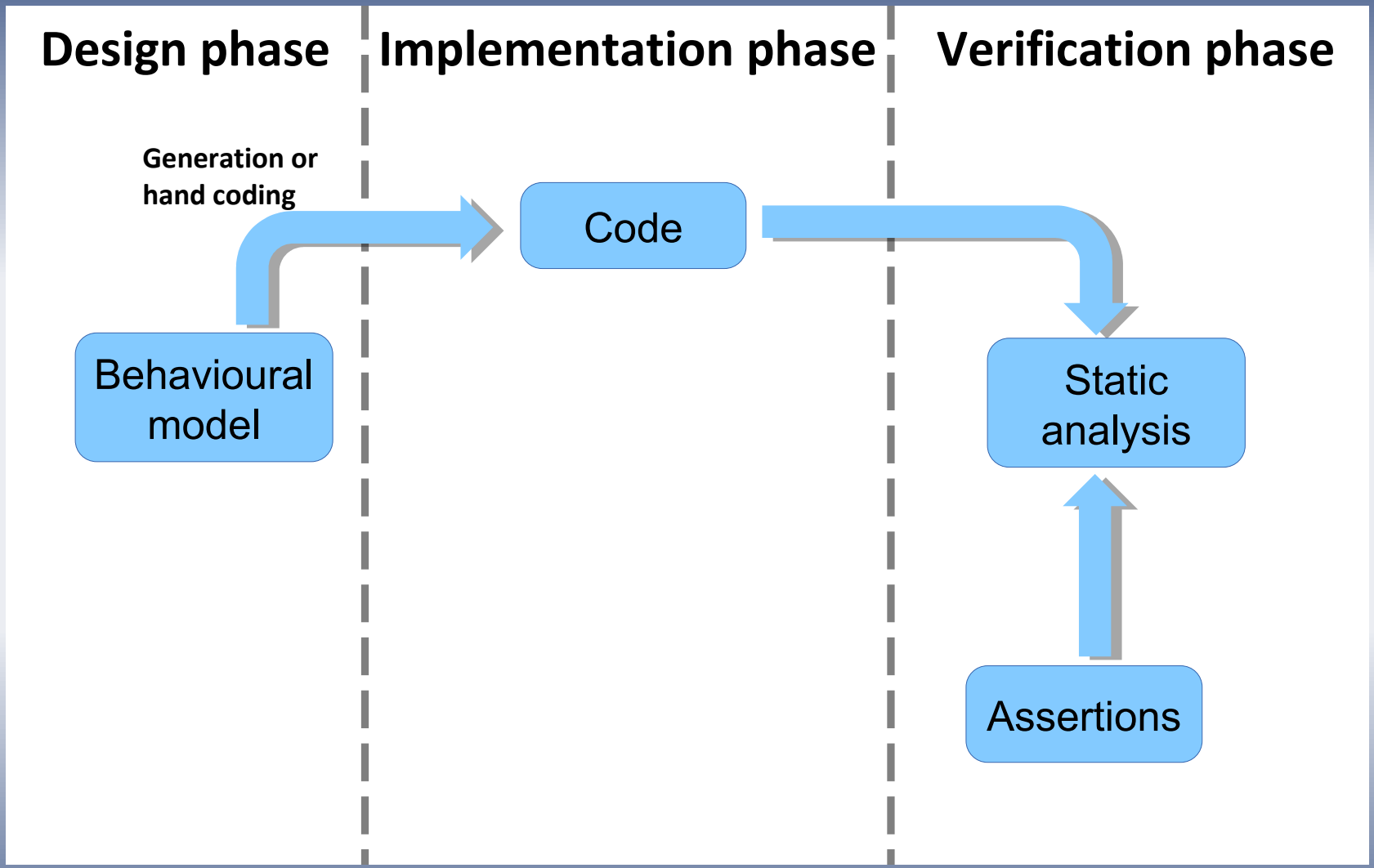
Objectifs


Design phase
Implementation phase
Verification phase
Code
Static analysis
Assertions

