An Overview of CADP 2014

Hubert Garavel

INRIA – Univ. Grenoble Alpes – LIG

http://convecs.inria.fr

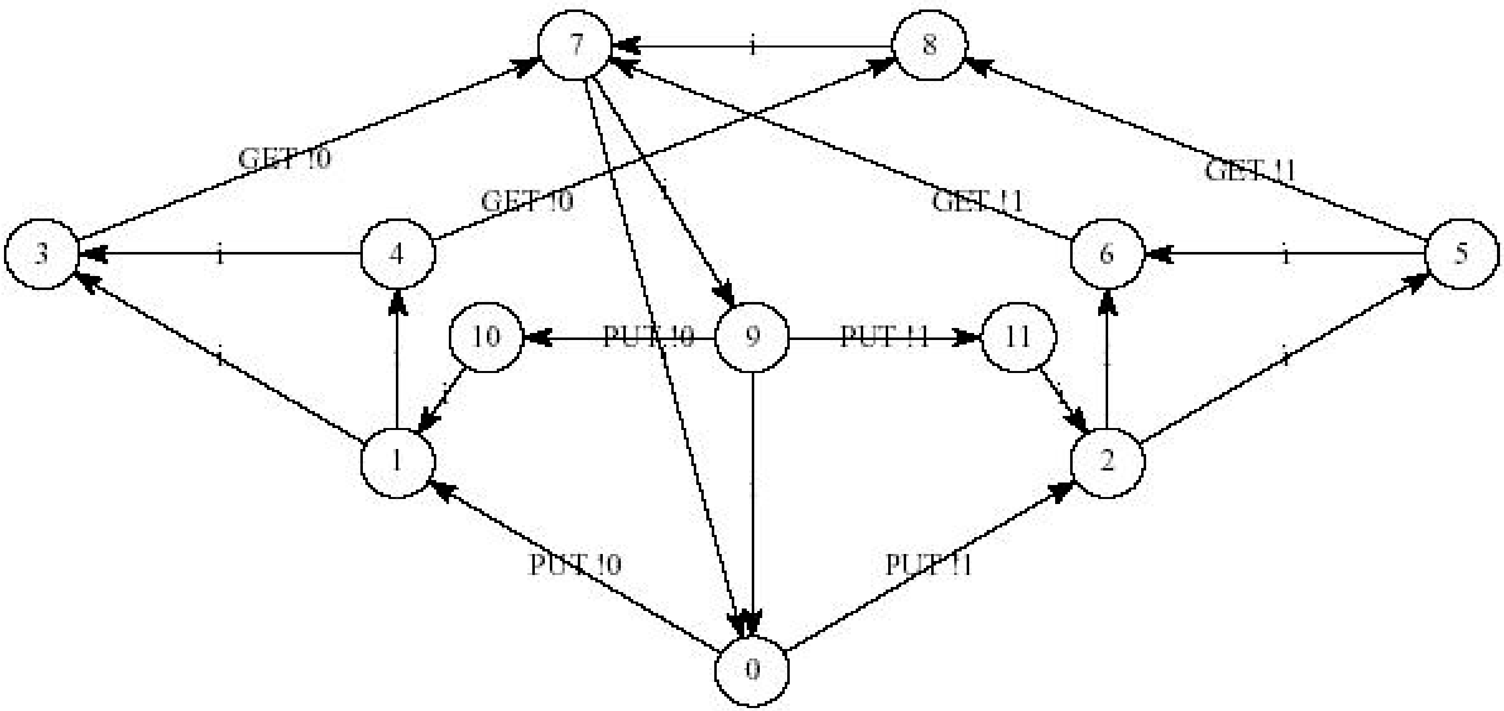
CADP

- A software toolbox for studying asychronous systems
- At the crossroads between:
  - concurrency theory
  - formal methods
  - computer-aided verification
  - compiler construction
- A continuous long-run effort:
  - development of CADP started in the mid 80s
  - initially: only 2 tools (CAESAR and ALDEBARAN)
  - today: nearly 50 tools

Semantic models and verification technologies

LTS (Labelled Transition Systems)

- LTS = state-transition graph
  - no information attached to states (except the initial state)
  - information ("labels" or "actions") attached to transitions

