An Overview of CADP 2001

Hubert Garavel

VASY team

INRIA Rhône-Alpes

655, avenue de l'Europe

38330 Montbonnot Saint Martin

CADP

- *CAESAR/ALDEBARAN Development Package*
- A toolbox for protocol and distributed systems engineering
- Main features:
  - modelling using process algebras (LOTOS)
  - equivalence checking (bisimulations)
  - model checking (modal mu-calculus)
  - exhaustive, partial, on the fly, compositional verification
  - C code generation, rapid prototyping
  - step by step simulation, random execution
  - test generation

Origins of CADP

- Work initiated in 1986
- Joint work between
  - the VASY team of INRIA
  - the Verimag laboratory

  with contributions of
  - the PAMPA team of IRISA
  - the FMT group at the University of Twente

Main applications of CADP

- Industrial case-studies
  - hardware, software, telecom, embeded systems…
  - formal specification of critical systems and protocols
  - simulation, rapid prototyping, verification, testing
- Research
  - analysis of new systems/protocols
  - experimentation of new verification/testing algorithms
  - implementation of new modelling languages
- Education
  - concurrency, process algebras, bisimulations, model checking
  - robust tools for lab exercises and student projects

Outline

- LOTOS and the Labelled Transition Systems model (LTSs)
- Tools for LOTOS
- Tools for explicit LTSs
- Tools for implicit LTSs
- Tools for compositional verification
- CADP architecture
- Conclusion

LOTOS and the Labelled Transition Systems (LTS) model

LOTOS

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

- A Formal Description Technique for the specification of protocols and distributed systems
- Two orthogonal sub-languages:

  Data: abstract data types (ActOne)

  - sorts and operations
  - algebraic equations

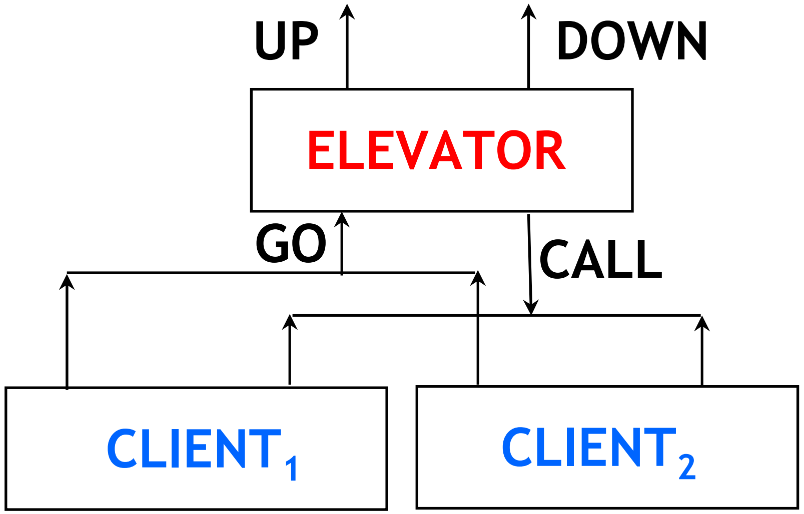
  Processes: process algebras (~CCS, CSP, Circal)

  - parallel processes (interleaving semantics)
  - message-passing communication

LOTOS types: An example

```
type FLOOR is  BOOLEAN
  sorts
    FLR

  opns
    LOWER (*! constructor *),
    MIDDLE (*! constructor *),
    UPPER (*! constructor *),
    ERROR (*! constructor *) :-> FLR
    INCR, DECR : FLR -> FLR
    _==_ , _<_ , _>_ : FLR, FLR -> BOOL

 eqns
   forall X, Y:FLR
   ofsort FLR
     INCR (LOWER) = MIDDLE;
     INCR (MIDDLE) = UPPER;
     (* else *) INCR (X) = ERROR;

   ofsort FLR
     DECR (MIDDLE) = LOWER;
     DECR (UPPER) = MIDDLE;
     (* else *) DECR (X) = ERROR;

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

   ofsort BOOL
     LOWER < MIDDLE = true;
     LOWER < UPPER = true;
     MIDDLE < UPPER = true;
     (* else *) X < Y = false;

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

LOTOS processes: An example



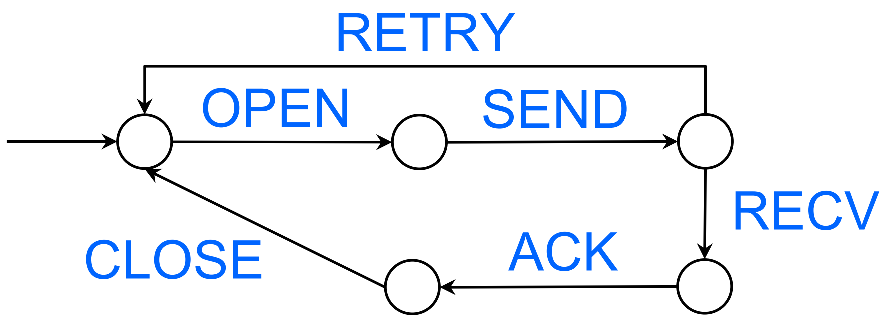
ELEVATOR [CALL, GO, UP, DOWN] (LOWER, LOWER)
|[CALL, GO]|
  (
  CLIENT [CALL, GO] (LOWER, UPPER)
  |||
  CLIENT [CALL, GO] (UPPER, MIDDLE)
)

LOTOS processes (cont'd)