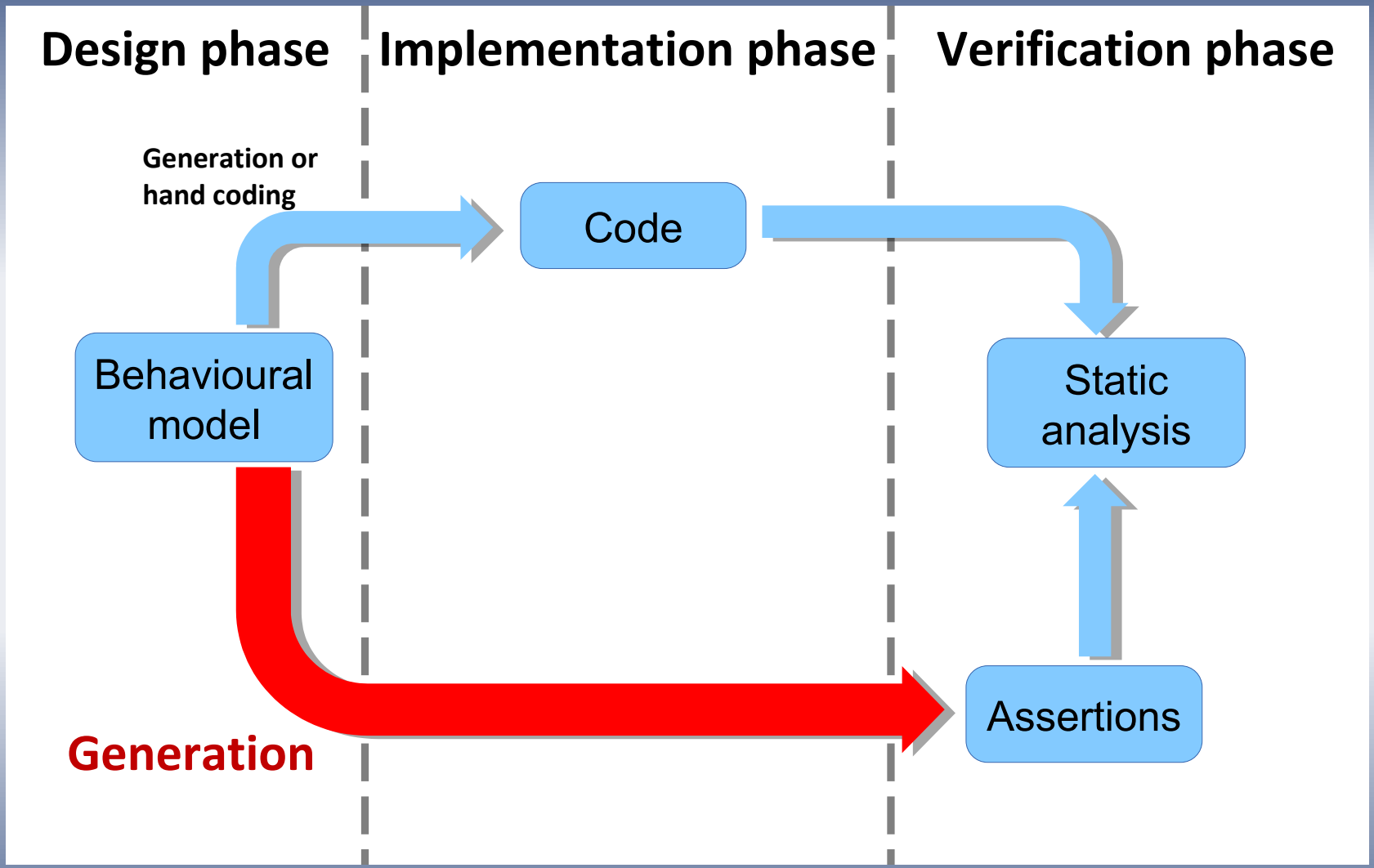
Objectifs


Design phase
Implementation phase
Verification phase
Code
Behavioural model
Static analysis
Assertions