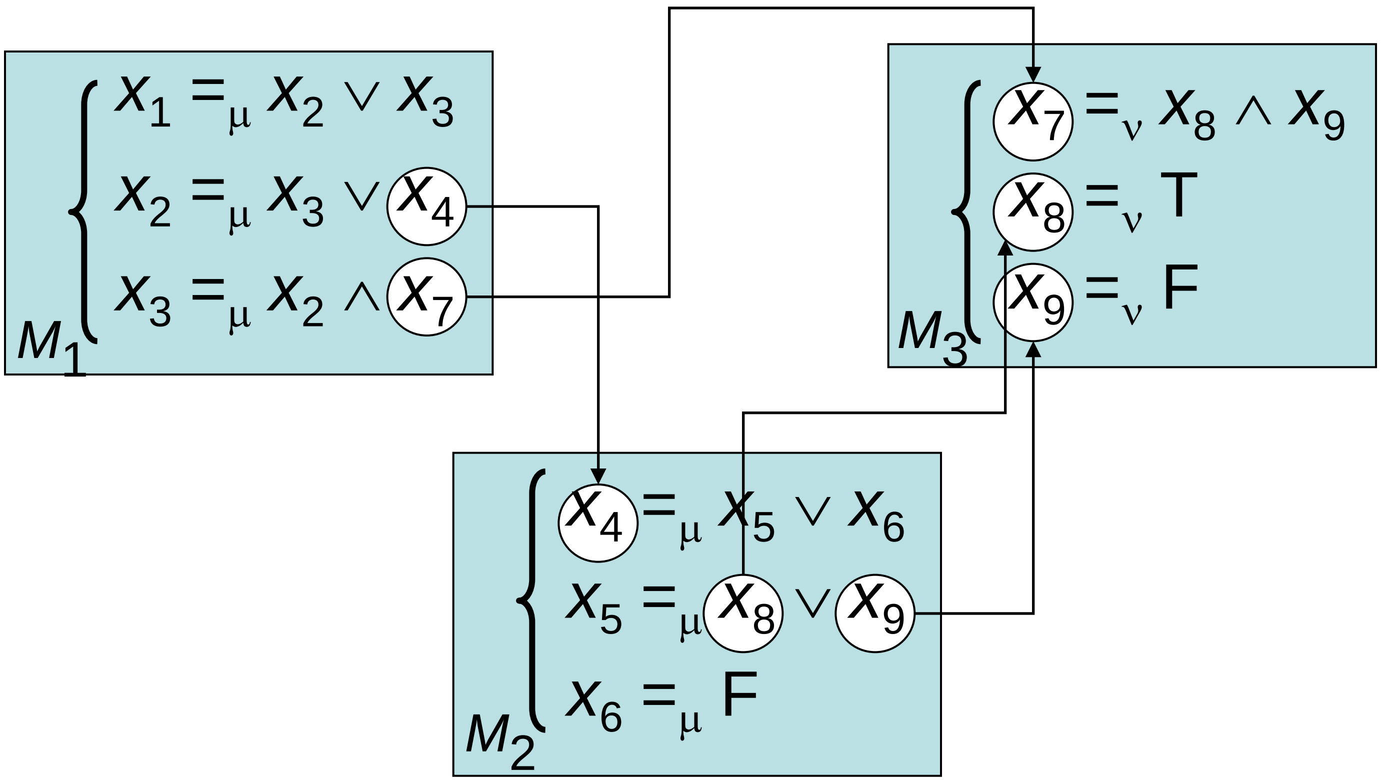


CADP technologies for LTSs

- "Explicit" LTS *(enumerative, global)*:
  - comprehensive sets of states, transitions, labels
  - BCG: a file format for storing large LTSs
  - up to 2^{44} states and transitions
  - a set of tools for handling BCG files
- "Implicit" LTS *(on the fly, local)*:
  - defined by initial state and transition function
  - Open/Caesar: a language-independent API
  - many languages connected to Open/Caesar
  - many tools developed on top of Open/Caesar

BES (Boolean Equation Systems)

- Boolean variables, constants, and connectors
- least (μ) and greatest (ν) fix points
- DAG of equation systems (no cycles – alternation-free)



