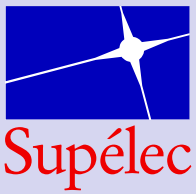
Composant d'intégration contrôle-traitements

Revue finale à T_0+24

Frédéric Boulanger
Christophe Jacquet
Dominique Marcadet

Supélec



Plan de la présentation

Rappels sur notre approche

Déploiement sur Inflexion

Démonstration : première partie

Vérification des propriétés

Démonstration : seconde partie

Conclusions et perspectives

Supélec

Plan de la présentation

Rappels sur notre approche

Déploiement sur Inflexion

Démonstration : première partie

Vérification des propriétés

Démonstration : seconde partie

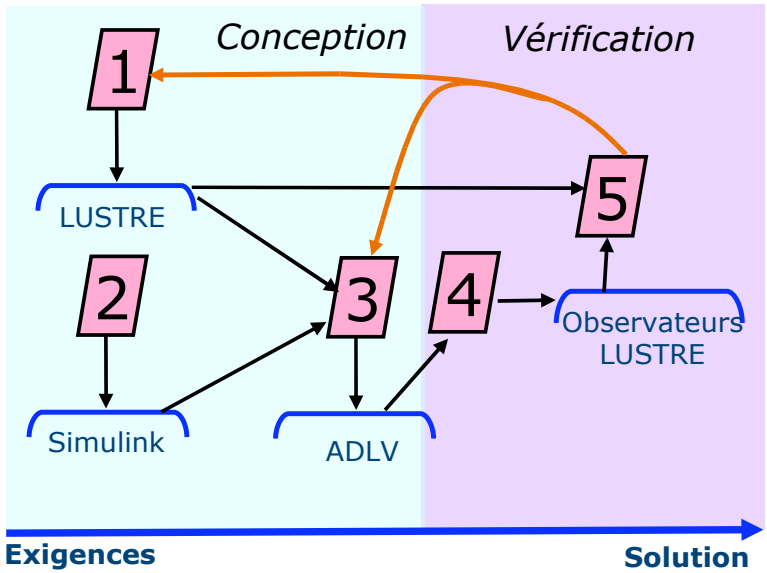
Conclusions et perspectives

Supélec

ADLV : Application Description Language for Verification

- Objectifs
  - Isoler le contrôle des traitements pour faciliter les vérifications sur le contrôle
  - Donner la possibilité au concepteur d'exprimer de manière naturelle les vérifications qu'il souhaite faire sur l'application
  - Automatiser le déploiement sur une infrastructure d'exécution
- Moyens
  - Un langage de description d'application et de propriétés
  - Traduction des propriétés dans le formalisme utilisé pour la conception du contrôle
  - Génération de tous les fichiers nécessaires à un déploiement sur Inflexion

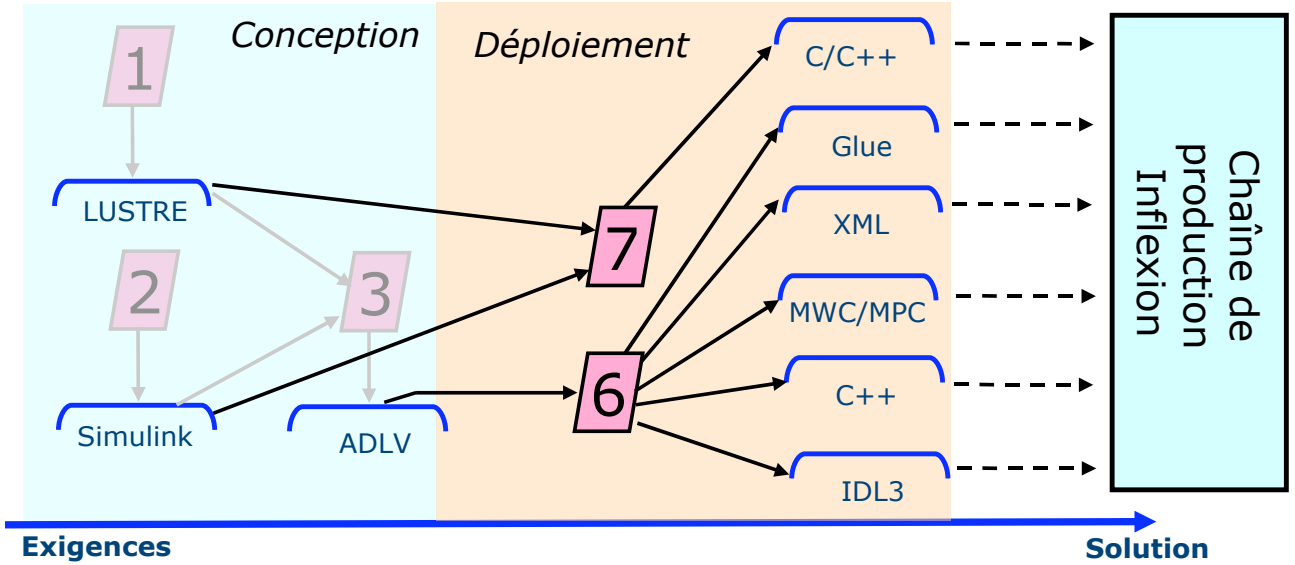
Design Flow ADLV pour la validation (exemple Lustre / Simulink)



1. Conception du composant contrôleur d'une application (par exemple avec SCADE, Esterel Studio ou Rhapsody).
2. Conception des composants Traitement dans des langages quelconques (C++, MATLAB, …)
3. Description de l'application en langage ADLV:
   - Description des interfaces des composants Contrôleur et Traitements,
   - Description des interfaces et des comportements des composants Internes. La partie comportement se limite à des connexions dynamiques entre flots, à des activations ou désactivations de composants, ou des créations d'événements booléens à partir de conditions de seuil sur des valeurs de flots de données,
   - Spécification de la dynamique d'activation des composants de l'application.
   - Spécification des propriétés de l'application (les assertions en logique temporelle).
4. Génération, dans le langage du contrôleur, des observateurs à partir des assertions ADLV et du code ADLV des composants Internes,
5. Vérification des assertions ADLV de l'application, sous forme de propriétés du « Contrôleur » par :
   - Assemblage en un seul programme (dans le langage d'implémentation du composant Contrôleur, ici Lustre) du code du Contrôleur et du code d'un observateur généré par ADLV,
   - Utilisation de l'outil de model-checking approprié sur le programme résultant.
   - Analyse du diagnostic, correction éventuelle du « Contrôleur », de l'assertion ou d'un composant Interne.

Remarque: Lustre et Simulink sont utilisés ici à titre d'exemples, d'autres langages peuvent être utilisés.

Design Flow ADLV pour le déploiement (exemple Lustre / Simulink)

