Formalizing OpenMP Performance Properties with ASL

T. Fahringer (Univ. of Vienna), M. Gerndt (TU Munich),
G. Riley (Univ. of Manchester), J. Träff (NEC Europe)

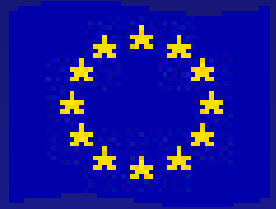
WOMPEI'2000
Yoyogi, Tokyo

October 2000

Outline

- Introduction:
  - Motivation
  - Esprit Apart Work-package 2
- Apart Performance Property Specification Language (ASL)
- Some example shared memory (OpenMP) properties (bottlenecks)
- Ongoing and Future Work

APART



- Esprit IV working group on Automatic Performance Analysis: Resources and Tools

- APART aims to identify

  - requirements for automatic performance analysis support
  - knowledge about typical performance bottlenecks (e.g. for OpenMP programs)
  - base implementation technologies

Apart Members

- European Universities/research centers: Dresden, Juelich, Malaga, Manchester, Muenchen, Pavia, Vienna
- US Universities: Lousiana State, Oregon, Wisconsin-Madison
- European Companies/Labs: Fecit, NEC, Pallas
- and a number of associated partners (academic and industry)

Visit our WebPage

www.fz-juelich.de/apart

Esprit APART WP2: Identification and Formalization of Knowledge

Goal of APART WP2: Support automatic performance-analysis by formalizing

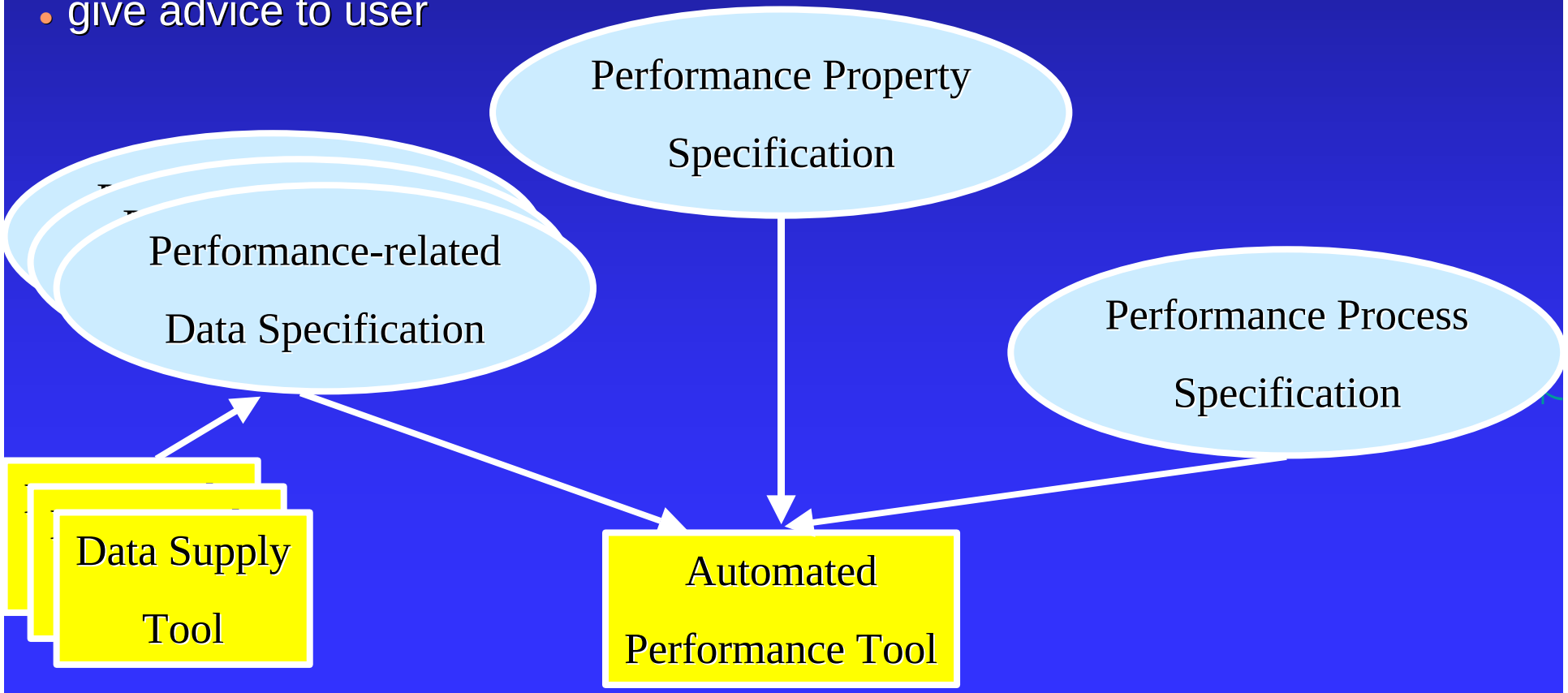
- performance related-data (experiments, programs, machines, etc.)
- performance properties (synchronization, cache misses, load imbalance, etc.)
- performance bottleneck (most severe performance property)
- search process for performance problems