CADP technologies for BESs

- BES can be given:
  - explicitly (stored in a file)
  - or implicitly (generated on the fly)
- CAESAR_SOLVE: a generic solver for BES
  - works on the fly: solves while building the BES
  - translates dynamically BES into Boolean graphs
  - implements 9 resolution algorithms A0-A8 (general or specialized)
  - generates diagnostics (examples or counter-examples)
  - fully documented API
- BES_SOLVE: a solver for explicit BES

Specification languages

Four specification languages in CADP

None of these languages is bound to a specific application domain

They have been used in software, hardware, telecom, bioinformatics…

- 1. LOTOS
  - process calculus combining CSP [Hoare] and CCS [Milner]
  - international standard ISO 8807:1989
  - tools: CAESAR, CAESAR.ADT, CAESAR.BDD
- 2. EXP
  - language for describing networks of communicating automata
  - parallel composition operators (LOTOS, CCS, CSP, mCRL, etc.) + MEC-like synchronization vectors
  - label hiding, renaming, cutting (using regexps), priorities
  - tools: EXP2C, EXP.OPEN (on-the-fly partial order reductions)

Four specification languages in CADP

- 3. FSP
  - process calculus designed for teaching purpose
  - by Jeff Kramer and Jeff Magee (Imperial College)
  - tools: FSP2LOTOS (translator to LOTOS+EXP), FSP.OPEN

- 4. LNT (formerly: LOTOS NT)
  - a modern specification language for concurrent systems
  - inspired from E-LOTOS (international standard ISO 15436:2001)
  - funded by Bull and the MULTIVAL project of Minalogic
  - tools: LNT2LOTOS (translation to LOTOS+C), LPP, LNT.OPEN

