C/C++ Code Generation for Embedded Devices

Daniel Siegl
CEO LieberLieber Software
daniel.siegl@lieberlieber.com
@danielsiegl
www.lieberlieber.com
Introduction – Daniel Siegl

- I am Enterprise Mobility Guy
- Developing Solutions for model based engineering
- Tools for “real” embedded
Introduction – LieberLieber Software

- Vienna, Austria www.lieberlieber.com
- Houston, Texas www.lieberlieber.us
- 20+ engineers
- OMG member

→ Solutions and consulting for model-based software and systems engineering
Generate Code from behaviour Models

- What is a Model?
- What is generated Code?
- What is a Behaviour Model?
- Why should I use this?
What it is about – UML Behavior?

- Attributes and Operations are declaration only
  ⇒ Same as HEADER files in C or C++
- And now - how to implement using UML??
- UML introduced behaviors to extend classic object oriented concept
- Behavior is UML concept for “implementation”
  ⇒ Same as SOURCE files in C or C++
Traffic Alert – while parking!
What shall we do now?
• Stop the car
• Ignore the alert
• ..... So many options .....
Generate Code from behaviour Models

Lot’s of new and old challenges ahead!

- Functional Safety (ISO 26262,...)
- UI complexity
- Multi and many core hardware
- Traceability
- Documentation requirements
Generate Code from behaviour Models

Inspiration:
- Higher level of abstraction than the generated code – especially State Charts are very powerful
- Render requirement and hazard information into the code automatically!
- Documentation = Product
class InterfaceTest
{
public:

/// auto generated virtual destructor
virtual ~InterfaceTest() {}

/// This is the description of method doA()
/// @param param1: this is the description of parameter param1
///
/// @covers REQ-1
virtual bool doA(uint8 param1) = 0;

/// this is the description of method doC()
virtual void doC() = 0;
};

REQ-1 My Requirement

Traceability from Requirement to Code
Generate Code from behaviour Models

- Full (Behavior) round trip is a myth
- 2017 forward only
- Reverse for legacy
- Optional: synchronization of method/function content
Reuse your source code, integrate it into your model

```c
typedef my_struct my_struct;
struct my_struct
{
    int my_attribute;
};
void my_operation(int my_parameter);

if(...)
{
    my_operation(1);
}
else
{
    my_operation(0);
}
```

Integrate into your model by reverse engineering or creating stub in the model

Generate new code by Forward Engineering
Template/VRA Programmed generator

- All modeling tools offer template based code generators – XTEXT, T4 and others
  - You need to learn something new
  - Hard to "debug" and find issues
- Old school code generators – are just a simple program
  - Every one in your Team should be able to understand the code
  - Very good Tool support to "debug" the generator
The missing Link – Debugging for Models

Abstraction

Model

Code

Assembler

? (Unknown)

IDE / DEBUG

Tools
The missing Link – Debugging for Models

- Engineers need feedback – Feedback means debugging
- We need them to debug with the model
- Ability to understand and fix issues in the model/generator and not in the “code”.
- Pure UML “Simulation” is not the best solution for embedded
- No Instrumentation – only Debugging!
Our Approach

- Magic Draw / Cameo Systems Modeler to build the models
- Programmed and debug able code generator (C#)
- C or C++ source code
- Visual Studio and Code Composer
- LieberLieber UML Debugger linked to TRACE32 from Lauterbach / I-System / PLS / Greenhills /....
bool Millionenrad_MainLogic(const Millionenrad* me, Millionenrad_MainLogic_STM* stm, Signals smg)
{
    bool evConsumed = 0;
    switch(stm->mainstate.activeSubstate)
    {
    case Millionenrad_MainLogic_INITIALIZATION:
        evConsumed = 1;
        /* Initialization */
        stm->mainstate.activeSubstate = Millionenrad_MainLogic_Main;
        Millionenrad_MainLogic_EnterDeepHistory(me, stm, &stm->standby);
        break;
    case Millionenrad_MainLogic_Main:
        /* do actions for state Main */
        Accelerometer_UpdateAxis(&me->acc, &me->accd);
        /* end of do actions for state Main */
        switch(stm->mainstate.activeSubstate)
        {
        }
DEMO
Our Approach/Demo

- Lauterbach, PLS, I-System, GHS, Visual Studio Debuggers
- Certification possible
- Fast/Extensible code generation
- Generation can be debugged using familiar techniques
- No framework
Our Approach

- Need to know what you want
- Not a turnkey solution
- Hardware breakpoints
- Very suitable for legacy projects
- Based on Magic Draw
Outlook

- Technology can also be used to create modules for Co-Simulation from a Systems Engineering Workflow (FMI/FMU)
- ALF to C/C++ currently under research
- Reverse Engineering of Sequence Diagrams from Debugger Traces
Conclusion

- Don’t wait for next “new” project – try to start with a legacy project
- Can you afford and “survive” not to generate code?
- Good “debugging” will raise acceptance and efficiency
- Code and model are in sync
Conclusion

- The API way of thinking....
  - Your operations are your API
  - State charts are used to build the Application
Conclusion

- “programmed” code generator can be smart – despite feeling old school
- Think about execution semantics! (Standards vs. Vendors vs. Legacy)

⇒ Start TODAY
C/C++ Code Generation for Embedded Devices

Thank you for your attention!

Daniel Siegl
CEO LieberLieber Software
daniel.siegl@lieberlieber.com
@danielsiegl
www.lieberlieber.com