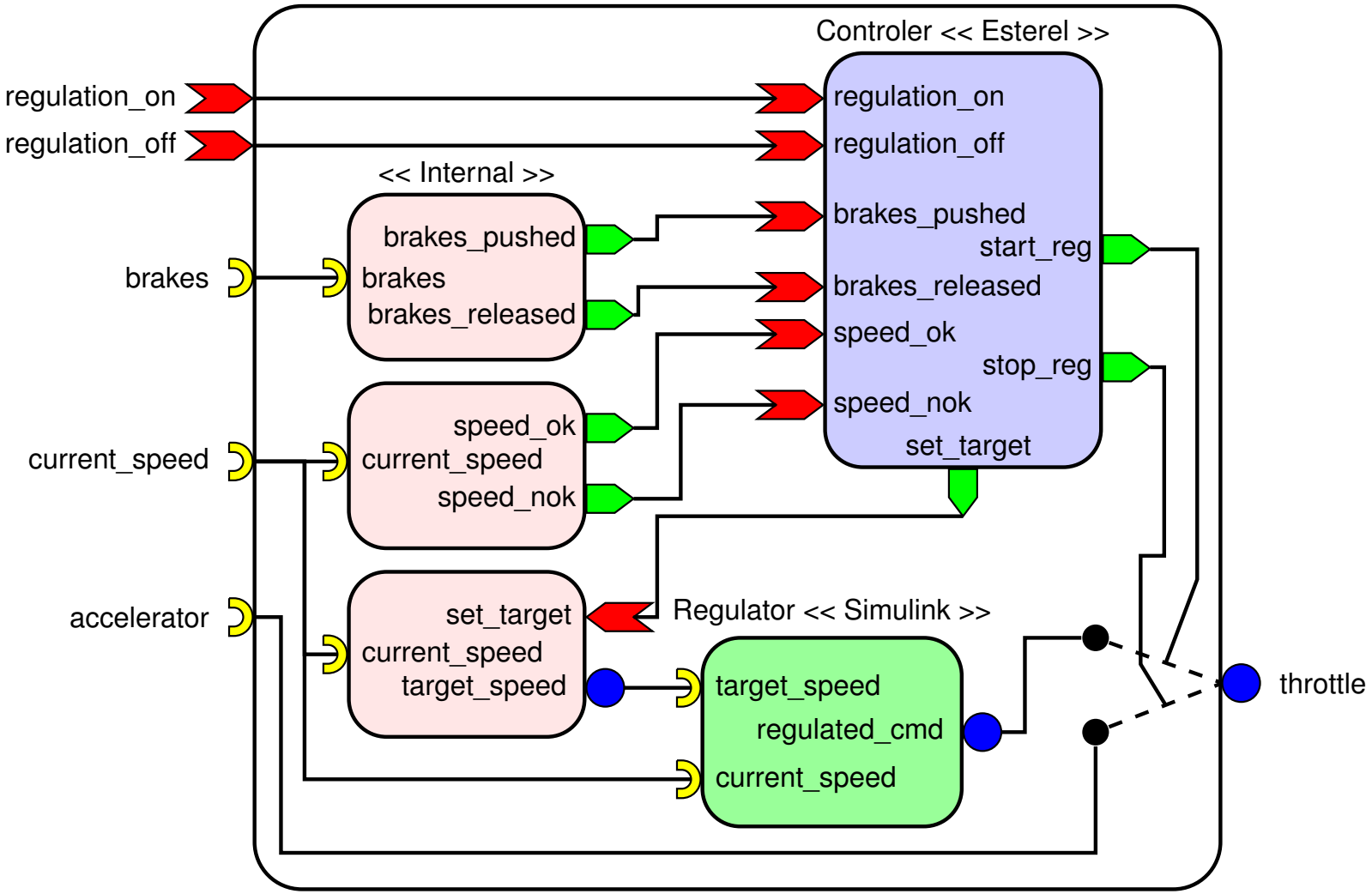


1. Génération:
   - De L'IDL3 des composants **Contrôleur**, **Traitement**, **Internes** et **Dynamique d'activation**
   - Du code C++ correspondant au comportement des composants **Internes** et **Dynamique d'activation**,
   - Des fichiers de construction pour Inflexion,
   - Des fichiers de déploiement pour Inflexion,
   - De la glue C++ d'adaptation des composants **Contrôleur** et **Traitements** aux conteneurs Inflexion.

- Génération du code C/C++ des composants **Contrôleur** et **Traitements**.

Remarque:
Lustre et Simulink sont utilisés ici à titre d'exemples, d'autres langages peuvent être utilisés

