Comparing Complexity of API Designs

An Exploratory Experiment on DSL-based Framework Integration
by Stefan Sobernig, Patrick Gaubatz, Mark Strembeck, and Uwe Zdun

September 23, 2011
Structure of the talk

1. **Context**
   - Motivation: Quantifying DSL Benefits?
   - Experiment Units: API, DSL, Structural Complexity
   - Experiment Design: Sneed Object-Points Analysis
   - Experiment Domain: Federated Identity Management & the xoSAML DSL

2. **Experiment procedure**

3. **Observations**

4. **Consequences and Threats**
Context (I) / DSL Development

- More than 14 DSL development projects (textual, external/embedded) for different domains, for example:
  - Process modelling
  - Data backup
  - RBAC
  - Software testing
  - Software documentation

- Engineering process driven by . . .
  - . . . an explicit language model
  - . . . a mockup language
  - . . . the extraction of a DSL concrete/abstract model from domain artifacts

- Tooling: XOTcl, Frag

- See [SZ09, Zdu10]
Context (II) / Promises

- Our DSLs serve as APIs on top of domain-specific frameworks

- DSL-based APIs target domain experts by wrapping framework functionality using domain abstractions

- Motivation: Positive impact on the quality attributes of the resulting DSL programs [vK98, HPvD09, GGA10, Fow10]
  - For example: Augmented expressiveness thanks to a “small” number of “higher-level” domain abstractions plus “declarative” concrete syntax → maintainability
  - Others: Comprehensibility, communicability, and reusability

- Empirical evidence is limited [GGA10, HPvD09], evaluation frameworks are missing.
  - Evidence is small in number [GGA10]
  - Qualitative vs. quantitative approaches
  - Quantitative approaches: Methodical issues (LOC; [MP07, ZME06, Bet02]) and, most importantly, not reusable/adoptable
Context (III) / Experiment Units

- **API**
  - Code units
  - Protocol
  - Configuration

- **Structural complexity**
  - Size
  - Interaction level
  - Parametric complexity

- **Program**
  - Source-code representation
  - UML representation

- **Object points**
  - Class points
  - Message points

Properties of the API integrate a working framework.
Context (IV) / Sneed Object-Points Analysis

\[
\begin{align*}
\text{Httpd::Wrk} & \quad +\text{respond()} : \text{void} \\
\text{Request} & \quad +\text{send(string dest)} : \text{void} \\
\text{Worker} & \quad +\text{respond()} : \text{void} \\
\text{AuthnRequest} & \quad +\text{ID(string id)} : \text{ID} \\
\text{Httpd} & \quad -\text{ipaddr} : \text{string}
\end{align*}
\]

\[
\begin{align*}
c \in C & \quad |A_c| |R_c| |O_c| N_c \\
\text{Httpd} & \quad 2 \quad 0 \quad 0 \quad 1 \\
\text{Httpd::Wrk} & \quad 0 \quad 1 \quad 1 \quad 1 \\
\text{Worker} & \quad 1 \quad 0 \quad 1 \quad 0.6 \\
\text{Request} & \quad 0 \quad 1 \quad 1 \quad 1 \\
\text{AuthnRequest} & \quad 0 \quad 1 \quad 1 \quad 0.5 \\
\text{5 classes} & \quad 3 \quad 3 \quad 4 \quad 0.82
\end{align*}
\]

\[
\begin{align*}
o \in O_M & \quad |P_o| |S_o| |T_o| N_o \\
\text{Worker.respond} & \quad 0 \quad 1 \quad 1 \quad 0.5 \\
\text{AuthnRequest.ID} & \quad 2 \quad 1 \quad 1 \quad 1 \\
\text{Request.send} & \quad 1 \quad 1 \quad 1 \quad 1 \\
\text{3 messages} & \quad 3 \quad 3 \quad 3 \quad 0.833 \\
\text{Message points (MP)} & \quad 17.5 \\
\text{MP} & \quad (2 \cdot 3 + 3 + 2 \cdot 3 + 3 \cdot 4) \cdot 0.833
\end{align*}
\]
Experiment Procedure

