Configuring Software Product Lines using Clafer and multi-objective optimization

Rafael Olaechea
Outline

- Background.
- Methodology including Research Problem.
- Tool
- Evaluation.
- Future Work.
- Conclusions.
Software Product Lines

Create family of software systems to be used in a specific domain.

- Domain Model.
- Reusable Assets.

Configuration Model:

- Feature Model.

Product Derivation Process

Examples: Medical Imaging Systems Software.

Thursday, 5 April, 12
Product Derivation Process

Variability Modelling → Feature Model → Measurement

Annotated Feature Model with Quality requirements values → Product Derivation

Configured Product
Berkeley DB Annotated Feature Model with Quality Requirements

Fig. 4 Product-line model of Berkeley DB with assigned properties. Footprint represents the actually measured binary size per feature. The up-arrow visualizes an improvement for a qualitative property.

Requires additional code (e.g., using nested #ifdefs). Additionally, feature interactions can cause deadlocks and bus overloads. In Berkeley DB, there is an exhaustive use of nesting a feature's code in another feature's code (e.g., to implement statistics for the hash search index; cf. Figure 2) resulting in different binary sizes depending on a certain feature combination. In Berkeley DB, we identified a feature interaction between features Replication and Statistics. We measured the influence of this interaction on footprint: A product with both features in combination has an increased binary size of 80 KB in addition to sum of the feature's sizes. Such feature interactions occur for many non-functional properties.


With SPL Conqueror, we propose a holistic approach to integrate measurement and optimization of non-functional properties in the product derivation process. With holistic we mean that we support the whole product derivation process starting from the definition of desired non-functional properties, over the measurement of properties, to the concrete feature selection and optimization by means of an objective function. We support the different kinds of non-functional properties described in Section 3.1. A stakeholder (i.e., an SPL vendor or domain expert) can assign properties to features to describe the influence of a feature on a specific property. In addition, a stakeholder can specify measurements and metrics in SPL Conqueror to measure either a single feature (e.g., the source-code complexity) or a whole variant. Once the measurement procedure is defined the process of selecting features and generating and measuring features is automatically performed.

The results of measurements are stored in the SPL's product-line model, which we described in Section 3.2. We use this model including all assignments and measurements during the product derivation phase to provide multiple optimization possibilities. Customers can define non-functional constraints (e.g., a footprint limit of 200 KB) as well.
Goal: Improve the product derivation process in Software Product Lines.

1. Collect SPL feature model examples.

2. Build on existing solutions (Clafer, Moolloy, Alloy partial instances)

3. Create extension to Clafer translator.

4. Evaluate Performance of tooling.
Annotate feature models with Quality Requirements:

- Binary Footprint.
- Performance
- Code Complexity
- Reliability
Tooling Pipeline

Clafer

Feature Model + Quality Objectives

Clafer Translator

Objective Ext

Partial Instance Ext

Alloy

Moolloy Ext

Partial Instances Ext

KodKod

Moolloy Ext

Sat
Berkeley DB Annotated
Feature Model

Fig. 4 Product-line model of Berkeley DB with assigned properties. Footprint represents the actually measured binary size per feature. The up-arrow visualizes an improvement for a qualitative property.

requires additional code (e.g., using nested #ifdefs). Additionally, feature interactions can cause deadlocks and bus overloads. In Berkeley DB, there is an exhaustive use of nesting a feature's code in another feature's code (e.g., to implement statistics for the hash search index; cf. Figure 2) resulting in different binary sizes depending on a certain feature combination. In Berkeley DB, we identified a feature interaction between features Replication and Statistics. We measured the influence of this interaction on footprint: A product with both features in combination has an increased binary size of 80 KB in addition to sum of the feature's sizes. Such feature interactions occur for many non-functional properties.

4S P L Conqueror: An Holistic Approach for the Optimization of Non-functional Properties

With SPL Conqueror, we propose a holistic approach to integrate measurement and optimization of non-functional properties in the product derivation process. With holistic we mean that we support the whole product derivation process starting from the definition of desired non-functional properties, over the measurement of properties, to the concrete feature selection and optimization by means of an objective function. We support the different kinds of non-functional properties described in Section 3.1. A stakeholder (i.e., an SPL vendor or domain expert) can assign properties to features to describe the influence of a feature on a specific property. In addition, a stakeholder can specify measurements and metrics in SPL Conqueror to measure either a single feature (e.g., the source-code complexity) or a whole variant. Once the measurement procedure is defined the process of selecting features and generating and measuring features is automatically performed.