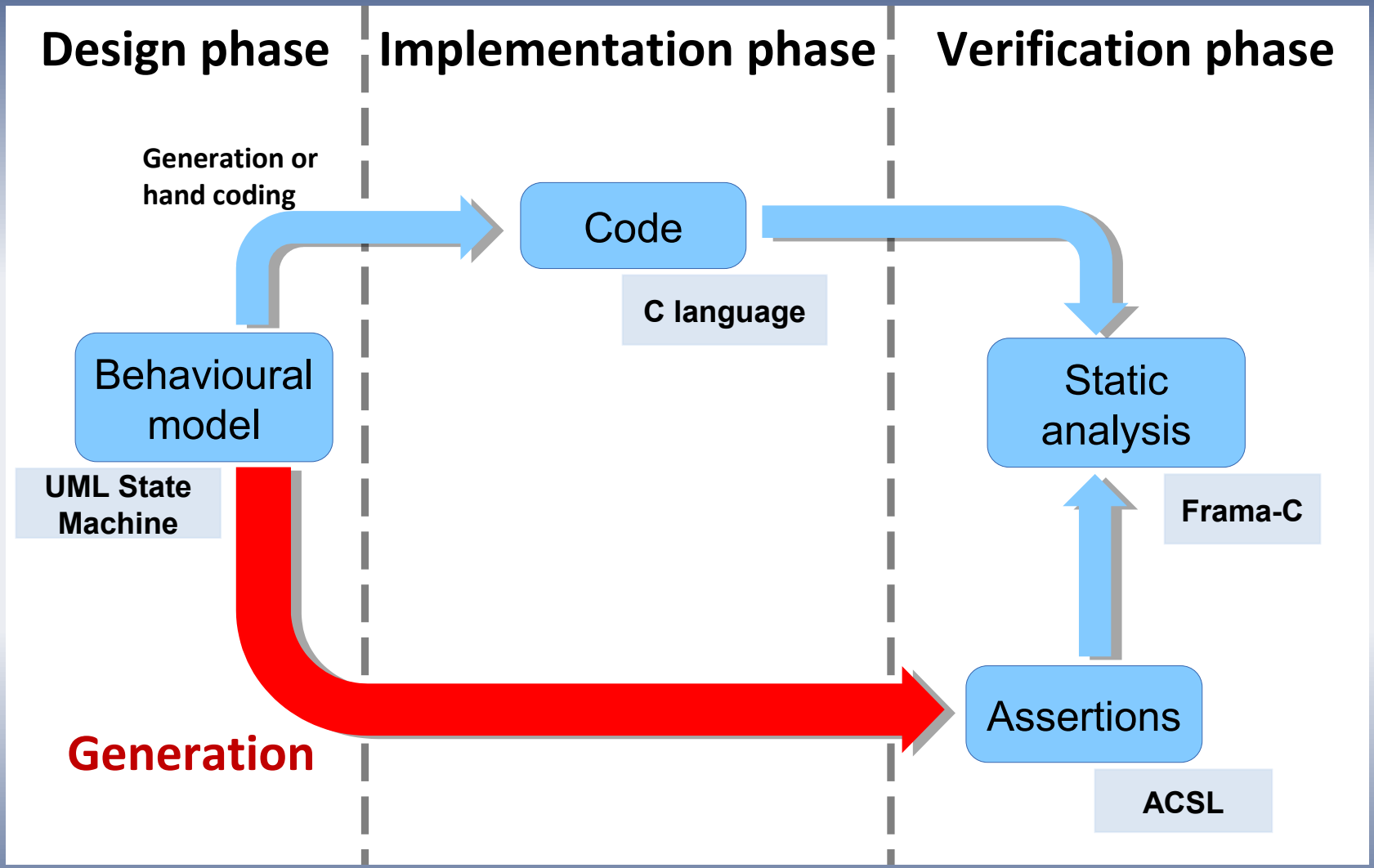
Objectifs


Design phase
Implementation phase
Verification phase
Generation or hand coding
Code
Behavioural model
Static analysis
Assertions

Objectifs


Design phase
Implementation phase
Verification phase
Generation or hand coding
Code
Behavioural model
Static analysis
Assertions
Generation

Objectifs


Design phase
Implementation phase
Verification phase
Generation or hand coding
Code
C language
Behavioural model
UML State Machine
Static analysis
Frama-C
Assertions
ACSL
Generation

Objectifs

Vérifier la conformité d'un code en fonction de son modèle de conception

- Établir des liens entre les modèles de conception et la vérification du code source
- Automatiser une partie de la preuve unitaire
- Travailler dans le contexte industriel actuel : UML, Frama-C

Vérification de deux types de propriétés

- La complétude : le code implémente totalement la spécification
- L'adéquation : le code implémente seulement la spécification