1. Application domain: Federated Identity Management, SAML

2. Application scenario: Single-Sign-On Service Provider (SSO SP)

3. Frameworks:
   - OpenSAML
   - simpleSAMLphp
   - JAXB
   - xoSAML

4. Model descriptors calculated using SLOCCount & SDMetrics
Experiment Scenario

A Single-Sign-On Service Provider (SSO SP) in SAML

The xoSAML DSL

Type/Class
Request
+ID
+send(dest)
---
Response
+RelayState
+receive(sender)
---
Assertion
+hasAttributes()
+getAttributes()
---
Subject
+name
---
Attribute
+name
+getName()
---
Status
+getStatusCode()

Instance/Object (ex.)
Request request
ID="identifier_1"
---
Response response
RelayState="res1"
---
Assertion a
hasAttributes
getAttributes
Subject bob "bob"
---
Attribute a1
name="n1"
getName
Attribute a2
name="n2"
---
Status s
getStatusCode
Quantitative Observations

<table>
<thead>
<tr>
<th>Program using...</th>
<th>OpenSAML</th>
<th>JAXB</th>
<th>simpleSAMLphp</th>
<th>xoSAML</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>151</td>
<td>120</td>
<td>48</td>
<td>144</td>
</tr>
<tr>
<td># classes</td>
<td>C</td>
<td>31</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td># operations</td>
<td>O</td>
<td>254</td>
<td>222</td>
<td>110</td>
</tr>
<tr>
<td># relations</td>
<td>R</td>
<td>30</td>
<td>28</td>
<td>15</td>
</tr>
<tr>
<td>Cohesion $H = \frac{</td>
<td>R</td>
<td>+ 1}{</td>
<td>C</td>
<td>}$</td>
</tr>
</tbody>
</table>

1. xoSAML
   - CP ($\bar{N}_C$) | 179.76 ($\approx 0.79$) | 87 (1) | 266.76
   - MP ($\bar{N}_{OM}$) | (1.51) | (0.99) | (1.34)

2. simpleSAMLphp
   - CP ($\bar{N}_C$) | 270.53 ($\approx 0.64$) | 85.81 (0.89) | 356.34
   - MP ($\bar{N}_{OM}$) | (4.2) | (2.1) | (3.54)

3. JAXB
   - CP ($\bar{N}_C$) | 755.46 ($\approx 0.98$) | 188 (1) | 943.46
   - MP ($\bar{N}_{OM}$) | (4.2) | (2.1) | (3.54)

4. OpenSAML
   - CP ($\bar{N}_C$) | 860.75 ($\approx 0.84$) | 224 (1) | 1084.75
   - MP ($\bar{N}_{OM}$) | (4.79) | (2.57) | (4.07)

- The program constructed from the xoSAML DSL features the *lowest* OP score: 266.76
- The OP scores of the high-OP programs JAXB/OpenSAML is 3.5 to 4 times the OP value of xoSAML.
- The program sizes in terms of the total number of classes $|C|$ and operations $|O|$ follow the pattern of the OP values.
- Despite its OP-based top rank, the xoSAML program does not have the smallest code base.
- The simpleSAMLphp program has a higher OP score than its xoSAML equivalent, while having a code base which is three times smaller than xoSAML's.
Qualitative Observations

- API Complexity and the xoSAML DSL
  → Low structural complexity: Few domain abstractions (15 classes), small interaction level (2.5 operations per class)

- xoSAML DSL vs. PHP Class Library
  → Modest OP score, when compared to xoSAML. HTTP handling is not part of the API feature chunk.