Exemple applicatif : un régulateur de vitesse



Plan de la présentation

Rappels sur notre approche

Déploiement sur Inflexion

Démonstration : première partie

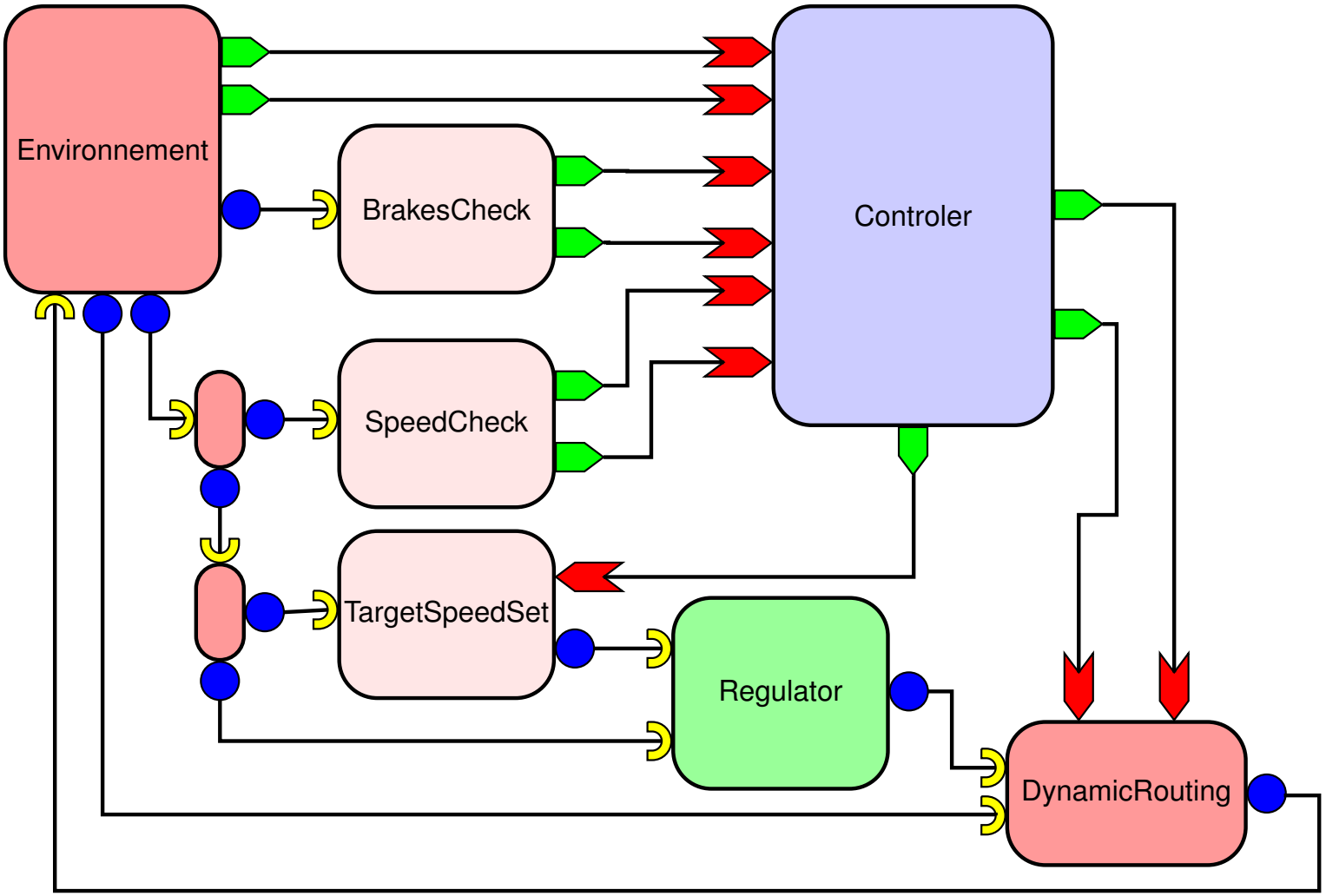
Vérification des propriétés

Démonstration : seconde partie

Conclusions et perspectives

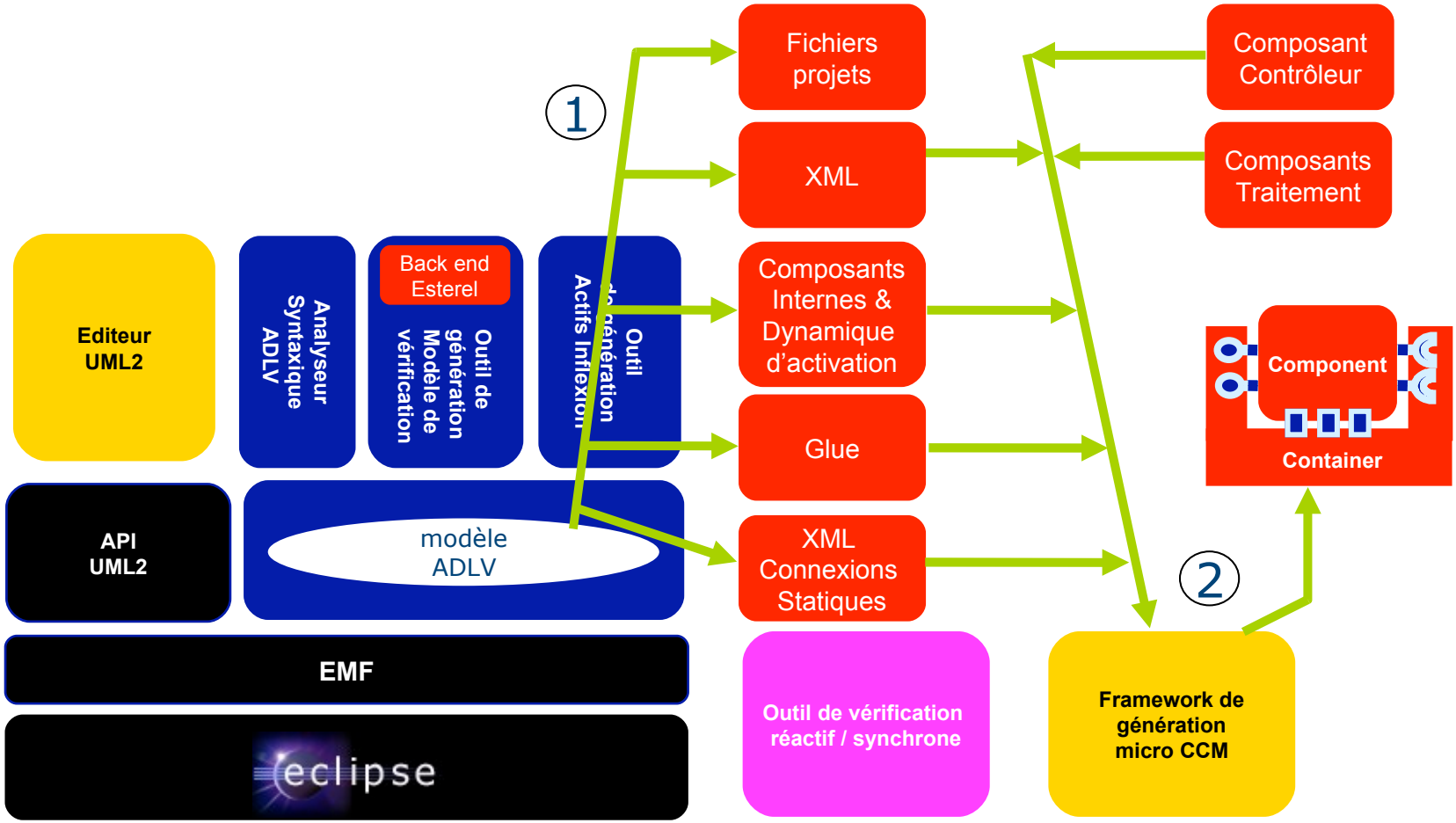
Supélec

Plateforme cible : LwCCM / Inflexion



Scénario de déploiement

1. Génération des actifs d'entrée à la chaîne de génération microCCM
   - Traduction des composants Internes en code C++.
   - Glue de mapping Composant / conteneur microCCM
   - IDL3 des composants Internes, Contrôleur et Traitements à partir de l'ADLV
   - Fichiers de déploiement correspondant aux connexions statiques
   - Fichiers de description de la construction
2. Génération des composants Inflexion déployables.



Plan de la présentation

Rappels sur notre approche

Déploiement sur Inflexion

Démonstration : première partie

Vérification des propriétés

Démonstration : seconde partie

Conclusions et perspectives

Supélec

Plan de la présentation

Rappels sur notre approche

Déploiement sur Inflexion

Démonstration : première partie

Vérification des propriétés

Démonstration : seconde partie

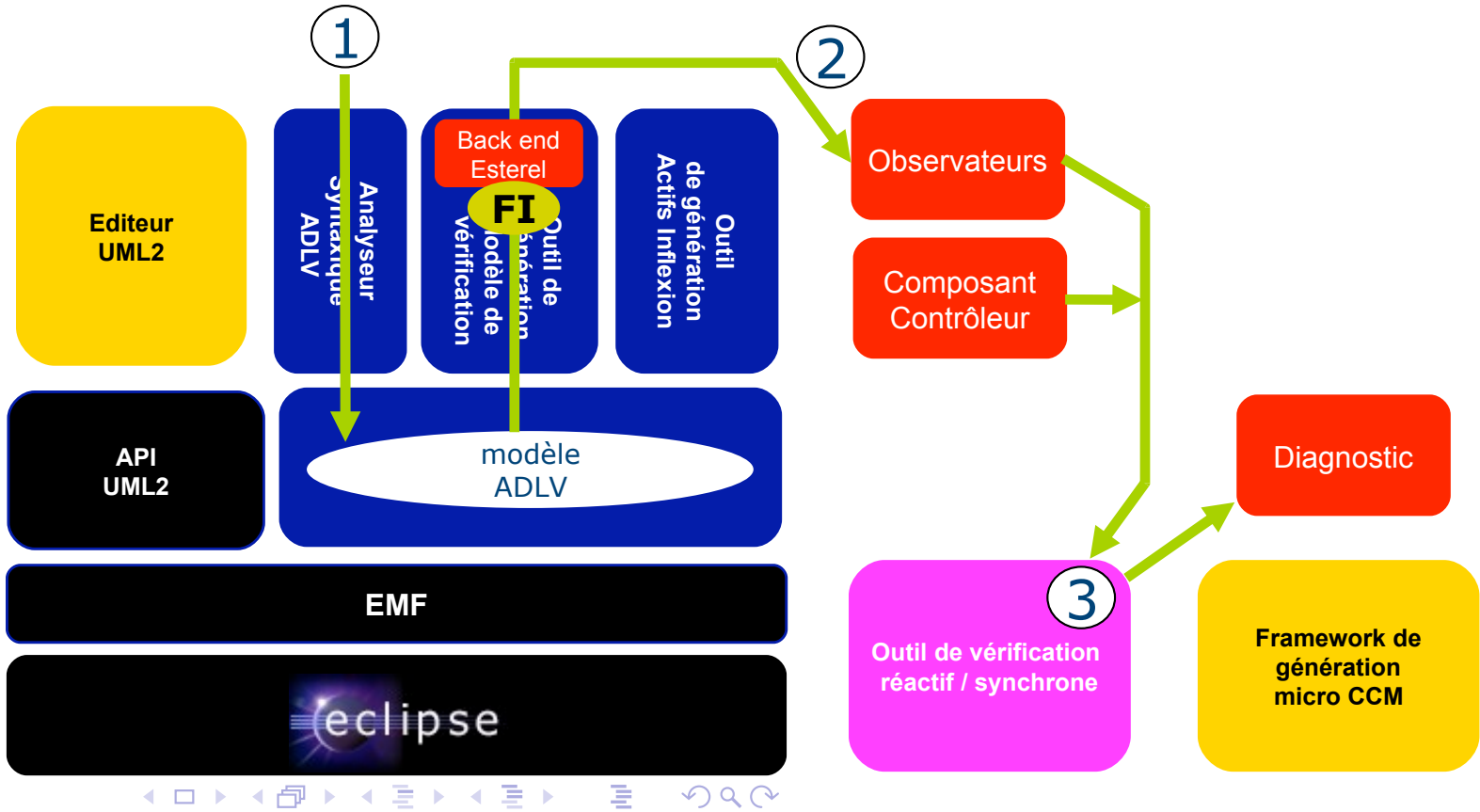
Conclusions et perspectives

Supélec

Scénario de vérification

1. Création (à partir d'une forme textuelle) du modèle ADLV
2. Assemblage du modèle réactif synchrone à vérifier:
   - Génération des observateurs du composant Contrôleur
   - Connexion avec le code réactif synchrone du composant Contrôleur
3. Vérification et diagnostic.

