Applying Concession-based Negotiation to Architectural Tradeoffs in SQuAT

Andrés Díaz-Pace, Alejandro Rago, Santiago Vidal, Sebastian Frank, and André van Hoorn

8th SPP
Karlsruhe, Germany
November, 2017
Brief introduction to SQuAT project

Exploration of PCM architectures for quality attributes of performance and modifiability → tradeoffs

PCM case-study

Search (only) strategy

Search + negotiation strategy

Initial results & comparison of strategies

Improvements & future work

Progress since SSP 2016

https://github.com/SQuAT-Team

Agenda

Based on a presentation given at SPP 2016
Motivation

A group decision-making problem

• Difficulties to manage and optimize all QAs in a satisfying way, i.e., make everybody happy

• **Monolithic** is unnatural

• Tradeoffs between different QA goals & negotiation among experts in the architecture design process

---

We must process 10 million transactions per sec.

If the database changes, we must be prepared (less than 1 day-effort)

The system must not fail if a server goes down

We have deadlines and we are running out of money

I want the system to be fast, adaptable, fail-proof, secure and cheap
The SQuAT Project

A distributed, multi-objective architecture optimization problem

If the database changes, we must be prepared (less than 1 day-effort)

We must process 10 million transactions per sec.

Heuristic Search & Negotiation algorithms

CANDIDATE (OPTIMIZED) ARCHITECTURES

SSP 2017 - Applying Concession-based Negotiation to Architectural Tradeoffs
What’s new since 2016?

*Focus on tooling, evaluation and negotiation*

- Implementation of **Monotonic Concession Protocol**
  - Basic implementation (utility function, egocentric concession)
  - Integration with heuristic search (for candidate architectures)
- Configuration of Dbots
  - **Modifiability**: KAMP extension + definition of modifiability tactics
  - **Performance**: PerOpteryx + identification of performance tactics
- **Initial case-study**: **SimpleTactics**+
  - A trip management system modeled in Palladio
  - Initial space of architectural candidates
To explain and exercise SQuAT, we use a system called IBusinessTripMgmt, with a focus on its search and negotiation strategies. In Section 4, we run a number of experiments on a given architecture, each dbot searches different dbots, or reimburses. The component is an alternative exporter, which can be replaced by the alternative exporter. The database can export in-storage the infrastructure, comprising servers interconnected via network connections. The rest of the paper is organized as follows. Section 2 introduces the system entry point for the users. Section 3 presents the best one according to a global, user-centric analysis of solution tradeoffs. The database can export information about trips and payments as a PDF file. Section 4 discusses analysis, or they could be combined with solutions from other dbots and lead to more interesting results. Each dbot might internally compute and discard many solutions, even if the architectural knowledge into separate components, called dbots, based on performance considerations. The database can export or reimbursements. The component is an alternative exporter, which can be replaced by the alternative exporter. The database can export information about trips and payments as a PDF file. The rest of the paper is organized as follows. Section 2 introduces the system entry point for the users. Section 3 presents the best one according to a global, user-centric analysis of solution tradeoffs. The database can export in-storage the infrastructure, comprising servers interconnected via network connections. The rest of the paper is organized as follows. Section 2 introduces the system entry point for the users. Section 3 presents the best one according to a global, user-centric analysis of solution tradeoffs. The database can export in-storage the infrastructure, comprising servers interconnected via network connections.
This tactic might cause a performance drop, since there is another scenario. These scenarios are further detailed in Section 3.

Alternative solutions that improve both scenarios. Thus, the architect would like to explore (with assistance from tool support) alternative solutions that improve both quality-attribute requirements for performance and modifiability.

From a performance perspective, possible tactics for the architect are captured by means of a repository (cf. Figure 2) and presented to the user for a (global) assessment of tradeoffs.

