Panorama de CADP 2001

Hubert Garavel

projet VASY

INRIA Rhône-Alpes

655, avenue de l'Europe

38330 Montbonnot Saint Martin

CADP

- *CAESAR/ALDEBARAN Development Package*
- Une boîte à outils pour l'ingénierie des protocoles et des systèmes répartis
- Caractéristiques essentielles :
  - modélisation par algèbres de processus (LOTOS)
  - vérification par équivalences (bisimulations)
  - vérification par logiques (mu-calcul modal)
  - vérification exhaustive, partielle, à la volée, compositionnelle
  - génération de code C, prototypage rapide
  - simulation pas à pas, exécution aléatoire
  - génération de tests

Origines de CADP

- Développement entrepris en 1986
- Travail conjoint entre
  - le projet VASY de l'INRIA
  - le laboratoire Verimag

  avec des contributions

  - du projet PAMPA de l'IRISA
  - du groupe FMT de l'Université de Twente

Principales applications de CADP

- Etudes de cas industrielles
  - matériel, logiciel, télécoms, systèmes embarqués…
  - spécification formelle de systèmes et protocoles critiques
  - simulation, prototypage rapide, vérification, test
- Recherche
  - analyse de nouveaux protocoles/systèmes
  - développement d'outils de vérification/test
  - implémentation de nouveaux langages de modélisation
- Enseignement
  - parallélisme, algèbres de processus, bisimulations, logiques temporelles
  - outils robustes pour TD/TP et projets d'étudiants

Plan

- LOTOS et le modèle des Systèmes de Transitions Etiquetées (STE)
- Outils pour LOTOS
- Outils pour les STE explicites
- Outils pour les STE implicites
- Outils pour la vérification compositionnelle
- Architecture de CADP
- Conclusion

LOTOS et le modèle des Systèmes de Transitions Etiquetées (STE)

LOTOS

Language Of Temporal Ordering Specification **[ISO-8807:1989]**

- Une Technique de Description Formelle pour la description des protocoles et systèmes répartis
- Deux sous-langages « orthogonaux » :

  Données: types de données abstraits (ActOne)

  - sortes et opérations
  - équations algébriques

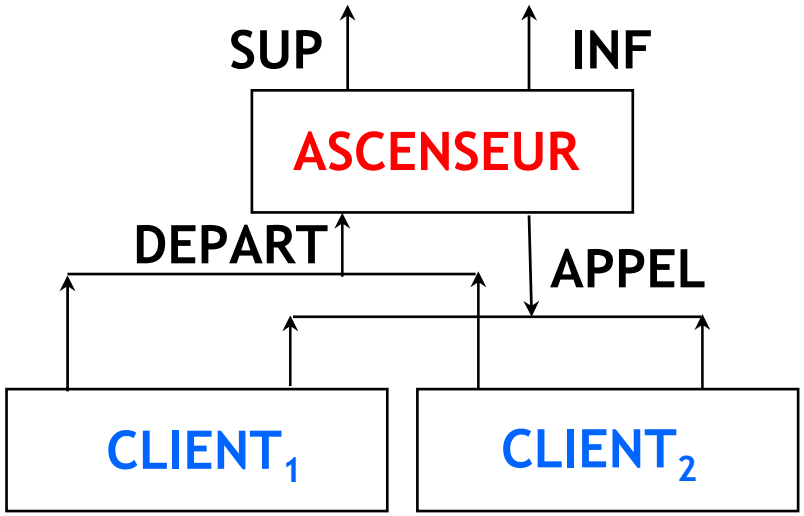
  Processus: algèbre de processus (~CCS, CSP, Circal)

  - parallélisme asynchrone (sémantique d'entrelacement)
  - communication par messages

Types LOTOS : un example

```
type ETAGE is  BOOLEAN
  sorts
    ETG

  opns
    BAS (*! constructor *),
    MILIEU (*! constructor *),
    HAUT (*! constructor *),
    ERREUR (*! constructor *) :-> ETG
    INCR, DECR : ETG -> ETG
    _==_ , _<_ , _>_ : ETG, ETG -> BOOL

  eqns
    forall X, Y:ETG
    ofsort ETG
       INCR (BAS) = MILIEU;
       INCR (MILIEU) = HAUT;
       (* else *) INCR (X) = ERREUR;

    ofsort ETG
       DECR (MILIEU) = BAS;
       DECR (HAUT) = MILIEU;
       (* else *) DECR (X) = ERREUR;

    ofsort BOOL
       X == X = true;
       (* else *) X == Y = false;

    ofsort BOOL
       BAS < MILIEU = true;
       BAS < HAUT = true;
       MILIEU < HAUT = true;
       (* else *) X < Y = false;

    ofsort BOOL
       X > Y = Y < X;
endtype
```

Processus LOTOS : un example



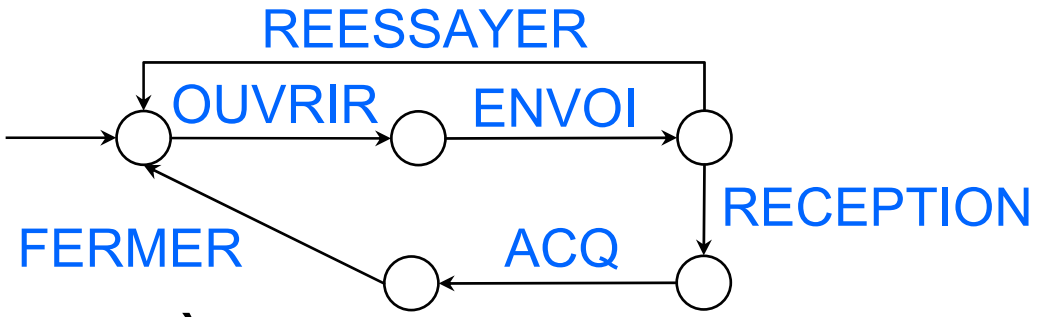

```
ASCENSEUR [APPEL, DEPART, SUP, INF] (BAS, BAS)
|[APPEL, DEPART]|
  (
  CLIENT [APPEL, DEPART] (BAS, HAUT)
  |||
  CLIENT [APPEL, DEPART] (HAUT, MILIEU)
  )
```

Processus LOTOS (suite)

```
process ASCENSEUR [APPEL, DEPART, SUP, INF] (COURANT, BUT: ETG) : noexit :=
    [BUT > COURANT] ->
      SUP !INCR (COURANT);
        ASCENSEUR [APPEL, DEPART, SUP, INF] (INCR (COURANT), BUT)
    []
    [BUT < COURANT] ->
      INF !DECR (COURANT);
        ASCENSEUR [APPEL, DEPART, SUP, INF] (DECR (COURANT), BUT)
    []
    [BUT == COURANT] ->
      (
      APPEL ?NOUVEAU_BUT:ETG;
        ASCENSEUR [APPEL, DEPART, SUP, INF] (COURANT, NOUVEAU_BUT)
      []
      DEPART ?NOUVEAU_BUT:ETG;
        ASCENSEUR [APPEL, DEPART, SUP, INF] (COURANT, NOUVEAU_BUT)
      )
endproc
```

Systèmes de transitions étiquetées (STEs)

- STE: le modèle sémantique standard pour les langages à actions (y compris LOTOS)



- $M = (S, A, T, s_0)$, où:
  - S : ensemble d'états
  - A : ensemble d'étiquettes (informations attachées aux transitions)
  - $T \in S \times A \times S$: relation de transition
  - $s_0 \in S$: état initial

STEs et vérification

- Les STEs constituent une base standard pour de nombreux algorithmes de vérification

- Exemples :
  - Analyse des états accessibles (exploration de STE)
  - Vérifications par équivalences (bisimulations)
  - Vérifications par logiques (mu-calcul modal)

