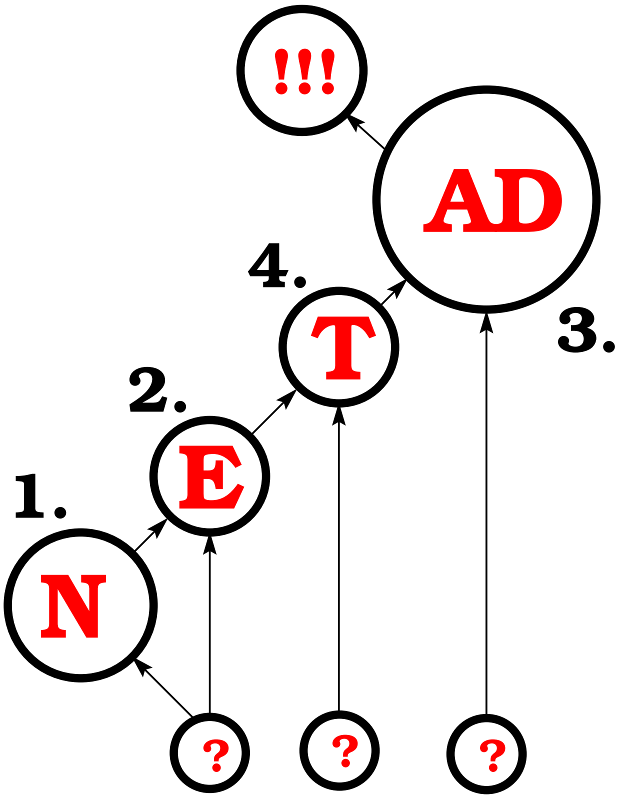
Generic Development of Application Program Enhancement and Transformation Algorithms

used for

NEADT - a **N**etwork **E**nabled **A**utomatic **D**ifferentiation **T**ool



Uwe Naumann

INRIA Sophia-Antipolis
Action TROPICS

Uwe.Naumann@sophia.inria.fr

What? Why? How? Who?

What?

1. **ALI** - Analyses de Languages Imperatifs
   Platform for general application code enhancement and transformation (ACET) accepting virtually any imperative programming language (F77-01,C,C++,Java,Ada,...)

2. Application Programmer's Interface (API) of ALI

3. **GDE** - Generic Differentiation Engine (Default modes → simple AD tool) + API

4. ACET Data Base (Derivative Code Data Base, Reference Solutions, Reports, UML Specifications)

5. Advanced AD Algorithms (Optimized pre-accumulation of Jacobians - static and runtime, Sparsity detection and exploitation, Checkpointing)

6. AD-specific Analyses (Trajectory, Differentiation unit argument checkpointing)

7. Intuitive User Interface

8. Servlet-based NEDS = Network Enabled Differentiation Server

Current Market Situation

1. End-users and application programmers (AP) from science and industry require ACET methods (Parallelization, AD, code optimization, proofs)

2. Simulation and optimization codes written in F77/90/95,C,C++,Ada,... (mixture, black box routines)

3. (to our knowledge) $\nexists$ general, open, accessible (free) ACET platform (neither in general nor for AD)

4. Fast and AP-friendly implementation of theoretical state of the art only by "universal experts without time limits"

5. Odyssee, ADOL-C/F, ADIFOR, ADIC, Padre2,... academic AD packages with many strengths (limitations), numerous theoretical results not implemented

6. Requirements of (potential) AD end-users are known, links exist

7. ESF pre-proposal: 19 academic institutions from 11 European countries, 11 industrial partners from 6 European countries $\Rightarrow$ interest in AD

Why? - It's the time to do it!

1. (Quasi-) Standardized platform for development of ACET algorithms by application programmers (AP) from multiple disciplines

2. (Java+UML ⇒) Portable, Re-usable and Extendible

3. Fast and AP-friendly implementation of known theoretical results

4. Elimination of the programming language problem

5. Interaction with other ACET methods (parallelization, code optimization, partial execution)

6. Acceptance of AD due to better applicability, user-friendliness, and efficiency (e.g. X-country? → speedup by factors > 3 possible)