```
process ELEVATOR [CALL, GO, UP, DOWN] (CURRENT, TARGET: FLR) : noexit :=
   [TARGET > CURRENT] ->
     UP !INCR (CURRENT);
       ELEVATOR [CALL, GO, UP, DOWN] (INCR (CURRENT), TARGET)
   []
   [TARGET < CURRENT] ->
     DOWN !DECR (CURRENT);
       ELEVATOR [CALL, GO, UP, DOWN] (DECR (CURRENT), TARGET)
   []
   [TARGET == CURRENT] ->
     (
     CALL ?NEW_TARGET:FLR;
       ELEVATOR [CALL, GO, UP, DOWN] (CURRENT, NEW_TARGET)
     []
     GO ?NEW_TARGET:FLR;
       ELEVATOR [CALL, GO, UP, DOWN] (CURRENT, NEW_TARGET)
     )
endproc
```

Labelled Transition Systems (LTSs)

- LTS: the standard semantic model for action-based languages (including LOTOS)



- $M = (S, A, T, s_0)$, where:
  - S: set of states
  - A: set of labels (information attached to transitions)
  - $T \in S \times A \times S$: transition relation
  - $s_0 \in S$: initial state

LTSs and verification

- LTSs provide a standard basis for many automated verification algorithms

- Examples:
  - Reachable state analysis (LTS exploration)
  - Equivalence checking (bisimulations)
  - Model checking (modal mu-calculus)

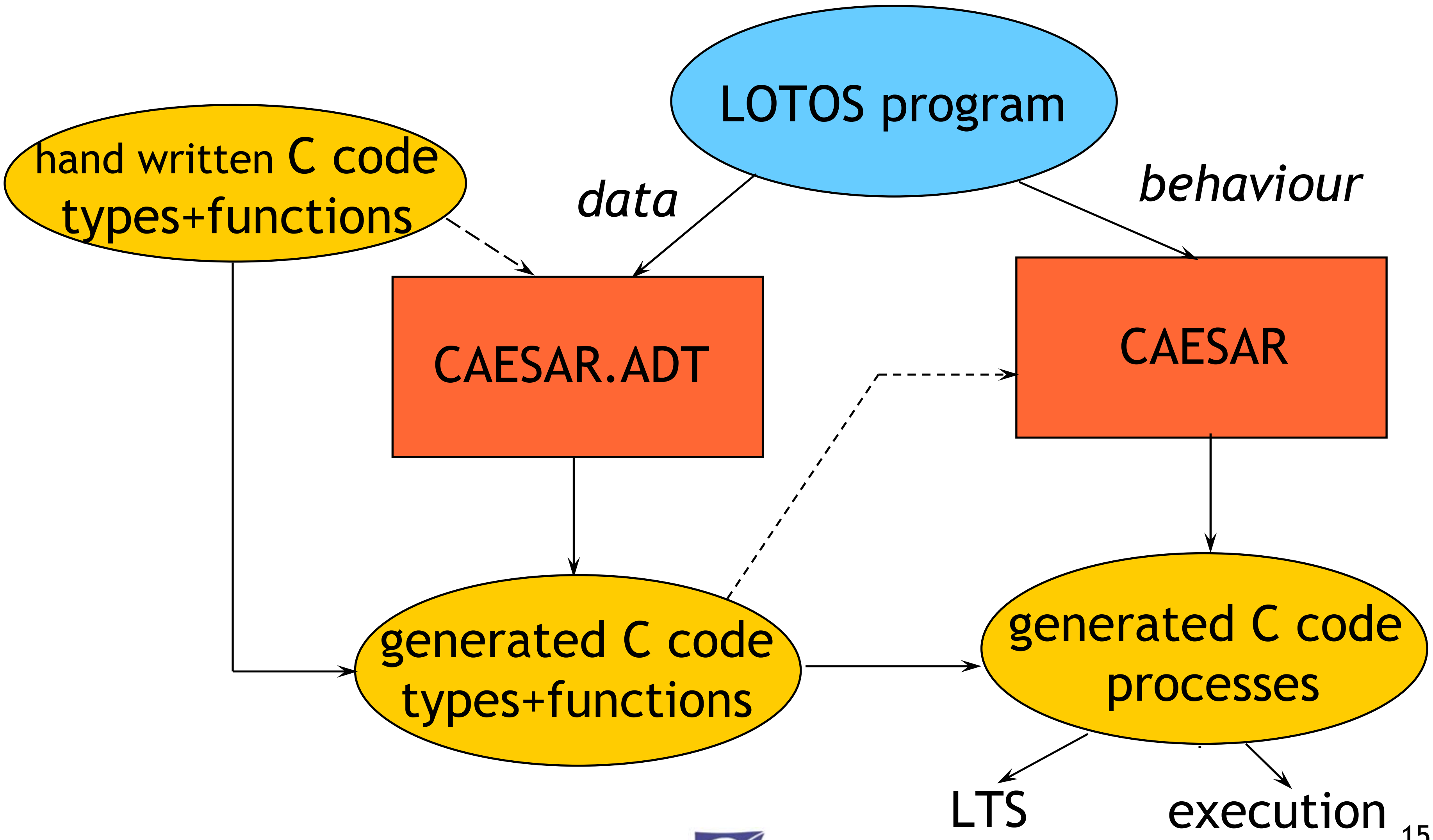
Computer representations of LTSs

- "Explicit" LTS (or LTS "in extenso"): LTS defined by the exhaustive list of its states and transitions
  - state successors and state predecessors are available: the LTS can be explored both forward and backward
  - this enables both global and local (on the fly) verification
  - CADP provides the BCG tools for explicit (finite) LTSs

- "Implicit" LTS (or LTS "in comprehension"): LTS defined by its initial state and successor function
  - state predecessors are not known: only forward exploration (local verification) is allowed
  - CADP provides the Open/Caesar tools for implicit LTSs

CADP tools for LOTOS

CAESAR.ADT and CAESAR



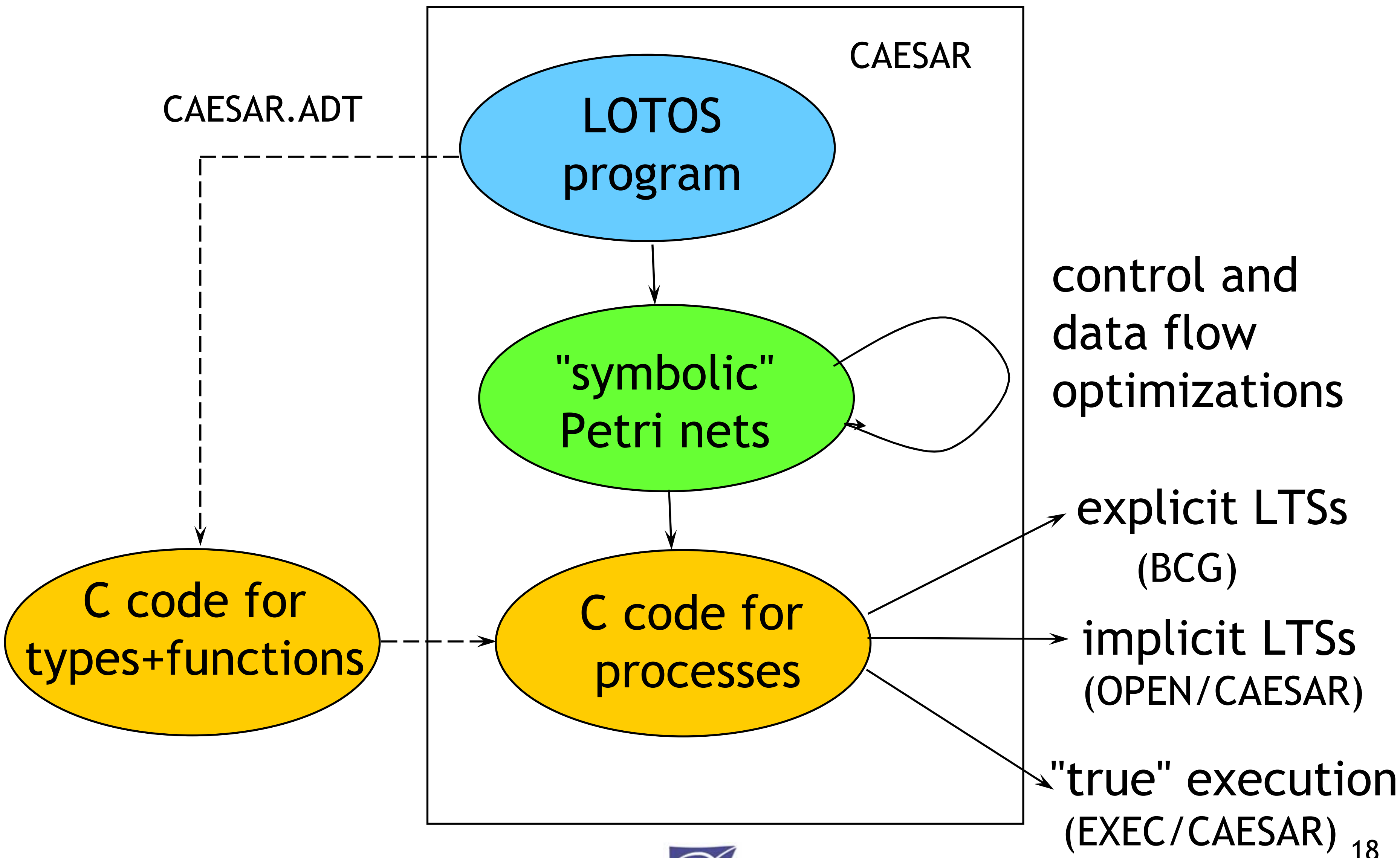
CAESAR.ADT

- **Translation LOTOS ADTs $\rightarrow$ C**
  - each LOTOS sort $\rightarrow$ one C type
  - each LOTOS operation $\rightarrow$ one C function
- **Assumptions wrt standard LOTOS**
  - difference between constructors and non-constructors
  - free constructors
  - equations seen as rewrite rules with pattern-matching and priorities
- **Specialized C code generation**
  - Oriented towards model checking
  - *Optimize memory first, then speed*

CAESAR.ADT (cont'd)

- **Compiling data strutures**
  - dynamic data structures (lists, trees, ...) allowed
  - optimized memory implementation:
    - minimal number of bits
    - permutation of "record" fields
    - common subterm sharing
- **Compiling functions**
  - pattern matching compiling algorithm
  - ad hoc optimisations

CAESAR



CAESAR
CAESAR.ADT
LOTOS program
"symbolic" Petri nets
C code for types+functions
C code for processes
control and data flow optimizations
explicit LTSs (BCG)
implicit LTSs (OPEN/CAESAR)
"true" execution (EXEC/CAESAR)

CADP tools for *explicit* LTSs

Motivations

- How to store **large** LTSs in computer files?
- Existing text-based formats are not satisfactory:
  - disk space consuming (hundreds of Mbytes, Gbytes)
  - slow (read/write operations are costly)
  - sometimes ambiguous

The BCG format of CADP

BCG (*Binary-Coded Graphs*):

- **a compact file format** for storing LTSs
- **a set of APIs**
- **a set of software libraries**
- **a set of tools (binary programs and scripts)**

implementation : 30,000 lines of C code

BCG is shipped as a component of CADP

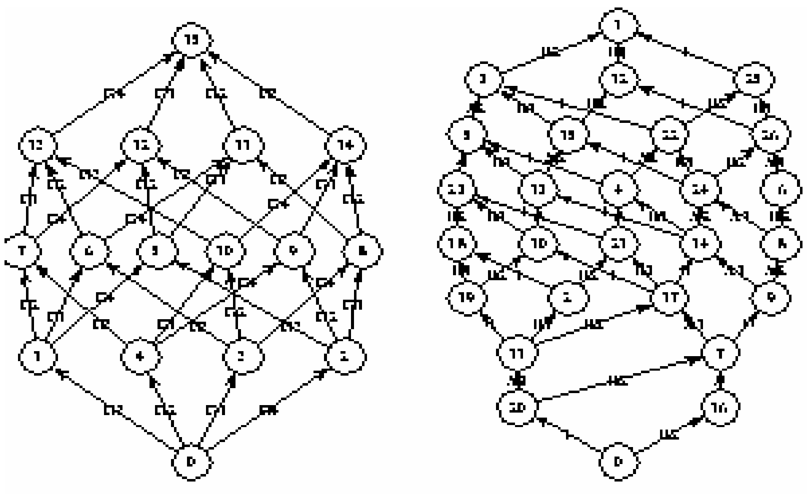
All the CADP tools use BCG consistently

The BCG libraries and APIs

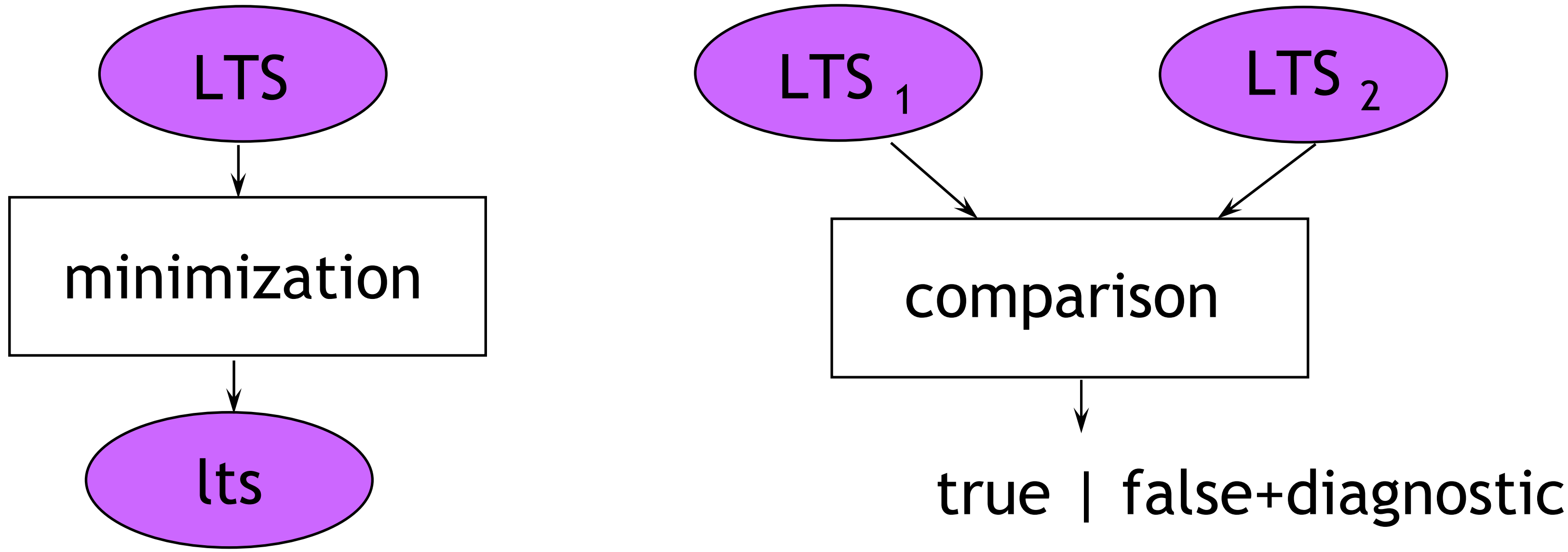
- **bcg_write:** API to create a BCG file
- **bcg_read:** API to read a BCG file
- **bcg_transition:** API to store a transition relation in memory:
  - successor function, or
  - predecessor function, or
  - successor and predecessor functions

The basic BCG tools

- **bcg_info:** extract info from a BCG file
- **bcg_io:** convert BCG ↔ other formats
- **bcg_labels:** hide and/or rename labels
- **bcg_draw, bcg_edit:** visualize LTSs



Equivalence checking tools



- CADP supports 3 such tools:
  - ALDEBARAN (Verimag)
  - FC2TOOLS (INRIA/Meije) – interfaced with CADP
  - BCG_MIN (INRIA/VASY) – the most recent
- Various equivalences supported: strong, observational, branching, safety…

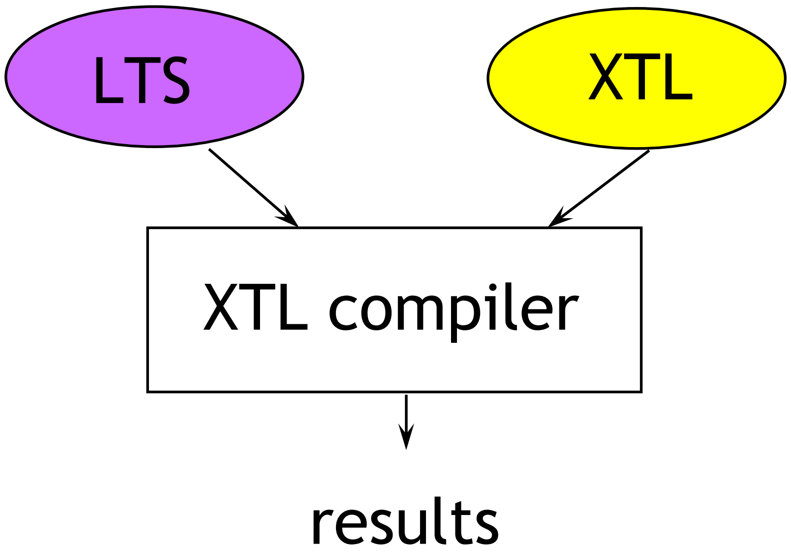
BCG_MIN: Minimization of LTSs

- This tool handles several types of LTSs:
  - standard LTSs
    - ✓strong bisimulation [~Kanellakis-Smolka]
    - ✓branching bisimulation [Groote-Vaandrager]
    - ✓better performances than Aldebaran and Fc2min
    - ✓better display of equivalence classes
  - probabilistic LTSs "**prob** p" transitions
  - stochastic LTSs "**rate** λ" transitions
  - mixed models "*label* ; **prob** p" or "*label* ; **rate** λ" transitions
- *Joint work with Holger Hermanns*

Model checking tools: XTL

XTL:

- a query language for LTSs (encoded in BCG)
- a compiler for this language



XTL: Principles and applications