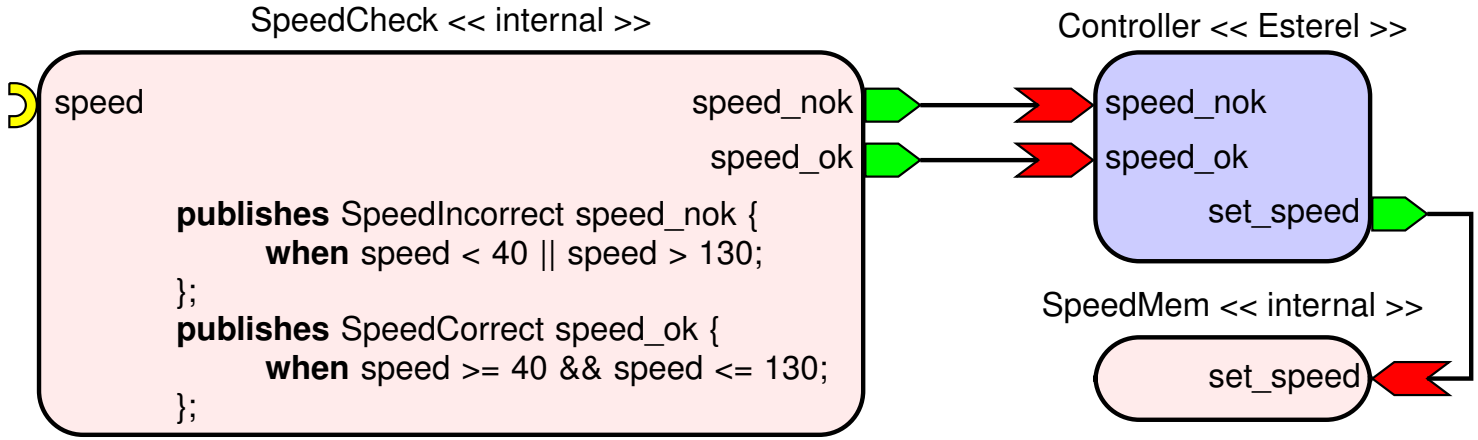
□ f où f ne contient que des opérateurs passés



Des signaux de l'application aux événements du contrôleur

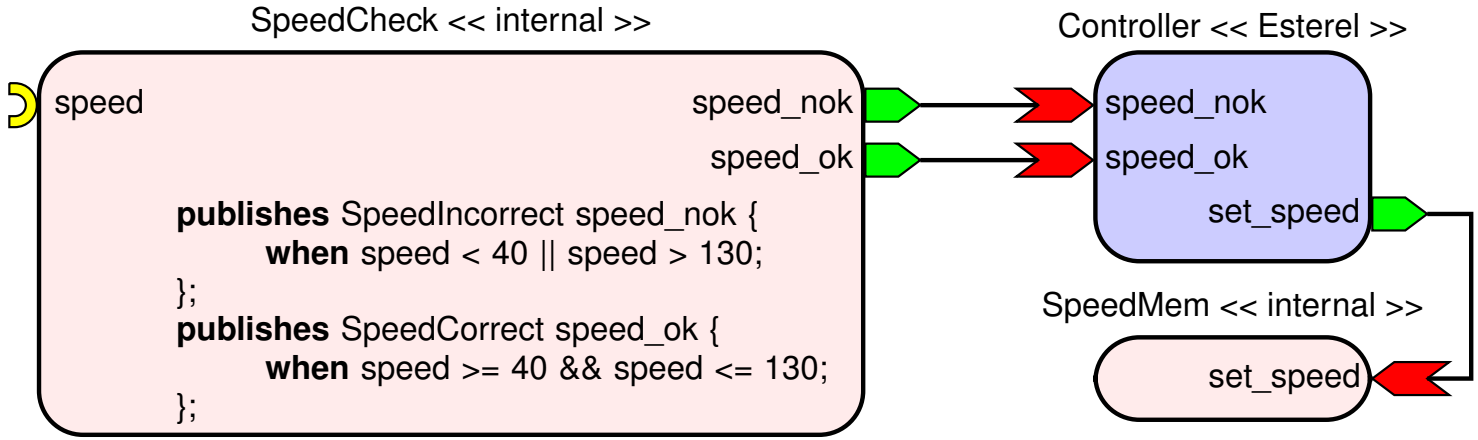
- Les prédicats d'une formule de sûreté portent sur des signaux de l'application (de types divers : entier, flottant, booléen...) ;
- les composants internes transforment ces signaux en événements à destination du contrôleur ;
- en effet, le contrôleur ne traite que des événements ;
- $\longrightarrow$ on utilise la description des composants internes pour remplacer tous les prédicats des formules de sûreté par des intervalles portant sur des événements du contrôleur : on obtient ainsi des formes intermédiaires ;
- ces formes intermédiaires servent alors à générer des observateurs dans le langage cible, qui permettent de vérifier la propriété.

Exemple



$\Box\neg f$, avec $f = (\underbrace{\text{speed} < 40 \ \vee \ \text{speed} > 130}_{g}) \ \wedge \ \text{set\_speed}$

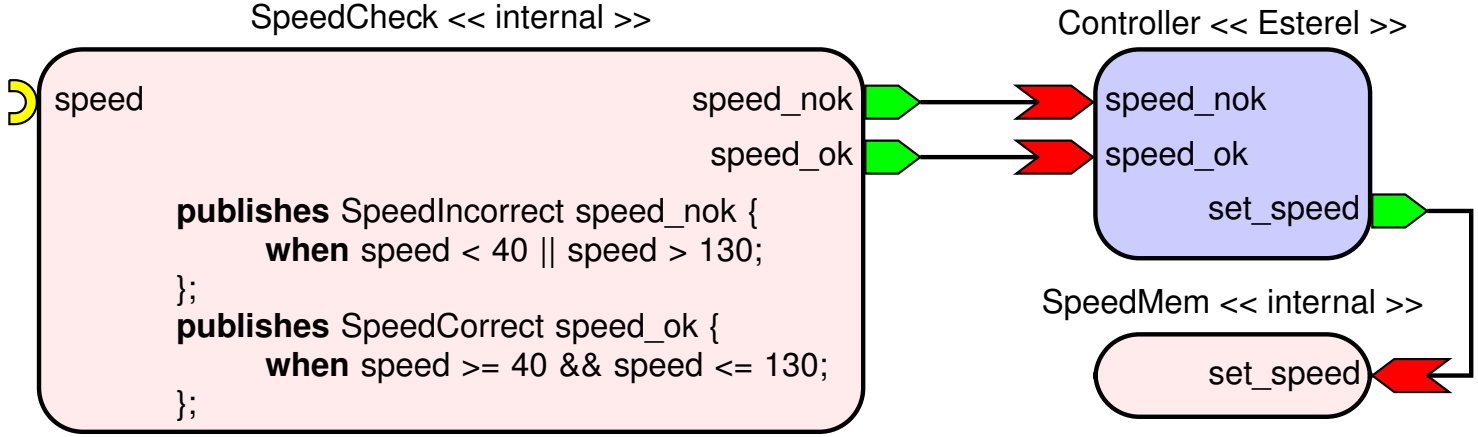
Exemple



$\Box\neg f$, avec $f = (\underbrace{\text{speed} < 40 \ \vee \ \text{speed} > 130}_{g}) \ \wedge \ \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements

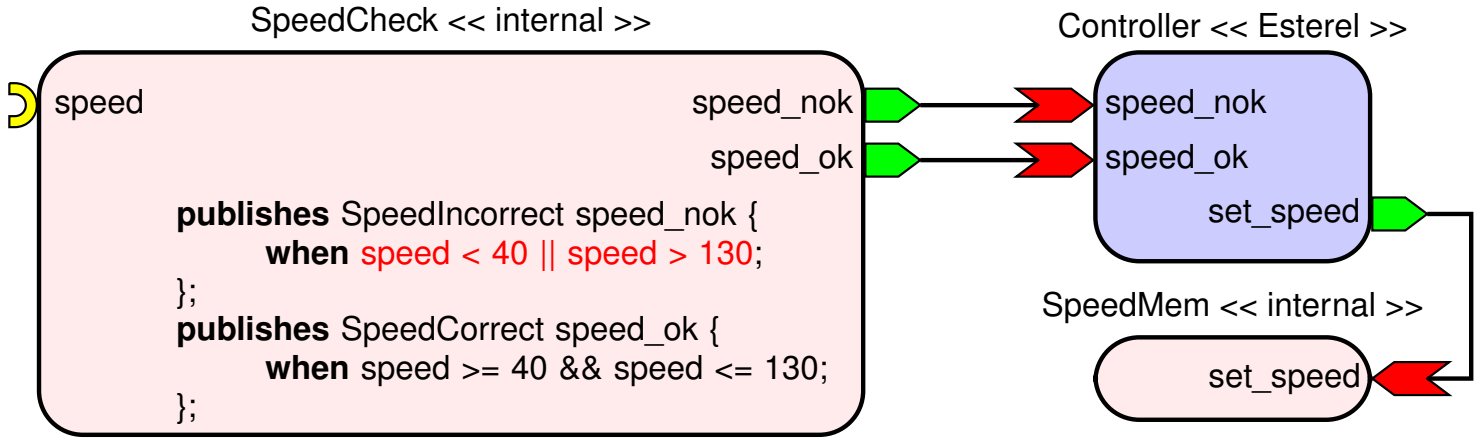
Exemple



$\Box\neg f$, avec $f = \underbrace{(\text{speed} < 40 \vee \text{speed} > 130)}_{g} \wedge \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements

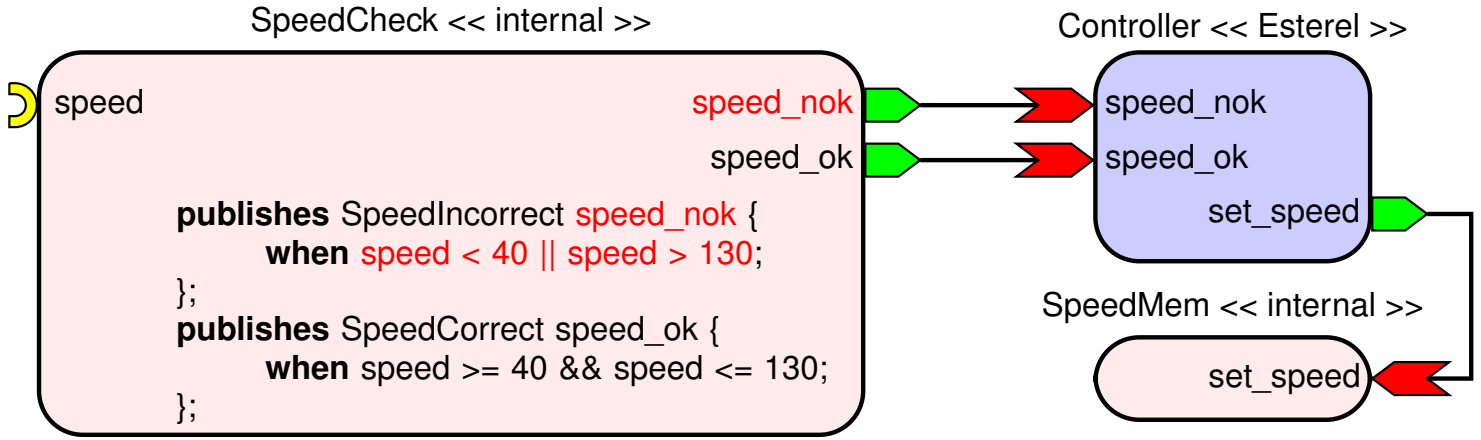
Exemple



$\Box\neg f$, avec $f = (\underbrace{\text{speed} < 40 \ \vee \ \text{speed} > 130}_{g}) \ \wedge \ \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements

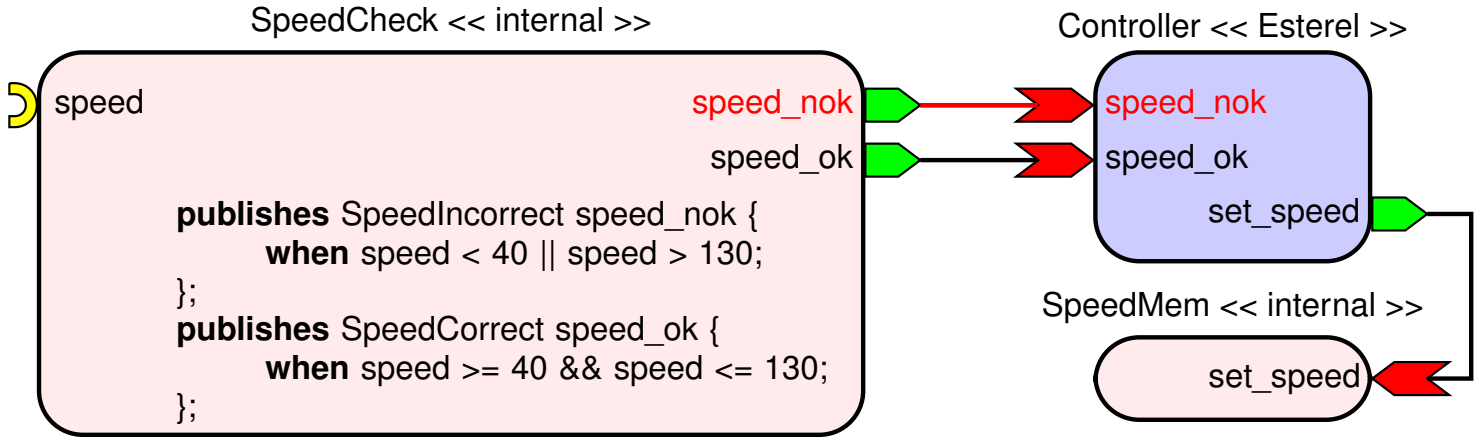
Exemple



$\Box\neg f$, avec $f = \underbrace{(\text{speed} < 40 \vee \text{speed} > 130)}_{g} \wedge \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements
- Identification des événements associés

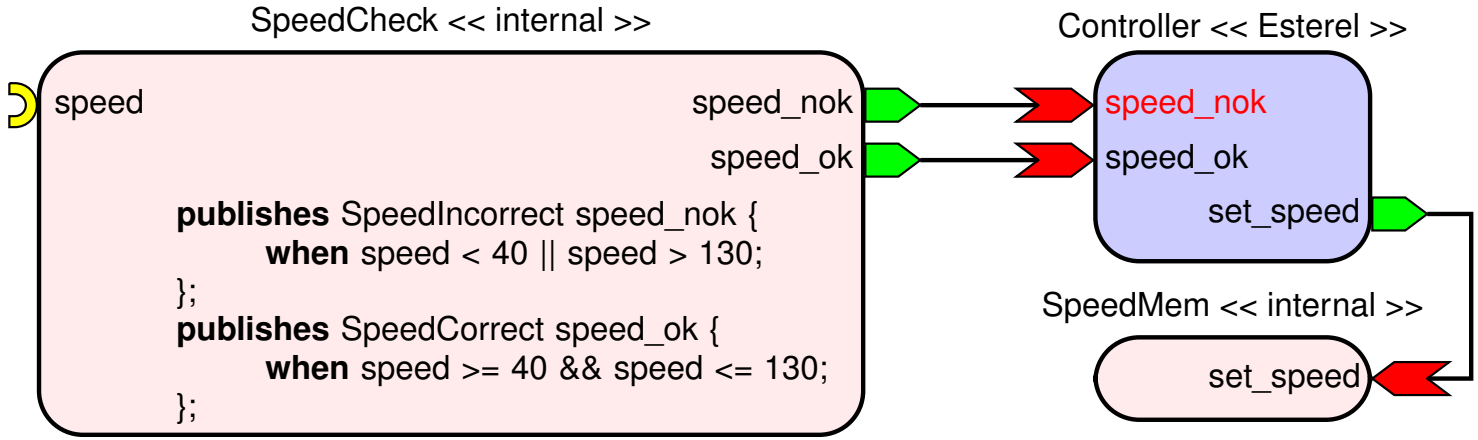
Exemple



$\Box\neg f$, avec $f = \underbrace{(\text{speed} < 40 \vee \text{speed} > 130)}_{g} \wedge \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements
- Identification des événements associés, vus du contrôleur