7. Unification of strengths of various teams developing ACET/AD algorithms (shared competences)

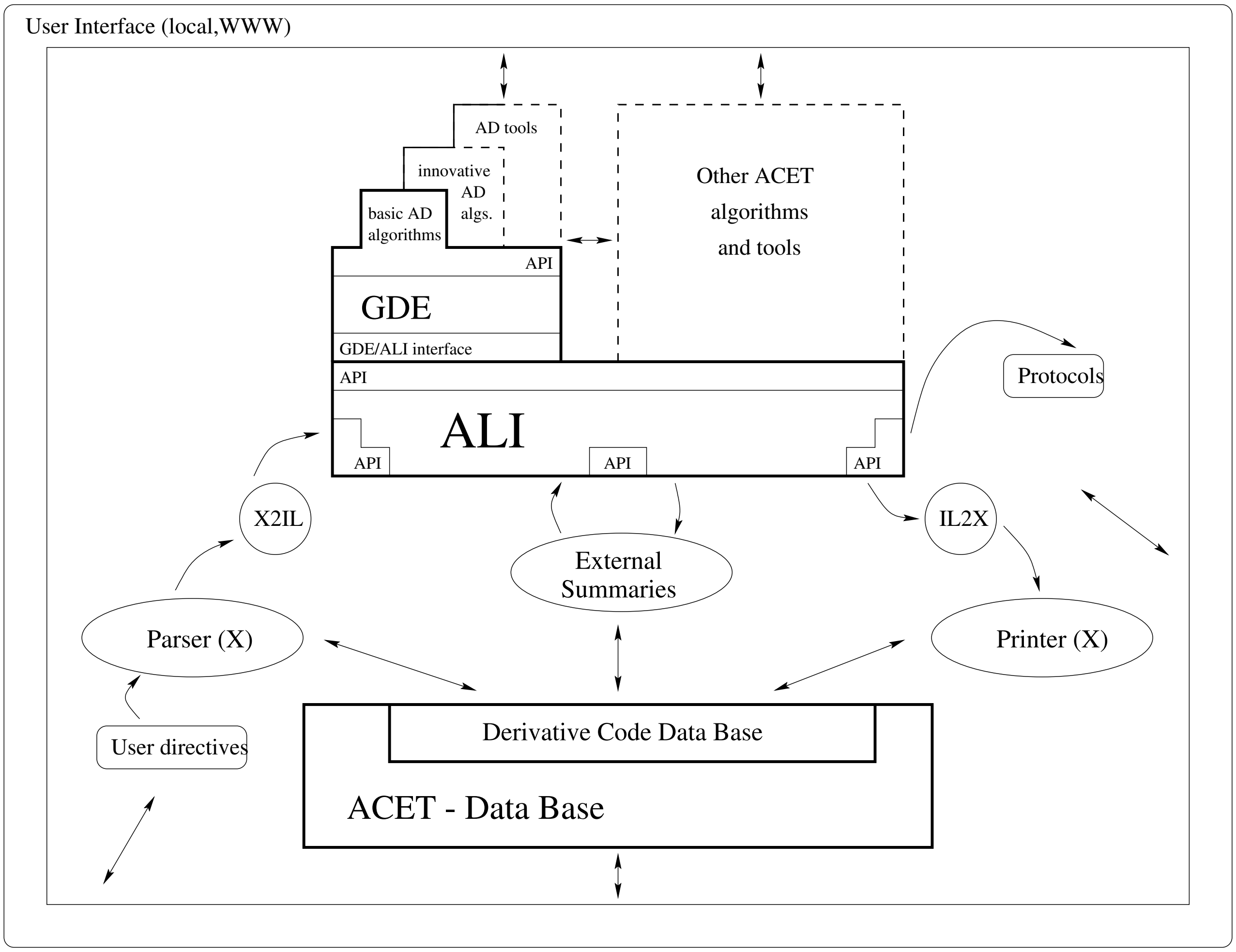
8. Developers and (potential) AD-users ask for it!

9. WWW-interface ⇒ possibility "to play" with AD ⇒ acceptance, teaching

Uwe Naumann (Uwe.Naumann@sophia.inria.fr)
Action TROPICS (http://www-sop.inria.fr/tropics)

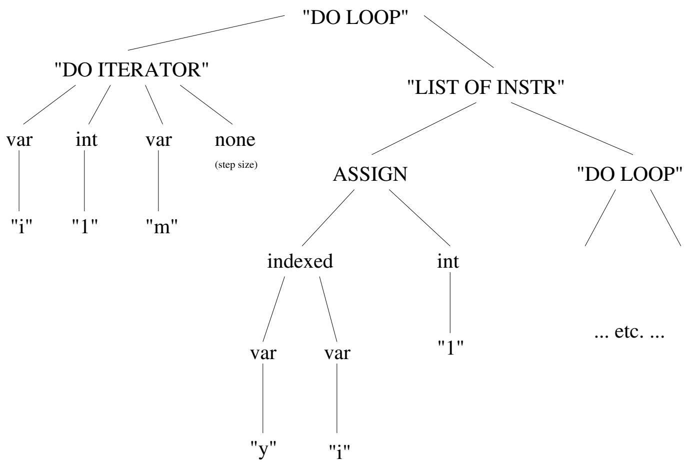
How?