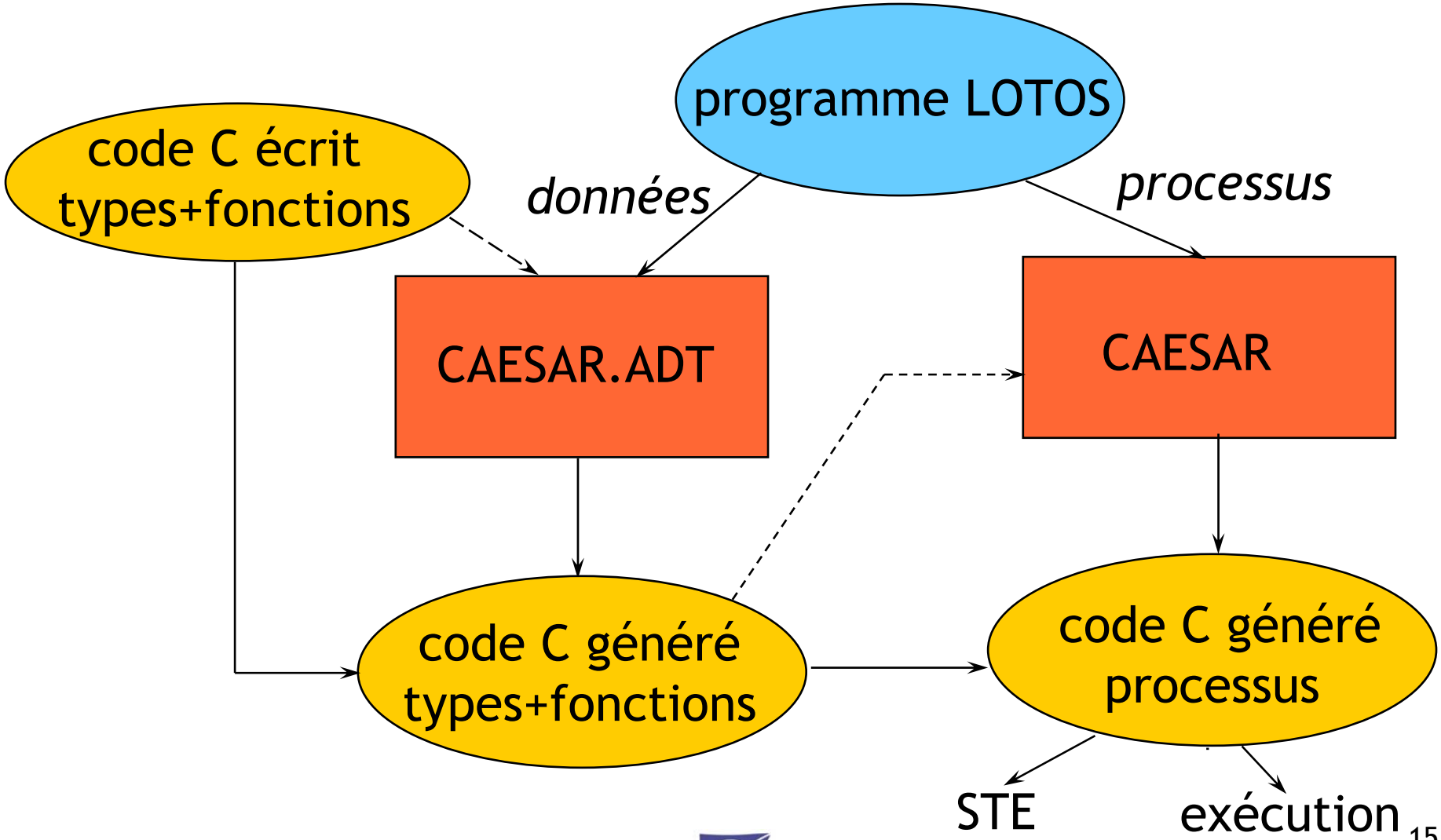
Représentations informatiques des STEs

- STE "explicite" (ou STE "en extension") : STE défini par la liste exhaustive de ses états et transitions
  - les successeurs et prédécesseurs des états sont connus : le STE peut être exploré en avant et en arrière
  - ceci permet la vérification globale et locale (à la volée)
  - CADP fournit les outils BCG pour les STEs explicites (finis)

- STE "implicite" (ou STE "en compréhension") : STE défini par son état initial et sa fonction successeur
  - les prédécesseurs des états ne sont pas connus : seule l'exploration en avant (vérification locale) est permise
  - CADP fournit les outils Open/Caesar pour les STEs implicites