Premières investigations

Un sous-ensemble des UML state machine

- entraînées par une horloge (tick) ;

Premières investigations

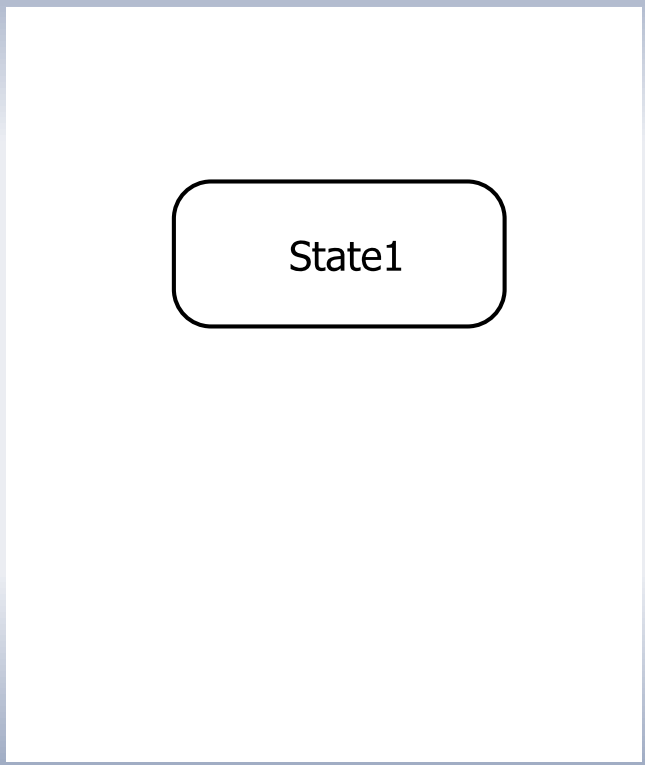
Un sous-ensemble des UML state machine

- entraînées par une horloge (tick) ;
- limitations :

Premières investigations

Un sous-ensemble des UML state machine

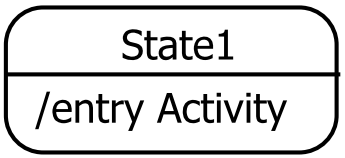
- entraînées par une horloge (tick) ;
- limitations :
  - états atomiques ;



Premières investigations

Un sous-ensemble des UML state machine

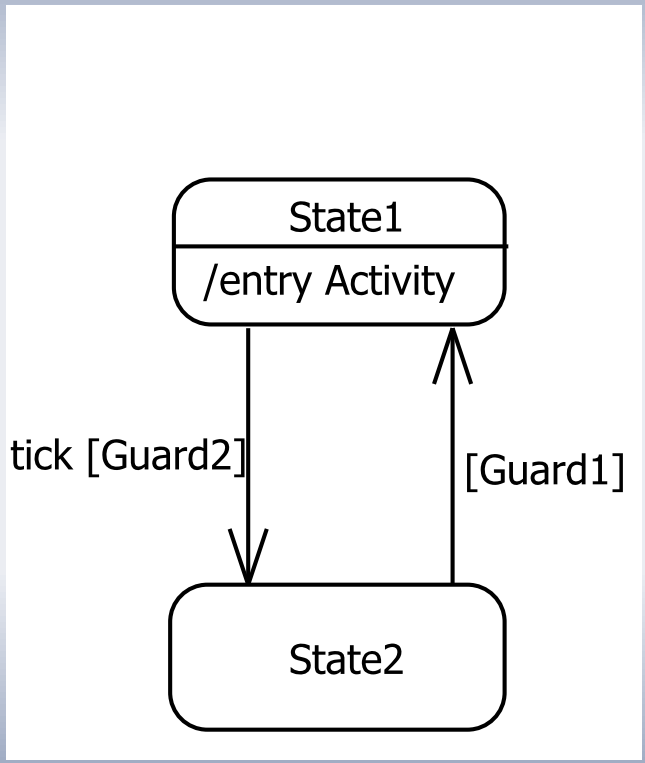
- entraînées par une horloge (tick) ;
- limitations :
  - états atomiques ;
  - actions uniquement dans les états ;