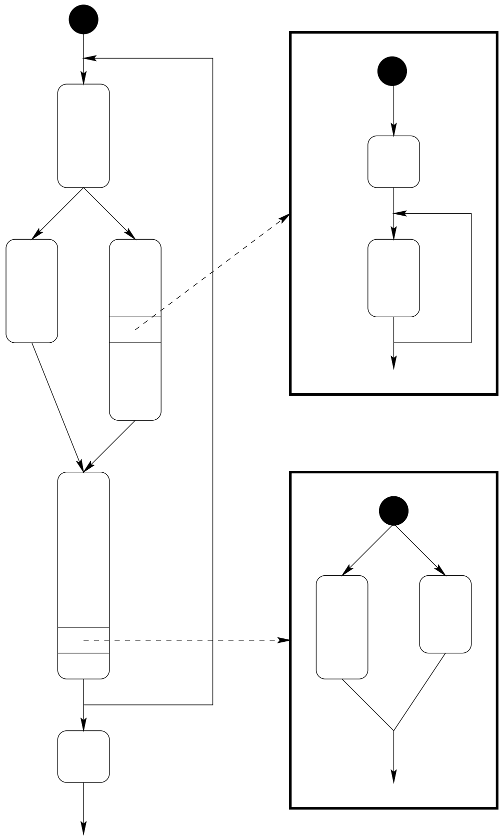
- Programs, subroutines in several imperative programming languages → abstract internal representation ALI based on IL (Imperative Language) → API → ACET methods and tools
- (Application codes + user directives + unit summaries) → parser → X2IL → ALI → GDE → DCDB → AD algorithm/tool → ALI → IL2X → printer → (derivative code + comments/warnings + protocol) → DCDB
- pre-/post-processing, logic codes for propagating boolean patterns, user interface, WWW interface, code optimization, parallelization



ALI

- X-Parser - AST(X) - X2IL protocol (e.g. generated with Centaur) - AST(IL) - Abstract Internal Representation (AIR)
- Flow graph, call graph, symbol table
- Analyses: type checking, dead code, loops, pointers, read-write, basic blocks (merging and ordering)
- ACET on AIR





Tasks within ALI Workpackage

1. **Interprocedural Framework (AIR)**
   - Flow graph
   - Call graph
   - temporary hard-disc storage/retrieval
   - summaries of call graph nodes
2. **General Interest Analyses**
   - data dependence
   - pointer
   - read-write
3. **Target Language I/O**
   - X2IL
   - IL2X
4. **C++**
   - C++2IL, IL2C++
   - impact of new features (objects,...) on AIR, analyses
   - focus on call graph
5. **Application Programmers Interface**
   - deduce from existing AD tools and GDE
   - UML representation of internals of ALI

Tasks within GDE Workpackage

1. **ACET Methods**
   - Transformations and enhancements on ALI's AIR related to AD
   - Interpretation of user-directives
   - Logic code generation (boolean patterns)
2. **Fundamental Differentiation and Analyses (Kernel - I)**
   - Basic pre- and post-processing
   - Local partial derivatives/Jacobians and forward vector mode
   - Warning mechanism
   - Basic analyses
3. **Advanced Differentiation and Analyses (Kernel - II)**
   - Adjoint elementals and reverse mode, trajectory analysis
   - Pre-accumulation, Derivative Code Data Base
   - Divided differences on black-box d-units
4. **Kernel Extensions (Kernel - III)**
   - D-unit call graph reversal modes
   - D-unit argument checkpointing
   - profiling facility and run-time dependent optimisations
5. **Application Programmers Interface**
   - API
   - Default modes

GDE - ACET Methods

- Transformations and enhancements on ALI's AIR related to AD

  (set of routines for adding AD-specific declarations, instructions and comments to ALI's AIR including tangent linear and adjoint instructions in vector mode, corresponding variable declarations, store/restore instructions, trajectory declarations, comments including warnings, references)

- Interpretation of user-directives

  (construction of d-unit hierarchy including call division, correctness tests, generation of d-unit call graph; recuperation of user directives associated with d-units, i.e context independent directives - associated with definition of d-unit - context dependent directives - associated with actual call)