for wide variety of programming paradigms and parallel/distributed architectures.

Design of an Integrated Performance Analysis Environment

Analyse trace or profile of program execution to

- discover performance problems
- classify according to severity
- give advice to user



Performance-related Data Specification

- Performance-related data:
  - static – determined without program execution

    (e.g. program structure, control and data flow information, machine information, predictions, ...)
  - dynamic – based on program execution
    - trace: collection of events, temporal and locality relation
    - profile: statistical summary, partial temporal and locality information

- Key issue: well-suited representation of performance related data and performance properties
- APART Performance Property Specification Language (ASL)

APART Performance Property Specification Language (ASL)

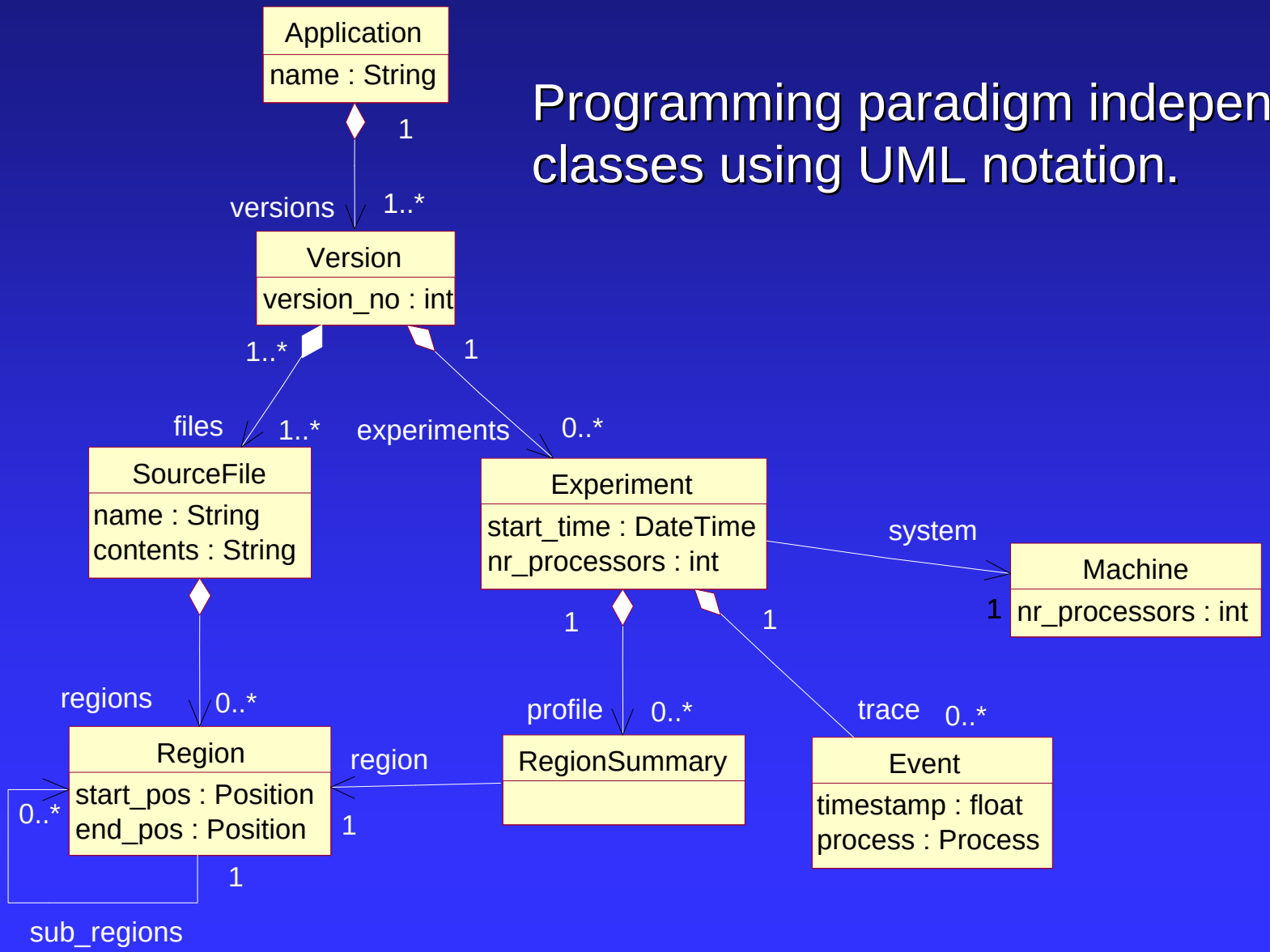
- **Performance related data**
  - object-oriented representation
  - single inheritance
  - no methods
- **Performance properties**
  - functions and constraints defined over performance-related data.

