Multi-Quality Auto-Tuning by Contract Negotiation

Verteidigung der Dissertation von Dipl.-Inf. Sebastian Götz

Betreuer: Prof. Dr. rer. nat. habil. Uwe Aßmann
Zweitgutachter: Prof. Dr. rer. nat. habil. Heinrich Hußmann
Fachreferent: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

17.07.2013
**Example:** Audio-Processing (https://auphonic.com/)

Qualities, Quality of Service (QoS), Non-functional Properties (NFPs)

- **Noise Reduction**
- **Generate Sound Effects**
- **Leveler**
- **Loudness Adjustment**
- **Configuration**

`audio file`
Goal: Self-adaptive Systems (SAS)

Robert Laddaga 1997:

"Self Adaptive Software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible." [L97]
Which variant of which software should be used?

How good is each variant in comparison to the others?

How to achieve the best possible user satisfaction for the least possible cost?

Which resources should be utilized?
**Motivation**

- **User objectives relate to qualities**: energy, performance, domain-specific qualities as noise-levels, etc.

- Often **multiple, competing qualities** are to be considered in combination [ST09]

---

A novel approach to **design & operate self-optimizing systems** covering **multiple objectives**.

**Multi-Quality Auto-Tuning (MQuAT)**
**Problem 1:** Developers *cannot reuse solutions* to build self-optimizing systems although many specific approaches exist.

- Fixed set of considered properties (e.g., bandwidth, response time)
- Fixed architecture (e.g., specific to servers, mobile phones or cars)
- Fixed optimization technique (e.g., integer linear programming)

**Goal:** A generic approach to self-optimizing systems.

**Solution:** A *model-driven development approach* to self-optimization

- A *component-based metamodel* enabling the developer to specify the properties of interest and the system’s architecture.
- **Technology bridges** to utilize multiple optimization techniques (generation of optimization problems).
Optimization Problem Description

Data-flow Graph

- Noise Reduction
- Generate Sound Effects
- Leveler
- Synchronization

Tree

Machine #1
- Arm
- Leg
  - CPU
  - RAM
  - Net

Machine #2
- Board #1
  - CPU
  - Net
  - RAM

Machine #3
  - CPU
  - RAM
  - Net

17.07.2013
Multi-Quality Auto-Tuning by Contract Negotiation
Slide 7
Problem 2: Existing (specific) approaches do not cover dependencies between qualities.

- Quality-contract-based approaches
  - COMQUAD → QoS characteristics (e.g., \( \text{response\_time} < 5\text{ms} \)) [RZ03]
  - THESEUS → SLAs; QoS intervals (e.g., \( 2\text{ms} < \text{response\_time} < 5\text{ms} \)) [S10]
  - No context-dependent QoS statements (e.g., \( \text{response\_time(size)} = f(\text{size}) \))
  - Both projects identified the need to cover QoS dependencies [ZM03, S10]

Goal: Explicit coverage of (context-dependent) interaction between qualities.

Solution:
- An extended notion of quality contracts and
- A process for quality contract refinement.
Problem 3: Competing qualities demand for **multi-objective optimization** having a **high computational complexity** (NP-hard) [NW99]

- Multi-objective approaches (e.g., OCTOPUS)
  - “a priori”: aggregation of objectives prior to optimization
  - “a posteriori”: optimization delivers set of multi-dimensional solutions (Pareto front)

- **Optimization at runtime** requires feasible, assessable time requirements

**Goal**: A generic, *assessable* runtime multi-objective optimization approach.

**Solution**:
- 4 runtime **technology bridges** to multi-objective optimization techniques.
- **Scalability analysis** of supported techniques.
Problem 3: Competing qualities demand for **multi-objective optimization** having a **high computational complexity** (NP-hard) [NW99]

- Multi-objective approaches (e.g., OCTOPUS)
  - "a priori": aggregation of objectives prior to optimization
  - "a posteriori": optimization delivers set of multi-dimensional solutions (Pareto front)
- **Optimization at runtime** requires feasible, assessable time requirements

**Goal**: A generic, **assessable runtime** multi-objective optimization approach.

**Solution**:
- 4 runtime technology **bridges** to multi-objective optimization techniques.
- **Scalability analysis** of supported techniques.
Overview