- Logic code generation (boolean patterns)

  (forward and reverse logic code, overloading by OR)

Uwe Naumann (Uwe.Naumann@sophia.inria.fr) November 1999
Action TROPICS (http://www-sop.inria.fr/tropics)

GDE - Fundamental Differentiation and Analyses (Kernel - I)

- Basic pre- and post-processing

  (canonicalisation, optimisation)

- Local partial derivatives/Jacobians and forward vector mode

  (local partials of intrinsics; explicit Jacobian-vector products; chain rule on instruction trees - statement level forward; chain rule on d-unit call graph - implicit Jacobian-vector products)

- Warning mechanism

  (possible non-differentiability)

- Basic analyses

  (dead code - independent and dependent variables specified; d-unit i/o; activity)

GDE - Advanced Differentiation and Analyses (Kernel - II)

- Adjoint elementals and reverse mode, trajectory analysis

  (implicit and explicit vector-Jacobian products; statement level reverse mode; location level TA - conservative, loop fixed point, two-pass subroutine output)

- Pre-accumulation, Derivative Code Data Base

  (Jacobian data structure declaration; forward and reverse pre-accumulation; which DBMS?; how to manage?)

- Divided differences on black-box d-units

  (algorithms provided by users; default algorithms including warnings)

GDE - Kernel Extensions (Kernel - III)

- D-unit call graph reversal modes

  (split; joint; joint with call division; mixed modes)

- D-unit argument checkpointing

  (analysis; data structures; store/restore methods)

- profiling facility and run-time dependent optimisations

  (run code $\rightarrow$ costly parts $\rightarrow$ optimisation)

