Using Objective Caml to Develop Safety-Critical Embedded Tools in a Certification Framework

Bruno Pagano, Olivier Andrieu, Thomas Moniot, Benjamin Canou, Emmanuel Chailloux, Philippe Wang, Pascal Manoury, Jean-Louis Colaço

Edinburgh, September, 1st, 2009
Plan

The SCADE Certified Software Factory
  Use of O’Caml

Context: Critical Software
  Certification in Avionics
  DO-178B
  Benefits of Using A Certified CG

Certifying an O’Caml binary
  Certifying O’Caml’s Runtime Library
  Certifying the Source Code
  Traceability from Source to Binary

Timeline Report

Conclusion
The SCADE Certified Software Factory

SYSTEM SPEC

Algorithm Design Capture
Architecture Design Capture

SYSTEM TEST

DESIGN

VERIFY

GENERATE

SCADE Suite KCG
RTOS Adaptors

MANAGE & TRACE

Requirements Management Gateway
Integrated Configuration Management
Automatic Design Documentation

DO-178B
IEC 61508
EN 50128
Certification Kits, Certificates & Handbooks

EDINBURGH
SEPTEMBER, 1ST, 2009
ICFP
PHILIPPE WANG (UPMC/LIP6)
3/20
The SCADE Certified Software Factory

SYSTEM SPEC
- Algorithm Design Capture
- Architecture Design Capture

SYSTEM TEST
- Aircraft
- Train
- Car

DESIGN
- Debugging & Simulation
- Formal Verification
- Time & Stack Analysis
- Model Coverage Analysis
- Object Code Verification

VERIFY

GENERATE
- SCADE Suite KCG
- RTOS Adaptors

MANAGE & TRACE
- Requirements Management Gateway
- Integrated Configuration Management
- Automatic Design Documentation
- DO-178B
  IEC 61508
  EN 50128
- Certification Kits, Certificates & Handbooks

EDINBURGH
SEPTEMBER, 1ST, 2009
ICFP
PHILIPPE WANG (UPMC/LIP6)
The SCADE Certified Software Factory

SYSTEM SPEC

ALGORITHM DESIGN CAPTURE

ARCHITECTURE DESIGN CAPTURE

SYSTEM TEST

DESIGN

VERIFY

GENERATE

SCADE SUITE KCG

RTOS ADAPTORS

MANAGE & TRACE

Requirement Management Gateway

Integrated Configuration Management

Automatic Design Documentation

DO-178B
IEC 61508
EN 50128
Certification Kits, Certificates & Handbooks

EDINBURGH
SEPTEMBER, 1ST, 2009
ICFP
PHILIPPE WANG (UPMC/LIP6)
SCADE Suite in a Certification Framework

SCADE Suite

- IDE for safety-critical software
- Based on synchronous dataflow language Lustre
- Graphical editor, model simulation, formal verification

KCG (Qualifiable Code Generator)

- C code generator
- Qualifiable DO-178B (level A) development tool
- Last version written in O’Caml
SCADE Suite in a Certification Framework

SCADE Suite

- IDE for safety-critical software
- Based on synchronous dataflow language Lustre
- Graphical editor, model simulation, formal verification

KCG (Qualifiable Code Generator)

- C code generator
- Qualifiable DO-178B (level A) development tool
- Last version written in O’Caml
Use of O’Caml

OCaml was very natural for R&D
- It is very-well suited for compilers (CG)
- Prototype already in O’Caml

However
- DO-178B: Use the best language for a given project
- Domain is very conservative: Use C or Ada

Need to
- Demonstrate the compatibility between DO-178B and O’Caml
- Find means to assess that generated code is under control!
  - This generated various activities detailed later...
Context: Critical Software

Certification of Safety-Critical Software

- Critical Code
  - Domains: avionics, railway, ...

- Norms
  - DO–178B   Aerospace and Defense
  - EN 50128   Rail Transportation
  - IEC61508   Industrial and Transportation
  - IEC60880   Nuclear
Certification in Avionics

Avionic industry is the most regulated one

- 1\textsuperscript{st} international conference in 1910!