Outils CADP pour LOTOS

CAESAR.ADT et CAESAR



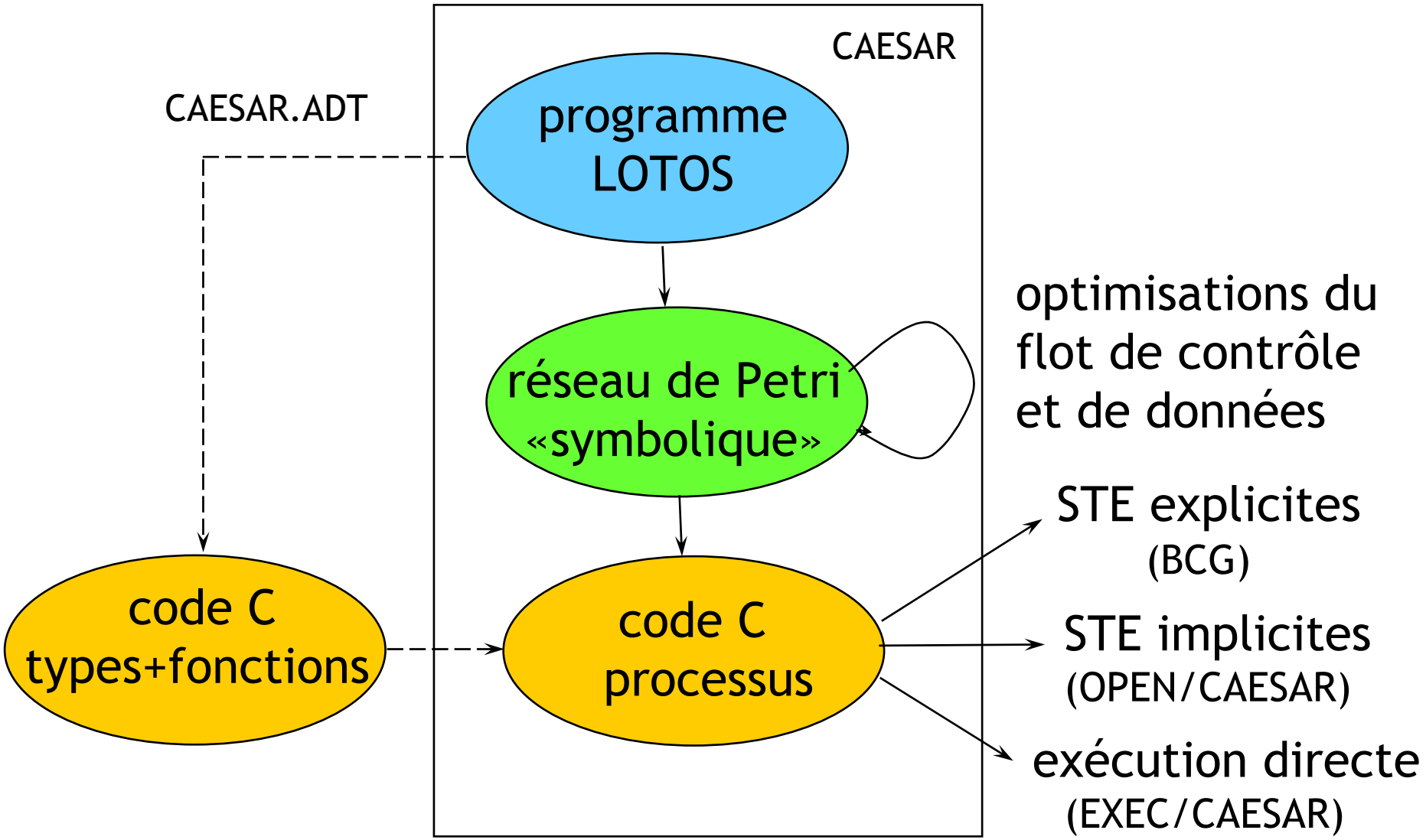
CAESAR.ADT

- **Traduction : types abstraits LOTOS → C**
  - chaque sorte LOTOS → un type C
  - chaque opération LOTOS → une fonction C
- **Hypothèses ajoutées à la norme LOTOS**
  - différence entre constructeurs et non-constructeurs
  - constructeur libres
  - équations vues comme des règles de réécriture avec filtrage et priorités
- **Génération de code C spécialisé**
  - Ciblé vers les besoins du « *model checking* »
  - *Optimiser d'abord la mémoire, puis la vitesse*

CAESAR.ADT (suite)

- **Compilation des structures de données**
  - structures de données dynamiques (listes, arbres...) permises
  - implantation mémoire optimisée :
    - nombre minimal de bits
    - permutation des champs d'enregistrements
    - mise en facteur des sous-termes communs
- **Compilation des fonctions**
  - algorithme de compilation du filtrage
  - optimisations ad hoc

CAESAR


CAESAR
CAESAR.ADT
programme LOTOS
réseau de Petri «symbolique»
code C types+fonctions
code C processus
optimisations du flot de contrôle et de données
STE explicites (BCG)
STE implicites (OPEN/CAESAR)
exécution directe (EXEC/CAESAR)

Outils CADP pour les STEs *explicites*

Motivations

- Comment stocker des STEs **très grands** dans des fichiers informatiques ?
- Les formats textuels ne conviennent pas :
  - coûteux en place disque (centaines de Mbytes, voire Gbytes)
  - lents (lectures/écritures coûteuses en temps)
  - parfois ambigus

Le format BCG de CADP

BCG (*Binary-Coded Graphs*) :

- **un format de fichier compact** pour les STEs
- **un ensemble d'APIs**
- **un ensemble de bibliothèques logicielles**
- **un ensemble d'outils (programmes binaires et scripts)**

Implémentation : 30,000 lignes de code C

BCG est fourni comme composant de CADP

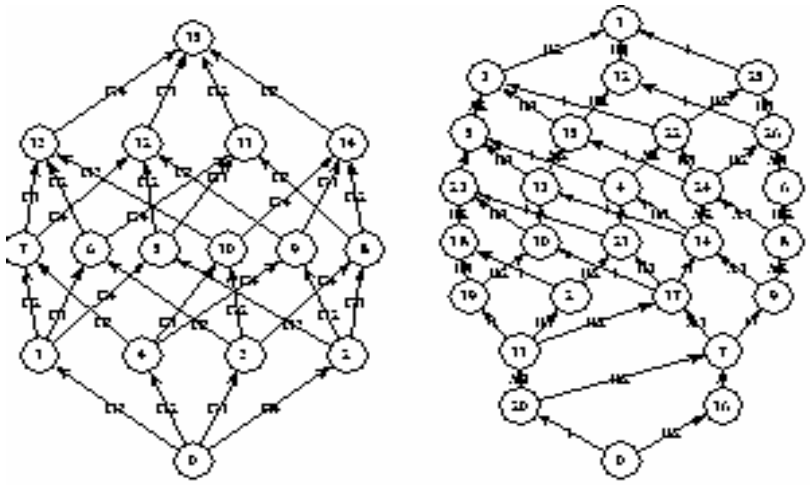
Tous les outils de CADP utilisent BCG uniformément

Bibliothèques et API BCG

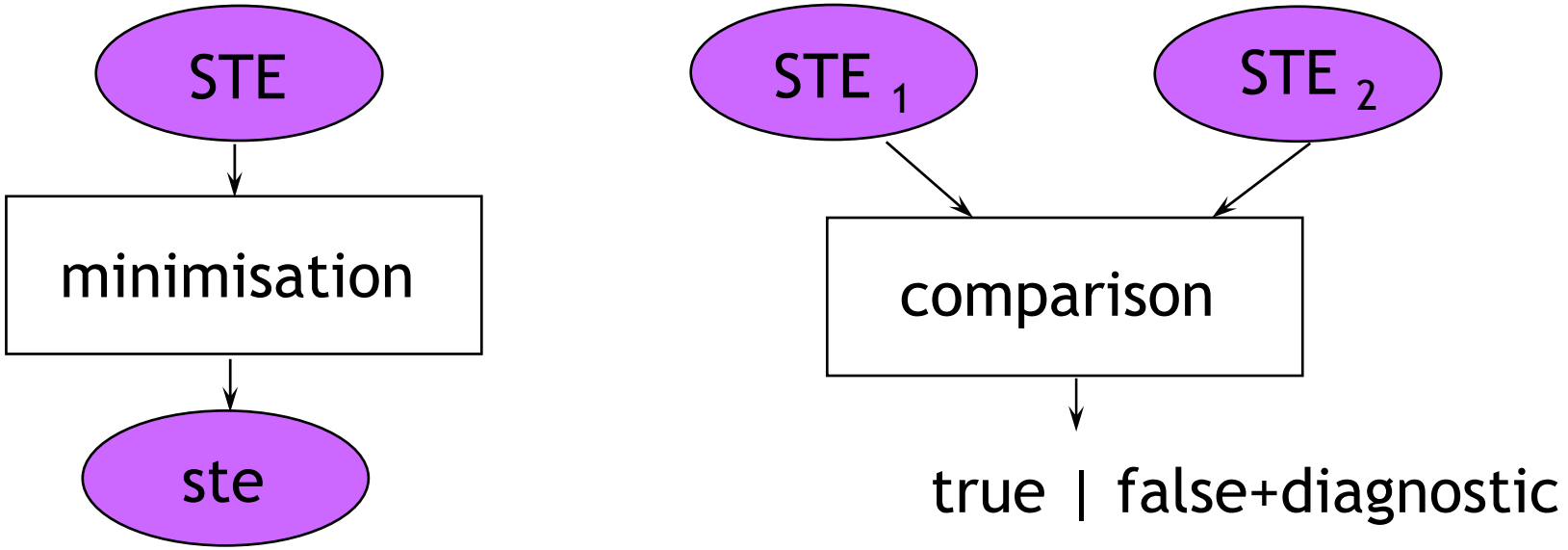
- **bcg_write:** API pour créer un fichier BCG
- **bcg_read:** API pour lire un fichier BCG
- **bcg_transition:** API pour stocker une relation de transition en mémoire :
  - fonction successeur, ou
  - fonction prédécesseur, ou
  - fonctions successeur et prédécesseur

Les outils BCG de base

- **bcg_info:** extrait les infos d'un fichier BCG
- **bcg_io:** convertit BCG $\leftrightarrow$ autres formats
- **bcg_labels:** masque/renomme les étiquettes
- **bcg_draw, bcg_edit:** visualise les STEs



Outils de vérification par équivalences



- CADP fournit 3 outils de ce type :
  - ALDEBARAN (Verimag)
  - FC2TOOLS (INRIA/Meije) – interfacé avec CADP
  - BCG_MIN (INRIA/VASY) – le plus récent
- Diverses équivalences disponibles : forte, observationnelle, de branchement, de sûreté…