GDE - Application Programmers Interface

- API

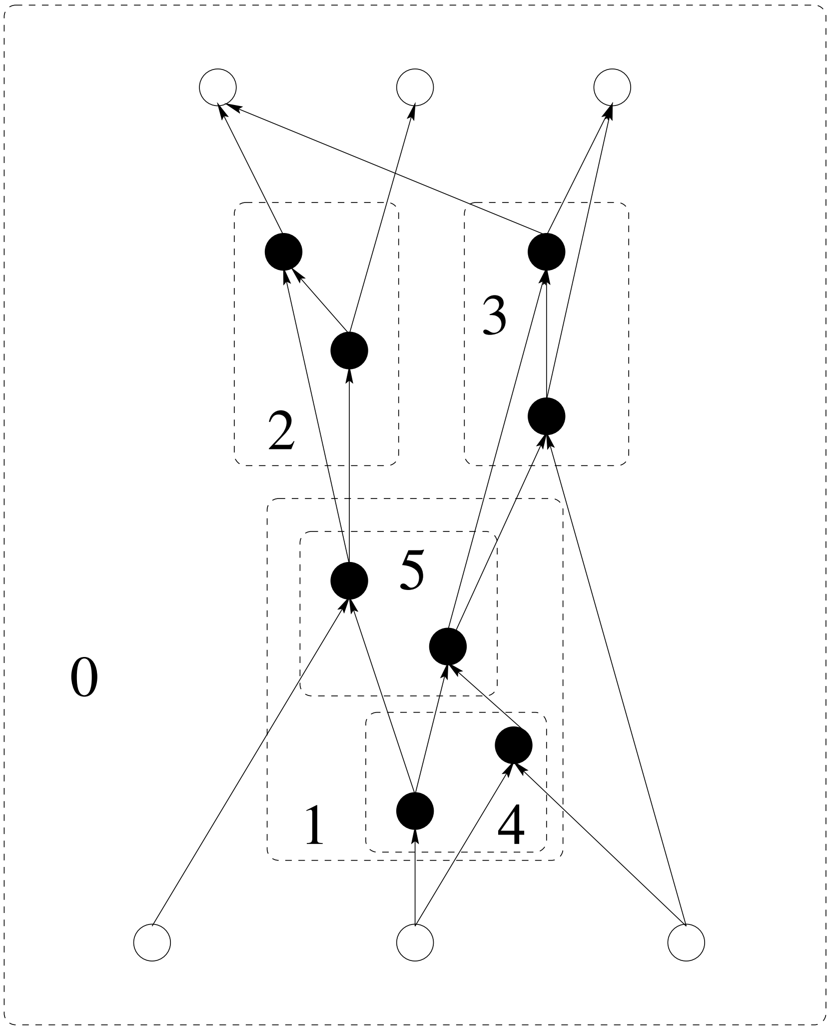
- Default modes

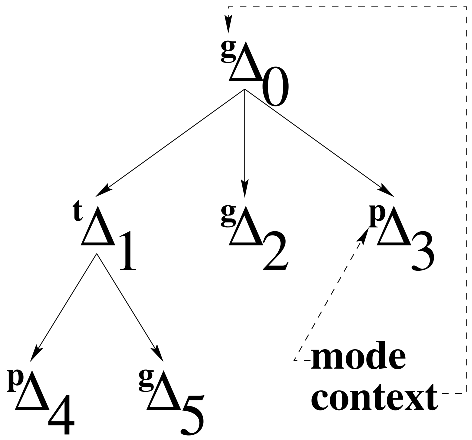
  (GDE as simple AD tool, e.g. forward/reverse mode on head routine $\Rightarrow$ correct derivative code)

User Directives

```
Cbegin_du[du_unit_nr=2;
          indep=[x,d];
          dep=[y];
          in=[a,b,x,d,e,f];
          out=[b,y,z,e,f];
          vector_size=300  (head)
          trj_dbl=1e7; ... (head)
          diff_code="mysubcl";
          jac_code="mysubjac"; ... ]

subroutine sub (a,b,x,y,z)

   parameter (trj_dbl_max=1e7),...
   double precision trj_dbl,...
   common /trj/trj_dbl(trj_dbl_max),...

   double precision a,b,x(10),y(10),z(5)
   double precision d,e,f
   common /globs/d(20),e(100),f(200)
        ...
end

Cend_du[du_nr=2]
...

Cbegin_du[du_unit_nr=3; ...

  call sub(a1,b1,x1,y1,z1)

Cend_du[du_nr=3]
```

How? (GDE)