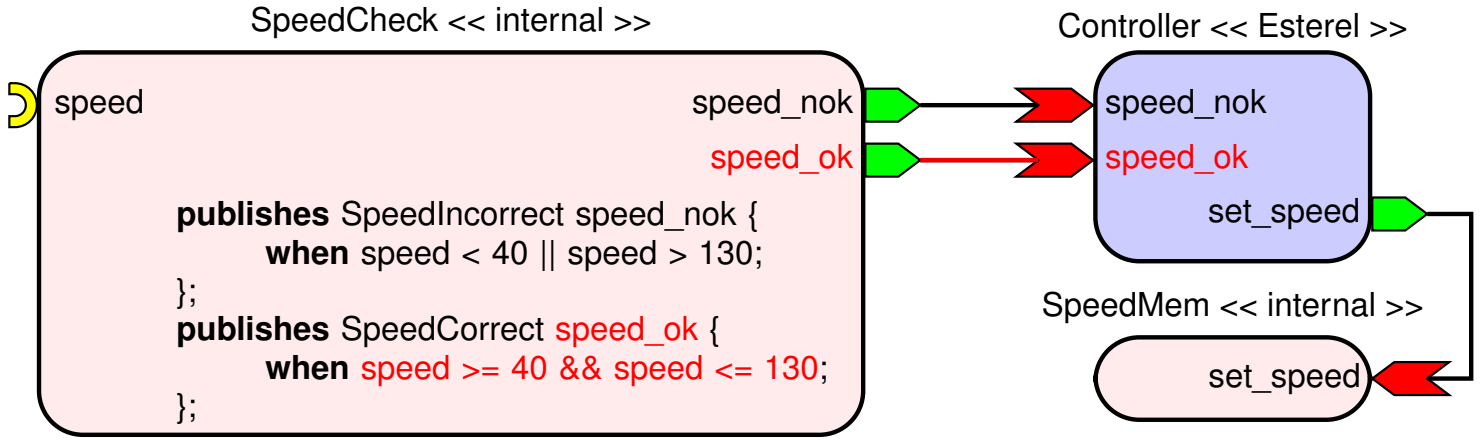
Exemple



$\Box\neg f$, avec $f = (\underbrace{\text{speed} < 40 \ \vee \ \text{speed} > 130}_{g}) \ \wedge \ \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements
- Identification des événements associés, vus du contrôleur
- Borne de début de l'intervalle associé à g : speed_nok

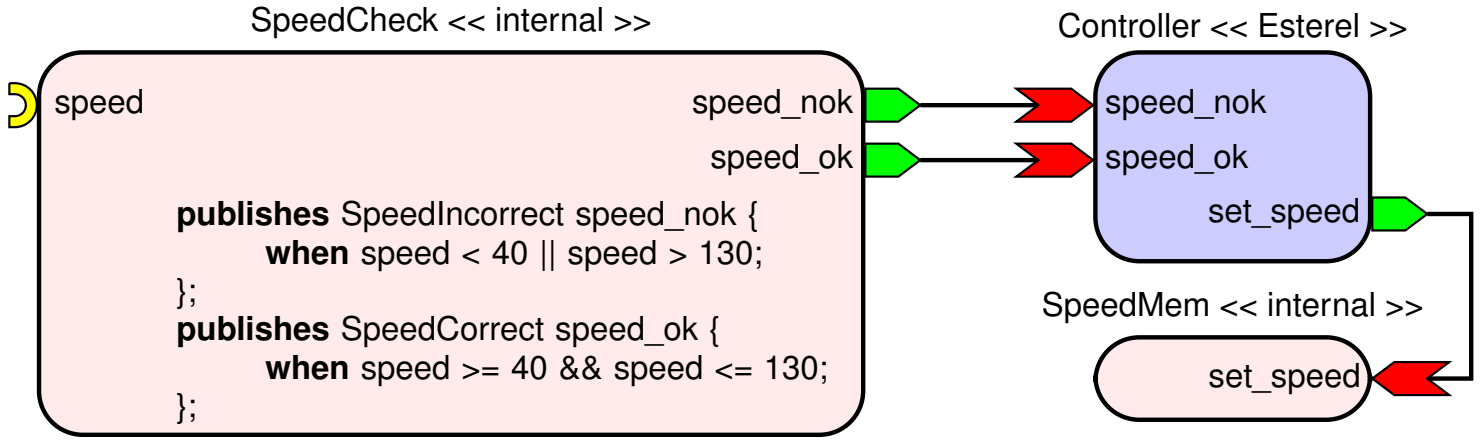
Exemple



$\Box\neg f$, avec $f = (\underbrace{\text{speed} < 40 \ \vee \ \text{speed} > 130}_{g}) \ \wedge \ \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements
- Identification des événements associés, vus du contrôleur
- Borne de début de l'intervalle associé à g : speed_nok
- Borne de fin (on recherche $\neg g$) : speed_ok

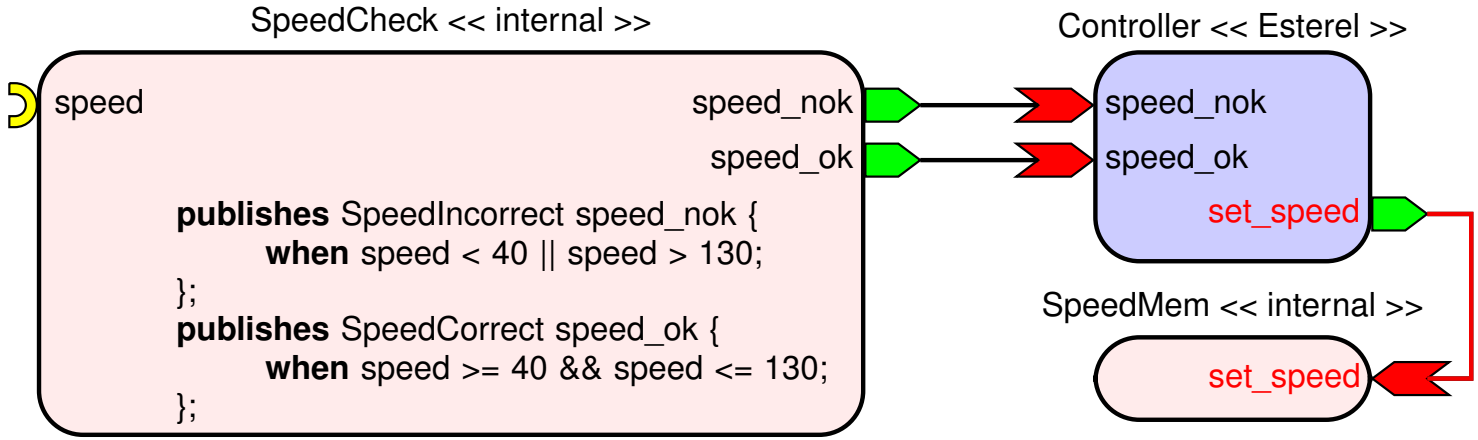
Exemple



$\Box\neg f$, avec $f = (\underbrace{\text{speed} < 40 \ \vee \ \text{speed} > 130}_{g}) \ \wedge \ \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements
- Identification des événements associés, vus du contrôleur
- Borne de début de l'intervalle associé à g : speed_nok
- Borne de fin (on recherche $\neg g$) : speed_ok
- Même chose pour set_speed

