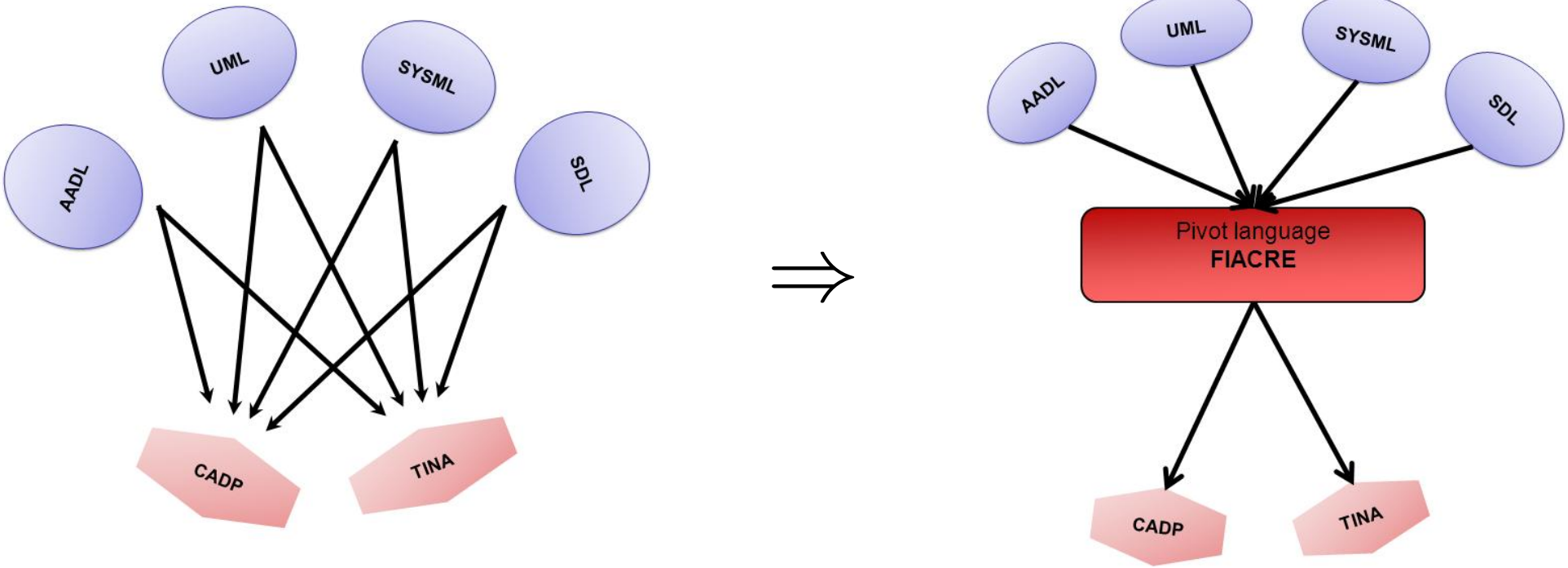
FIACRE – Un langage formel pour systèmes temps-réels

Bernard Berthomieu Didier le Botlan
Silvano Dal Zilio François Vernadat

LAAS/CNRS/INSA

AFSEC
Paris, 18 novembre 2010

Justification



Traits

Langage formel de description de systèmes TR

Process = composants de base

- États explicites, transitions symboliques
- Structures de contrôle classiques
- Riches types de données
- Communications (variables partagées ET messages)

Component = compositons + contraintes

- Description des interactions entre sous-composants
- Contraintes temporelles sur interactions
- Contraintes de priorité sur interactions

Types de données

Basiques :

int, nat, bool

Intervalles :

type byte is 0..255

Unions et énumérations :

type color is union red | green | blue end

type response is union Bat of nat | Bool of bool end

Enregistrements :

type size is record width : nat, height : nat end

Tableaux et files de taille fixe :

type table is array 10 of nat end

type fifo is queue 5 of byte end

Composants de base

Process, paramétrés par :

des canaux de communications, typés et orientés

des variables partagées, typées et avec contrôle d'accés

des valeurs typées

process P [p : **in nat**] (&x : **read** 0..7, z : **bool**)

Transitions :

définies par un état initial s et un bloc d'instructions S

définissant les actions à effectuer depuis l'état cible : **from** s S

Instructions (1/2)

Affectations

$P_1, \ldots, P_n := E_1, \ldots, E_n$

$P_1, \ldots, P_n :=$ **any where** E

case E **of** $P_1 \rightarrow S_1 | \ldots | P_n \rightarrow S_n$ **end**

Séquences

$S_1 ; \ldots ; S_n$

Boucles et branchements

if E **then** S_1 **else** S_2 **end**

foreach E **do** S **end**

while E **do** S **end**

Instructions (2/2)

Alternatives

select S_1 [] ... [] S_n**end**

Synchronisation et Communication

Synchronisation : p

Emission : $p!E_1, \ldots, E_n$

Réception filtrante : $p?E_1, \ldots, E_n$ **where** E

Saut vers état suivant

to s

loop

Un exemple (1/2)

type index **is** 0..3

type request **is union** getSum | getValue **of** int **end**

type data **is array** 4 of **nat**

Un exemple (2/2)

```
process ATM [req : in request, resp : out nat ] is
  states ready, sendSum, sendValue
  var c : request, i : index, sum : nat, val : data := [6,2,7,9]
  Init to ready
  from ready
    req ? c
    case c of
        req ? getSum → to sendSum
        req ? getValue i → to sendValue
    end
  from sendValue
    resp ! val[i] ;
    to ready
  from sendSum
    sum := 0 ;
    foreach i do sum := sum + val [i] end ;
    resp ! sum ;
    to ready
```

Sémantique des transitions

Peuvent avoir plusieurs chemins d'éxécution

en raison des **if**, **case**, ...

en raison du non déterminisme (**select**, **any**, ?, ...)