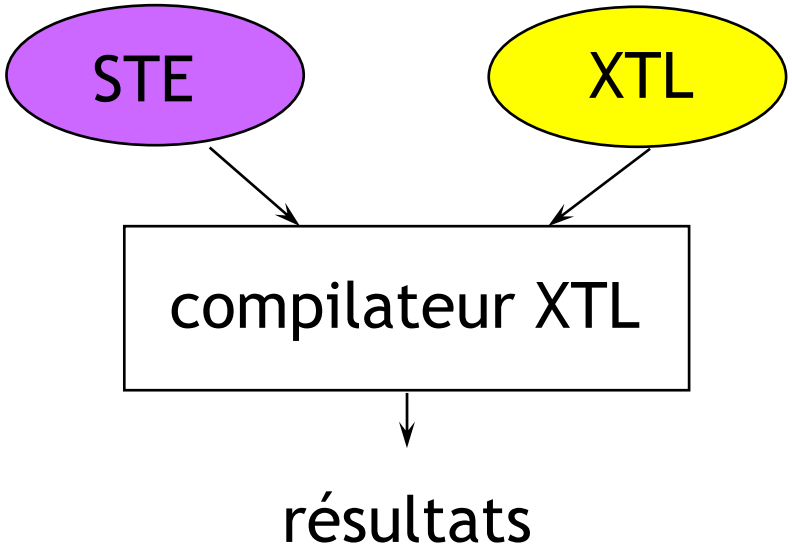
BCG_MIN : Minimisation de STEs

- Cet outil traite divers types de STEs :
  - **STE standards**
    - ✓ bisimulation forte [~Kanellakis-Smolka]
    - ✓ bisimulation de branchement [Groote-Vaandrager]
    - ✓ performances supérieures à Aldebaran et Fc2min
    - ✓ meilleur affichage des classes d'équivalence
  - **STE probabilistes** transitions « **prob** *p* »
  - **STE stochastiques** transitions « **rate** λ »
  - **modèles mixtes** transitions « *label* ; **prob** *p* » ou « *label* ; **rate** λ »
- *Travail commun avec Holger Hermanns*

Outils de vérification par logiques : XTL

XTL :

- un langage d'interrogation pour les STEs (encodés en BCG)
- un compilateur pour ce langage



XTL : principes et applications

- Caractéristiques de XTL
  - langage fonctionnel avec extensions « *model checking* »
  - types prédéfinis : **states, state sets, transitions, transition sets, labels…**
  - accès aux objets typés du graphe BCG
- Applications de XTL
  - bibliothèques: HML, CTL, ACTL, µ-calcul
  - prototypage rapide de logiques temporelles
  - logiques temporelles étendues avec valeurs

XTL: Un exemple

La modalité $\langle A \rangle F$ de HML (Hennessy-Milner logic) peut être exprimée en XTL

$\langle A \rangle F$ dénote l'ensemble des états S qui

- précèdent un état satisfaisant F
- en suivant des transitions satisfaisant A

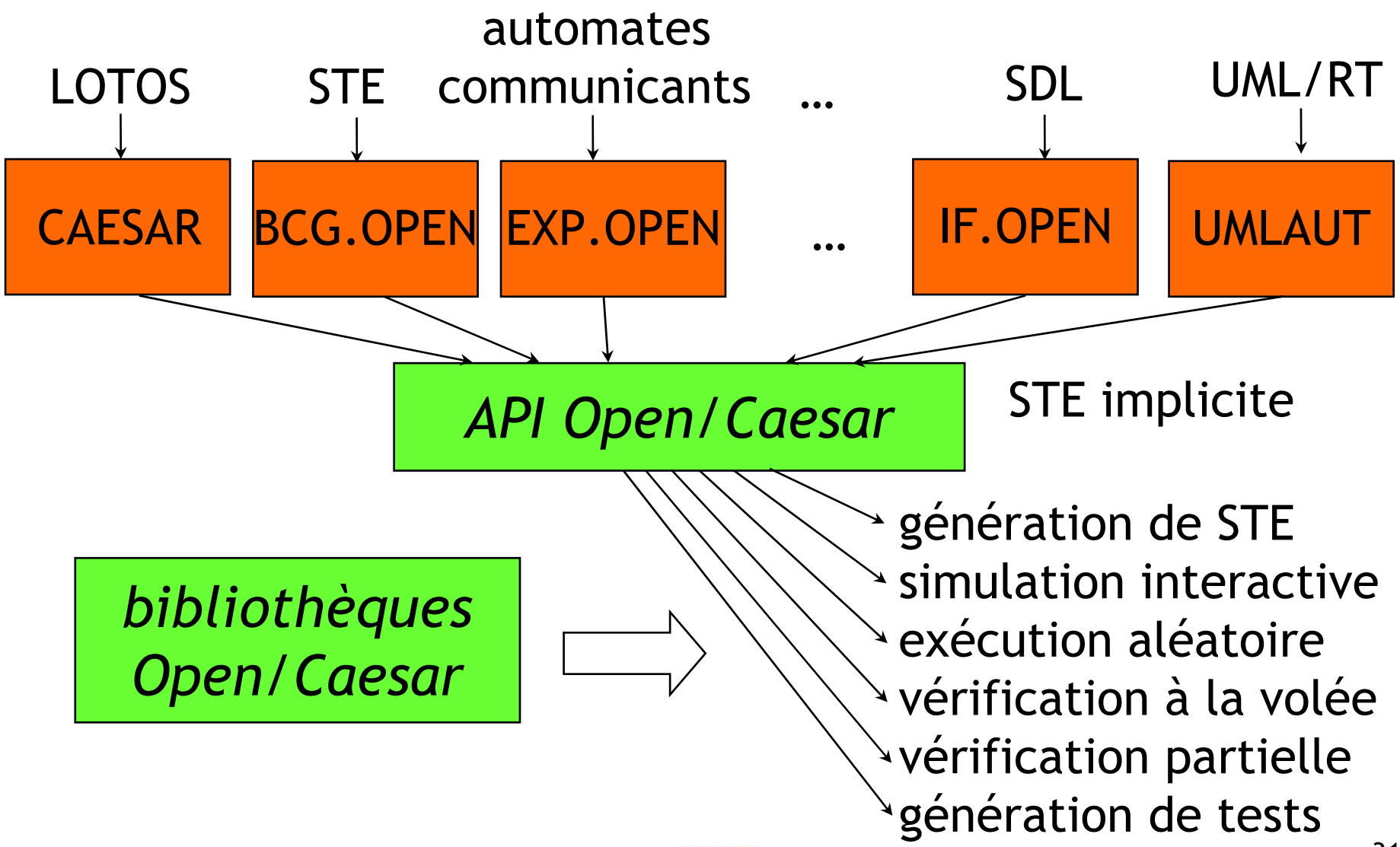
```
def Diamond (A:labelset, F:stateset):stateset =
    { S:state where
      exists T:edge among out (S) in
          (label (T) among A) and (target (T) among F)
      end_exists }
end_def
```

Outils CADP pour les STEs *implicites*

Motivations

- La plupart des « *model checkers* » sont dédiés à un langage d'entrée particulier (Spin, SMV…)
- Ils sont difficiles à réutiliser pour d'autres langages
- Idée : introduire de la modularité en séparant
  - **les aspects dépendants du langage source :** compilation du langage source vers un STE
  - **les aspects indépendants du langage source :** algorithmes pour l'exploration des STEs

OPEN/CAESAR


LOTOS
STE
automates communicants
...
SDL
UML/RT
CAESAR
BCG.OPEN
EXP.OPEN
...
IF.OPEN
UMLAUT
API Open/Caesar
STE implicite
bibliothèques Open/Caesar
génération de STE
simulation interactive
exécution aléatoire
vérification à la volée
vérification partielle
génération de tests

Bibliothèques OPEN/CAESAR

Des structures de données prédéfinies

- EDGE : listes de transitions (ex.: listes de successeurs)
- HASH : catalogue de fonctions de hash-code
- STACK_1 : piles d'états et/ou d'étiquettes
- DIAGNOSTIC_1 : ensemble de traces d'exécution
- TABLE_1 : tables d'états
- BITMAP : tables « *bit state* » de Holzmann

Des primitives dédiées à la vérification à la volée

- possibilité d'attacher des informations aux états
- débordement de pile ou de table ⇒ « *backtracking* »
- etc.