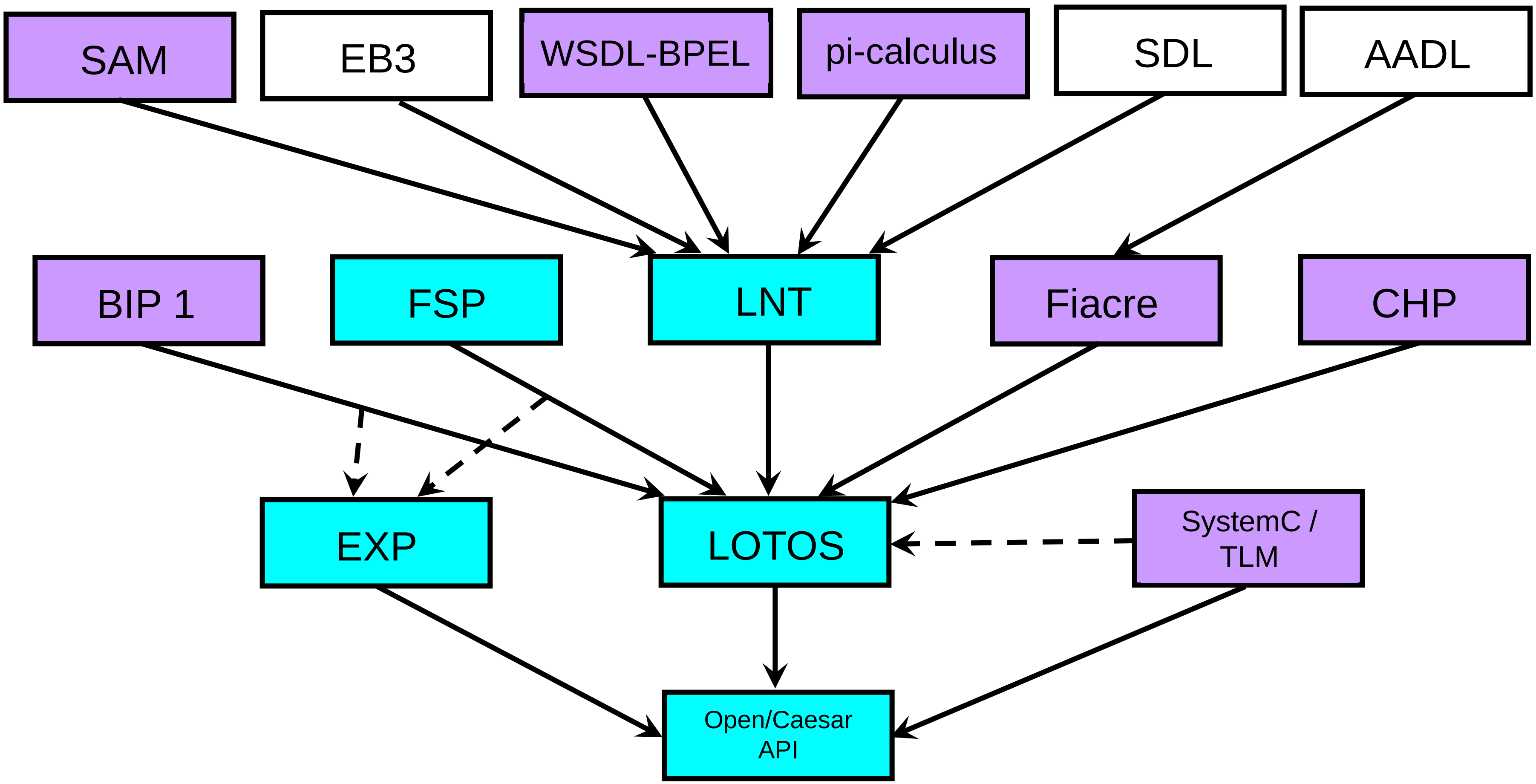
Main features of LNT

A careful mix of process calculi and functional languages

Key idea: be closer to mainstream programming language

- Types
  - predefined types: boolean, integer, real, character, string, etc.
  - ML-like inductive types + subranges, sets, lists, sorted lists, etc.
- Functions
  - if, for, while, case + pattern-matching, return
- Processes: superset of functions
  - nondeterministic choice, nondeterministic value selection
  - multiway rendezvous, typed communication channels
- Modules

Connecting other languages to CADP



Model checking

Three model-checkers in CADP

- EVALUATOR 3.6
  - alternation-free modal μ-calculus
  - extended with regular expressions on labels and action paths
  - libraries of standard property patterns
  - on-the-fly model checker built on top of Caesar_Solve BES solver
  - automatic generation of diagnostics (sequences, trees, or graphs with cycles) to explain why a formula is true or false
- EVALUATOR 4.0
  - extends μ-calculus formula with typed data
  - **if**, **case**, **let** statements ; quantifiers over finite domains
  - on-the-fly model checking based on PBES (Parameterized Boolean Equation Systems) ; automatic generation of diagnostics

Three model-checkers in CADP

- XTL
  - functional language to express queries on explicit LTSs encoded in the BCG format
  - data types: booleans, integers, reals, character, strings
  - LTS types: states, labels, edges, state sets, edge sets
  - rapid prototyping of LTS exploration algorithms
  - easy encoding of temporal logics: HML, CTL, ACTL, μ-calculus
  - "non-standard" properties involving data: counting actions
  - XTL compiler: translates XTL to C code
  - possibility to import external C code

Equivalence checking

Equivalence checking

- An alternative approach to model checking:
  formal verification without temporal logic formulas
- Principles:
  - Old idea of program equivalence
  - Compare two programs $\rightarrow$ generate and compare their LTSs
  - **Equivalence relations** between LTSs:
    LTSs are equivalent iff they have "the same" observable behaviour
    many possible equivalence relations exist
  - **Bisimulations**: a subclass of equivalence relations
    states are equivalent iff they have the same future
    stronger than usual trace (or language) equivalence
    several bisimulation relations: strong, branching, etc.
    efficient algorithms exist to compute bisimulations
  - **Preorder relations** between LTSs:
    An LTS contains another LTS if it can do all what the other does,
    and possibly more (~ refinement and implementation relations)

Equivalence checking

- Practically:
  - a large, complex LOTOS/LNT specification is compared against a small, visibly correct LTS
  - a large LTS is minimized to yield a smaller, equivalent one

- Equivalence checking is efficiently implemented in CADP
  BCG_MIN, BISIMULATOR, EXP.OPEN, REDUCTOR
  - minimization and comparison of LTSs
  - explicit-state and on-the-fly algorithms (based on BES solving)
  - 7 equivalence relations supported, with their preorders
  - generates diagnostics to explain why comparison fails