Premières investigations

Un sous-ensemble des UML state machine

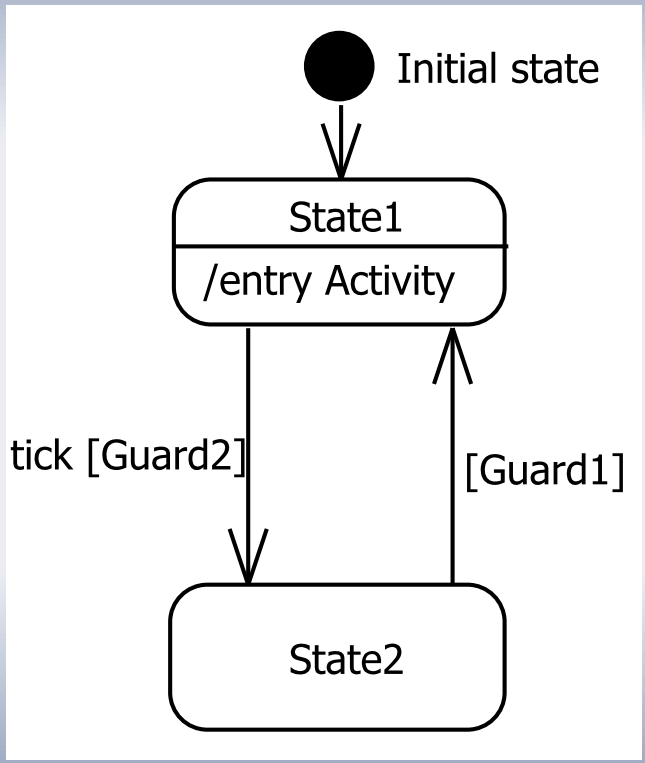
- entraînées par une horloge (tick) ;
- limitations :
  - états atomiques ;
  - actions uniquement dans les états ;
  - transitions composées d'un événement et d'une garde ;
  - seulement deux événements : *completion event* (généré automatiquement à la fin des actions d'un état) et *tick* ;



Premières investigations

Un sous-ensemble des UML state machine

- entraînées par une horloge (tick) ;
- limitations :
  - états atomiques ;
  - actions uniquement dans les états ;
  - transitions composées d'un événement et d'une garde ;
  - seulement deux événements : *completion event* (généré automatiquement à la fin des actions d'un état) et *tick* ;
  - un seul pseudo-état : l'état initial.



Premières investigations

Hypothèses

- Pas de concurrence entre les transitions.
- Chaque séquence d'activités commencée avec un tick d'horloge se termine avant le prochain tick d'horloge.

Premières investigations

Complétude (a)

Pour l'état actuel de la machine d'état (représenté par la variable *current_state*), si la garde de la transition est vraie alors la fonction de transition renvoie l'état ciblé spécifié.

ACSL

```
assumes current_state==<current state>;
ensures <guard of outgoing transition 1>
   .   ==> \result == <target state of outgoing transition 1>;
   .
   .
ensures <guard of outgoing transition N>
      ==> \result == <target state of outgoing transition N>;
```

Premières investigations

Complétude (b)

La fonction de transition est sans effet de bord.

ACSL

```
assigns \nothing;
```

Premières investigations

Adéquation (a)

Pour l'état courant, si une transition a été déclenchée alors sa garde doit être vraie.

ACSL

```
assumes current_state==<current state>;
ensures \result == <target state of outgoing transition 1>
   .          ==> <guard of outgoing transition 1>;
   .
   .
ensures \result == <target state of outgoing transition N>
               ==> <guard of outgoing transition N>;
```

Premières investigations

Adéquation (b)

Pour l'état courant, si aucune garde d'aucune transition sortante n'est vraie, alors aucune transition n'est déclenchée (dans notre patron de code la fonction de transition renvoie Null).

ACSL

```
assumes current_state==<current state>;
ensures (!<guard of outgoing transition 1>
          && ...
          && !<guard of outgoing transition N>)
                    ==> \result == Null;
```