- ASL covers typical structure of programs and architectures without claiming completeness.

- Applied ASL to OpenMP, HPF, and MPI

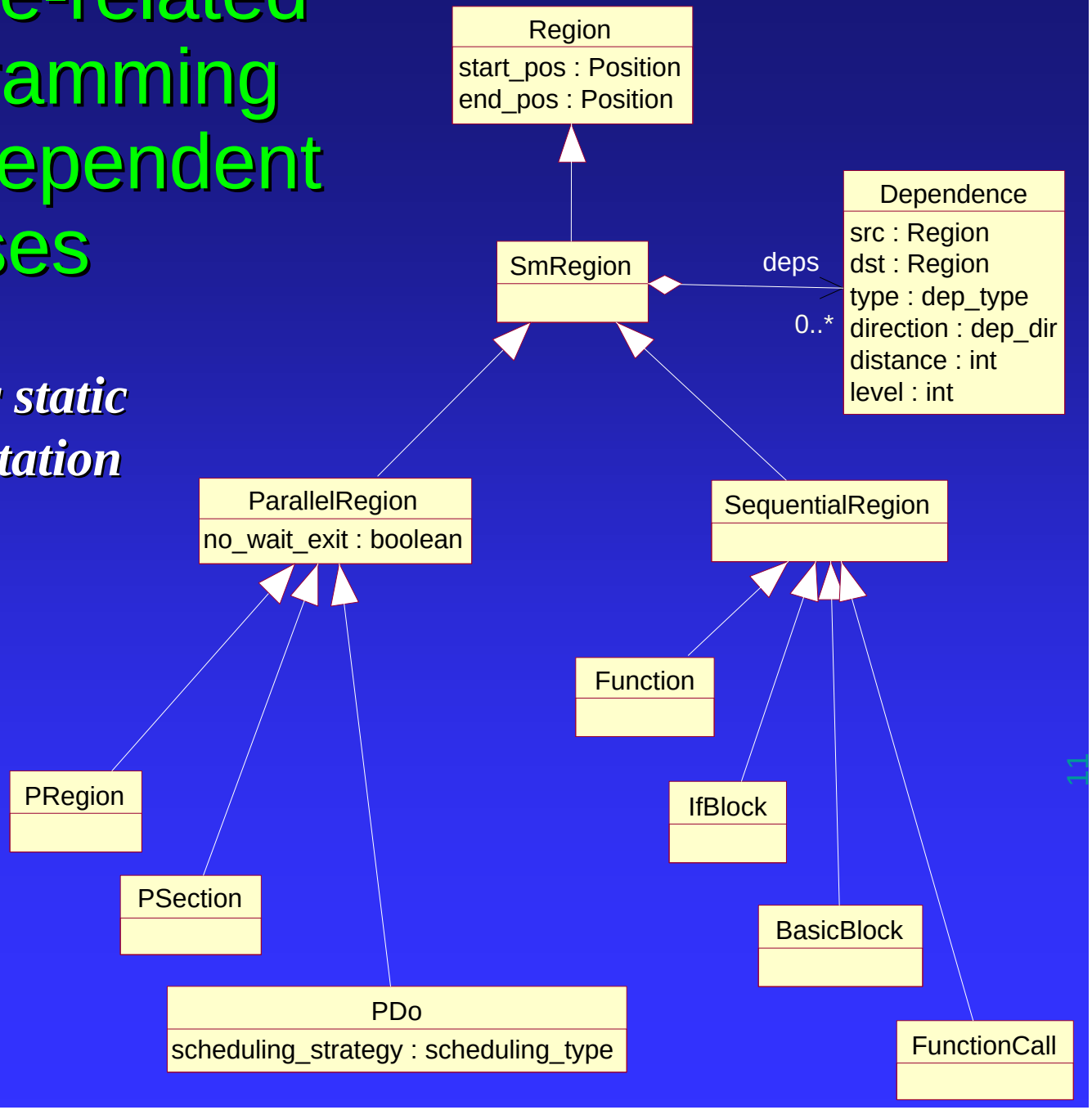