- xoSAML DSL vs. Java OO Frameworks
  → Considerable structural complexity: 20-31 classes; configuration overhead due to container and beans management in the API feature chunk; boiler plate due to X/O data binding

- Expressiveness and API Complexity
  → Number of abstractions (classes) vs. interaction level (operations per class)

- LOC Measurement
  → The program sizes expressed in LOC, and the LOC-based ranking of the programs, do not relate to the order by OP value.
Contributions:

1. Applied the Sneed Object-Points analysis to obtain indicators of API complexity comparing four different approaches to API design, including an embedded DSL.
2. First indication that a textual, embedded DSL can reduce API complexity.
3. Reproducible experiment for a given domain; code bases and raw data sets are available from [http://swa.univie.ac.at/~patrick/op.zip](http://swa.univie.ac.at/~patrick/op.zip) and [https://bitbucket.org/pgaubatz/xosaml](https://bitbucket.org/pgaubatz/xosaml).
4. Reusable experiment design.

Threats to Validity:

1. **Construct validity**: Alienation of the OP analysis; missing empirical CP/MP validation for our setting; value loading of standard OP weightings.
2. **Internal validity**: Comparability of the four experiment programs; implementor bias; continued learning effect; surrogate measurement (API chunk); UML intermediate representation (abstraction mismatch).
3. **External validity**: Generalizability is limited (single application domain, xoSAML not representative, limited working notion of API complexity).
Shooting time!

Thanks for bearing with me!

→ Any questions on the nitty-gritty (or the big picture)?
 References


• Uwe Zdun. A DSL toolkit for deferring architectural decisions in DSL-based software design. Information and Software Technology, 52(7):733–748, 2010


• Martin Fowler. Domain Specific Languages. The Addison-Wesley Signature Series. Addison-Wesley Professional, 1st edition, 2010


Component Structure of Software Artifacts (I)

- xoSAML SP Example
  - Transport Protocol
  - SAML API
  - xoSAML - Dependencies
    - xoComm Httpd
    - xoSAML
    - tDOM
    - xoXSD
    - XML Parser

- simpleSAMLphp SP example
  - Transport Protocol
  - SAML API
  - simpleSAMLphp - Dependencies
    - Apache HTTP Server
    - simpleSAMLphp
Component Structure of Software Artifacts (II)

JAXB SP Example
- Transport Protocol
- Servlet Container
- SAML API
- JAXB - Dependencies
  - Apache Tomcat
  - Apache Commons
  - JAXP
  - SAML Classes
  - JAXB Binding Compiler
  - JAXB

OpenSAML SP Example
- Transport Protocol
- Servlet Container
- SAML API
- OpenSAML - Dependencies
  - Apache Tomcat
  - Apache Velocity
  - Joda Time
  - OpenSAML
  - JAXP
**Weights in the Sneed Object-Points analysis**

<table>
<thead>
<tr>
<th>Weight</th>
<th>Loading</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_C$</td>
<td>4</td>
<td>The value of 4 has been adopted from the standard weight of entities in the Data Points analysis; see [Sne95].</td>
</tr>
<tr>
<td>$W_{Rc}$</td>
<td>2</td>
<td>Also adopted from the standard Data Points method; a weight of 2 reflects the fact that a single relation affects exactly two entities; see [Sne95].</td>
</tr>
<tr>
<td>$W_{Oc}$</td>
<td>3</td>
<td>This weight reflects the average size of methods found for a representative code base in the programming language under investigation. The value 3 is the standard value proposed in [Sne95], expressing a ratio of 15:5 between the avg. method statement size of more high-level (e.g. Smalltalk) and more low-level programming languages (e.g. C++).</td>
</tr>
<tr>
<td>$W_{OM}$</td>
<td>2</td>
<td>Similarly to $W_{Rc}$, the weight 2 reflects that a single message occurrence involves two entities – the sender and the receiver; taken from [Sne95].</td>
</tr>
<tr>
<td>$W_{So}$</td>
<td>2</td>
<td>The standard value adopted from [Sne95].</td>
</tr>
<tr>
<td>$W_{To}$</td>
<td>2</td>
<td>The standard value adopted from [Sne95].</td>
</tr>
</tbody>
</table>