SEQ.OPEN: A Tool for Efficient Trace-Based Verification

Hubert Garavel and Radu Mateescu

INRIA Rhône-Alpes
655, avenue de l'Europe
38330 Montbonnot Saint Martin
France

http://www.inrialpes.fr/vasy

Motivations

- Verification of complex industrial systems
- Formal methods are not always applicable to existing (i.e., already running) systems
- These systems are often "opaque" (black box)
- Only the inputs/outputs events are visible
- Traces = chronological list of inputs/outputs
- Goal: Check the correctness of traces

Off-line vs On-line Traces

- Off-line trace = trace stored in a "log file"
- On-line trace = trace generated on the fly as the system executes
- Two different approaches to verification
  - For off-line traces: trace-based model checking
  - For on-line traces: run-time monitoring
- Pros and cons:
  - with off-line traces: it is easier to verify several temporal formulas (also, multiple runs may not produce the same trace due to nondeterminism)
  - with on-line traces: errors can be detected earlier

In this paper

- We focus on off-line traces ("log files"),which are easier to obtain in practice
- We follow a pragmatic approach:
  - We do not want to develop a yet another model checker that would be dedicated to traces
  - We want to reuse already existing tools as much as possible
  - The amount of new software development should be as limited as possible

The CADP toolbox

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

- Many features:
  - LOTOS -> C compilers
  - equivalence checking (bisimulations)
  - model checking (modal mu-calculus)
  - visual checking (graph drawing)
  - exhaustive, partial, on the fly, compositional verification
  - step by step simulation, random execution
  - C code generation, rapid prototyping
  - test case generation
- A wide dissemination:
  - license agreement signed with 310 organizations
  - installed on 840 machines in 2003
  - 72 case studies done with CADP
  - 16 research tools connected to CADP
  - 17 academic courses using CADP

Assumptions on the trace file format

- We want to handle large traces
- No *a priori* limitation on:
  - the length of traces (i.e., number of I/O events)
  - the size of labels (which is application-dependent)
- Traces should be encoded in standard text files
- The trace format should be simple
- We reuse the SEQUENCE format of CADP
  - human-readable text files
  - one event per line, enclosed between "..."
  - multiple traces allowed, separated by []