Contrainte

au plus un événement de communication/synchronisation

ou écriture VP par chemin d'éxécution

Transitions sont atomiques

éxécutées entièrement (= jusqu'à un saut) ou pas du tout

Composants

Compositions, paramétrés par :

- canaux de commnication/synchronisation typés et orientés (**in**, **out**)
- variables {partagées} typées et avec contrôle d'accés (**read**, **write**)

Canaux locaux et variables partagées locales

- variables {partagées}
- canaux locaux {temporellement contraints} **port** p : 0..7 **in** $[2, 5[$
- priorités **priority** p | q | r < a | b

Comportement défini par :

- produit synchronisé d'instances de processus ou composants
- contraintes temporelles sur interactions
- contraintes de priorité sur interactions

Exemple de composant

```
component C [r : in 0..7, s : bool ] is
   var a : nat := 5, val b : array 4 of bool
   port p : 0..7 in [4,8], q : nat
   priority p > q, q > r
   par p -> A [p,r] || p, s -> B [p,s] || s -> C [s,q] (&a,b) end
```

Outillage Fiacre (1/2)

(2007-2010)

Meta-modèle Fiacre

Sémantique formelle

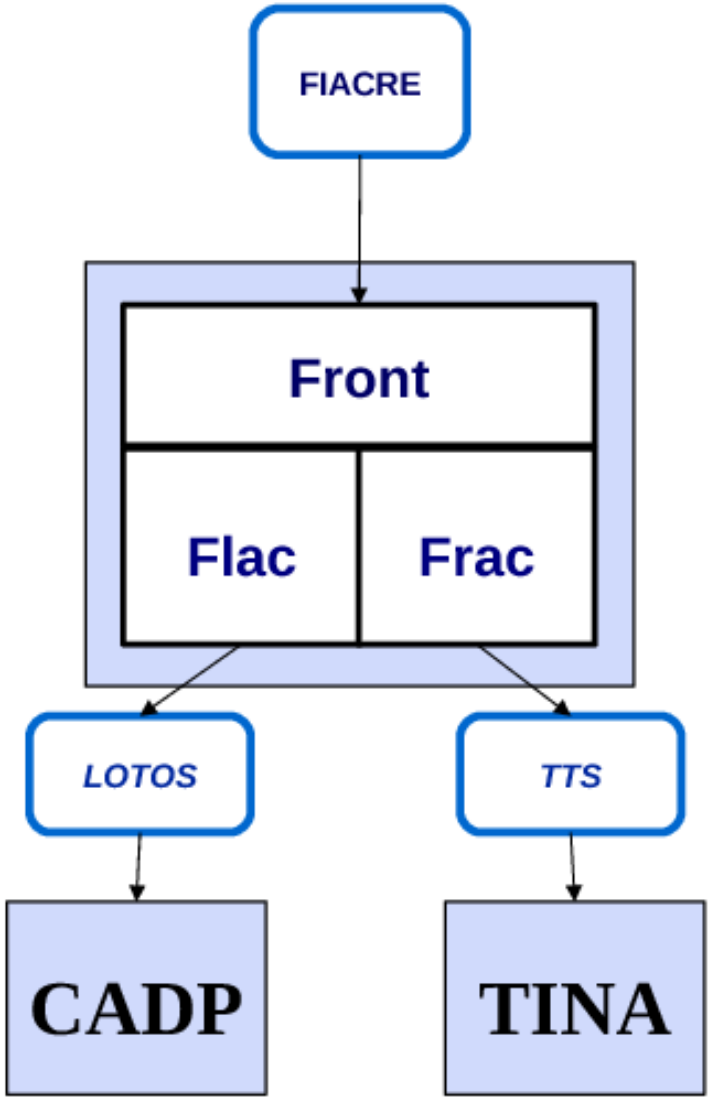
LAAS-CNRS

- Front : Fiacre → AT
- Frac : AT → TTS/TINA

(2007-2010)

INRIA/VASY

- Flac : AT → LOTOS/CADP



Outillage Fiacre (2/2)

Front : front-end (commun pour Flac et Frac)

Parser (lex, yacc) + Contrôle de types + ver. statiques → AST annoté

Frac : back-end pour Tina-TTS

Réduction des constructions dérivées (select, any, etc) → Core-AST

Composition statique des composants → TTS abstrait

Mise à plat, Instantiations, Passage des paramètres

Optimisations → TTS optimisé

Analyse des variables
Normalisation des transitions

Génération → .tts

.tts = Réseau de Petri (.net) + Traitement données (.c selon API)
Librairie partagée (.dll/.so/.dylib) produite depuis .c

Liens

tina:

http://www.laas.fr/tina

frac:

http://www.laas.fr/~bernard/fiacre

Démonstration