Premières investigations

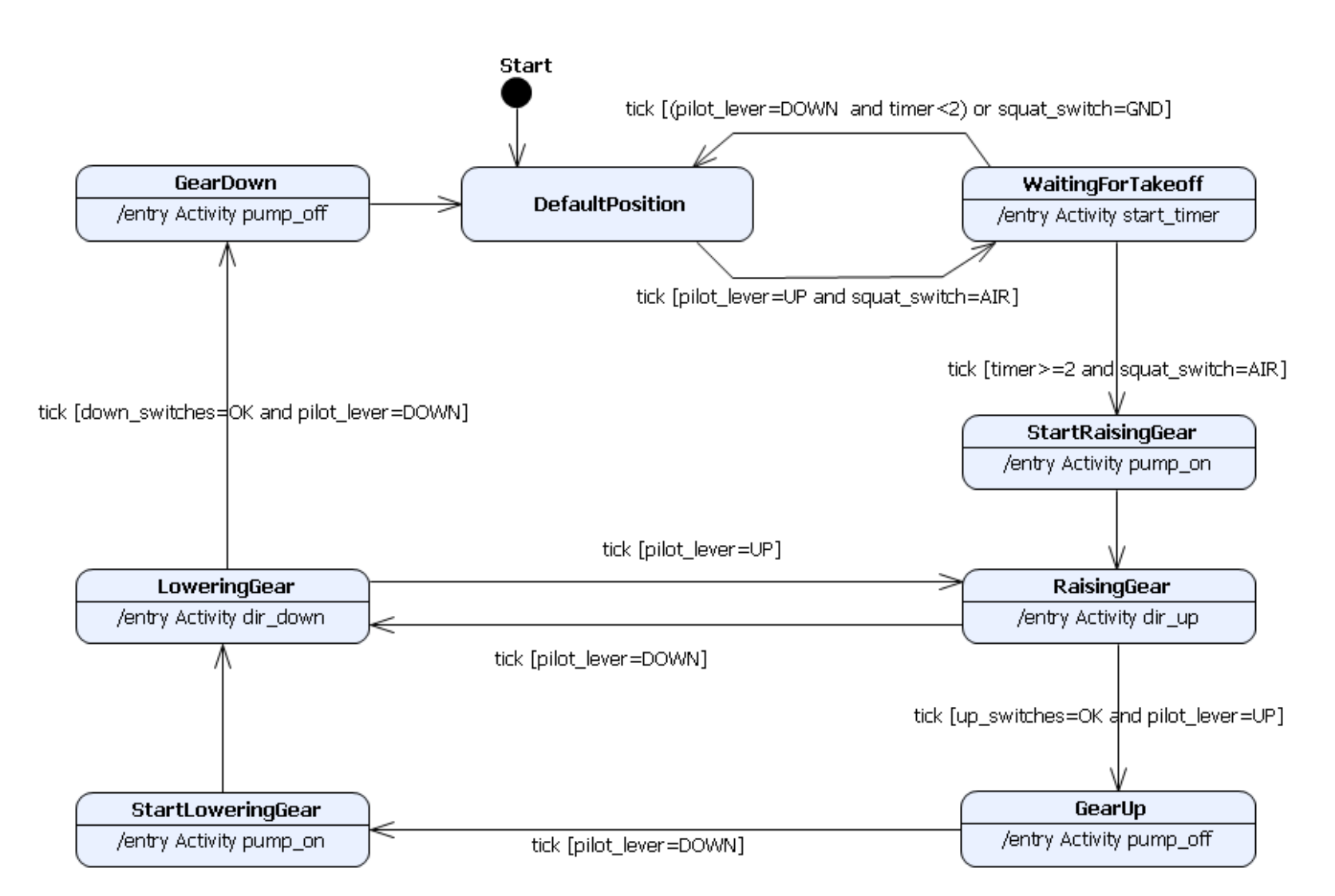
Adéquation (c)

Si un état n'est pas géré par une fonction de transition (aucune transition sortante de cet état n'est déclenchée par l'événement géré par cette fonction), cette fonction de transition ne déclenche aucune transition pour cet état.