- Main features of XTL
  - functional language with model checking features
  - special types: **states, state sets, transitions, transition sets, labels…**
  - access to the typed objects of the BCG file
- Applications of XTL
  - libraries: HML, CTL, ACTL, mu-calculus
  - rapid prototyping of temporal logics
  - temporal logics extended with value passing

XTL: An example

The $\langle A \rangle F$ modality of HML (Hennessy-Milner logic) can be expressed in XTL

$\langle A \rangle F$ denotes the set of states S that

- lead to states satisfying F
- following transitions satisfying A

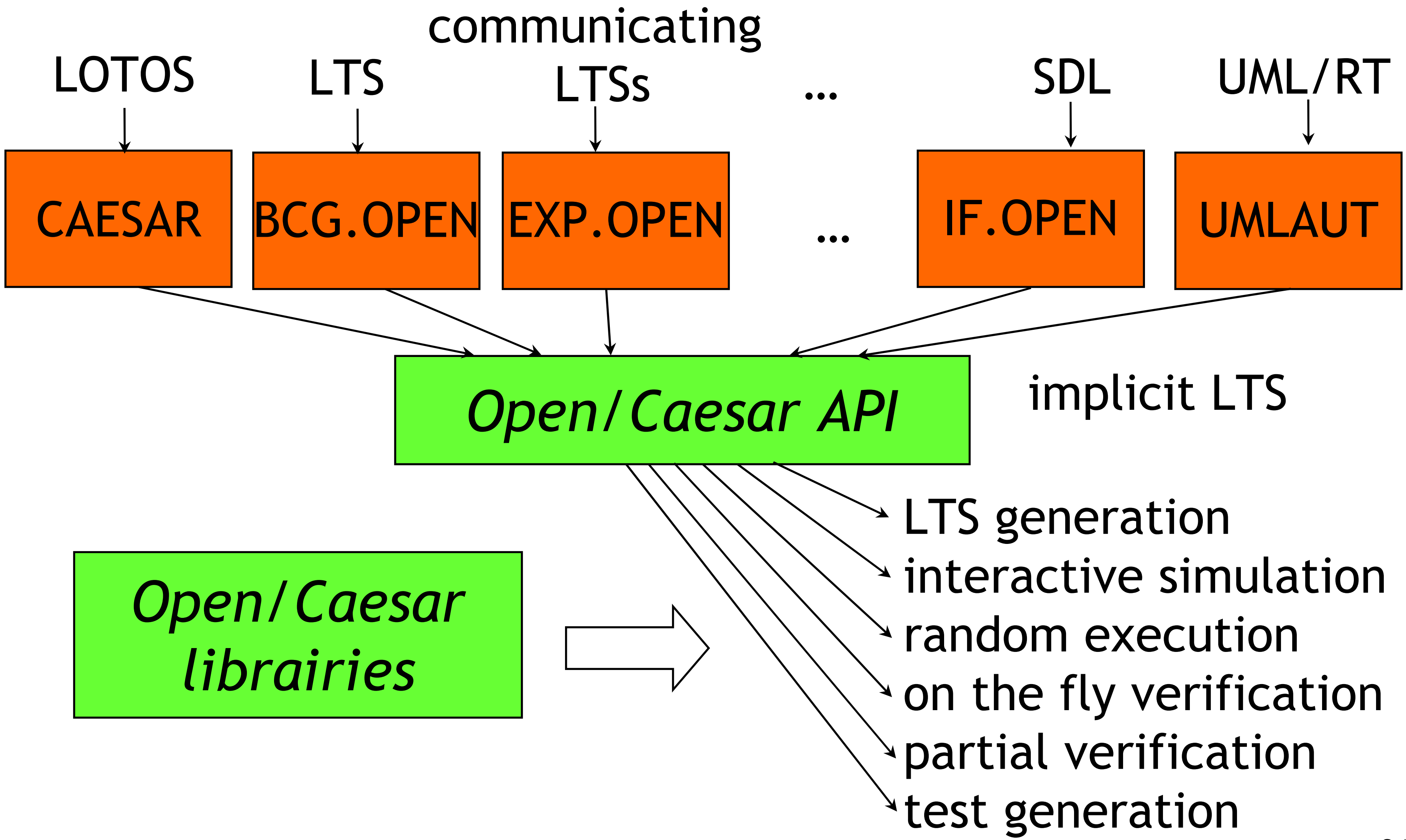
```
def Diamond (A:labelset, F:stateset):stateset =
    { S:state where
      exists T:edge among out (S) in
         (label (T) among A) and (target (T) among F)
      end_exists }
end_def
```

CADP tools for *implicit* LTSs

Motivations

- *Most model checkers are dedicated to one particular input language (Spin, SMV, …)*
- *They can't be reused easily for other languages*
- Idea: introduce modularity by separating
  - **language-dependent aspects:** compiling language into LTS model
  - **language-independent algorithms**: algorithms for LTS exploration

OPEN/CAESAR


LOTOS
LTS
communicating LTSs
...
SDL
UML/RT
CAESAR
BCG.OPEN
EXP.OPEN
...
IF.OPEN
UMLAUT
Open/Caesar API
implicit LTS
LTS generation
interactive simulation
random execution
on the fly verification
partial verification
test generation
Open/Caesar librairies

OPEN/CAESAR libraries

A set of predefined data structures

- EDGE : list of transitions (e.g., successor lists)
- HASH : catalog of hash functions
- STACK_1 : stacks of states and/or labels
- DIAGNOSTIC_1 : set of execution paths
- TABLE_1 : state tables
- BITMAP : Holzmann's "bit state" tables

Specific primitives for on the fly verification

- possibility to attach additional information to states
- stack or table overflow => backtracking
- etc.

OPEN/CAESAR applications

- EXECUTOR : random walk
- SIMULATOR : interactive simulation (textual)
- XSIMULATOR : interactive simulation (graphical)
- GENERATOR : exhaustive LTS generation
- REDUCTOR : LTS generation with safety reduction
- PROJECTOR : LTS generation with constraints
- TERMINATOR : Holzmann's bit-space algorithm
- EXHIBITOR : search paths defined by reg. expr.
- EVALUATOR : evaluation of mu-calculus formulas
- TGV : test sequence generation

An example: 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);
}
```