The results of measurements are stored in the SPL's product-line model, which we described in Section 3.2. We use this model including all assignments and measurements during the product derivation phase to provide multiple optimization possibilities. Customers can define non-functional constraints (e.g., a footprint limit of 200 KB) as well.
abstract BerkeleyDbC
  STATISTICS : IMeasurable ?
    [ this.footprint = 285]
  CRYPTO : IMeasurable ?
    [ this.footprint = 19]
  INDEXES : IMeasurable
    [this.footprint = 0]
 xor BTREE
    [this.footprint = 0]
  BTREE_SMALL : IMeasurable
    [ this.footprint = 340]
  BTREE_FAST : IMeasurable
    [ this.footprint = 1800]
  HASH : IMeasurable ?
    [ this.footprint = 113]
  QUEUE : IMeasurable ?
    [ this.footprint = 58]
  REPLICATION : IMeasurable ?
    [ this.footprint = 89]
(...)

Thursday, 5 April, 12
Optimizing Quality Requirements Workflow

Annotated Feature Model

User selects some features

User sets objective function over quality requirements

System selects other features based on such objectives
## Evaluation on Sample Feature Models

<table>
<thead>
<tr>
<th>SPL</th>
<th>Features</th>
<th>Time (s)</th>
<th>Binary Footprint (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedList</td>
<td>18</td>
<td>71</td>
<td>4.43</td>
</tr>
<tr>
<td>LinkedList [ Print and Measurement]</td>
<td>18</td>
<td>21</td>
<td>10.64</td>
</tr>
<tr>
<td>SQLite</td>
<td>80</td>
<td>32278</td>
<td>1200</td>
</tr>
<tr>
<td>Berkeleydb</td>
<td>8</td>
<td>23.6</td>
<td>1804</td>
</tr>
</tbody>
</table>

Thursday, 5 April, 12
Other Feature Models

- Violet UML - UML Diagramming Tool, ~ 200 features.
- ZipMe - Zipping program.
- Prevayler - Java Persistence framework.
- PKJab - Instant Messenger Application.
- Apache - Web Server.
- BerkeleyDB Java Version - Database.
Extending Clafer with Objectives: LinkedList Feature

abstract LinkedList
xor AbstractElement : IMeasurable
[ this.footprint = -12]
ElementA : IMeasurable
[ this.footprint = 12]
ElementB : IMeasurable
[ this.footprint = 0]
(...)
xor AbstractSort : IMeasurable
[ this.footprint = 57]
BubbleSort : IMeasurable
[ this.footprint = 17]
MergeSort : IMeasurable
[ this.footprint = 32]
(...)
print : IMeasurable
[ this.footprint = 44]
Measurement : IMeasurable
[ AbstractSort ]
[ this.footprint = 484]
(...)
total_footprint : integer =
sum(IMeasurable .footprint)
Extending Clafer with Objectives

- LL_Configuration: LinkedList_FeatureModel
  - [print && Measurement]
  - << min LL_Configuration.total_footprint >>
  - << max LL_Configuration.performance >>
objectives o_global { **minimize**
[c229_simpleConfig.@r_c121_total_footprint.@c121_total_footprint_ref], **maximize**
[c229_simpleConfig.@r_c122_total_performance.@c122_total_performance_ref] }

Get a set of solutions in the optimal front between performance and footprint.
Reasoning Optimization: Partial Instances in Alloy

- Alloy Extension to express scope in terms of concrete instances.

-Clafer translator generates a partial instance block to improve performance of reasoning in alloy.

inst partialinstance {
    12 int, // bitwidth
    relation_footprint in ...
}
Translate Clafer Objectives into Alloy:

objectives o_global { minimize
    [c229_simpleConfig.@r_c121_total_footprint. @c121_total_footprint_ref] }

Execute using Multi-Objective Alloy:
Found base solution. At time: 3, Improving on [586]
Found a better one. At time --: 3, Improving on [586]
Found a better one. At time --: 6, Improving on [467]
Found a better one. At time --: 27, Improving on [444]
Found a better one. At time --: 43, Improving on [443]
GIA ----: [443]
Future Work

- Integrate Sparse Integer Support from Alloy.
- Breadth-Width Search could create set of all reachable integers.
- Integrate partial non-optimal results from the alloy solver before reaching the optimal answers.
- For Sqlite it took 13 hours, but last 7 hours gave only marginal improvement.
Challenges

- Partial Instances in Alloy:
  - Ongoing work from Vajih Montaghami.
  - Getting Realistic Software Product Line Models
  - Wrote translator to get real models from SPLConqueror work by Norbert Siegmund et al.
Conclusions

- Product Configuration in Clafer
- Explore Space of Product Configurations
- Helps Stakeholders consider quality properties.
- Quality of Annotated Feature Models.
References


- Extending Alloy with Partial Instances. V. Montaghami, D. Rayside.


- The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization. Rayside et al.