Performance-Driven Interface
Contract Enforcement for
Scientific Components

10th International Symposium on
Component-Based Software Engineering
Medford, MA USA
July 9-11, 2007

Tamara “Tammy” Dahlgren (dahlgren1@llnl.gov)
http://www.llnl.gov/comp/bio.php/dahlgren1

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

UCRL-PRES-232517
Performance-Driven Interface Contract Enforcement for Scientific Components

Motivation
- Babel Toolkit
- Experiments
- Summary
Applications built using plug-and-play components depend on common interfaces.

- Multiple implementations conform to the same specification
  - Provide same basic services
  - Require same basic inputs
  - Implementations can vary significantly to include differences in…
    - Algorithms
      - Solution accuracies
    - Underlying data structures
    - Implementation language(s)
      - C, C++, Fortran 77/9x/20xx, Python, Java

Different organizations ➔ different development processes (and rigor)
Contracts added to common interfaces can be used to improve software quality.

```java
package vector version 1.0 {
    interface Utils { …

    double norm (in array<double> u, in double tol)
    throws /* Exceptions */
        sidl.PreViolation, NegativeValueException,
        sidl.PostViolation;

    require /* Preconditions */
        not_null : u != null;
        u_is_1d : dimen (u) == 1;
        non_negative_tolerance : tol >= 0.0;

    ensure /* Postconditions */
        no_side_effects : is pure;
        non_negative_result : result >= 0.0;
        nearEqual (result, 0.0, tol) iff isZero (u, tol);

    … }
}
```

Example based on Babel’s vector.sidl (class) specification

vector.Utils.isZero (u, tol), which would typically be $O(|u|)$
Computational Scientists are typically willing to incur no more than 10% overhead.

Hence, this research focuses on performance-driven sampling.
Performance-Driven Interface Contract Enforcement for Scientific Components

- Motivation
- Babel Toolkit
- Experiments
- Summary
Enforcement automatically generated by Babel* language interoperability toolkit.

*Experiments were conducted using an experimental version of Babel based on release 0.10.8.
Enforcement tracing currently provides simple timing dumps on exercised contracts.
Global enforcement options are based on two parameters: frequency and type.

<table>
<thead>
<tr>
<th>Enforcement Frequency</th>
<th>Contract [Clause] Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>All</td>
</tr>
<tr>
<td>Always</td>
<td>Constant-time</td>
</tr>
<tr>
<td>Periodic</td>
<td>Linear-time</td>
</tr>
<tr>
<td>Random</td>
<td>Preconditions*</td>
</tr>
<tr>
<td>Adaptive Fit (AF)</td>
<td>Postconditions*</td>
</tr>
<tr>
<td>Adaptive Timing (AT)</td>
<td>Invariants*</td>
</tr>
<tr>
<td>Simulated Annealing (SA)</td>
<td>Simple Expressions</td>
</tr>
<tr>
<td></td>
<td>Method Calls</td>
</tr>
<tr>
<td></td>
<td>Results</td>
</tr>
</tbody>
</table>

*All combinations of the three Eiffel method clause types are actually available.
Performance-Driven Interface Contract Enforcement for Scientific Components

- Motivation
- Babel Toolkit
- Experiments
- Summary
Input sets were varied for three of five programs, forming a total of **thirteen** trials.

<table>
<thead>
<tr>
<th>Component</th>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicial Mesh</td>
<td>MA</td>
<td>Retrieve all faces from the mesh then, for each face, retrieve the adjacent vertices.</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>Retrieve all faces from the mesh in sets based on size of input array. Sizes 1, 1,4587 (10%), and 14,5870 (100%).</td>
</tr>
<tr>
<td></td>
<td>AA</td>
<td>Retrieve faces as in A plus, for each set of faces, retrieve their corresponding adjacent vertices. The <em>same input sizes</em> were used.</td>
</tr>
<tr>
<td>GRUMMP 0.2.2b’s Volume Mesh</td>
<td>MT</td>
<td>Exercise and check consistency of five mesh interfaces: core, single entity query and traversal, entity array query and traversal, single entity mesh modification, and entity array mesh modification.</td>
</tr>
<tr>
<td>Vector Utilities</td>
<td>VT</td>
<td>Exercise all supported functions to include successful execution; one or more precondition violations; and one or more postcondition violations. Sizes 6 (original), 10, 100, 1000, and 10000.</td>
</tr>
</tbody>
</table>
Baseline experiments resulted in a variety of profiles using contract enforcement traces.
Enforcement performance was generally better without tracing instrumentation.