Outils OPEN/CAESAR

- EXECUTOR : exécution aléatoire
- SIMULATOR : simulation interactive (textuel)
- XSIMULATOR : simulation interactive (graphique)
- GENERATOR : génération exhaustive de STE
- REDUCTOR : génération de STE avec réduction
- PROJECTOR : génération de STE sous contrainte
- TERMINATOR : algorithme «*bit space*» de Holzmann
- EXHIBITOR : recherche de chemins selon exp. rég.
- EVALUATOR : évaluation de formules de μ-calcul
- TGV : génération de séquences de tests

Un exemple : GENERATOR

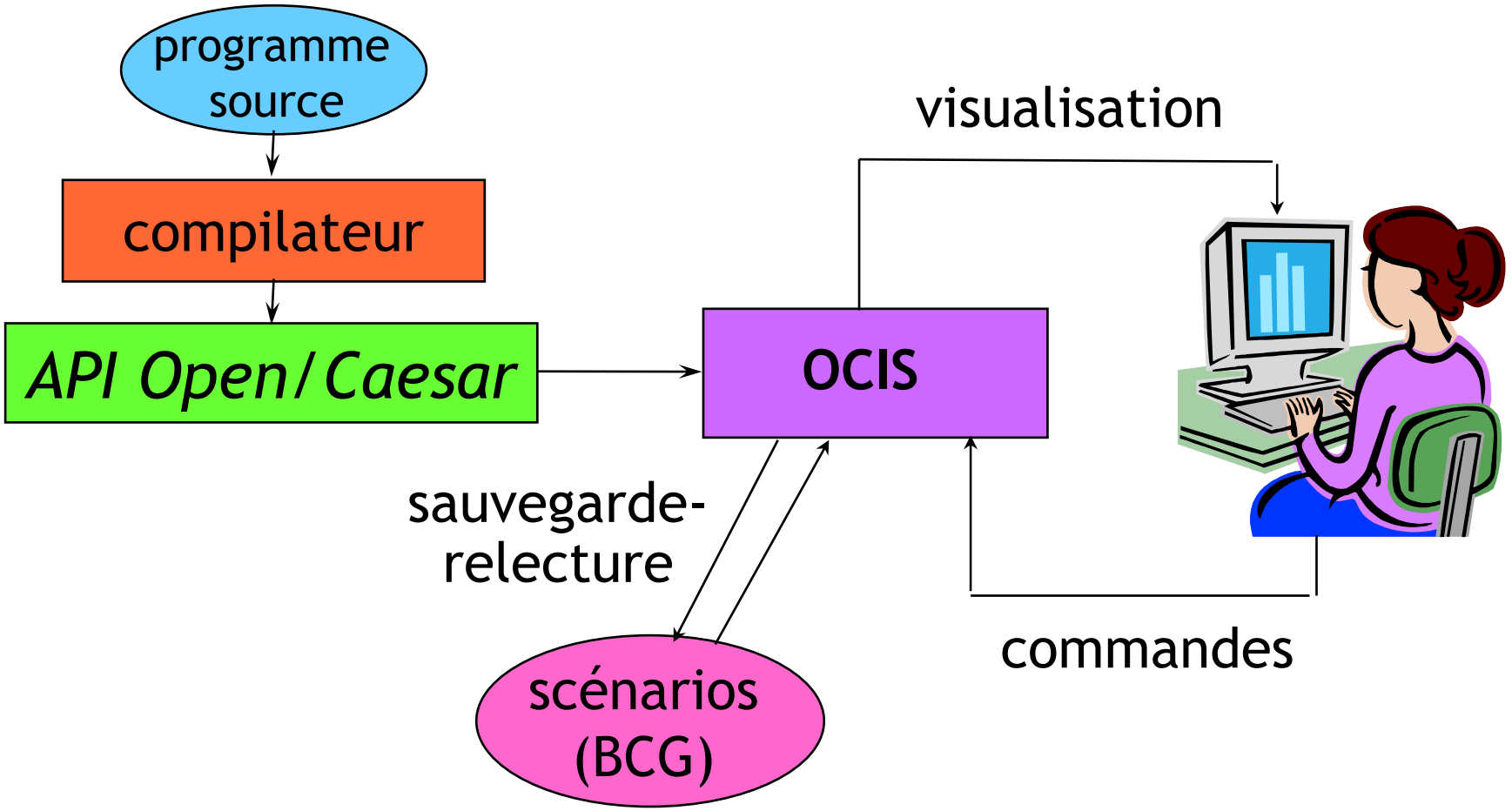
```
#include "caesar_graph.h"
#include "caesar_edge.h"
#include "caesar_table_1.h"

TYPE_TABLE_1 t;     TYPE_STATE s1, s2;                TYPE_EDGE e1_en, e;
TYPE_LABEL l;       TYPE_INDEX_TABLE_1 n1, n2         TYPE_POINTER dummy;

INIT_GRAPH ();
INIT_EDGE (FALSE, TRUE, TRUE, 0, 0);
CREATE_TABLE_1 (&t, 0, 0, 0, 0, TRUE, NULL, NULL, NULL, NULL);
if (t == NULL) ERROR ("not enough memory for table");

START_STATE ((TYPE_STATE) PUT_BASE_TABLE_1 (t));
PUT_TABLE_1 (t);
while (!EXPLORED_TABLE_1 (t)) {
    s1 = (TYPE_STATE) GET_BASE_TABLE_1 (t);
    n1 = GET_INDEX_TABLE_1 (t);
    GET_TABLE_1 (t);

    CREATE_EDGE_LIST (s1, &e1_en, 1);
    if (TRUNCATION_EDGE_LIST () != 0) ERROR ("not enough memory for edge lists");

    ITERATE_LN_EDGE_LIST (e1_en, e, l, s2) {
        COPY_STATE ((TYPE_STATE) PUT_BASE_TABLE_1 (t), s2);
        (void) SEARCH_AND_PUT_TABLE_1 (t, &n2, &dummy);
        print_edge (n1, STRING_LABEL (l), n2);
    }
    DELETE_EDGE_LIST (&e1_en);
}
```