Exemple of a trace file

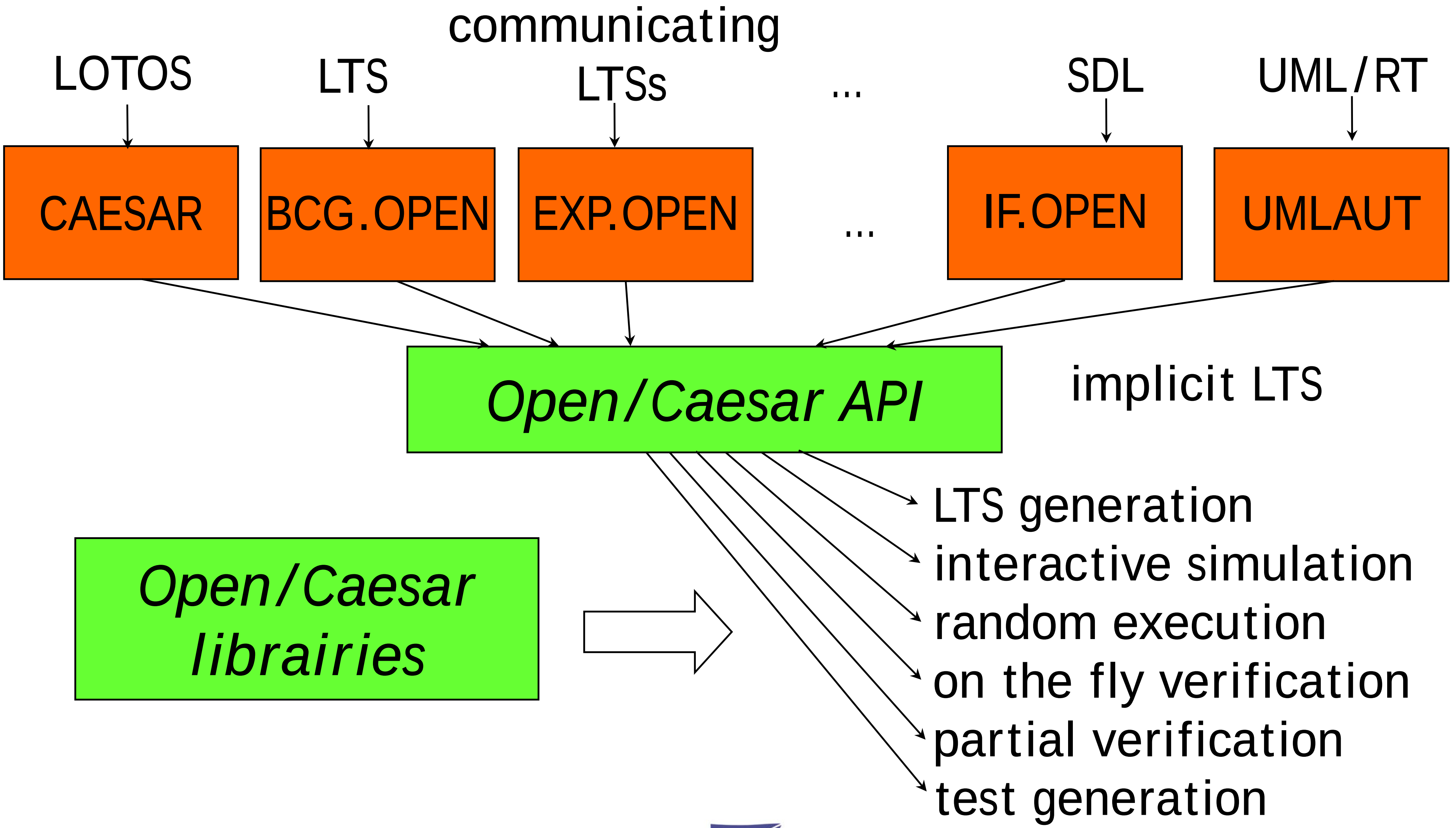
1st trace begins here

```
"PRI_INIT !PRB0 !COH !PMWI_D !ADDR !WB !VECTOR(00001) !PRB0 !SRCTXNID !M0_SNC0 !0"
"ILU_RESP !PRB0 !COH !NOCMP !IRETRY !NORMAL !0 !VECTOR(1000000)"
"ILU_REQ !OP_NIL !SRCTXNID !M0_SNC0 !WB !PMWI_D !ADDR !COH !VECTOR(0000000) !PRB0 !0"
"PRR_UPD_CLCOL !PRB0 !COH !VECTOR(0000000) !NODATA !CMP !PCMP !NORMAL !0"
"ILU_RELEASE !PCMP !COH !WB !SRCTXNID !M0_SNC0 !CMP"
"PRR_UPD_CLCOL !PMWW !DIRUPDATE !DIR_E !VECTOR(1000000) !ADDR"
"PRBE_RELEASE !PRB0"
"PRR_REQ !OP_NIL !SRCTXNID !M0_SNC0 !UC !PRLC !COH !VECTOR(0000000) !VECT"
"ILU_RESP !PRB0 !COH !NOCMP !IRETRY !NORMAL !0 !VECTOR(0001000)"
...
[]
```

2nd trace begins here

```
"PRI_INIT !PRB0 !COH !PRLD !ADDR !WB !VECTOR(00001) !PRB0 !SRCTXNID !M0_SNC0 !0"
"PRR_UPD_CLCOL !PRB0 !COH !VECTOR(0000001) !DATA !NOCMP !PDATA !NORMAL !0"
"ILU_RESP !PRB0 !COH !NOCMP !NULL !NORMAL !0 !VECTOR(0001000)"
"PRR_UPD_CLCOL !NOT_PMWW !DIRNOUPDATE !DIR_NOE !VECTOR(1001000) !ADDR"
"PRI_INIT !PRB0 !COH !PMWE_D !ADDR !WM !VECTOR(00100) !PRB0 !SRCTXNID !M0_SNC0 !0"
"PRR_UPD_CLCOL !PRB0 !COH !VECTOR(0000000) !NODATA !CMP !PCMP !NORMAL !0"
"ILU_RELEASE !PCMP !COH !WM !SRCTXNID !M0_SNC0 !CMP"
"PRR_UPD_CLCOL !PMWW !DIRUPDATE !DIR_E !VECTOR(1000000) !ADDR"
...
```

