SOCLe:
Integrated Design of Software Applications and Security

Frédéric Painchaud
Defence Scientist
Trusted C2IS Group
High-Assurance Software: A Question of Correct Design

• Software abounds in mission-critical systems
  – Banks, hospitals, emergency services, public utilities, governments, military organizations, etc.

• High reliability and security requirements
  – But very complex and fragile

• Three main issues:
  – Design problems (system architecture)
  – Implementation problems (programming languages and platforms used)
  – Malicious code (viruses, trojans, worms, backdoors, etc.)

• Correcting errors during or after implementation is costly
  – 50 to 100 times more expensive than at design-time

Validating Design: Our Key Approach

- Integrate design and system specification
- Use an innovative mix of UML and OCL
UML and OCL

• UML is the *de facto* standard software design notation
  – Used in the industry and in C2IS and other governmental critical systems

• OCL is a complementary constraint language
  – Eliminates ambiguities in software design
  – Formulates pre-/post-conditions and invariants
  – Part of UML

UML: Unified Modeling Language
OCL: Object Constraint Language
Plan of the Presentation

1. SOCLe Architecture
2. Caveat-Separation System: A Simplified Case Study
3. SOCLe Prototype
4. Conclusion
SOCLe Architecture

[Diagram depicting the SOCLe Architecture flow, showing connections between UML Editing Tool, OCL Constraints (.xmi), UML Model (.xmi), ASM Rule (.asm), Executable UML Model (.asm), UML Compiler, Model-Checker, Execution Graph, Model-Checking Diagnosis, and ASM Interpreter.]
Plan of the Presentation

1. SOCLe Architecture
2. Caveat-Separation System: A Simplified Case Study
3. SOCLe Prototype
4. Conclusion
**Caveat-Separation System: A Simplified Case Study**

- **Users**
  - **Security clearance**: Enhanced, Confidential, Secret, or Top Secret
  - **Country of origin**: Canada, United States, United Kingdom, Germany, Egypt, etc.
  - **Military rank**: General, Officer, or Civilian

- **Documents**
  - **Classification level**: Unclassified, Confidential, Secret, or Top Secret
  - **Caveat**: CEO, CANUS, CANUSUK, NATO, UN Coalition, USUK, etc.
  - **Affiliation with a project**: A, B, C, etc.
Caveat-Separation System: A Simplified Case Study

• Access Control Tables
  – **Classification table**: access relation between the security clearance of users and the classification level of documents
  – **Caveat table**: caveats dictate acceptable countries of origin for users who want to access documents
  – **Need-to-Know table**: determines the need-to-know, in terms of countries of origin and military ranks, that is necessary to access the documents of a particular project

• System property
  – At all times, a user can access a document if and only if the access is granted with respect to the three access control tables.
## Classification Table

<table>
<thead>
<tr>
<th>Classification level / Security clearance</th>
<th>Enhanced</th>
<th>Confidential</th>
<th>Secret</th>
<th>Top Secret</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclassified</td>
<td>Granted</td>
<td>Granted</td>
<td>Granted</td>
<td>Granted</td>
</tr>
<tr>
<td>Confidential</td>
<td>Denied</td>
<td>Granted</td>
<td>Granted</td>
<td>Granted</td>
</tr>
<tr>
<td>Secret</td>
<td>Denied</td>
<td>Denied</td>
<td>Granted</td>
<td>Granted</td>
</tr>
<tr>
<td>Top Secret</td>
<td>Denied</td>
<td>Denied</td>
<td>Denied</td>
<td>Granted</td>
</tr>
</tbody>
</table>
# Caveat Table

<table>
<thead>
<tr>
<th>Caveat / Country of origin</th>
<th>Canada</th>
<th>United States</th>
<th>United Kingdom</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEO</td>
<td>Granted</td>
<td>Denied</td>
<td>Denied</td>
<td>…</td>
</tr>
<tr>
<td>CANUS</td>
<td>Granted</td>
<td>Granted</td>
<td>Denied</td>
<td>…</td>
</tr>
<tr>
<td>CANUSUK</td>
<td>Granted</td>
<td>Granted</td>
<td>Granted</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>Project / Country of origin</td>
<td>Canada</td>
<td>United States</td>
<td>United Kingdom</td>
<td>…</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>---------------</td>
<td>----------------</td>
<td>---</td>
</tr>
<tr>
<td>A</td>
<td>G – O – C</td>
<td>G – O – C</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>B</td>
<td>G</td>
<td>G – O</td>
<td>G – O – C</td>
<td>…</td>
</tr>
<tr>
<td>C</td>
<td>G – O</td>
<td>G</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

G: General  
O: Officer  
C: Civilian
Plan of the Presentation

1. SOCLe Architecture
2. Caveat-Separation System: A Simplified Case Study
3. SOCLe Prototype
4. Conclusion
SOCLe Prototype

- Design the system in the usual way
  - Class diagrams, object diagrams, collaboration diagrams, statecharts, etc.
  - Add OCL constraints
- SOCLe constructs an underlying model
  - Formally verified by under-the-hood model-checking
  - Transparent to the designer
context : Controller :: access(u : User, d : Document)

pre : u.clearance ≥ d.level
\∧ (self.tabC).co → filter\{x|x.codeP = u.country\}.cav →
   exists\{y|y.codeC = d.caveat\}
\∧ (self.tabP).cell → filter\{x|x.country.codeCt = u.country\} →
   filter\{y|y.proj.codePj = d.project\}.ra.codeR ≤ u.rank

post : true
Defence R&D Canada – Valcartier # 19

SOCLE  Compiling Execution Diagram

Socle Info

== Layer 6: Done ==

== Layer 7: Method Constraints ==
  Extracting Constraints (.xmi):
    Class Diagram...  Done
  Elaborating Constraints:
  == Layer 7: Done ==

== Generating .asm File ==

*** Computing .asm file ***

[1] Opening caveatFinal.asm... done
[2] Creating caveatFinal directory... done
[3] Parsing input file...
  Parsing OCL Expressions... done
  Type-checking OCL Expressions... done
done in 0.05 s
[4] Computing execution graph...

------------------------------
CRAC's ASM-Compiler & OCL-Model-checker v3
Danien Azambre & Mathieu Bergeron - 2003-2004
------------------------------
Plan of the Presentation

1. SOCLe Architecture
2. Caveat-Separation System: A Simplified Case Study
3. SOCLe Prototype
4. Conclusion
Conclusion

- Critical software design should be and can be formally verified to some extent.
- Reduces the risk of design problems and defects in the implementation.
- Error detected
  - Falsified property when the access rules for documents can be updated while the point-to-point transfer is still in progress.
- Reasonably scalable
  - But state explosion on large systems.
- Work in progress
  - State reduction techniques
  - On-the-fly validation.
Conclusion

• Technical reports
  – Caveat Case Study
  – An ASM Semantics of UML and OCL
  – Java Source Code Generation from UML
  – etc.
• Quebec Provincial FIQ OCTAS 2005 Winner
  – Finalist in the Canadian CIPA Awards
• IBM Canada at Ottawa is particularly interested in SOCLe technology transfer
  – Inline with their current vision
  – PolyMTL is currently preparing a proposition for IBM on how to integrate this technology into their tools