Trois outils OPEN/CAESAR récents

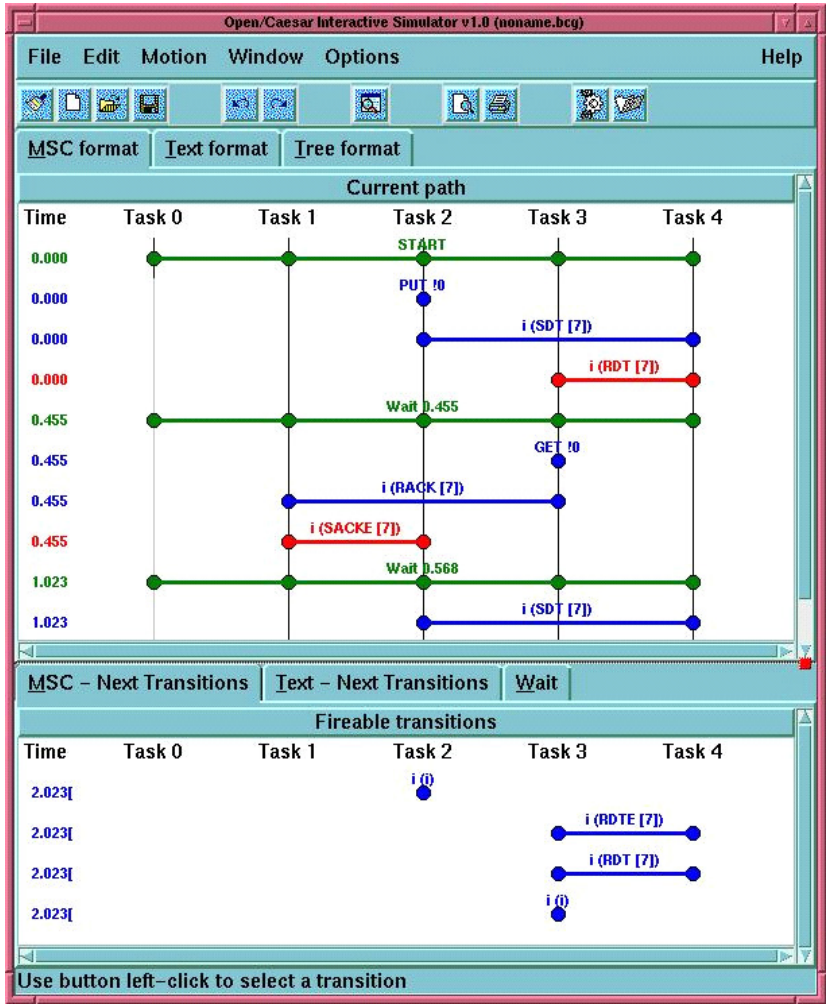
Trois domaines d'application différents :

- Simulation :
  => OCIS (*Open/Caesar Interactive Simulator*)
- Vérification par formules logiques :
  - => EVALUATOR 3.0
- Génération de tests :
  => TGV

OCIS *(Open/Caesar Interactive Simulator)*



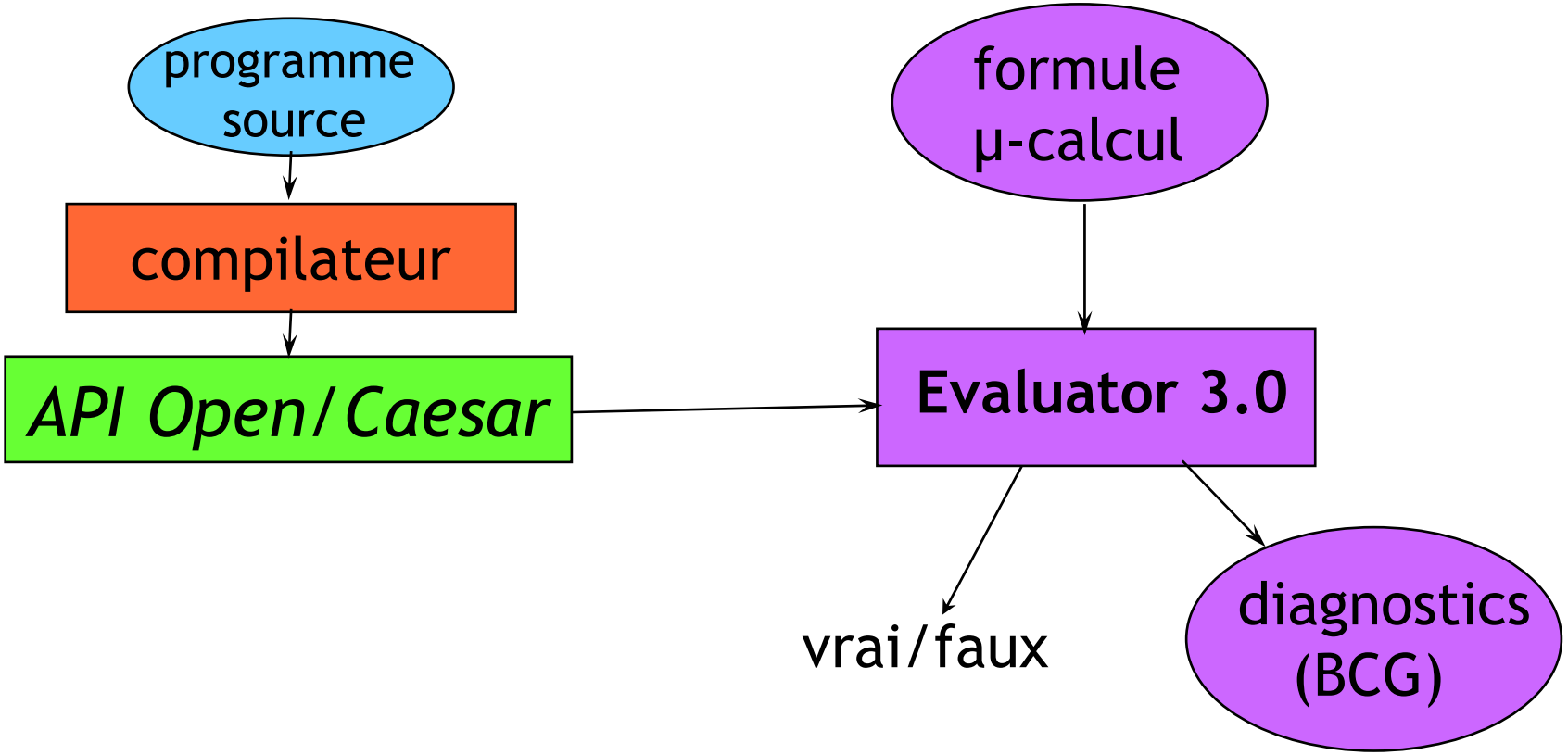
OCIS *(Open/Caesar Interactive Simulator)*



- indépendant du langage
- scénarios arborescents
- sauvegarde/relecture de scénarios
- remontée au code source
- recompilation dynamique
- support pour tâches et MSCs