Three recent OPEN/CAESAR tools

Three different application areas:

- Simulation:

  => OCIS (*Open/Caesar Interactive Simulator*)

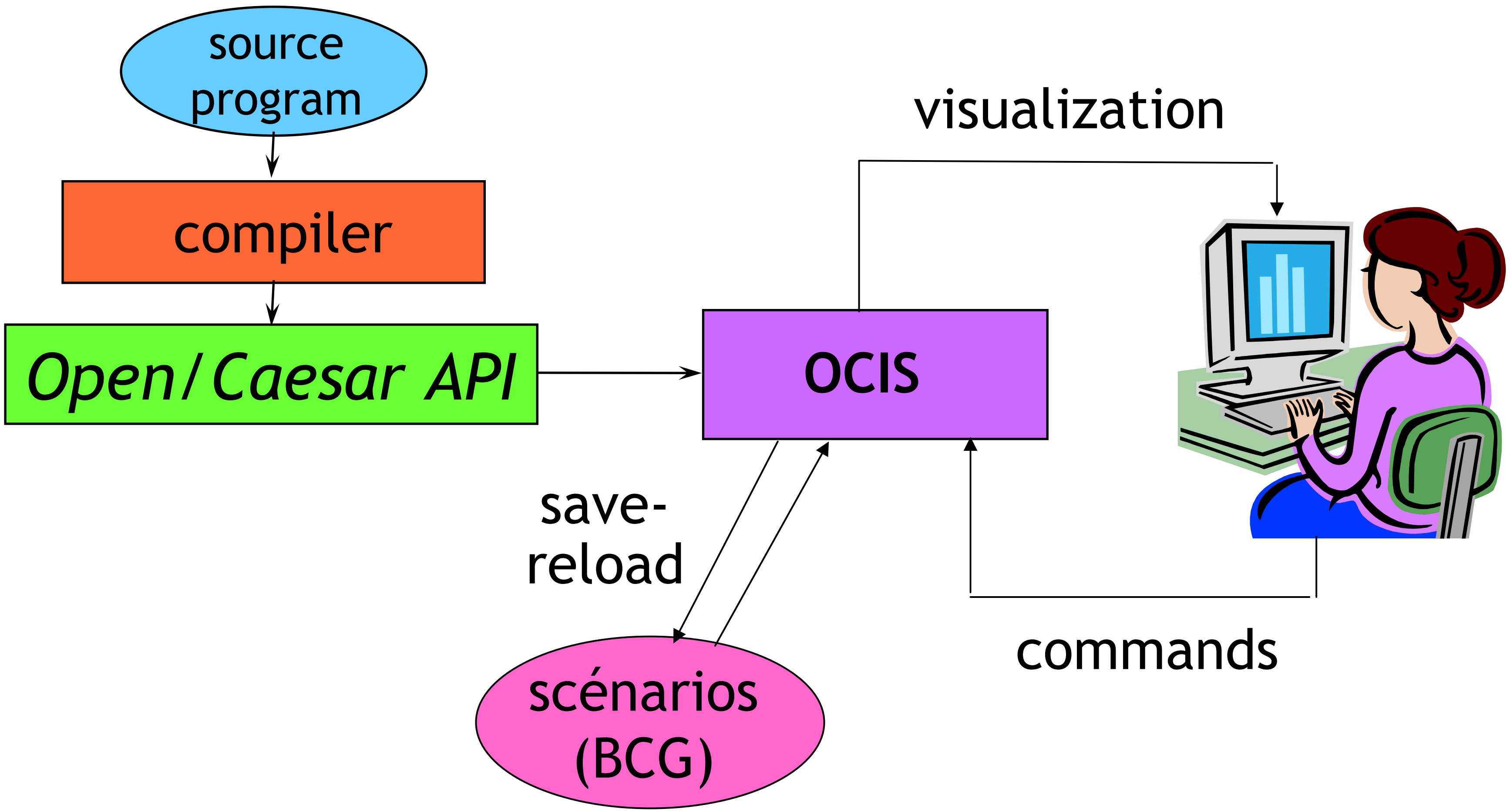
- Model checking:

  => EVALUATOR 3.0

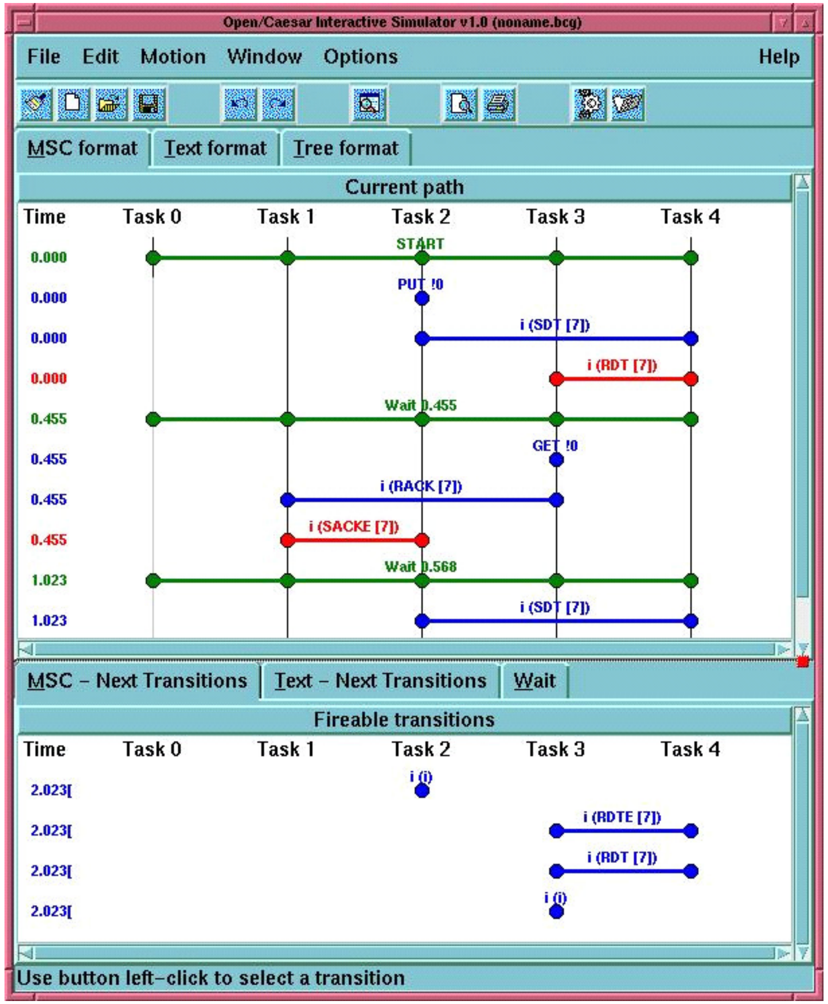
- Test generation:

  => TGV

OCIS *(Open/Caesar Interactive Simulator)*



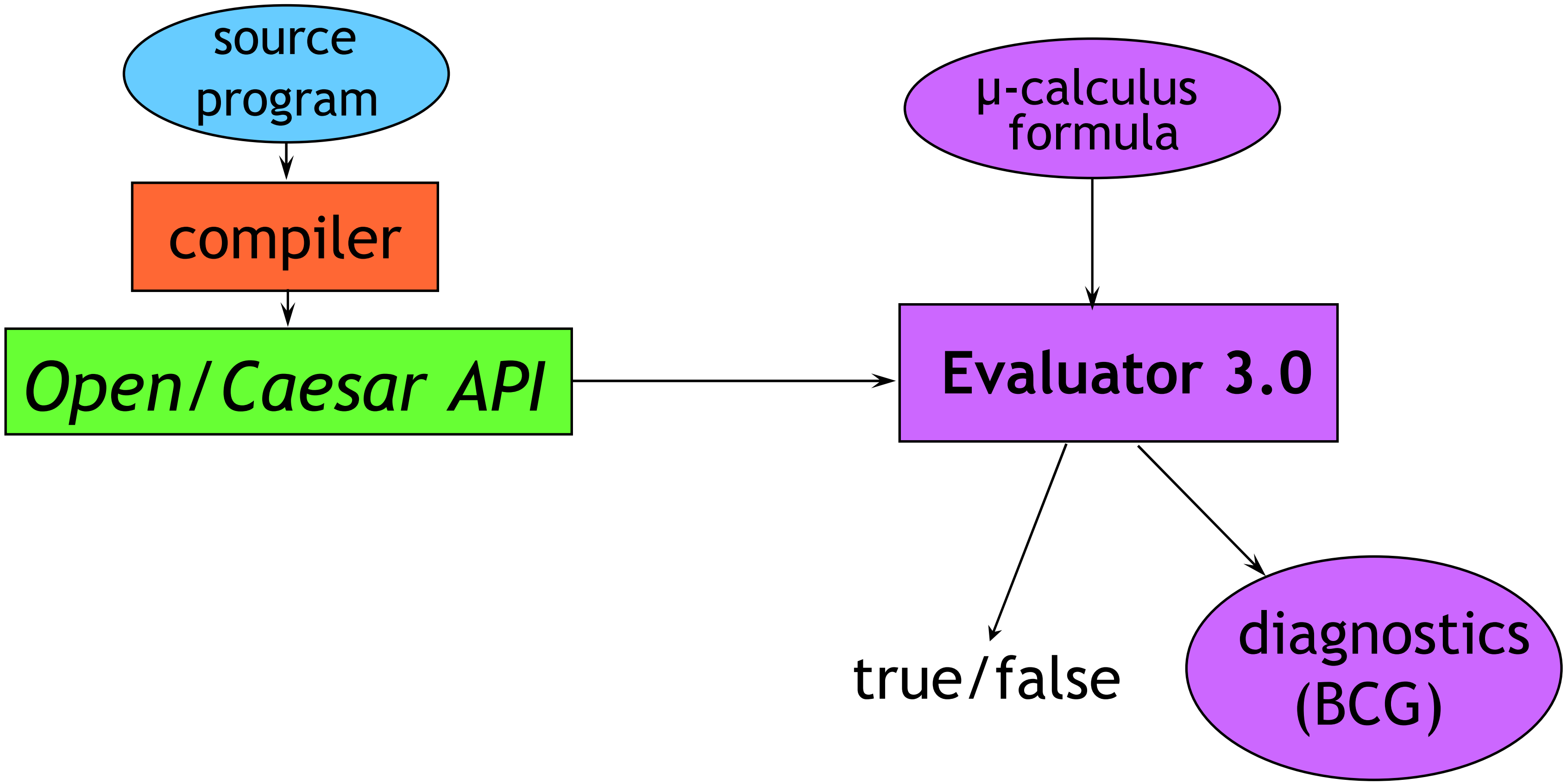
OCIS *(Open/Caesar Interactive Simulator)*



- language-independent
- tree-like scenarios
- save/load scenarios
- source code access
- dynamic recompile
- support for tasks and MSCs