Part 1: Development
- Models
- QoS Contracts
- Code
- Benchmarks
- CCM
- QCL
- Refinement

Part 2: Runtime System
- Running Components
- Runtime Model
- Runtime Optimization (Contract Negotiation)
- Multiple Objective Function Computation

Requirements:
- Users: Request
- Multiple Objectives

Genericity / Reuse
QoS Dependencies
Runtime MOO

硬件基础设施
PART 1: DEVELOPMENT
Cool Component Model

SAS Layer
- Requests
- Reconfigurations
- Workloads

Core Layer
- Quality Contract Language
- <refined by>>
- Structure Models (i.e., types)
- <<instance of>>
- Cool Component Model
- Variant Models (i.e., instances) \(\rightarrow\) runtime
- <<enrich>>
- Behavior Models
- Cool Component Model

Base Layer
- Expressions
- Units
- DataTypes

17.07.2013
Multi-Quality Auto-Tuning by Contract Negotiation
Slide 13
### Example CCM Structure Model for Servers:

<table>
<thead>
<tr>
<th>&lt;&lt;container&gt;&gt; Server</th>
<th>1..*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong> 1..*</td>
<td></td>
</tr>
<tr>
<td>clock_rate : GHz</td>
<td></td>
</tr>
<tr>
<td>performance : FLOP/s</td>
<td></td>
</tr>
<tr>
<td>cpuLoad : Percent</td>
<td></td>
</tr>
<tr>
<td>cpu_time : Second</td>
<td></td>
</tr>
<tr>
<td><strong>Net</strong> 1..*</td>
<td></td>
</tr>
<tr>
<td>bandwidth : Mb/s</td>
<td></td>
</tr>
<tr>
<td><strong>RAM</strong> 1..*</td>
<td></td>
</tr>
<tr>
<td>free : GB = total – used</td>
<td></td>
</tr>
<tr>
<td>used : GB</td>
<td></td>
</tr>
<tr>
<td>total : GB</td>
<td></td>
</tr>
<tr>
<td>throughput : GB/s</td>
<td></td>
</tr>
<tr>
<td><strong>DbxCard</strong> 1..*</td>
<td></td>
</tr>
<tr>
<td>time : Second</td>
<td></td>
</tr>
<tr>
<td>threshold : dB</td>
<td></td>
</tr>
<tr>
<td>amplification : dB</td>
<td></td>
</tr>
</tbody>
</table>

### Example Unit Library

```java
library {
    simple unit Watt : Integer
    simple unit Second : Integer;
    simple unit dB : Real;

    complex unit Joule = Watt Second;

    factor KW = 1000 Watt;
}
```

### Example CCM Structure Model for Sort:

```java
apply NoiseReduction

<meta> audio_length : Second

response_time : Second
noiseReductionLevel : dB
```
Quality Contract Language [GWC+12a]

```plaintext
1 contract Dbx implements NoiseReduction.apply {

2    
3        mode professional {
4            requires component SpecialNoiseReduction {
5                min capability: 100 [percent]
6            }
7        }
8
9        requires resource DbxCard {
10            min <time>(audio-length) [ms]
11            }
12
13        provides min noiseReductionLevel: 25 dB
14        provides min <response_time>(audio_length) [s]
15    }
16
17    mode amateur {
18        /* More requirements and provisions here ... */
19    }
20 }
```
• Target systems and user input are unknown to developer.
• Developer creates **contract templates**:

```java
contract Dbx implements NoiseReduction.apply {
    mode professional {
        ...
        provides min response_time:
            <response_time>(audio_length) [s];
    }
    ...
}
```

• Developer creates **Benchmark Suite** using **Profiler Framework** [WGR13]

```java
for(i = 0; i <= N; i++) {
    Profiler.getProfiler(„response_time”).start();
    dbx.apply(sample_files[i]);
    Profiler.getProfiler(„response_time”).stop();
}
```
• Target systems and user input are unknown to developer.
• Developer creates **contract templates**:

```
contract Dbx implements NoiseReduction.apply {
  mode professional {
    ... provides min response_time:
    <response_time>(audio_length) [s];
  }
  ...
}
```

• Benchmarks executed at **deployment** time on each **target machine**:

<table>
<thead>
<tr>
<th>audio_length</th>
<th>response_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s</td>
<td>945ms</td>
</tr>
<tr>
<td>2s</td>
<td>1823ms</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>120s</td>
<td>110215ms</td>
</tr>
</tbody>
</table>

\[
<\text{response_time}>(\text{audio_length}) \ [s] \\
1.147 \times 10^{-6} \times \text{audio_length}^2 - 1922 \ [s];
\]

**One contract per machine and implementation.**
PART 2: RUNTIME
... denotes a **global optimization problem** of a system of **components**, which are **known and controllable** by **central coordinators** as known from the self-adaptive system’s community.
• **Base:** Integer Linear Programming (ILP)

![Diagram](image)

• **Goal:** determine the variable assignment, which
  - Maximizes objective function and
  - Adheres to the constraints.

• Avoids pruning of whole search space (worst case)
• Integer Linear Programming (ILP)

CCM Variant Model
Runtime Description of Hard- & Software Infrastructure

CCM Structure Model
Architecture of Hard- & Software System

CCM Behavior Models

QCL Contracts
Characterizing Non-functional Behavior of Implementations

ILP

Decision Variables

Map to HW
Select Impl.

Usage Variables

Objective Function

Constraints

NFP Requirements

NFP Provisions

Resource Requirements

Resource Provisions
fixed

Architectural Constraints

Knapsack

Knapsack

17.07.2013 Multi-Quality Auto-Tuning by Contract Negotiation Slide 21
/* objective function: minimize energy consumption (based on cpu_time) */

\[
\text{min: } 5700.0 \text{ b#Quicksort#delayed#R1} + 495.0 \text{ b#UnsortedFilter#slow#R1} \\
+ 10285.0 \text{ b#Quicksort#immediate#R1} + 6160.0 \text{ b#Javasort#immediate#R1} \\
+ 385.0 \text{ b#UnsortedFilter#fast#R1} + 2250.0 \text{ b#Random#slow#R1} \\
+ 5940.0 \text{ b#Javasort#delayed#R1} + 2695.0 \text{ b#Random#fast#R1};
\]

/* architectural constraints */

\[
\text{b#Random#fast#R1} + \text{b#Random#slow#R1} = \text{b#Quicksort#delayed#R1} + \text{b#Quicksort#immediate#R1} \\
+ \text{b#Javasort#immediate#R1} + \text{b#Javasort#delayed#R1};
\]
\[
\text{b#UnsortedFilter#fast#R1} + \text{b#UnsortedFilter#slow#R1} = 1;
\]
\[
\text{b#Quicksort#immediate#R1} + \text{b#Quicksort#delayed#R1} \\
+ \text{b#Javasort#immediate#R1} + \text{b#Javasort#delayed#R1} = \text{b#UnsortedFilter#slow#R1} \\
+ \text{b#UnsortedFilter#fast#R1};
\]

/* resource negotiation */

\[
\text{usage#R1#Core[TM]_i7_CPU_Q_720_@_1.60GHz#frequency} \leq 1596.0;
\]
\[
\text{usage#R1#Core[TM]_i7_CPU_Q_720_@_1.60GHz#frequency} \geq 0;
\]
\[
\text{usage#R1#Core[TM]_i7_CPU_Q_720_@_1.60GHz#frequency} = \\
100 \text{ b#Javasort#delayed#R1} + 100 \text{ b#UnsortedFilter#slow#R1} + 100 \text{ b#Quicksort#delayed#R1} \\
+ 300 \text{ b#Random#fast#R1} + 300 \text{ b#Quicksort#immediate#R1} + 100 \text{ b#Random#slow#R1} \\
+ 300 \text{ b#Javasort#immediate#R1} + 300 \text{ b#UnsortedFilter#fast#R1};
\]

...
... 