LJFC step for d-unit 1

LJBC step for d-unit 5

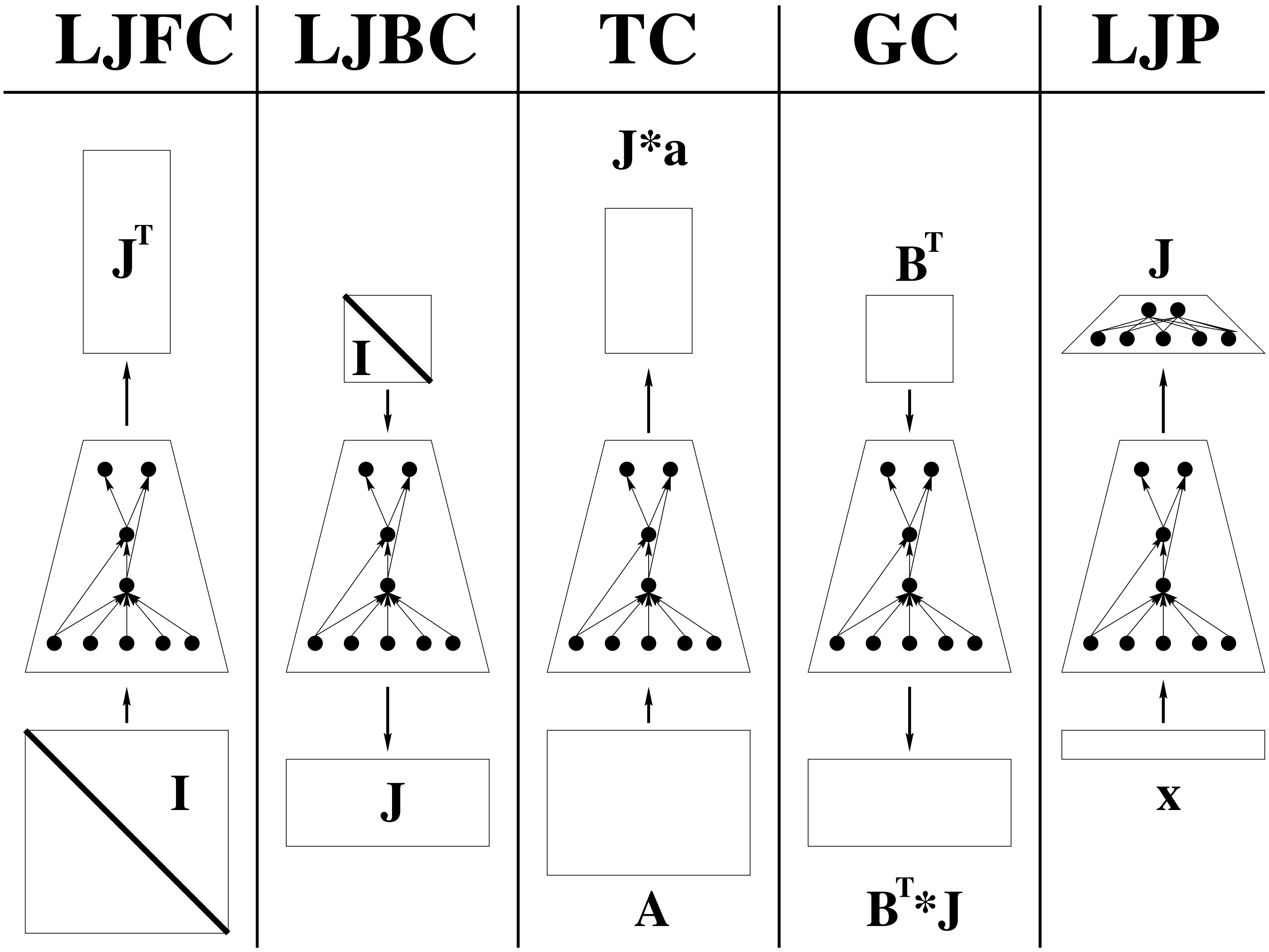
LJP step for d-units 3 and 4

TC step for no d-unit

GC step for d-unit 2

- Differentiation unit hierarchy
- D-unit call graph
- Differentiation mode and context
- Reversal mode

How? (GDE)



- TC, GC
  → implicit Jacobian-vector, vector-Jacobian products
- LJFC, LJBC, LJP
  ⇒ explicit Jacobian-vector, vector-Jacobian products

How? (GDE)

How to use the linearized flow graph of

$$F : \mathbb{R}^n \to \mathbb{R}^m?$$

1. Local "Jacobian" Forward Computation (LJFC)
   (n (or 1) forward sweeps)

2. Local "Jacobian" Backward Computation (LJBC)
   (m (or 1) reverse sweeps)

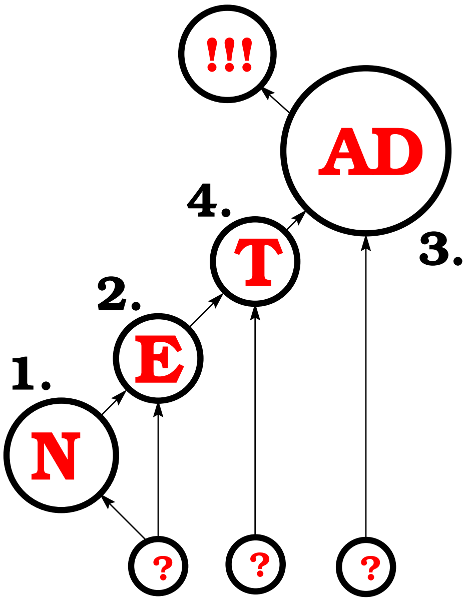
3. "Tangent" Computation (TC)
   (q_A (or 1) forward sweep)

4. "Gradient" Computation (GC)
   (q_B (or 1) reverse sweep)

5. Local "Jacobian" Pre-accumulation (LJP)
   (edge elimination in computational graph)

Uwe Naumann (Uwe.Naumann@sophia.inria.fr)
Action TROPICS (http://www-sop.inria.fr/tropics)

Optimised Edge Elimination Sequences



- pre-accumulated Jacobian $\hat{=}$ linearized bipartite computational graph
- elimination of intermediate vertices (edges), cost = # pred * # succ
- dense FM: 3*5=15, RM: 1*7=7 (mults)
- sparse FM & RM: 7 (mults)
- x-country: 6 (mults)
- edge elimination superior (conjecture)
- heuristics, dynamic programming, simulated annealing
- speedup > 3 for most problems

Uwe Naumann (Uwe.Naumann@sophia.inria.fr)
Action TROPICS (http://www-sop.inria.fr/tropics)

Who will pay for it?

EC: Fifth RTD Framework Programme (IST)
(http://www.cordis.lu/ist/)

- 1.10.1999 - 2nd call
- Key Action IV - Essential Technolgies and Infrastructures
- Action Line IV.3 - Technologies and engineering for software, systems and services, including high-quality statistics
- Action IV.3.1. Component-based Software Engineering
  - *Develop and validate the innovative processes, methods and tools necessary to design, implement and manage software-intensive systems using a component-based approach*
  - *Re-use, incorporation of new technology COTS components and evolutionary re-configuration*
  - *Definition of processes, methods and their supporting technologies that enable the smooth and auditable integration of components from multiple independent sources into complex systems and services, possibly taking advantage of the "system families" concepts*
  - *Work is to be complemented by technology-transfer and best practice initiatives to stimulate both real-life practice improvement and the take-up of the associated technologies*
- fixed deadline: 17.01.2000, 3 month evaluation, one-step submission
- minimum 2 partners
- 50% of personnel cost
- accompagnying measures, trials

Who will do it?