Everything is ruled

- Conception, ...
- Transportation, Crew, ...
- Noise, Population health, ...
- Leisure

Components must be conceived such that

- Defects WRT flight security take-off or landing are EXTREMELY IMPROBABLE, and do not result from simple cause
- Any other defects are IMPROBABLE
Activities Required by Certification

Traceability: reviews & tests

- System Requirements Analysis
- Software High-Level Specifications
- Architectural Design
- Detailed Design
- Coding
- Software Receipt
- Validation Tests
- Integration Testing
- Unary Testing

EDINBURGH  SEPTEMBER, 1ST, 2009  ICFP  PHILIPPE WANG (UPMC/LIP6)
Software Considerations in Airborne Systems and Equipment Certification

Mean of conformity for *embedded software*

“It is in general not feasible to assess the number or kinds of software errors, if any, that may remain after the completion of system design, development, and test. DO-178B/ED-12B, provides acceptable means for assessing and controlling the software used to program digital computer-based systems”
Benefits of Using A Certified CG

<table>
<thead>
<tr>
<th>Objective</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Low-level requirements comply with high-level requirements.</td>
<td>Not eliminated</td>
</tr>
<tr>
<td>2 Low-level requirements are accurate and consistent.</td>
<td>Automated</td>
</tr>
<tr>
<td>3 Low-level requirements are compatible with target computer.</td>
<td>Not eliminated</td>
</tr>
<tr>
<td>4 Low-level requirements are verifiable.</td>
<td>Eliminated</td>
</tr>
<tr>
<td>5 Low-level requirements conform to standards.</td>
<td>Automated</td>
</tr>
<tr>
<td>6 Low-level requirements are traceable to high-level requirements.</td>
<td>Not eliminated</td>
</tr>
<tr>
<td>7 Algorithms are accurate.</td>
<td>Not eliminated</td>
</tr>
<tr>
<td>8 Software architecture is compatible with high-level requirements.</td>
<td>Not eliminated</td>
</tr>
<tr>
<td>9 Software architecture is consistent.</td>
<td>Automated</td>
</tr>
<tr>
<td>10 Software architecture is compatible with target computer.</td>
<td>Not eliminated</td>
</tr>
<tr>
<td>11 Software architecture is verifiable.</td>
<td>Eliminated</td>
</tr>
<tr>
<td>12 Software architecture conforms to standards.</td>
<td>Automated</td>
</tr>
<tr>
<td>13 Software partitioning integrity is confirmed.</td>
<td>Not eliminated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objective</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Source Code complies with low-level requirements</td>
<td>Eliminated</td>
</tr>
<tr>
<td>2 Source Code complies with software architecture</td>
<td>Eliminated</td>
</tr>
<tr>
<td>3 Source Code is verifiable</td>
<td>Eliminated</td>
</tr>
<tr>
<td>4 Source Code conforms to standards</td>
<td>Eliminated</td>
</tr>
<tr>
<td>5 Source Code is traceable to low-level requirements</td>
<td>Eliminated</td>
</tr>
<tr>
<td>6 Source Code is accurate and consistent</td>
<td>Eliminated</td>
</tr>
<tr>
<td>7 Output of software integration process is complete and correct</td>
<td>Not eliminated</td>
</tr>
</tbody>
</table>

HIGH LEVEL REQUIREMENTS  ==  LOW LEVEL REQUIREMENTS
Certifying an O’Caml binary

O’Caml specificities

- High-level language: functional model, pattern matching, polymorphism, exceptions, etc
- Significant Runtime Library: garbage collector, polymorphic comparison, exceptions, etc

Necessities

- Runtime Library coverage
- New tools for O’Caml code coverage
- New means for O’Caml code traceability

Target and Version

- OCaml 3.09.3 Native compiler for x86 (32-bit)
Certifying an O’Caml binary

Test the Runtime library
- Memory management (allocator and collector)
- Language features implementation (apply, exception, ...)
- Language built-in functions (IO, operators, ...)

Test the Source Code
- Program source code + Standard library

Traceability from Source to Binary
- From Source to Assembly
- Translation of Explicit Control of the Source Code
- Controls Introduced by the Compiler Itself
Certifying O’Caml’s Runtime Library

O’Caml’s runtime library

- Complex Low-level C code
- Very efficient garbage collector
- Unsafe or extra features

Solution

- **Simplify** the garbage collector
  - simple *Stop & Copy* (125 lines of C instead of 1’200)
- **Remove** unneeded stuff (by KCG)
  - Threads, Marshaling, Weak pointers
- Really **easier to specify and test!**
  - 4’500 lines of C code instead of 16’000
Certifying the Source Code

Coding standard

- Functional and imperative subset of O’Caml
  - No objects, no polymorphic variants, no lazy evaluation,...
  - Reduced standard library

- Coding rules
  - Identifier naming conventions
  - No variable hiding
  - Limited use of anonymous functions
  - etc.