Performance-related Data: Base Classes

Programming paradigm independent classes using UML notation.



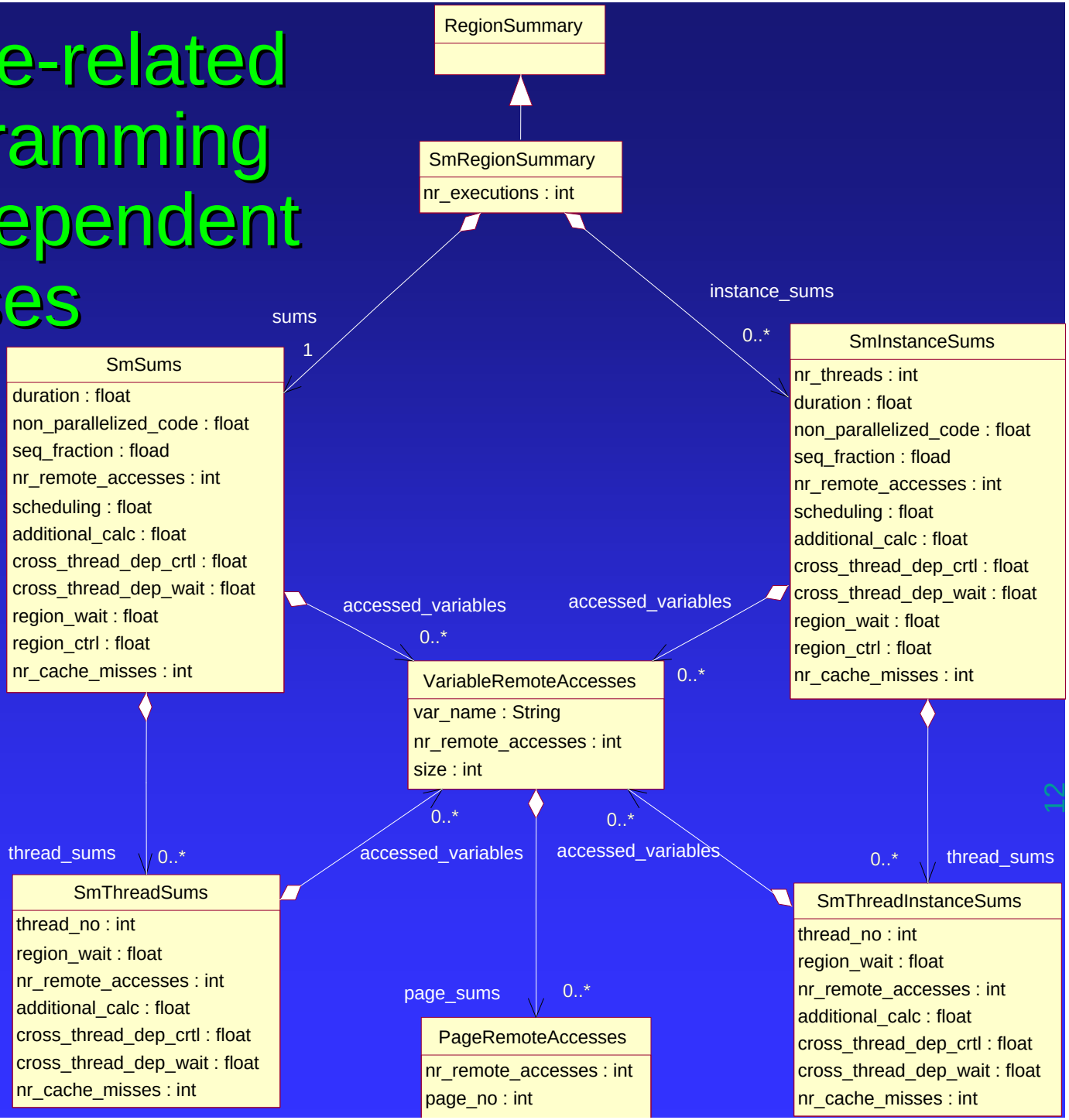
Performance-related Data:Programming Paradigm Dependent Classes