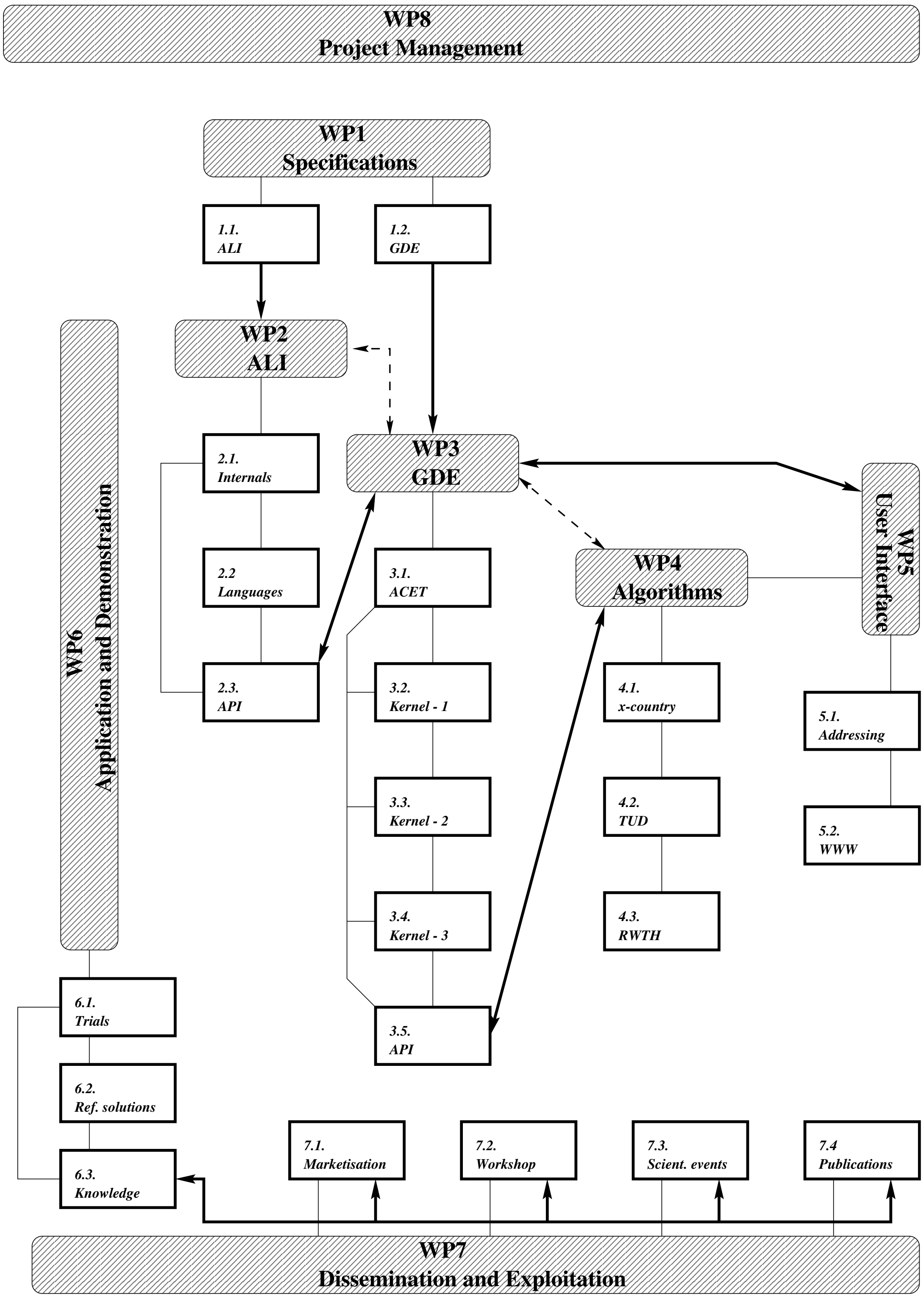
- **Principal contractors:**
  University of Hertfordshire, UK; INRIA, France; RWTH Aachen; TU Dresden, Germany
- **Industrial Partners**
  DERA - British Aerospace, CAD-FEM Siemens, Engel Maschinenbau, Cadoe
- **Sub-contractors**
  Univ of Bayreuth, Germany

When will it happen?