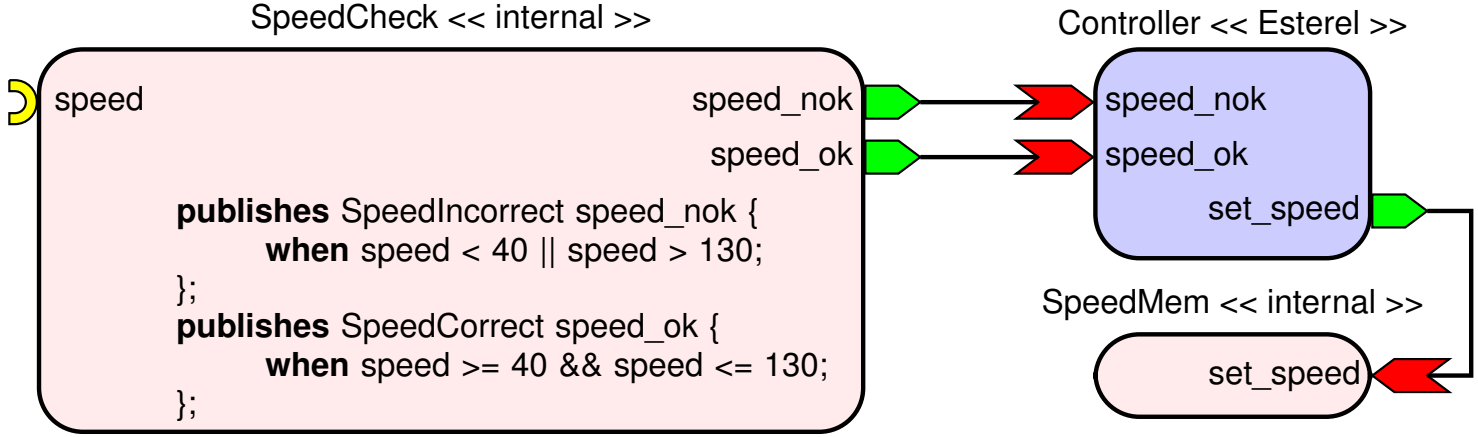
Exemple



$\Box\neg f$, avec $f = \underbrace{(\text{speed} < 40 \vee \text{speed} > 130)}_{g} \wedge \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements
- Identification des événements associés, vus du contrôleur
- Borne de début de l'intervalle associé à g : speed_nok
- Borne de fin (on recherche $\neg g$) : speed_ok
- Même chose pour set_speed : [set_speed, $\overline{\text{set\_speed}}$[

Exemple



$\Box\neg f$, avec $f = (\underbrace{\text{speed} < 40 \ \vee \ \text{speed} > 130}_{g}) \ \wedge \ \text{set\_speed}$

- Recherche de la formule ou si nécessaire de ses sous-formules dans les clauses `when` et événements
- Identification des événements associés, vus du contrôleur
- Borne de début de l'intervalle associé à g : speed_nok
- Borne de fin (on recherche $\neg g$) : speed_ok
- Même chose pour set_speed : [set_speed, $\overline{\text{set\_speed}}$[
- Forme intermédiaire de f : [speed_nok, speed_ok[$\wedge$ [set_speed, $\overline{\text{set\_speed}}$[

Génération exacte d'un observateur

Pour une formule f de logique temporelle passée :

1. si f ne contient pas d'opérateur temporel, on recherche f et $\neg f$ dans les clauses `when`
   - recherche à l'identique grâce à des *formes canoniques* basées sur une forme normale (conjonctive ou disjonctive)
   - Cas simple : *déduction directe d'une forme intermédiaire*

$$\begin{array}{l} s_1 \texttt{ when } f \\ s_2 \texttt{ when } \neg f \end{array} \longrightarrow \begin{array}{c} \text{forme intermédiaire exacte} \\ [s1, s2[\end{array}$$

2. si l'étape (1) échoue, on décompose f ($f = f_1 \star f_2$), et on applique l'algorithme à f_1 et f_2
   ($\star$ est un opérateur de logique temporelle passée)
3. la forme intermédiaire est traduite en *observateur* dans le langage cible (Esterel, Lustre, etc.)

Sous- et sur-vérification (1/2)

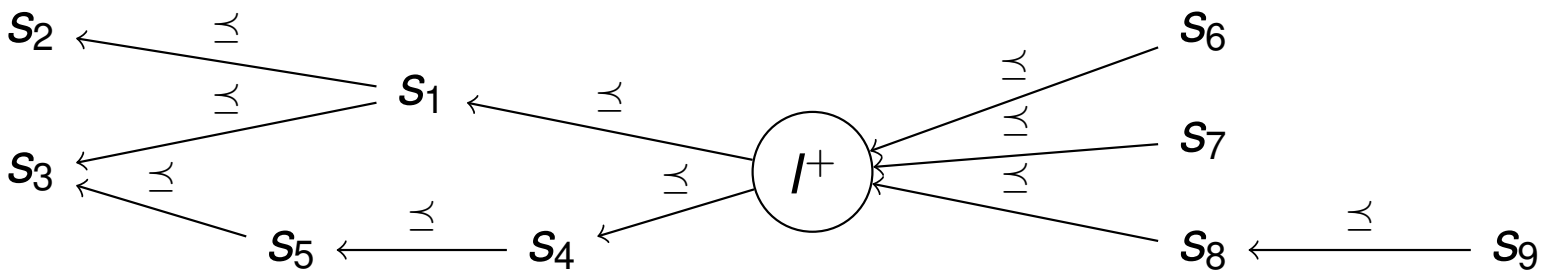
Lorsqu'on recherche une formule f dans des clauses `when`, on ne la trouve pas toujours telle-quelle... On peut avoir :

$$\begin{cases} s_1 \texttt{ when } a_1 \\ s_9 \texttt{ when } a_9 \end{cases} \quad \text{avec} \quad \begin{cases} f \Rightarrow a_1 \\ a_9 \Rightarrow f \end{cases}$$

En notant I^+ l'événement de début de I, l'intervalle de validité de f, ceci induit une relation d'ordre (temporel) entre événements :

$$s_1 \preceq I^+ \preceq s_9 \qquad (\preceq \text{ est un ordre partiel})$$

On encadre I^+ au plus juste par les ensembles S_O^+ et S_I^+ :



On encadre de même I^- (événement de fin de I) par S_I^- et S_O^-.

Sous- et sur-vérification (1/2)

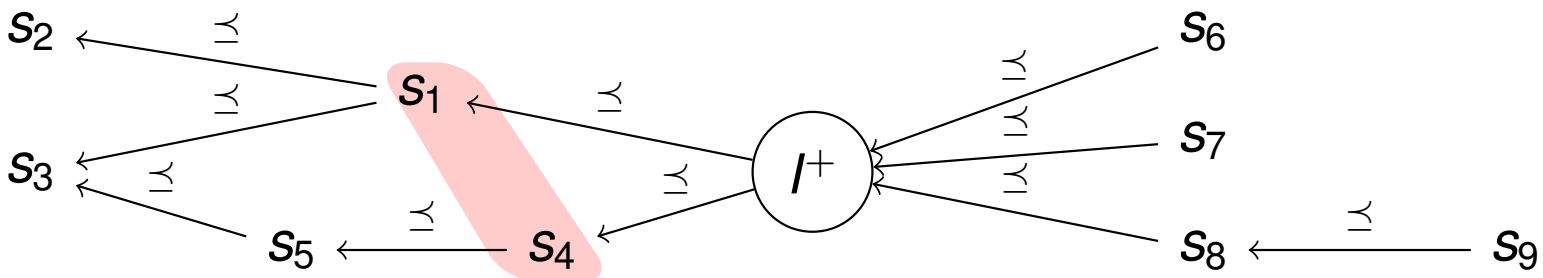
Lorsqu'on recherche une formule f dans des clauses `when`, on ne la trouve pas toujours telle-quelle... On peut avoir :

$$\begin{cases} s_1 \texttt{ when } a_1 \\ s_9 \texttt{ when } a_9 \end{cases} \quad \text{avec} \quad \begin{cases} f \Rightarrow a_1 \\ a_9 \Rightarrow f \end{cases}$$

En notant I^+ l'événement de début de I, l'intervalle de validité de f, ceci induit une relation d'ordre (temporel) entre événements :

$$s_1 \preceq I^+ \preceq s_9 \qquad (\preceq \text{ est un ordre partiel})$$

On encadre I^+ au plus juste par les ensembles S_O^+ et S_I^+ :



On encadre de même I^- (événement de fin de I) par S_I^- et S_O^-.