/* software NFP negotiation */
Sort#response_time = 382.05714282441 b#Quicksort#delayed#R1
+ 377.31428570997804 b#Quicksort#immediate#R1
+ 399.771428570494 b#Javasort#immediate#R1
+ 416.34285718949195 b#Javasort#delayed#R1;
Filter#response_time = 23.921216866850248 b#UnsortedFilter#slow#R1
+ 28.407017552658598 b#UnsortedFilter#fast#R1;
ListGen#response_time = 107.6078431285458 b#Random#slow#R1
+ 106.7843137012918 b#Random#fast#R1;
Sort#response_time >= 50 b#UnsortedFilter#fast#R1;
ListGen#response_time >= 50 b#Quicksort#delayed#R1 + 50 b#Javasort#immediate#R1
+ 50 b#Javasort#delayed#R1;

/* user request */
Filter#response_time <= 200.0;

/* boolean restriction */
binary b#Quicksort#delayed#R1, b#UnsortedFilter#slow#R1, b#Quicksort#immediate#R1,
b#Javasort#immediate#R1, b#UnsortedFilter#fast#R1, b#Random#slow#R1,
b#Javasort#delayed#R1, b#Random#fast#R1;
Klein und Hannan ´82

Pareto Front

Solution

Variables

Objective 1
Objective 2
Objective 3
Constraints

Solution

Variables

Objective 1
Objective 2
Objective 3
Constraints

Solution

Variables

Objective 1
Objective 2
Objective 3
Constraints

Derived Constraints

Derived Constraints
Contract Negotiation by MOILP

Pareto Front

Quadratic Growth until Termination
Scalability Analysis [GWR+13]

- Performed on **data-flow graphs (pipe-and-filter style)**

<table>
<thead>
<tr>
<th>C components</th>
<th>S servers</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

- Measurements taken for $C \times S$ systems from $C = [2..100]$ and $S = [2..100]$

- **All measurements made on Alienware X51** (Win7 64bit, SSD HDD, 8GB DDR1600 RAM, Intel Core i7-2600 with 4 physical cores at 3.4GHz)

- Concrete numbers will differ on other machines, solvers, etc.

- Focus on **principle findings**.
Scalability Analysis: ILP [GWR+13]

Solving Time [ms]

Timeout: 2min

Feasible up to 100x100

Predictable up to 25 Components

Reason: Worst-case situations

3rd Quartile: 26,58s

17.07.2013 Multi-Quality Auto-Tuning by Contract Negotiation
Solving Time for 2 Objective Functions

3rd Quantile: 62.92 s

The jump is due to heuristics in solver.
Scalability Analysis: MOILP

Size of Pareto Front for 2 Objective Functions

Large Pareto-fronts even for small systems

Number of Components

Size of Pareto Front

Objective 1

Objective 2

Number of Servers

Large Pareto-fronts even for small systems

17.07.2013
Multi-Quality Auto-Tuning by Contract Negotiation
Slide 29
Solving Time for 3 Objective Functions

Infeasible due to quadratic explosion.
Contributions

Part 1: Development
- Genericity / Reuse
- QoS Dependencies

Models
- CCM
- QCL
- Refinement

QoS Contracts
- Code
- Benchmarks

Part 2: Runtime System
- Running Components
- Runtime Model
- Runtime Optimization (Contract Negotiation)

- Multiple Objective Function Computation

Users
- Request
- Multiple Objectives

Developer

17.07.2013 Multi-Quality Auto-Tuning by Contract Negotiation Slide 31
Part 1: Development

Developers are not restricted to prescribed non-functional properties and architectural elements.

Context-dependent interdependencies of multiple qualities are supported.

Realized and analyzed four runtime MOO techniques as technology bridges.

Part 2: Runtime System

Genericity / Reuse

QoS Dependencies

Runtime MOO

Multi-Quality Auto-Tuning by Contract Negotiation

17.07.2013

Contributions
Future Work

- **Bootstrapping**: MQuAT for Monitoring, Optimization and Reconfiguration
  - Both are components with different implementations, too.
  - Scalability analysis is a first step for the optimization component
  - Collaboration planned with Prof. Fischer (Numerical Optimization Group)

- **Green Software Engineering** (CRC 912: HAEC, NFG ZESSY)
  [WGR+11, WRP+12, WRP+13, WGR13, GMT+13, WRG+13a, WRG+13b]
  - *Open Challenges*: Sustainability, Negotiation of Energy-Sources (Solar, Battery, Provider, etc.)

- **Software Engineering for Robotic and Cyber-Physical Systems**
  [GLR+11, GLP+12, PRG+12]
  - *Open Challenge*: Optimization across discrete and continuous system parts