Fighting state explosion…

Compositional verification

- A significant means of fighting state explosion
  - A "silver bullet" applicable to process calculi only
  - Implemented in several co-operating CADP tools
    BCG_MIN, CAESAR, EXP.OPEN, PROJECTOR, SVL
- Principle:
  - Divide the system into concurrent processes
  - Generate the LTS of each separate processes
    (possibly adding "interface" constraints to restrict this LTS)
  - Minimize all the LTSs (for strong or branching bisimulation)
  - Recombine in parallel all the minimized LTSs
    (during LTS generation, interface constraints are checked)
  - Result: a smaller, yet (strongly- or branching-) equivalent LTS

Distributed verification

- Exploit NoWs, clusters, and grids
- Cumulate RAM and CPU of many remote machines
- Distributed LTS exploration

  DISTRIBUTOR, PBG_MERGE, PBG_CP, PBG_INFO, PBG_MV,PBG_RM
  - The LTS is built on the fly and partitioned into fragments
  - Each fragments is a set of states and transitions
  - Each fragment is built and stored on a different machine
  - PBG = distributed LTS consisting of remote fragments
- Distributed BES resolution

  BES_SOLVE
  - The BES is built, partitioned, and solved on the fly
  - Each fragment is a set of Boolean variables and logical dependencies between variables
- In practice, linear scalability is observed

Beyond verification…

Model-based testing

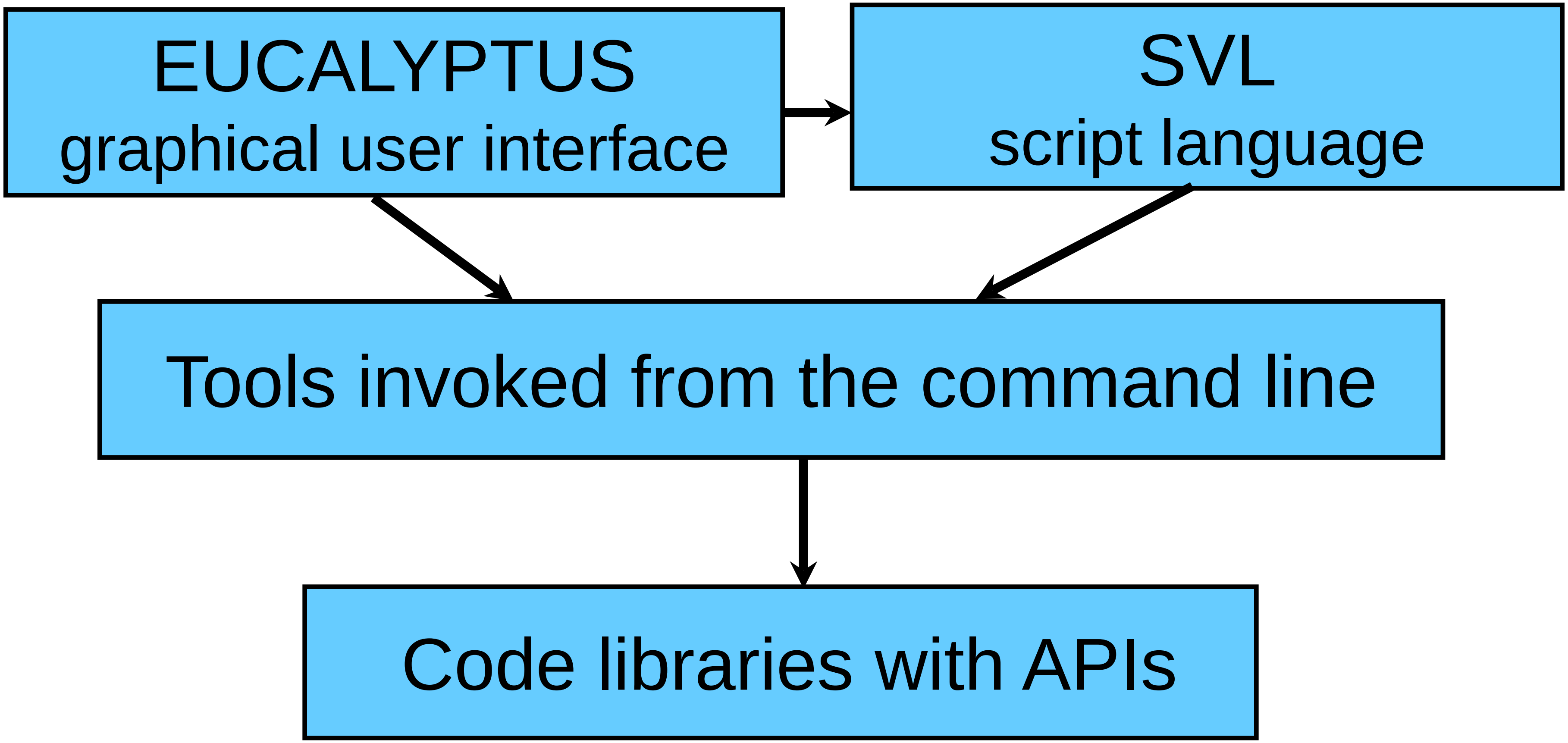
- Comparison between:
  - a formal model (LOTOS, LNT)
  - an actual implementation (software, hardware)
- On-line testing (co-simulation) EXEC/CAESAR
  - simultaneous execution of model and implementation
  - detection of diverging behaviour
- Off-line testing (test-case generation) TGV
  - test cases automatically generated from the model
  - test purposes (scenarios), pass/fail verdicts
- Trace checking (off-line analysis of log files) SEQ.OPEN