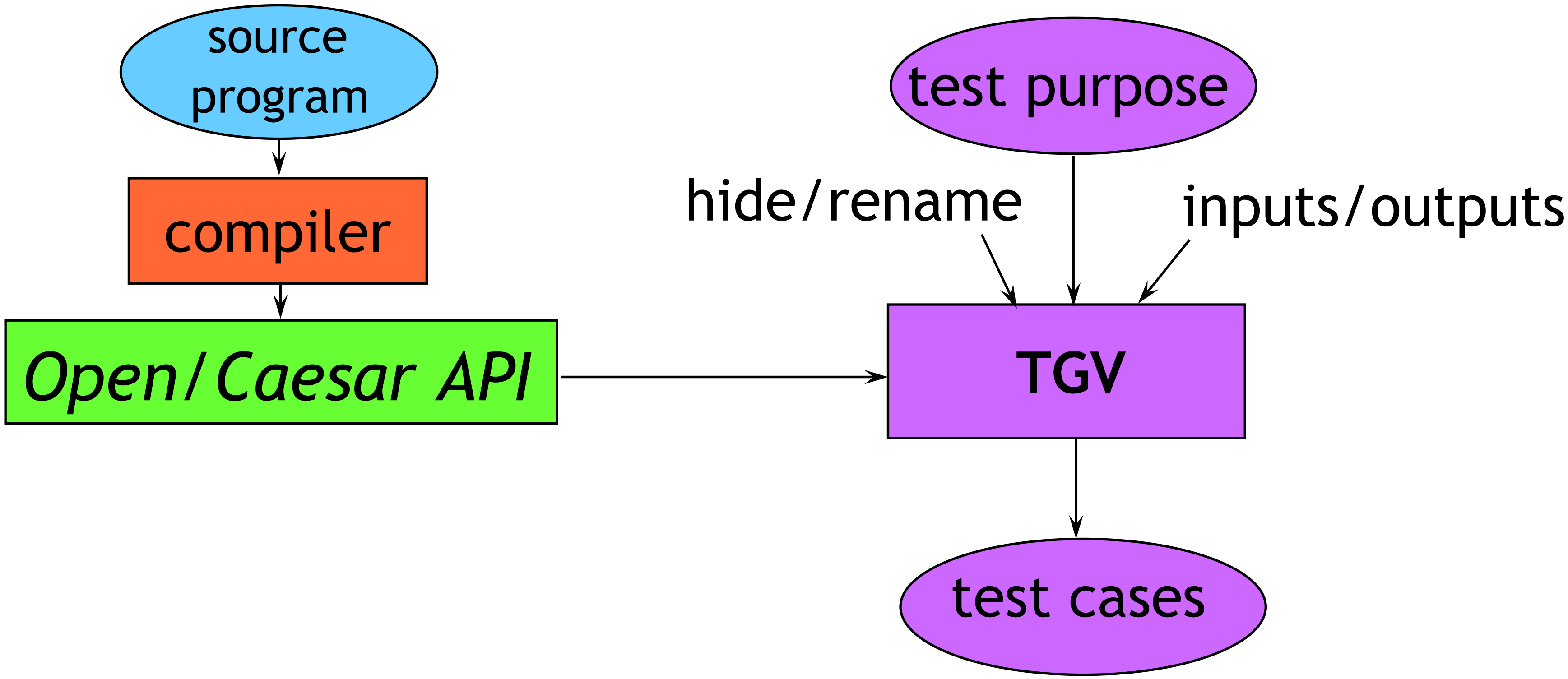
Evaluator 3.0

On-the-fly model checking of
regular alternation-free mu-calculus



TGV

On-the-fly generation of test cases
according to hand-written test purposes



joint work between IRISA and Verimag *[and VASY]*

CADP tools for compositional verification

Compositional verification

- A significant mean to fight state explosion
- Principle:
  - Generation of separate processes
  - Minimization of processes
  - Recombination of minimized processes
- CADP provides numerous tools for compositional verification

The SVL language

- SVL: a scripting language supplied with CADP
- Two motivations:
  - Provide a textual interface for all the tools of CADP (+ Fc2Tools)
  - Enable an easy writing of compositional verification scenarios
- Targeted audience:
  - Novice users (simple verifications)
  - Expert users (sophisticated verifications, namely compositional)