The OPEN/CAESAR framework



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

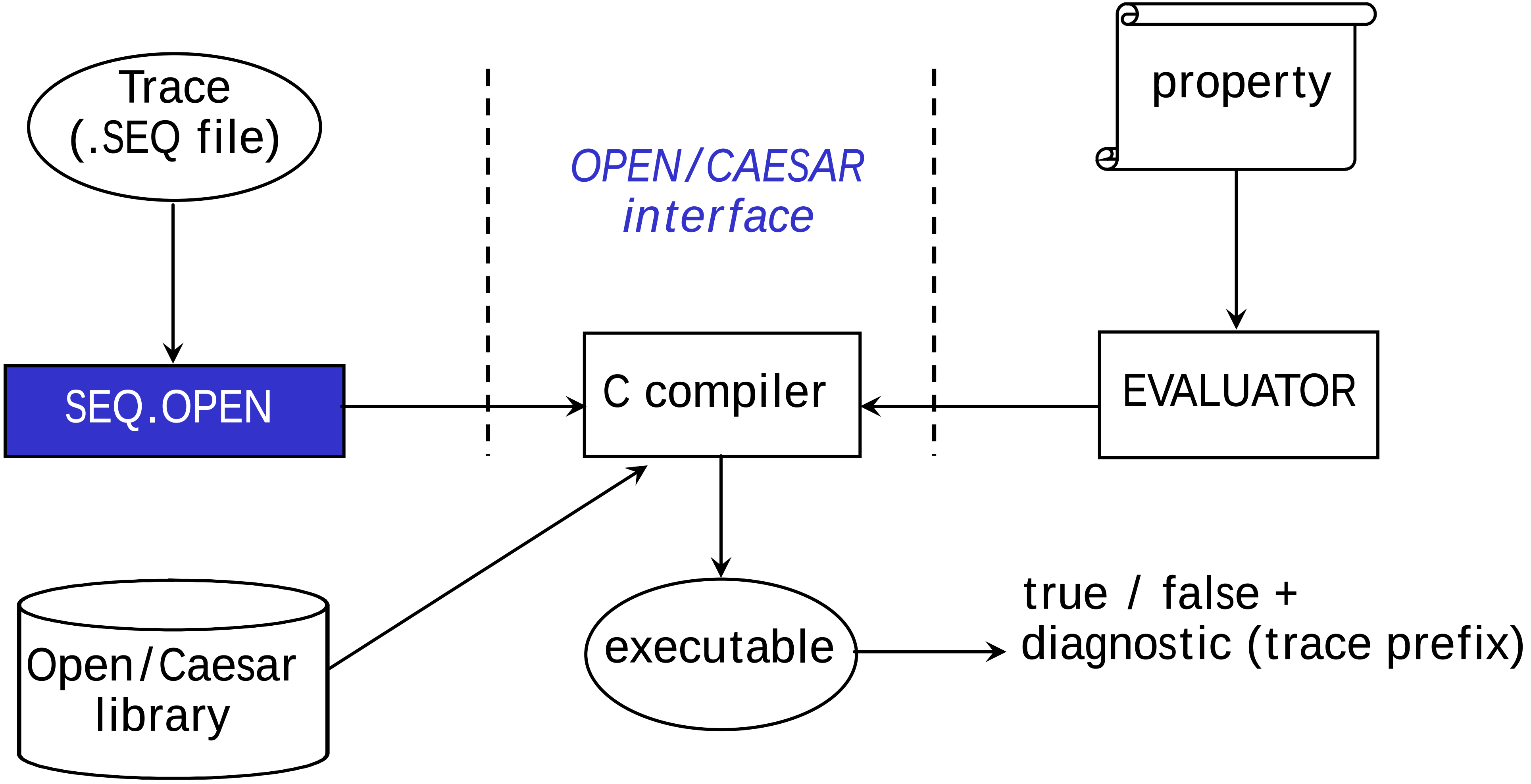
test generation

Open/Caesar librairies

The SEQ.OPEN tool

- A new OPEN/CAESAR tool
- SEQ.OPEN reads a ".SEQ" file containing traces encoded in the SEQUENCE format
- Traces are viewed as an *implicit* LTS, accessed using the OPEN/CAESAR interface
- Traces can be verified using various OPEN/CAESAR tools, such as:
  - EVALUATOR (model-checking of mu-calculus formulas)
  - EXHIBITOR (search for regular expressions)
  - BISIMULATOR (check for trace inclusion in an LTS)

The SEQ.OPEN tool (with EVALUATOR)



Implementation of states

- Three kinds of states in SEQ.OPEN:
  - initial state (any number of successors)
  - "ordinary states" (exactly one successor)
  - deadlock states (zero successor)
- Implementation: SEQ.OPEN does not load the entire trace (which can be large) in memory
- Instead: state = offset in the .SEQ file (+ 2 special offsets for initial/deadlock states)
- State offsets are canonical: equality of file offsets <=> equality of states