OpenMP classes for static data using UML notation



Performance-related Data: Programming Paradigm Dependent Classes

OpenMP dynamic performance-related data



Performance Property Specification: APART Specification Language ASL

- **Performance property** characterizes a specific performance behavior (communication, load imbalance, cache misses, etc.) of a program.
  - **Conditions:** check for existence of performance property
  - **Confidence:** degree of certainty of property presence [0,1]
  - **Severity:** importance of property

- Combined specification of performance data and properties with ASL

performance-property-spec is **PERFORMANCE DATA**
*class-def **
PERFORMANCE PROPERTIES
[LET
*def **
IN]
*property **
END

ASL – APART Specification Language

- ASL is a language for describing performance data, and specifying performance properties.
- Performance data: object oriented, single inheritance, no methods
- built-in types: int, float, ..., enum, setof
- auxiliary functions for extracting summary information for a region from an experiment:

```
SmRegionsummary summary(SmRegion r, Experiment e) =
UNIQUE({sumr IN e.profile WITH sumr.region== r});
```

- auxiliary functions to extract total time spent in a region:

```
float duration(SmRegion r, Experiment e) =
summary(r,e).sums.duration;
```

Example OpenMP Performance Property: Load Imbalance

```
Property load_imbalance (SmRegion r, Experiment e, Region
rank_basis) {

LET

    float cost = summary(r,e).sums.region_wait;

IN

    CONDITION: cost > 0;

    CONFIDENCE: 1;

    SEVERITY: cost/duration(rank_basis,e);

}
```

Example OpenMP Performance Property: Irregular synchronization across instances

Property irregular_sync_across_instances (SmRegion r,

Experiment e, Region rank_basis) {

LET float inst_sync(SmInstanceSums sum) = sum.region_wait +
sum.region_ctrl + sum.cross_thread_dep_wait +
sum.cross_thread_dep_ctrl;

IN CONDITION: stdev(inst_sync(inst_sum) WHERE inst_sum IN
summary(r,e).instance_sums) >

irreg_behavior_threshold * sync(r,e)/r.nr_executions;

CONFIDENCE: 1;

SEVERITY: sync(r,e)/duration(rank_basis,e);}

Example OpenMP Performance Property:Remote Variable Accesses

```
Property remote_access_to_variable (SmRegion r, Experiment e,
String var, Region rank_basis) {


LET    VariableRemoteAccesses var_sum = UNIQUE({info IN
summary(r,e).sums.accessed_variables WITH info.var_name==var});


IN   CONDITION : var_sum.nr_remote_accesses > 0;
     CONFIDENCE: 1;
     SEVERITY:  var_sum.nr_remote_accesses *
                e.system.remote_access_time/duration(rank_basis,e);
}
```

Ongoing and Future Work

- **Case-studies**
  - Cray T3E
  - SGI Origin 2000
  - SMP Clusters
- **Language extensions**
  - Mechanism for specifying patterns in traces
  - Templates for expressing similar performance properties
  - Metaproperties to specify properties based on existing ones.
- Formalize **search process for bottlenecks** based on ASL (APART WP2 and WP3).
- **Local projects** to implement concepts of Apart in Munich, Manchester, Juelich, and Vienna.
- More info about APART: www.kfa-juelich.de/apart