Approximating Quality Contracts for Energy Auto-Tuning Software

Sebastian Götz, Claas Wilke, Sebastian Richly and Uwe Assmann

03.06.2012

GREENS 2012, Zürich, Switzerland
Context
- Prediction of (component-based) software’s non-functional properties (NFPs)
- QoS Contracts used to specify non-functional behavior
- Self-Optimization w.r.t. energy consumption (and other NFPs)
  - Energy / Multi-Quality Auto-Tuning (EAT / MQuAT)

Problem
- Derivation / Computation of QoS Contracts for EAT/MQuAT
- Concrete values of NFPs depend on
  - Utilized hardware
  - User request

Goal/Solution
- 3-Phase approach:
  1. contract template (hardware- and user-independent),
  2. contract (user-independent),
  3. contract instance
Approach: Quality Contract Creation

**Phase 1: Design Time**
- **1. Application Development**
  - Contract Template
  - Code
- **2. Benchmark Development**
  - Benchmark
  - Parameter Metadata

**Phase 2: Deployment Time**
- **4. Approximation**
  - Measurement Results
  - Available Resources
- **5. Instantiation**
  - User Request

**Phase 3: Runtime**
- Contract Instance

Legend:
- Activity
- Information

Design Time: Available Resources Unknown
Deployment Time: User Demand Unknown
Runtime
Approach: Quality Contract Creation

1. Application Development
2. Benchmark Development

Design Time:
Available Resources
Unknown

Deployment Time:
User Demand

Deployment Time:
User Demand

Runtime

contract HeapSort implements Sort.sort {
  /* Quality mode for fast CPUs. */
  mode fast {
    requires resource CPU {
      min frequency: 1 * cpu.frequency.max [GHz];
      min time: <f_cpu_time>(size_of_list) [ms];
    }
    provides min response_time:
      <f_response_time>(size_of_list) [ms];
  }
  /* Quality mode for slower CPUs. */
  mode slow {
    requires resource CPU {
      min frequency:
        0.4 * cpu.frequency.max [GHz];
      min time: <f_cpu_time>(size_of_list) [ms];
    }
    provides min response_time:
      <f_response_time>(size_of_list) [ms];
  }
}
**Approach: Quality Contract Creation**

```
contract HeapSort implements Sort.sort {
  /* Quality mode for fast CPUs. */
  mode fast {
    requires resource CPU {
      min frequency: 2.0 [GHz];
      min time: 1.147*10^(-6)*size_of_list^2-1922 [ms];
    }
    provides min response_time:
      2.152*10^(-6)*size_of_list^2-1917 [ms];
  }
  /* Quality mode for slower CPUs. */
  mode slow {
    requires resource CPU {
      min frequency:
        0.8 [GHz];
      min time: 1.552*10^(-6)*size_of_list^2-1821 [ms];
    }
    provides min response_time:
      3.552*10^(-6)*size_of_list^2-1901 [ms];
  }
}
```
Approach: Quality Contract Creation

1. Application Development
2. Benchmark Development

Design Time:
Available Resources
Unknown

Deployment Time:
User Demand

Runtime

Phase 1: Design Time

//size_of_list = 500.000
contract HeapSort implements Sort.sort {
  /* Quality mode for fast CPUs. */
  mode fast {
    requires resource CPU {
      min frequency: 2.0 [GHz];
      min time: 47 [ms];
    }
    provides min response_time:
    104 [ms];
  }
  /* Quality mode for slower CPUs. */
  mode slow {
    requires resource CPU {
      min frequency:
      0.8 [GHz];
      min time: 140.2 [ms];
    }
    provides min response_time: 244 [ms];
  }
}

Phase 2: Deployment

Time
Available Resources

Phase 3: Runtime

User Request

Contract Instance

Legend

Activity
Information
1. Are there further dependencies not covered by the presented approach?

2. Does it make sense to *directly cover energy consumption* in QoS contracts or is it better to *compute the potential consumption* based on the derived resource utilization?

3. Measureability of NFPs
   - Minimum runtime requirement – OS tasks and Hardware’s SMM introduce deviations of >20ms.
   - Reproducability – How to determine a sufficient amount of context factors to consider for a benchmark setup?
Contact

http://www.inf.tu-dresden.de/~sebgoetz
sebastian.goetz@acm.org