Coverage measurement

- MC/DC measure required
Modified Condition/Decision Coverage

Statement Coverage (SC)
- Each statement is executed at least once
  \[ \Rightarrow 1 \text{ test} \]

Decision Coverage (DC)
- SC + each Boolean decision must be evaluated to true and false
  \[ \Rightarrow 2 \text{ tests} \]

Multiple Condition Coverage (MCC)
- DC + all combinations of Boolean conditions
  \[ \Rightarrow 2^n \text{ tests} \]

Modified Condition/Decision Coverage (MC/DC)
- Each cond takes on every possible outcome
- Each cond independently affects decision’s outcome
  \[ \Rightarrow n + 1 \text{ tests} \]
MLcov: code coverage for O’Caml

MC/DC measure capable tool for O’Caml

```
let all_positive1 a b c =
    (a > 0) && (b > 0) && (c > 0);;

let all_positive2 a b c =
    (a > 0) && (b > 0) && (c > 0);;

let all_positive3 a b c =
    (a > 0) && (b > 0) && (c > 0);;
```

(* all_positive1 1 1 1 ;; *)
(* all_positive1 1 1 0 ;; *)
(* all_positive2 1 1 1 ;; *)
(* all_positive2 1 0 1 ;; *)
(* all_positive2 1 1 0 ;; *)
(* all_positive3 1 1 1 ;; *)
(* all_positive3 0 1 1 ;; *)
(* all_positive3 1 0 1 ;; *)
(* all_positive3 1 1 0 ;; *)
### Structural coverage statistics

<table>
<thead>
<tr>
<th>Function name</th>
<th>Covered points</th>
<th>Total points</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_positive1</td>
<td>1</td>
<td>1</td>
<td>100 %</td>
</tr>
<tr>
<td>all_positive2</td>
<td>1</td>
<td>1</td>
<td>100 %</td>
</tr>
<tr>
<td>all_positive3</td>
<td>1</td>
<td>1</td>
<td>100 %</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>3</strong></td>
<td><strong>3</strong></td>
<td><strong>100 %</strong></td>
</tr>
</tbody>
</table>

### MC/DC statistics

<table>
<thead>
<tr>
<th>Decision number</th>
<th>Covered conditions</th>
<th>Total conditions</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 (all_positive1)</td>
<td>1</td>
<td>3</td>
<td>33 %</td>
</tr>
<tr>
<td>#2 (all_positive2)</td>
<td>2</td>
<td>3</td>
<td>66 %</td>
</tr>
<tr>
<td>#3 (all_positive3)</td>
<td>3</td>
<td>3</td>
<td>100 %</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>6</strong></td>
<td><strong>9</strong></td>
<td><strong>66 %</strong></td>
</tr>
</tbody>
</table>
Traceability from Source to Binary

Making relation between Source & Assembly explicit

- Translation of Explicit Control of the Source Code
  - pattern matching, exceptions handling, ...
- Controls Introduced by the Compiler Itself
  - memory allocation, array access, functional application mechanism
- Primitive functions
  - translated either to assembly or external function calls

Thanks to different intermediate languages from Source to Assembly
Timeline Report

« 2005 – 2009 »

- Prototype for KCG written in O’Caml
- From prototype to final product, different ways:
  1. Rewrite prototype in C or Ada,
  2. Using an O’Caml to C compiler to certify generated C code
  3. Or Directly certify O’Caml code
- For choice 3, modify the runtime library and build test tools for O’Caml
- Industrial development process by Esterel-Technologies
- Certification IEC 61508 & EN 50128 for railways
- Used in several DO-178B projects
Successful use of a functional language to develop certified safety-critical software

High-level requirements $\equiv$ Low-level requirements
- shorter certification process

Last version of KCG written in O’Caml
- easier & faster to write a compiler

Certification Norms are Open
- Made it possible to use a functional language to write a compiler to develop safety-critical embedded tools in a Certification Framework

A new domain for ICFP languages