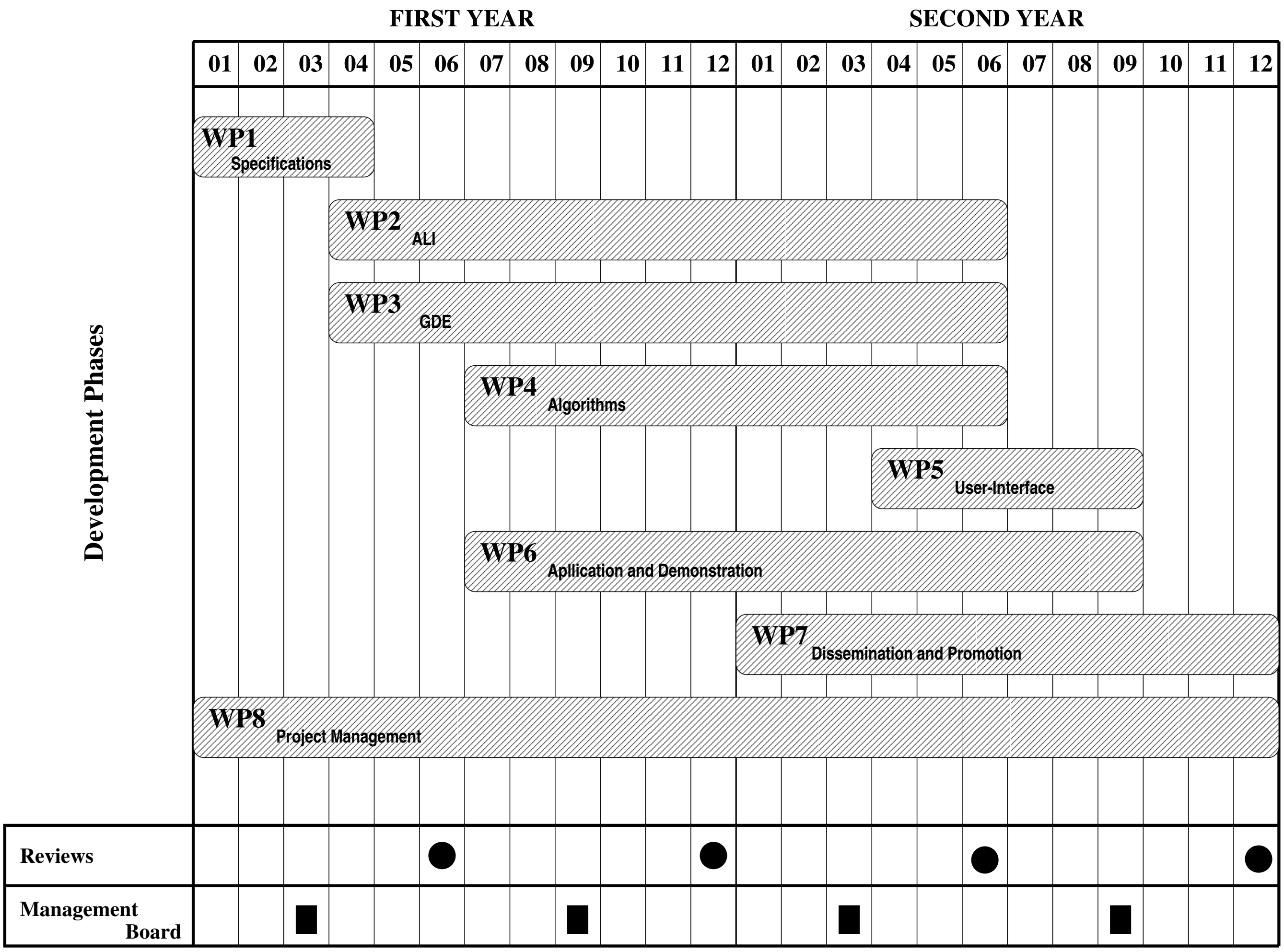
- **January 2001 - December 2002**

→ **Work has already started!**

Workpackages



Timetable


FIRST YEAR
SECOND YEAR
01 02 03 04 05 06 07 08 09 10 11 12 01 02 03 04 05 06 07 08 09 10 11 12
Development Phases
WP1 Specifications
WP2 ALI
WP3 GDE
WP4 Algorithms
WP5 User-Interface
WP6 Aplication and Demonstration
WP7 Dissemination and Promotion
WP8 Project Management
Reviews
Management Board

Application Programmer's View