Candidate A

M1: Complexity: 110.5
P1: Response time: 119.6 ms

Initial Architecture

M1: Complexity: 163
P1: Response time: 60 ms

Candidate B

M1: Complexity: 123
P1: Response time: 8.3 ms

In the example, we assume that the (initial) ST+ architecture has been created, the dbots apply a recommendation strategy that looks for can-

SSP 2017 - Applying Concession-based Negotiation to Architectural Tradeoffs in SQuAT
List of available Tactics (so far)
Adapted to PCM representation

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Modifiability</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Split component:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Divide interface and component, e.g., in case of low cohesion.</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Add wrapper/intermediary:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wrap a component by a façade, e.g., to add functionality.</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Component selection:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection of alternative components that provide and require the same interface(s).</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Component re-deployment:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Re-deploy in case load is not evenly distributed or many remote calls are detected.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>Resource scaling:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change processing rate in case high resource utilization is detected.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>Moving distributed components:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move components to a single server in case many remote calls are detected</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td><strong>Change passive resources:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity of passive resources are changed.</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td><strong>Selection, mutation, crossover:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default genetic operations based on automatically determined degrees of freedom.</td>
<td>(Y)</td>
<td>(Y)</td>
</tr>
</tbody>
</table>

- There are interdependencies among tactics (and PCM models)
The role of the Human Architect

*Mixed-initiative schema for SQuAT*

- **At configuration time**
  - Provide initial architecture
  - Define scenarios and map them to architecture
  - Configure dbots with tactics (default)
  - Configure utility function & concession strategy (default)
  - Set k levels of search (default = 2)

- **At runtime**
  - Run the search & negotiation algorithm
  - Inspect candidate solutions returned by SQuAT
  - Pick a given candidate as the initial architecture, and run algorithm again
Improving Search with Negotiation

Looking for “interesting” tradeoffs rather than exhaustive search

1. HEURISTIC SEARCH, using tactics as operators (k levels of depth in design tree)

2. NEGOTIATION based on MCP

• How to provide alternative tradeoffs based on Dbots’ preferences over the candidate solutions?
The Monotonic Concession Protocol

Try to mimic how humans negotiate about items

- Originally proposed by Endriss for agents (2006)
- Previous work on group recommendation for movies

Architectures are more complex items than movies

Start with $arq_{ini}$

1. Each agent searches for architectural solutions in multiple levels

2. Each agent makes an initial proposal (architectural solution)

3. Check for agreement of all agents

Agreement criterion (A)

Conflict

Deal

4. Determine agent(s) that must concede

Concession decision rule (B)
(Zeuthen, product-increasing, etc.)

5. An agent makes a concession (i.e. Proposes a different architectural solution)

Concession strategy (C)
(Egocentric)

Some agent can concede

No agent can concede

End with agreement ($arq_{neg}$ with a different tradeoff)

End with conflict

Negotiation process

SSP 2017 - Applying Concession-based Negotiation to Architectural Tradeoffs in SQuAT
Evaluation – Experiment #1
With SimpleTactics+

- Goals
  - Investigate tradeoff solutions in Pareto space
  - Analyze whether negotiation strategy leads to better candidates

- Setup
  - 1 application
    - with 4 scenarios
  - 1 dbot per scenario
  - $K = 2$ search levels
    (33 + 520 candidates)

Boulware

\[
\begin{align*}
\text{if } SR_i(a_j) &\leq ER_i \text{ then } U_i(a_j) = 2 - \frac{ER_i}{SR_i(a_j)}; \\
\text{if } SR_i(a_j) &> ER_i \text{ then } U_i(a_j) = 2 - 1.10 \times \frac{SR_i(a_j)}{ER_i}. 
\end{align*}
\]
Experiment #1 – Tradeoff space

Only P1 versus M1

Figure 5: Solution space and trade-offs between QA scenarios (i.e., scenario) of a given dbot, making it possible to quickly find solutions that can fulfill many scenarios simultaneously.