Reason: A combination of tracing instrumentation and program speed.
Adaptive Fit (AF) sampling tuned clause enforcement based on estimated overheads.
Adaptive Timing (AT) is biased toward ‘fast’ clauses relative to the cost of their methods.

91% of violations in ‘fast’ clauses with linear-time expressions!
Performance-Driven Interface Contract Enforcement for Scientific Components

- Motivation
- Babel Toolkit
- Experiments

Summary
Performance-Driven Interface Contract Enforcement for Scientific Components

- **Goal**: Improve quality of applications built of automatically swapped, plug-and-play, third-party components

- **Approach**: Use performance criteria to tune contract [clause] enforcement to the program
  - Reduce overhead compared to full enforcement (i.e., *Always*)
  - Increase coverage over other sampling techniques (when appropriate)
    - Increase probability of detecting more violations

- **Findings**: Performance-driven enforcement appears to be most suited to contract clauses that are at most moderately expensive to check
  - Based on user-specified overhead limit
  - Relative to program/methods
Thank you for your attention.

Any Questions?
For more information related to this work, refer to the following web sites.

- **Components Project**
  - [http://www.llnl.gov/casc/components](http://www.llnl.gov/casc/components)
  - Note: Experiments conducted using experimental prototype of the Babel toolkit

- **Common Component Architecture (CCA) Forum**
  - [http://cca-forum.org](http://cca-forum.org)

- **Center for Technology for Advanced Scientific Component Software (TASCS)**
  - SciDAC’s Plug and Play Supercomputing
  - [http://www.scidac.gov/compsci/TASCS.html](http://www.scidac.gov/compsci/TASCS.html)
Supplemental Material
What can be done to ensure plug-and-play components used and implemented correctly?

<table>
<thead>
<tr>
<th>Year</th>
<th>Theory/Proofs of Correctness</th>
<th>Applied Research/Demonstrations of Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
<td>Assertions (Routines) 1950 (Turing)</td>
<td>Executable Assertions 1977 (Saib)</td>
</tr>
<tr>
<td>1960</td>
<td>Assertions (Programs) 1967/68 (Floyd/Hoare)</td>
<td>Design by Contract 1985 [Eiffel]</td>
</tr>
<tr>
<td>1970</td>
<td></td>
<td>High-level Component Specs 1994+ [ADL]</td>
</tr>
<tr>
<td>1980</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This research is based on contracts on the [common] interfaces.
Performance overhead concerns lead to no or partial enforcement during deployment.

<table>
<thead>
<tr>
<th>Most common</th>
<th>All-or-nothing</th>
<th>Eiffel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion type</td>
<td>Package/Class</td>
<td>JAF, Jass</td>
</tr>
<tr>
<td>Selective</td>
<td>Method</td>
<td>iContract</td>
</tr>
<tr>
<td>Severity</td>
<td></td>
<td>APPC</td>
</tr>
<tr>
<td>Individual</td>
<td></td>
<td>SIFT, ConFract</td>
</tr>
<tr>
<td>Sampled</td>
<td></td>
<td>Frequency-based*, Random**</td>
</tr>
</tbody>
</table>

Enforcement decisions are made on a contract clause basis.

Performance-driven variants execute contracts only if accumulated enforcement costs do not exceed user-specified overhead limit.
Contract characteristics varied across programs, in one case across input sets.

<table>
<thead>
<tr>
<th>Program</th>
<th>Array Size</th>
<th>Const.</th>
<th>Linear</th>
<th>SE</th>
<th>MC</th>
<th>Precond</th>
<th>Postcond</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>n/a</td>
<td>50%</td>
<td>50%</td>
<td>0%</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>All</td>
<td>100%</td>
<td>0%</td>
<td>50%</td>
<td>50%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>1</td>
<td>75%</td>
<td>25%</td>
<td>25%</td>
<td>75%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td>14587</td>
<td>77%</td>
<td>24%</td>
<td>26%</td>
<td>73%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>145870</td>
<td>88%</td>
<td>20%</td>
<td>30%</td>
<td>63%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MT</td>
<td>n/a</td>
<td>99.995%</td>
<td>.005%</td>
<td>73%</td>
<td>27%</td>
<td>58%</td>
<td>42%</td>
</tr>
<tr>
<td>VT</td>
<td>All</td>
<td>95%</td>
<td>8%</td>
<td>0%</td>
<td>100%</td>
<td>80%</td>
<td>33%</td>
</tr>
</tbody>
</table>

One annotated method ≤ Two contract clause enforcement opportunities
Simulated Annealing (SA) sampling performed similar to AF, except in presence of lots of checks.