Quantitative analysis

- Combining functional verification (Boolean results) and performance evaluation (numerical results)
- *Interactive Markov Chains* (IMCs) *[Hermanns-98]*
  - combination of LTSs and continuous-time Markov chains
  - parallel composition ("rate" transitions do not synchronize)
  - theory permits compositional generation/minimization of IMCs
- Supported by CADP:
  - Compositional generation of IMCs
    BCG_MIN, DETERMINATOR, EXP.OPEN, SVL
  - Steady-state and transient solvers for IMCs
    BCG_STEADY and BCG_TRANSIENT
  - Simulation for IMCs CUNCTATOR
- Also: *Interactive Probabilistic Chains* (IPCs)
  - combination of LTSs and discrete-time Markov chains

Integration between CADP tools

A layered software architecture

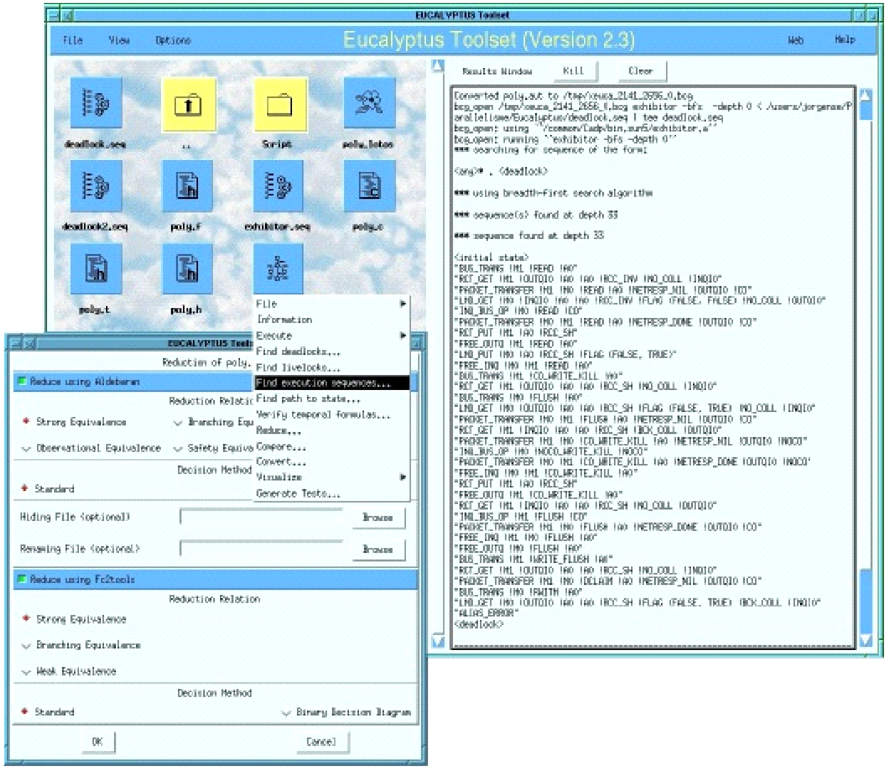


EUCALYPTUS graphical user interface

A simple user interface:

- File types
- Contextual menus
- Dialog boxes
- Online help

Minimalistic, yet usable



User contributions:

- configuration files for various editors: emacs, jedit, a2ps
- several Eclipse plugins for CADP

SVL script language

- SVL is both:
  - a script language to describe verification scenarios
  - a compiler that translates SVL scripts to shell scripts

- Using SVL is optional (as well as EUCALYPTUS)
- Advantages:
  - higher level than command-line tool invocations
  - provides a unified textual interface above CADP tools
  - eases writing of compositional verification scenarios
  - implements automated verification tactics
  - targets both naive and expert users
- SVL is being regularly enhanced

Forthcoming SVL extension for traceability

property DigitReadiness (d)

"It is always possible for the subscriber to press on digit $d"

is

-- verification using model-checking

"system.lnt" |= [true*] (< "DIAL !$d" > true) ;

expected TRUE ;

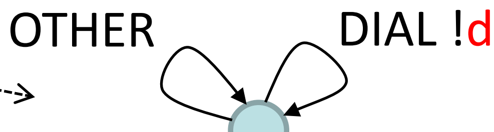
-- verification using equivalence checking

strong comparison

(**total rename** "DIAL !$d" -> "DIAL !$d", ".*" -> "OTHER" **in**
"system.lnt") == "result_$d.aut" ;

expected TRUE ;

end property



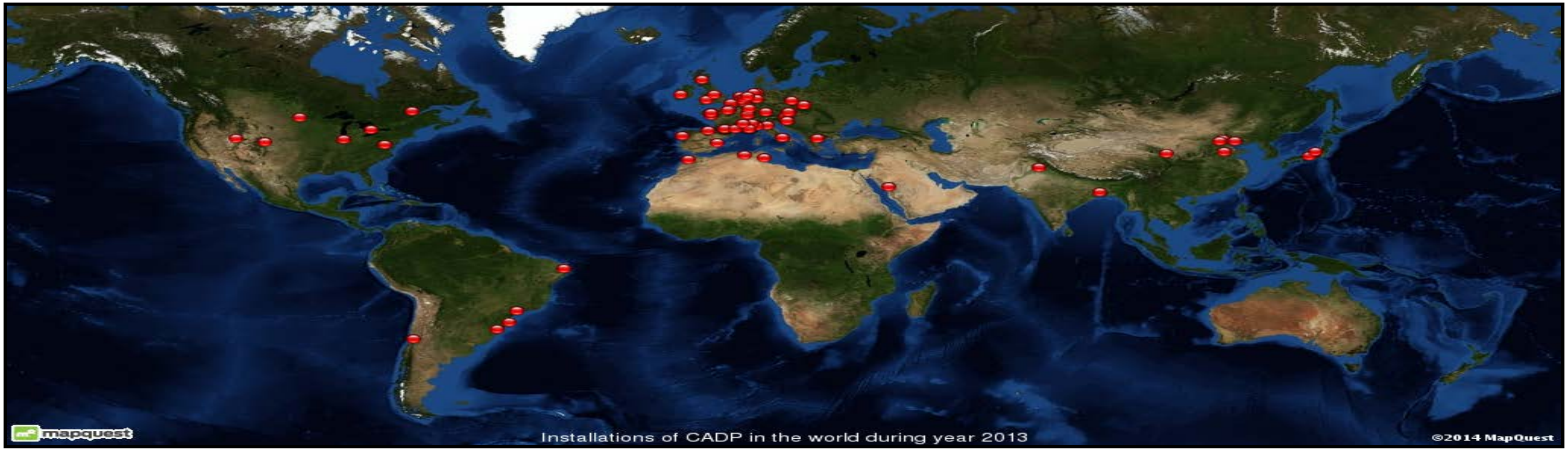
Conclusion

Conclusion

CADP: bringing concurrency theory to practice

- A comprehensive toolbox:
  - 50 tools, 20 code libraries
  - modular, extensible using well-defined, stable APIs
- A significant software development effort:
  - platforms: Linux, MacOS X, Solaris, Windows
  - large documentation (700+ pages)
  - emphasis on quality and backward compatibility
- Free for academic users

Dissemination and impact



- CADP licenses granted for 10,000+ machines
- 60+ university lectures based on CADP since 2002
- 170+ case studies tackled using CADP
- 80+ academic software tools reusing CADP components

More: ……………………………………………… http://cadp.inria.fr