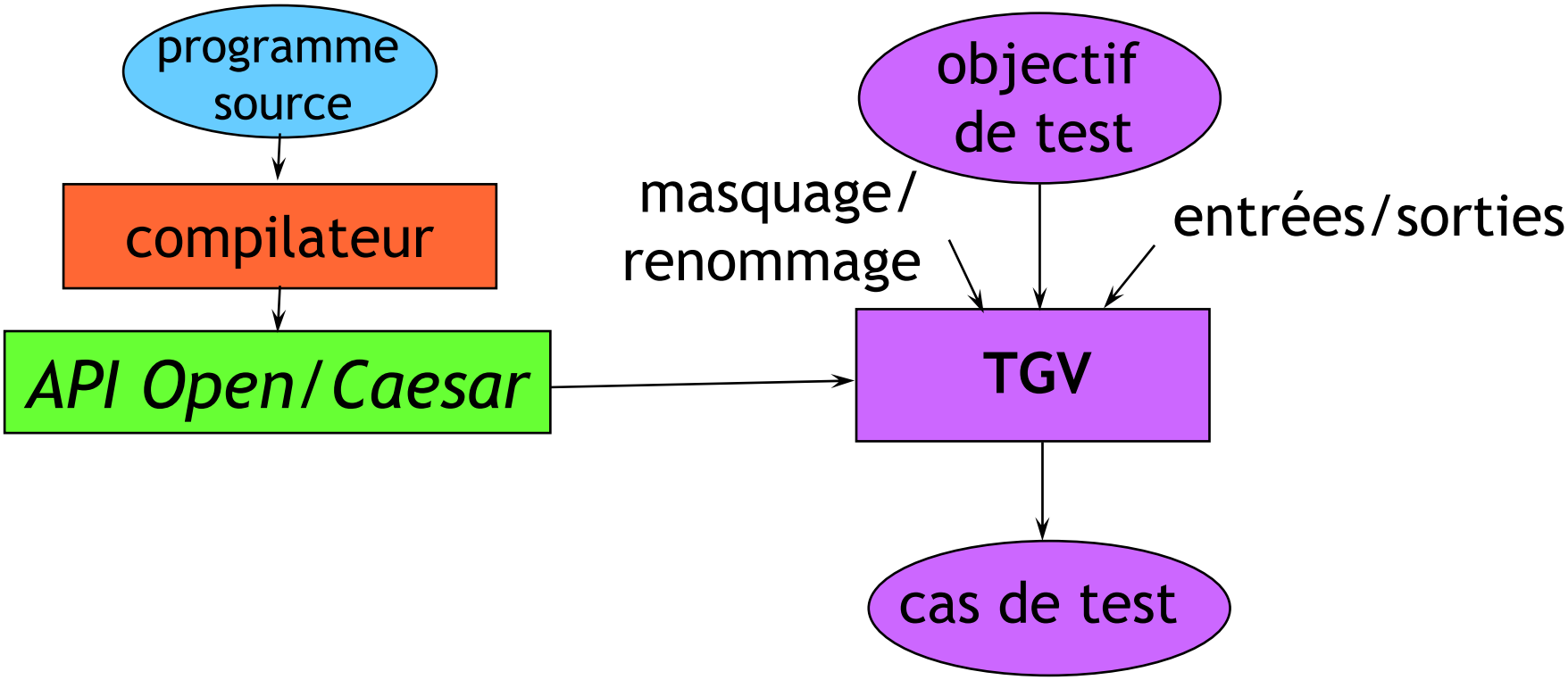
Evaluator 3.0

« *Model checking* » à la volée de formules de µ-calcul régulier sans alternance



TGV

Génération à la volée de cas de test définis par des objectifs de tests écrits manuellement



travail commun entre l'IRISA et Verimag *[et VASY]*

Outils CADP pour la vérification compositionnelle

Vérification compositionnelle

- Un moyen significatif de lutte contre l'explosion d'états
- Principe :
  - Génération de processus séparés
  - Minimisation des processus
  - Recombinaison des processus minimisés
- CADP fournit de nombreux outils pour la vérification compositionnelle

Le langage SVL

- SVL : un langage de scripts fourni avec CADP
- Deux motivations :
  - Fournir une interface textuelle pour tous les outils de CADP (+ Fc2Tools)
  - Permettre l'écriture aisée de scénarios de vérification compositionnelle
- Public visé :
  - Utilisateurs novices (vérifications simples)
  - Utilisateurs experts (vérifications sophistiquées, notamment compositionnelles)

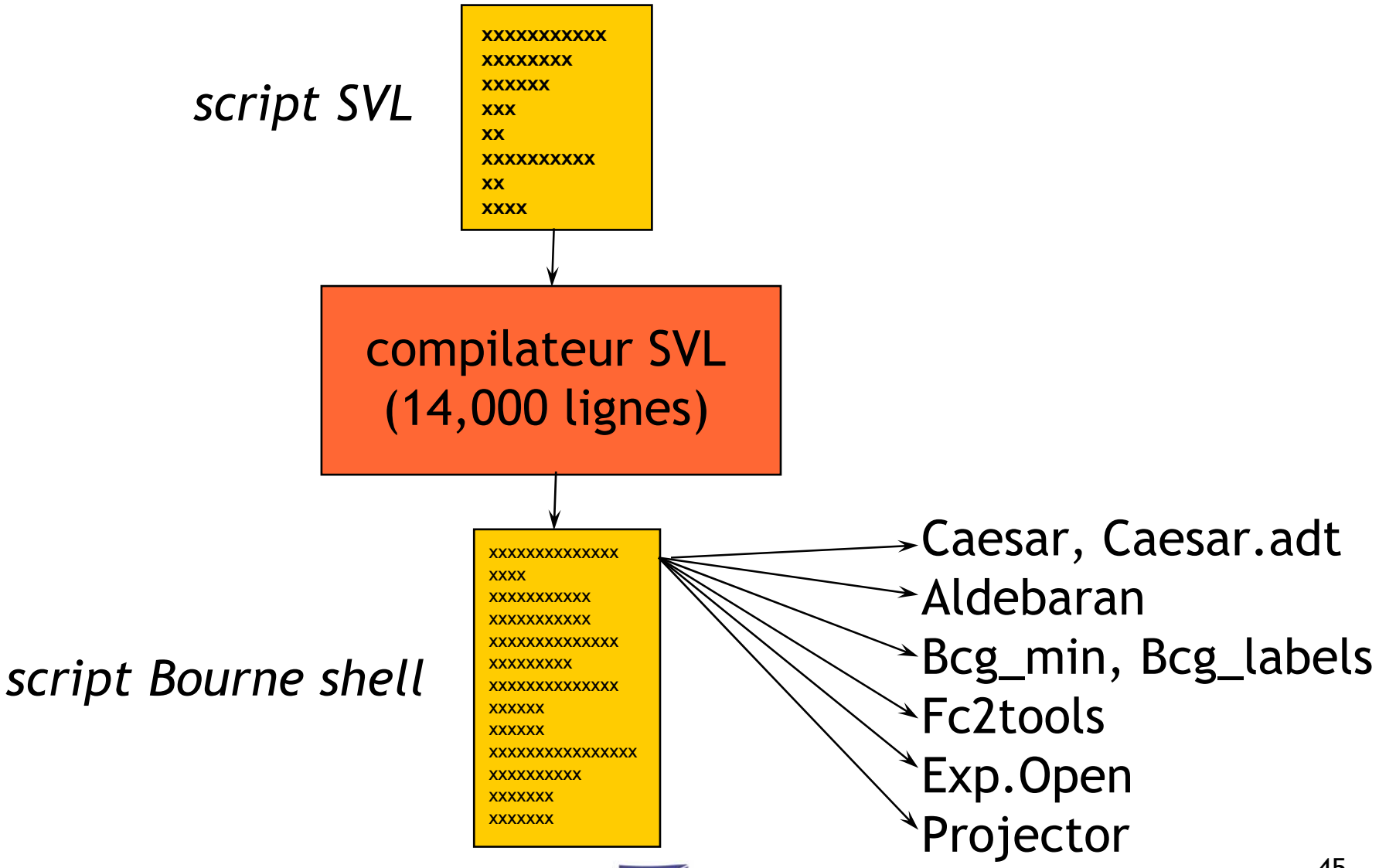
Script SVL : exemple 1

```
% DEFAULT_LOTOS_FILE="bitalt_protocol.lotos"
"bitalt_protocol.exp" =
    leaf strong reduction of
      hide SDT, RDT, RDTe, RACK, SACK, SACKe in
          (
              (BODY_TRANSMITTER ||| BODY_RECEIVER)
              |[SDT, RDT, RDTe, RACK, SACK, SACKe]|
             (MEDIUM1 ||| MEDIUM2)
          );
"bitalt_dead.seq" = deadlock of "bitalt_protocol.exp";
"bitalt_live.seq" = livelock of "bitalt_protocol.exp";
branching comparison using fly with aldebaran
          "bitalt_protocol.exp" == "bitalt_service.lotos";
```