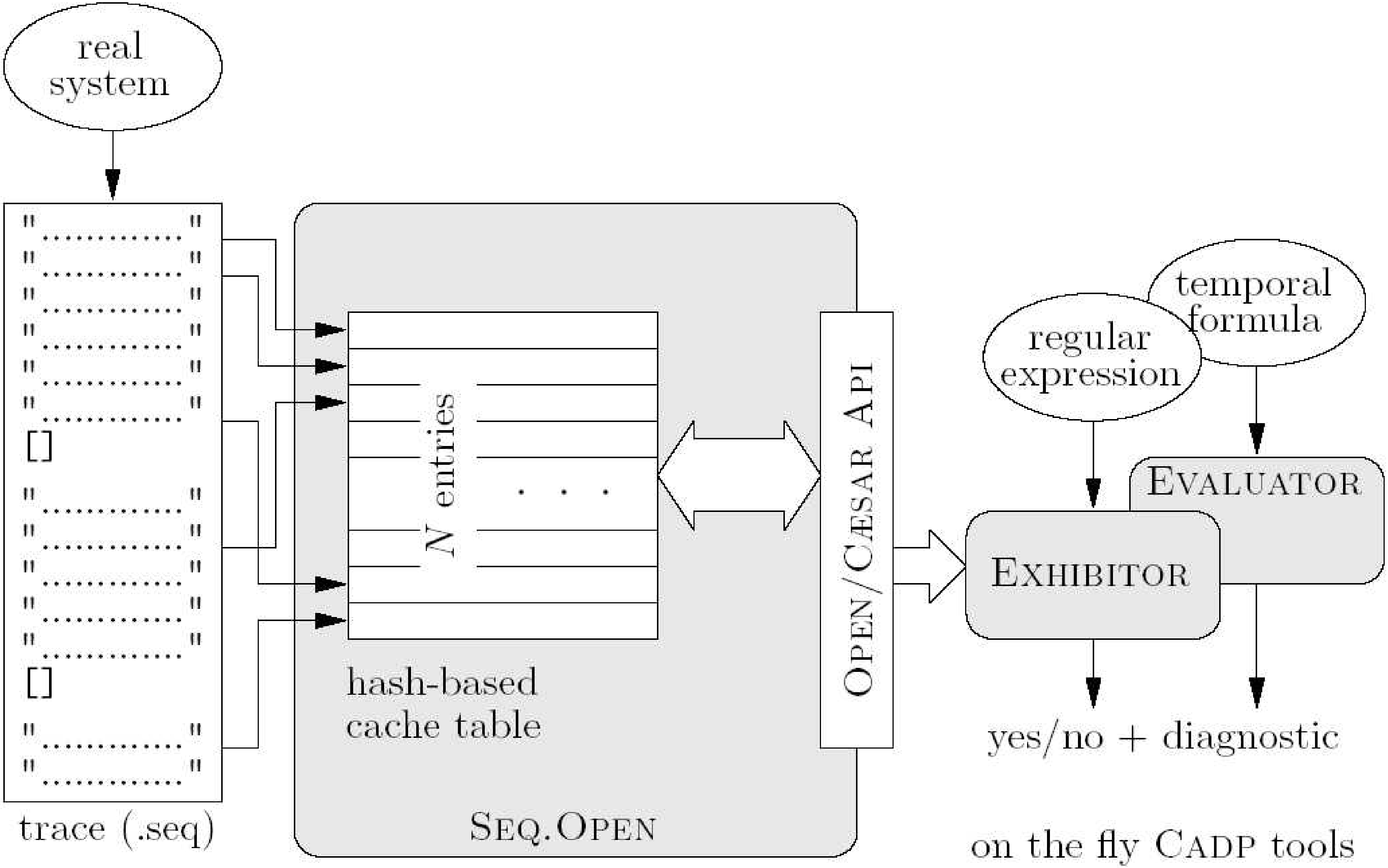
Implementation of labels

- Labels are character strings contained in the .SEQ file; their number and size are unbounded
- SEQ.OPEN does not store all labels in memory
- Instead: label = file offset (pointing to the opening double quote of the label)
- Label offsets are not canonical (contrary to states): two different offsets can point to equal character strings

Hash-based caching

- SEQ.OPEN uses an internal cache table to:
  - avoid redundant accesses in the .SEQ file
  - speed up the computation of state successors
  - speed up the mapping from file offsets to character strings
- Principle: hash-based caching
  - to each state offset => label and successor state
  - to each label offset => character string
  - collisions resolved by overwriting existing entries

Architecture of SEQ.OPEN



Two significant applications

- Hardware design: Multiprocessor architectures
  - Bull "NovaScale" servers ("FormalFame" project)
  - Random/guided simulation of Verilog designs → very large traces (>100,000 events)
  - Correctness and coverage checking using SEQ.OPEN
- Software architectures
  - IST Project "Archware"
  - Traces generated by the execution of a multi-threaded virtual machine
  - Correctness checking using SEQ.OPEN

Conclusion

- A pragmatic approach
  - trace checking is easily accepted in industry
  - do not develop a new model-checker for traces
  - reuse the existing CADP technology
  - use a simple, general format for traces
- A software implementation available
  - SEQ.OPEN (1,200 lines of code)
  - distributed as part of CADP (since Dec. 2002)
- Several non-trivial applications