4.4 Lessons Learned

Applying SQuAT to PCM models revealed several insights about distributed architecture optimization. First, making transformations to multi-grained models such as PCMs is not straightforward. Using Henshin was not always sufficient to codify modifiability tactics, and some PerOpteryx assumptions (e.g., no changes in the repository model) somehow constrained our tradeoff analysis. Second, there were differences between our two quality attributes and their supporting tools. The modifiability analysis and transformations involved high-level changes (e.g., components and interfaces), while the performance counterparts consisted of lower-level changes (e.g., interchanging components in the system model, or allocating assemblies to other resources). Third, the tool maturity is uneven for the two attributes. In the performance dbots, we had well-established and tested technologies for simulating and optimizing system performance and reliability. Regarding modifiability dbots, a perceived drawback is that the change effort computed by KAMP is still very basic. For this reason, a cyclomatic complexity metric adapted to repository components was developed. However, a careful evaluation of such a metric is necessary to assess the results. Another modifiability-related problem was the current optimizer, which led to vertical blocks of discrete solutions in the charts. We conjecture that using genetic algorithms, like in PerOpteryx, can alleviate this issue.

SSP 2017 - Applying Concession-based Negotiation to Architectural Tradeoffs in SQuAT
Experiment #1 – Tradeoff space

*All pairs of scenarios for Modifiability against Performance*
Evaluation – Experiment #2

*With CTAS (non-PCM architecture)*

- Using Dbots for modifiability and performance
  - Change Impact analysis & Rate Monotonic analysis
  - Assignment of priorities to scenarios
    - Low-priority Dbots should concede more than high-priority Dbots
    - Priority is part of the utility function

- Goals
  - Compare results of 3 strategies:
    a) pure search (selfish)
    b) Negotiation #1: search and then negotiation (sequential)
    c) Negotiation #2: search interleaved with negotiation

- Setup
  - 1 application with 8 scenarios
Evaluation – Experiment #2

Negotiated versus “selfish” solutions

- ER is a harder constraint than CR
- Some scenarios might still be difficult to satisfy
- More “balanced” (or equalized) improvements with negotiation

(a) All Designbots and Individual Analysis - Improvement rate of Expected response

Weighted average of improvements, based on priority of scenarios
Lessons Learned

• More balanced improvements (i.e., group utility) with negotiation strategies than with pure search
  - Strategy #2 (interleaved) performed better than strategy #1 (sequential), but with fewer dbots (e.g., only high-priority one) in the negotiation
  - Some scenarios (e.g., performance) might still be difficult to satisfy, due to inherent architecture characteristics
• Computation time was often reduced
  - But assuming agreements in a few rounds of negotiation

• Limitations
  - All dbots used the same priority function and concession type
  - Quality-attribute analyzers (and their results) as well as tactics are representative of the domain, but still need to be calibrated
  - Utility function needs to be validated against architects’ preferences
Microservices Architecture for SQuAT

Aim at scalability and faster optimization

- Initial modular design of SQuAT is not enough (still monolithic)
- Prototype
  - Container-based technology stack
  - PCM to JSON
  - REST interfaces
  - Stateless Dbots
  - Asynchronous negotiator
- Minimize overhead
  - Network communication
  - Object conversion
  - Complexity of setup

Performance Dbot
- Java 8
- Phusion

negotiator
- Java 8
- Phusion

Modifiability DBot
- Java 8
- Phusion

Docker Engine

Operating System

Infrastructure
Future Work

Improvements & experimentation

• Larger case studies
  - CoCoME system
  - Expose more modifiability and performance tradeoffs
• More sophisticated change-impact analysis (CIA)
• Apply different negotiation variants in PCM
  - Other types of concession
• Additional quality attributes (reliability, security)
• A more abstract architectural glue model, less dependent on PCM
• Comparison with microservices approach
  (and also against centralized optimization)
THANK YOU!

- Andrés Diaz Pace  
  adiaz@exa.unicen.edu.ar

- Santiago Vidal  
  svidal@exa.unicen.edu.ar

- Alejandro Rago  
  arago@exa.unicen.edu.ar

- Sebastian Frank  
  s.frankausowen@googlemail.com

- André van Hoorn  
  van.hoorn@informatik.uni-stuttgart.de


Open-source project available at: https://github.com/SQuAT-Team