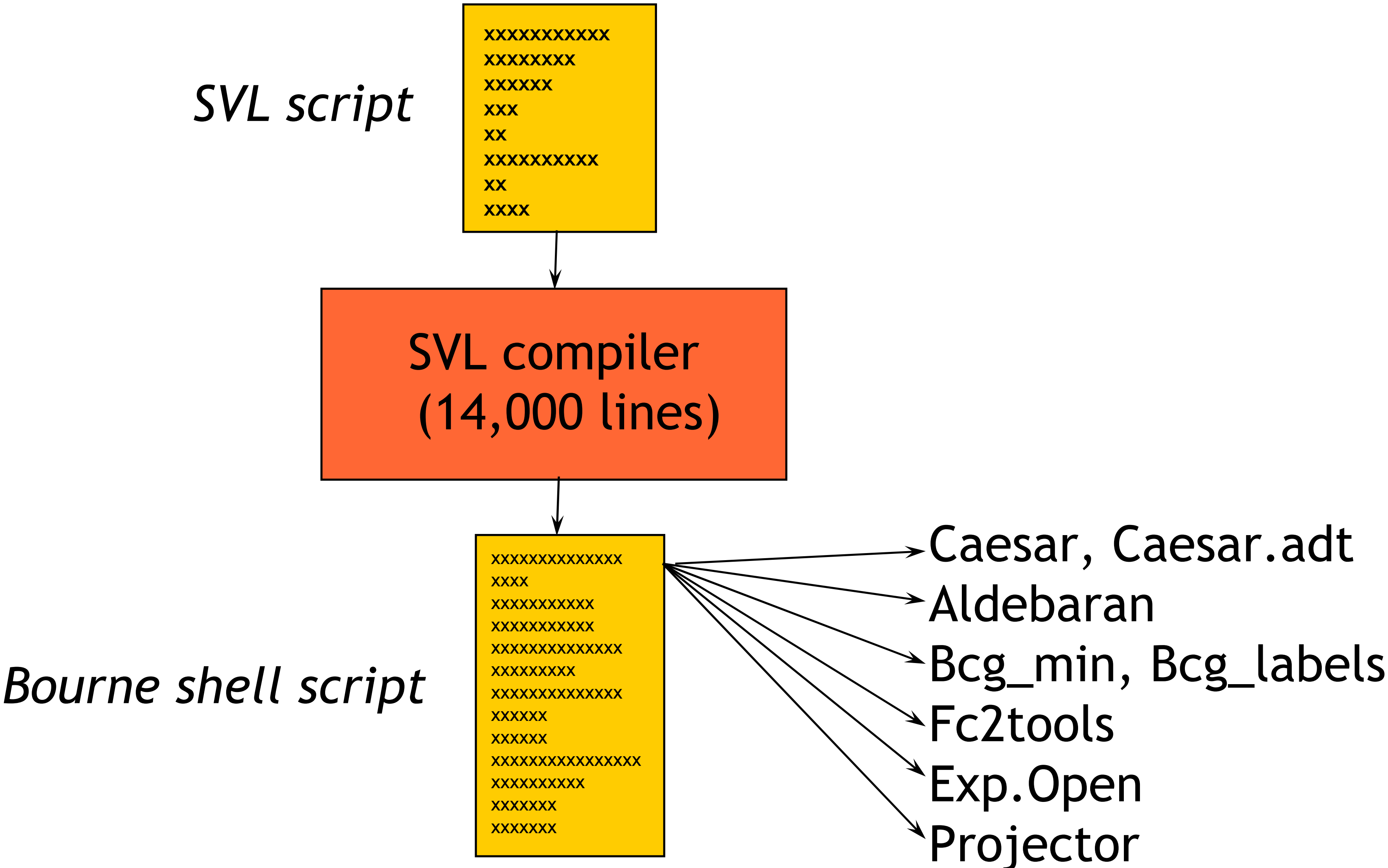
Script SVL: Example 1

```
% 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: Example 2

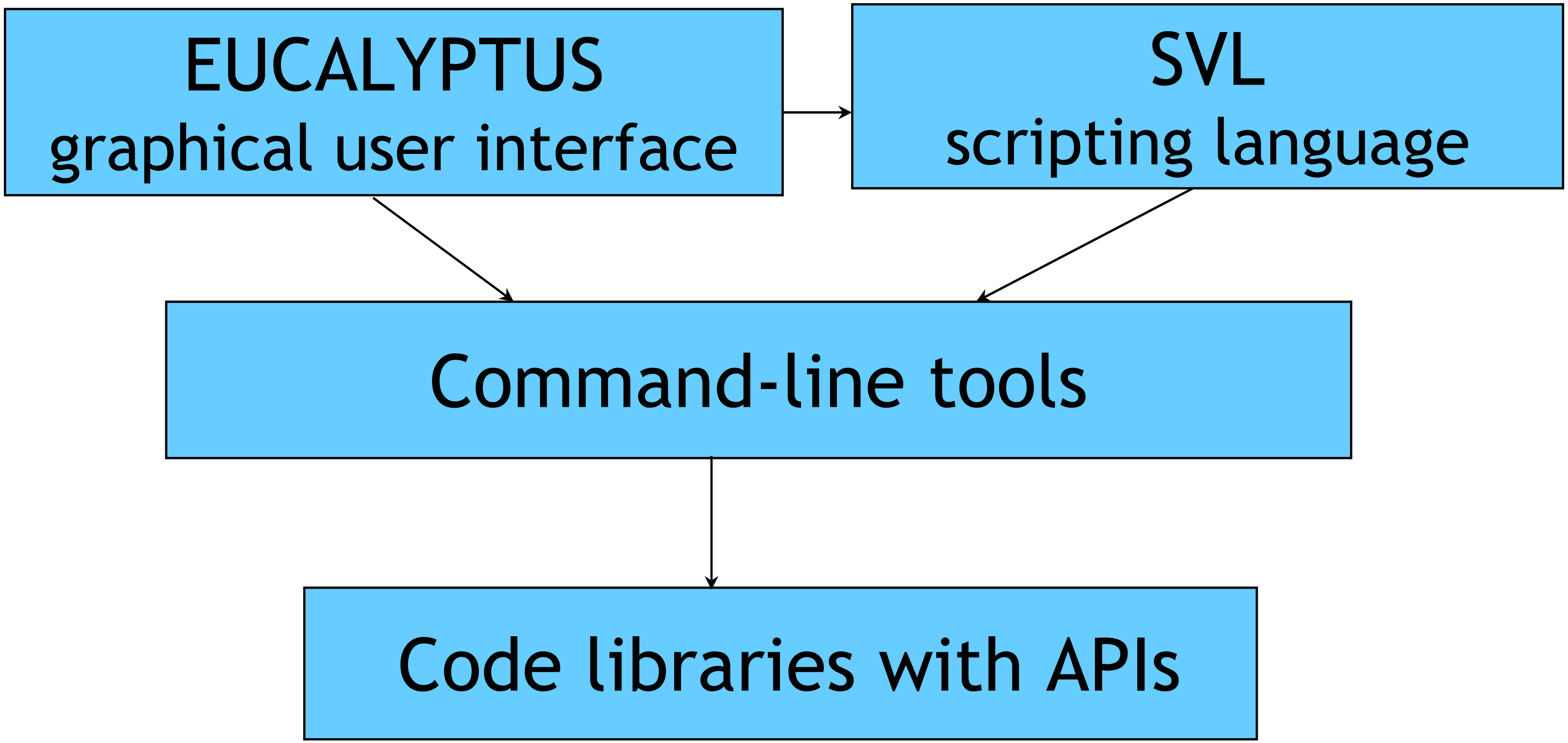
```
% DEFAULT_LOTOS_FILE="rel_rel.lotos"
"crash_trans.bcg" = strong reduction of CRASH_TRANSMITTER ;
"rel_rel.bcg" = generation of leaf strong reduction of
   hide R_T1, R_T2, R_T3, R12, R13, R21, R23, R31, R32  in
   (  (  (  (RECEIVER_NODE_1 -||? "r1_interface.lotos")
            |[R12, R21, R13, R31]|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

The SVL compiler



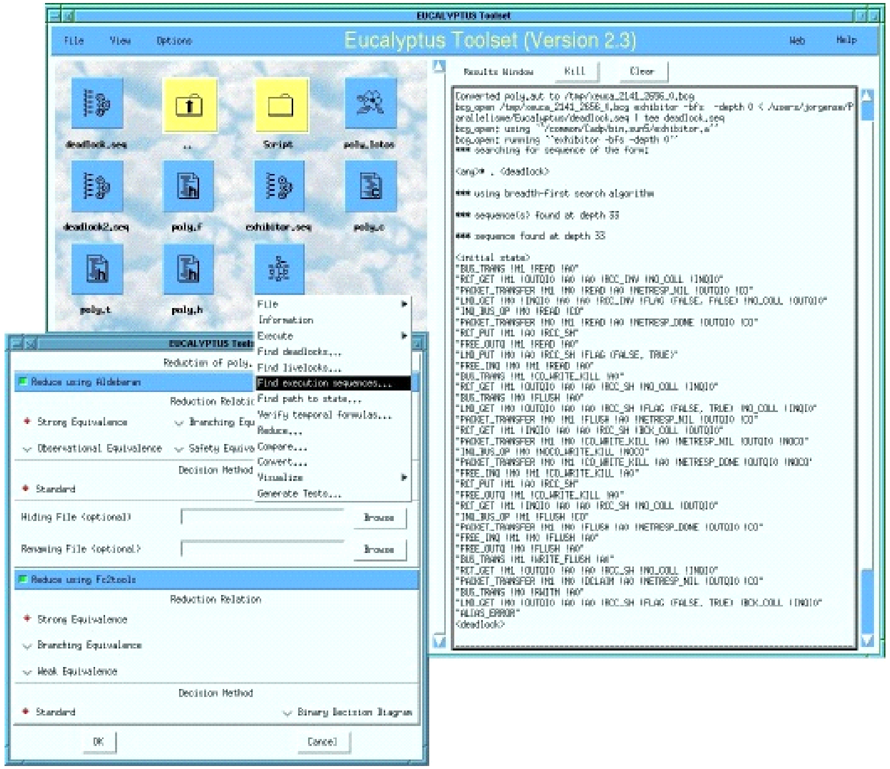
CADP Architecture

A layered software architecture



The EUCALYPTUS graphical user interface

- File types
- Contextual menus
- Dialog boxes
- Multiple tools: CADP, FC2
- Online help



Conclusion

Conclusion

- CADP: a rich platform for protocol and distributed systems engineering
- An open, extensible toolbox through well-defined APIs
- Several architectures supported:
  - Sun running SunOS or Solaris
  - PC running Linux
  - PC running Windows
- Large dissemination (figures dated 2001):
  - CADP licensed to **256 sites**
  - licenses granted for **950 machines in 2001**
  - **53 case-studies** done using CADP
  - **11 research tools** built using CADP

More information…

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