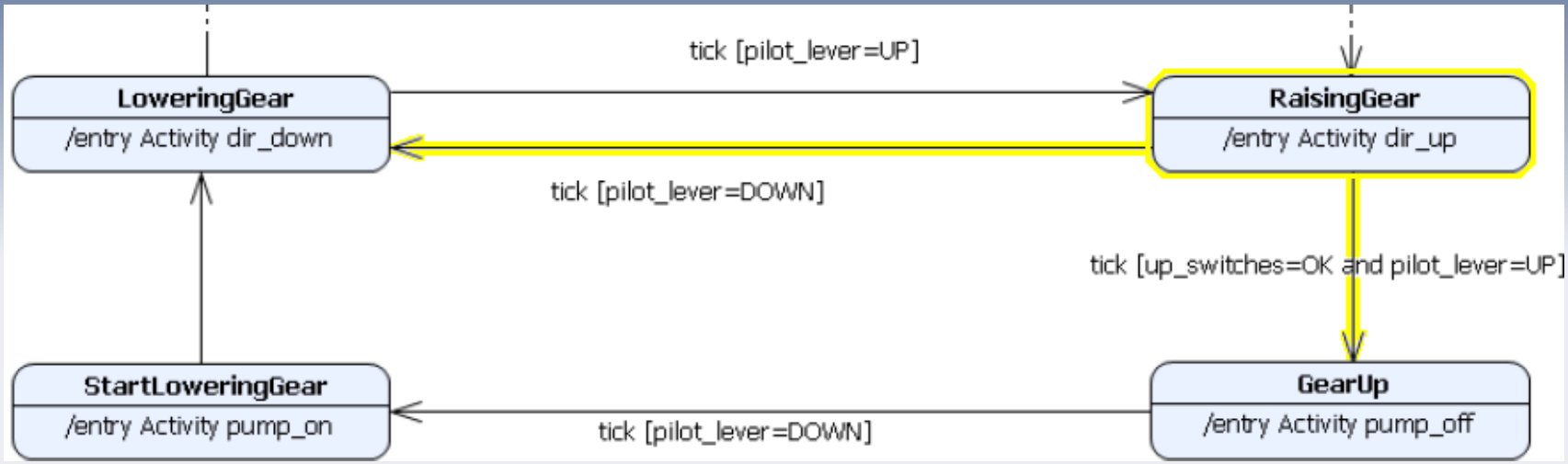
ACSL

```
behavior OtherStates :
        assumes current_state != stateA
                && ...
                && current_state != stateN ;
        assigns \ nothing ;
        ensures \ result == Null ;
```

Exemple


Start
tick [(pilot_lever=DOWN and timer<2) or squat_switch=GND]
GearDown
/entry Activity pump_off
DefaultPosition
WaitingForTakeoff
/entry Activity start_timer
tick [pilot_lever=UP and squat_switch=AIR]
tick [timer>=2 and squat_switch=AIR]
tick [down_switches=OK and pilot_lever=DOWN]
StartRaisingGear
/entry Activity pump_on
tick [pilot_lever=UP]
LoweringGear
/entry Activity dir_down
RaisingGear
/entry Activity dir_up
tick [pilot_lever=DOWN]
tick [up_switches=OK and pilot_lever=UP]
StartLoweringGear
/entry Activity pump_on
tick [pilot_lever=DOWN]
GearUp
/entry Activity pump_off

Exemple



ACSL

```
behavior RaisingGear:
 assumes current_state==RaisingGear;
 assigns \nothing;
 ensures (pilot_lever==DOWN) <==> \result==LoweringGear;
 ensures (pilot_lever==UP && up_switches==OK) <==> \result==GearUp;
 ensures (!(pilot_lever==DOWN)
            && !(pilot_lever==UP && up_switches==OK))
              ==> \result==Null;
```

Exemple

Adéquation (c)

Si un état n'est pas géré par une fonction de transition (aucune transition sortante de cet état n'est déclenchée par l'événement géré par cette fonction), cette fonction de transition ne déclenche aucune transition pour cet état.

ACSL

```
behavior OtherStates :
        assumes current_state != LoweringGear
                && current_state != DefaultPosition
                && current_state != WaitingForTakeoff
                && current_state != RaisingGear
                && current_state != GearUp ;
        assigns \ nothing ;
        ensures \ result == Null ;
```

Exemple

Démonstration

`http://code.google.com/a/eclipselabs.org/p/agrum/`

▸ Link

Conclusion

Conclusion

- Lier la vérification du code formel et Ingénierie dirigée par les modèles.
- Etablir partiellement la preuve de conformité d'un code source en fonction de sa specification à base de modèle.
- Eclipse plugin AGrUM `http://code.google.com/a/eclipselabs.org/p/agrum/`

Perspectives

- Etablir les liens avec les objectifs de certifications définis dans la DO-178C.
- Définir d'autres patrons de code.
- Etendre le sous ensemble UML.