Script SVL : exemple 2

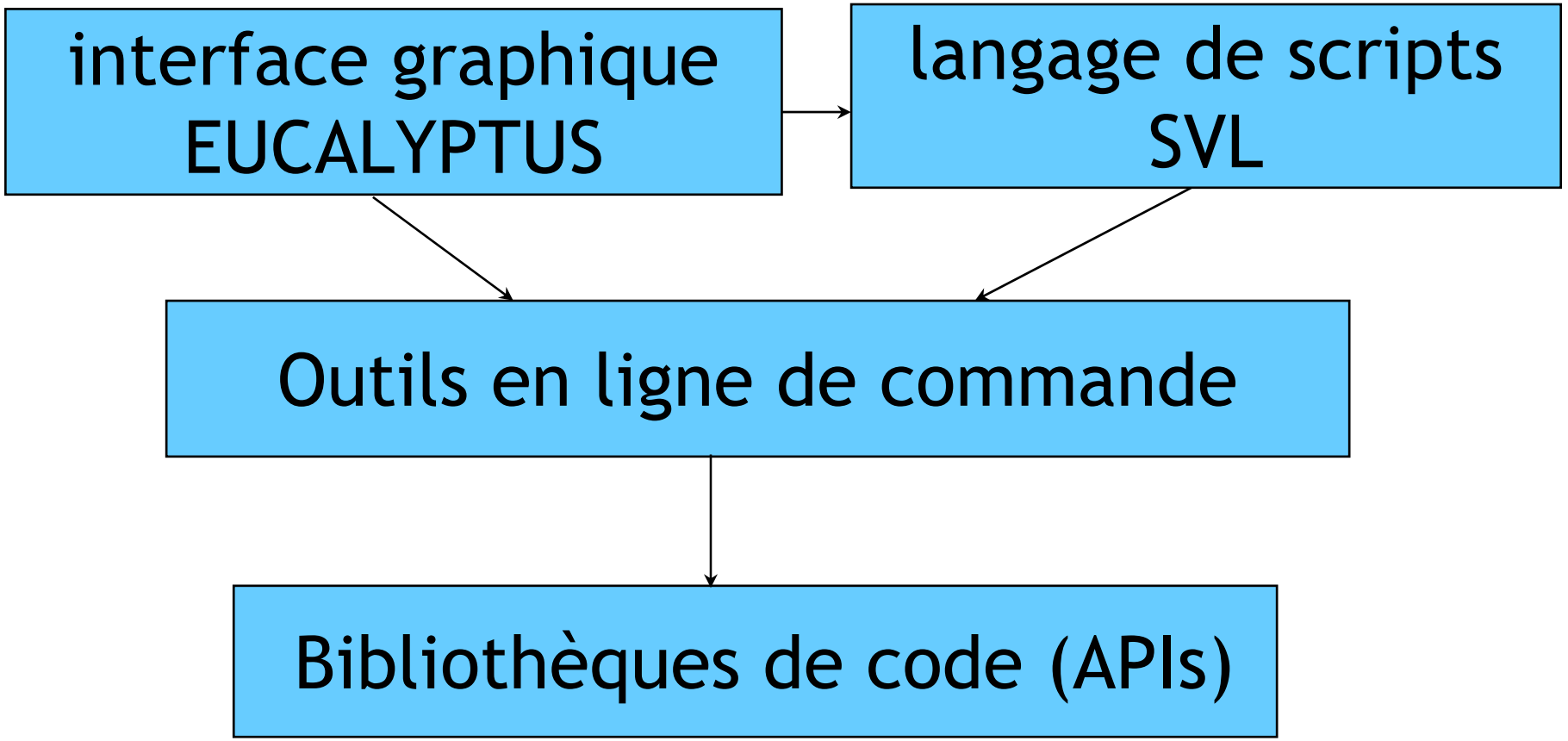
```
% DEFAULT_LOTOS_FILE="rel_rel.lotos"
"crash_trans.bcg" = strong reduction of CRASH_TRANSMITTER ;
"rel_rel.bcg" = generation of leaf strong reduction of
   hide R_T1, R_T2, R_T3, R12, R13, R21, R23, R31, R32  in
   (  (  (  (RECEIVER_NODE_1 -||? "r1_interface.lotos")
         ||[R12, R21, R13, R31]||
         (   (RECEIVER_NODE_2 -||? "r2_interface.lotos")
            ||[R23, R32]||
             (RECEIVER_NODE_3 -||? "r3_interface.lotos")
         ) -||[R_T2, R_T3]|| "crash_trans.bcg"
         ) -||[R_T1, R_T2, R_T3]|| "crash_trans.bcg"
      )
    ||[R_T1, R_T2, R_T3]||
   "crash_trans.bcg");
```

Le compilateur SVL



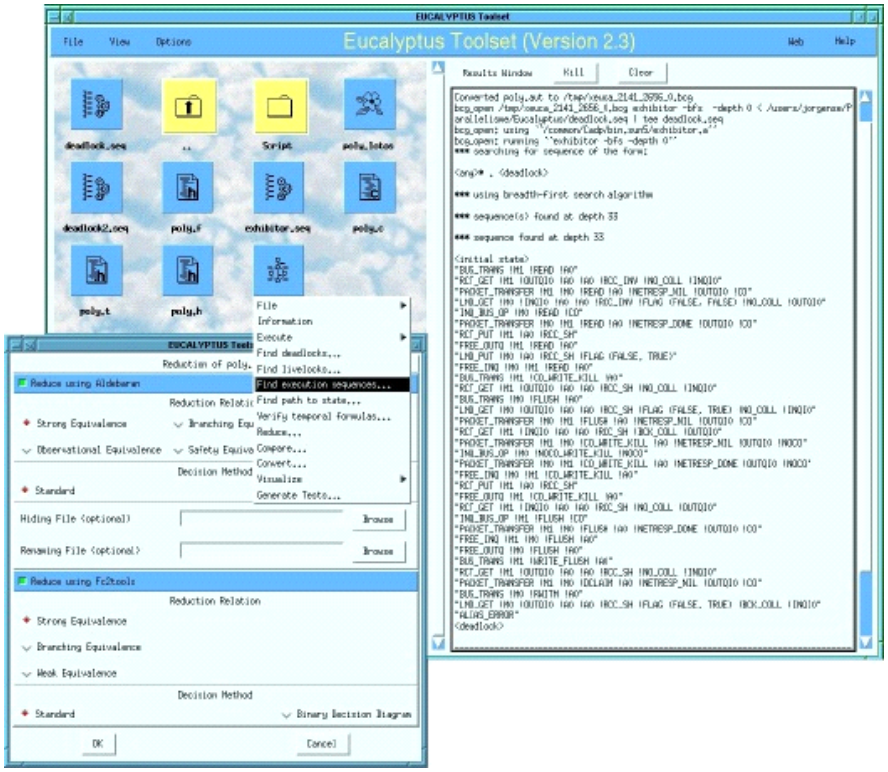
Architecture de CADP

Une architecture stratifiée



L’interface graphique EUCALYPTUS

- Typage des fichiers
- Menus contextuels
- Boîtes de dialogue
- Multi-outils : CADP, FC2
- Aide en ligne



Conclusion

Conclusion

- CADP: une riche plate-forme pour l'ingénierie des protocoles et des systèmes distribués
- Une boîte à outils ouverte et extensible via des APIs bien définies
- Plusieurs architectures supportées :
  - Sun sous SunOS ou Solaris
  - PC sous Linux
  - PC sous Windows
- Large dissémination (chiffres de 2001):
  - CADP diffusé à **256 sites**
  - licences accordées pour **950 machines** in 2001
  - **53 études de cas** effectuées avec CADP
  - **11 outils de recherche** construits avec CADP

Pour plus d'information…

http://www.inrialpes.fr/vasy/cadp