Sous- et sur-vérification (1/2)

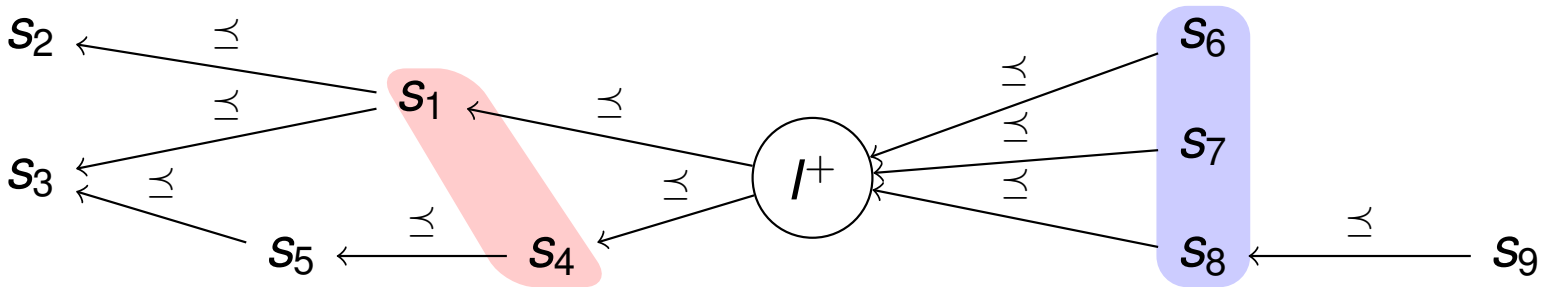
Lorsqu'on recherche une formule f dans des clauses `when`, on ne la trouve pas toujours telle-quelle... On peut avoir :

$$\begin{cases} s_1 \texttt{ when } a_1 \\ s_9 \texttt{ when } a_9 \end{cases} \quad \text{avec} \quad \begin{cases} f \Rightarrow a_1 \\ a_9 \Rightarrow f \end{cases}$$

En notant I^+ l'événement de début de I, l'intervalle de validité de f, ceci induit une relation d'ordre (temporel) entre événements :

$$s_1 \preceq I^+ \preceq s_9 \qquad (\preceq \text{ est un ordre partiel})$$

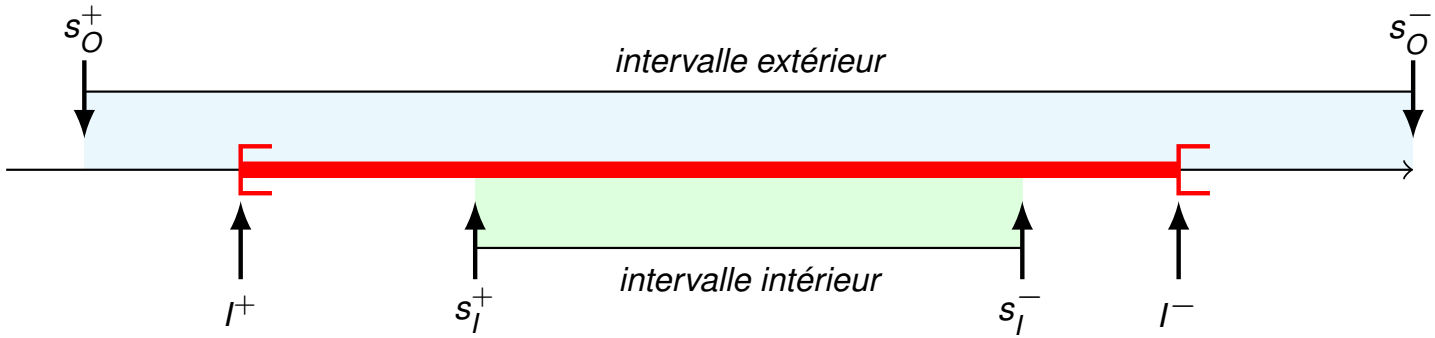
On encadre I^+ au plus juste par les ensembles S_O^+ et S_I^+ :



On encadre de même I^- (événement de fin de I) par S_I^- et S_O^-.

Sous- et sur-vérification (2/2)

- Pour un intervalle I, $\langle S_O^+, S_I^+, S_I^-, S_O^- \rangle$ définit 2 intervalles :



- On peut obtenir au final deux formes intermédiaires $\longrightarrow$ deux observateurs approchés :
  - observateur trop lâche : tout défaut détecté est un vrai défaut, mais il peut y avoir des cas d'erreur non détectés ;
  - observateur trop strict : tous les cas d'erreur sont détectés, mais on peut trouver des « défauts » qui n'en sont pas.

Plan de la présentation

Rappels sur notre approche

Déploiement sur Inflexion

Démonstration : première partie

Vérification des propriétés

Démonstration : seconde partie

Conclusions et perspectives

Supélec

Plan de la présentation

Rappels sur notre approche

Déploiement sur Inflexion

Démonstration : première partie

Vérification des propriétés

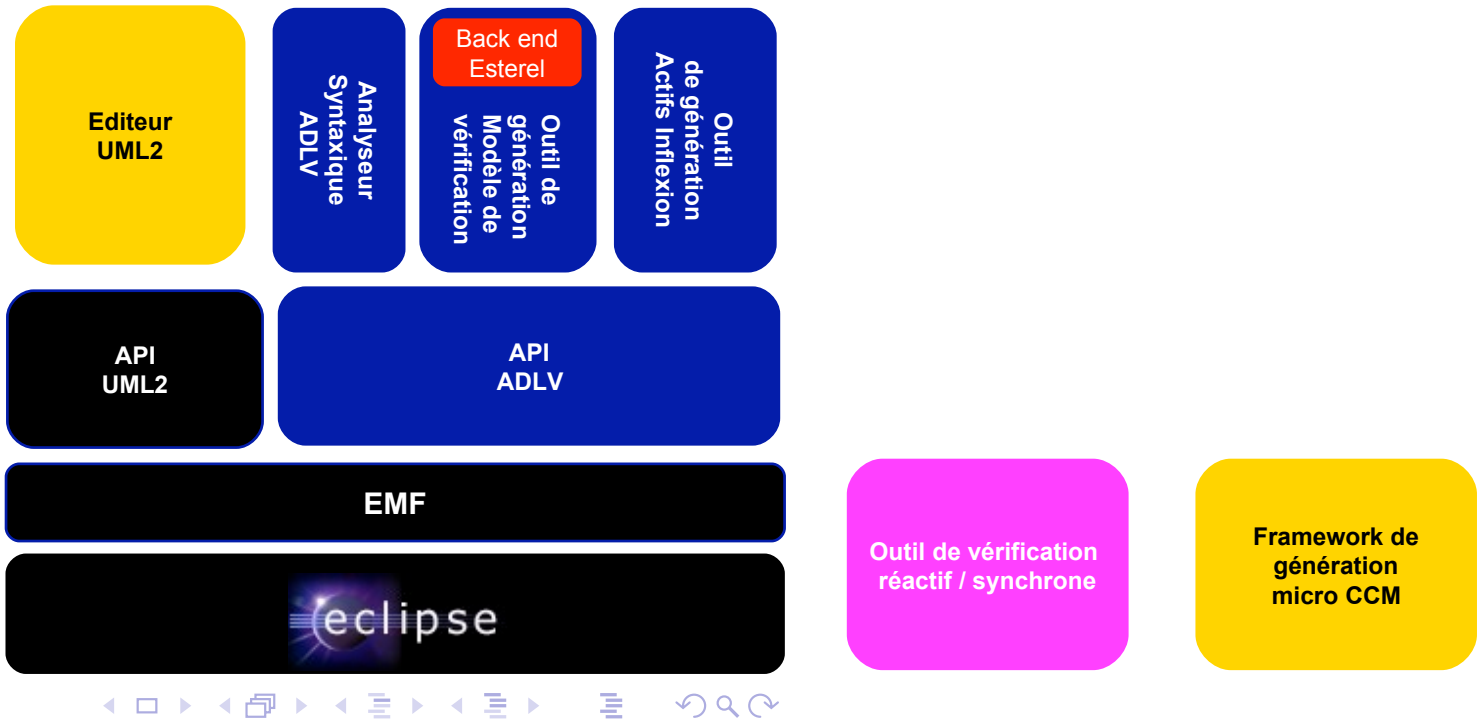
Démonstration : seconde partie

Conclusions et perspectives

Supélec

Résultats de l'activité

- Composant d'intégration contrôle-traitements (D2.3)
  - Logiciel disponible : http://wwwdi.supelec.fr/logiciels
  - Licence EPL
- Rapport sur le composant d'intégration contrôle-traitements



Perspectives de poursuite des travaux et de valorisation

- Deux publications en préparation
  - une sur l'aspect vérification des propriétés
  - une sur l'aspect architecture logicielle et déploiement sur une infrastructure d'exécution
- Des contacts sont en cours avec Thales Communications pour étendre ADLV en direction de la reconfiguration
- Les parties « vérification de propriétés » et « déploiement sur Inflexion » sont faiblement couplées, et peuvent être valorisées indépendamment