C/C++ Code Generation for Embedded Devices

Daniel Siegl

CEO LieberLieber Software

daniel.siegl@lieberlieber.com

@danielsiegl

www.lieberlieber.com

LieberLieber

Introduction – Daniel Siegl

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

LieberLieber

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

LieberLieber

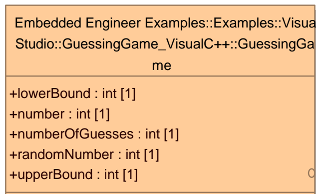
Generate Code from behaviour Models

- What is a Model?
- What is generated Code?
- What is a Behaviour Model?
- Why should I use this?

LieberLieber

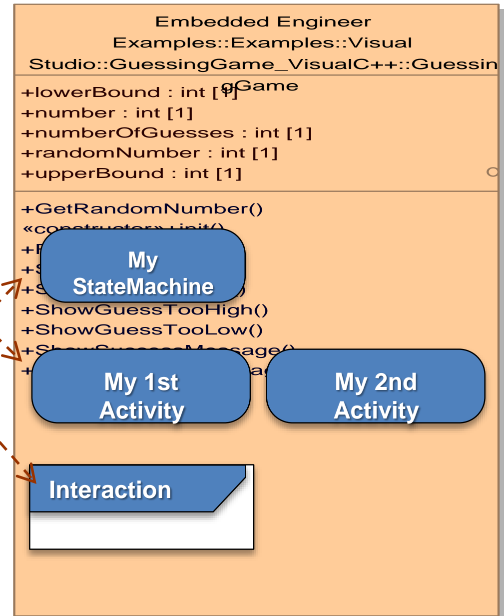
What it is about – UML Behavior?

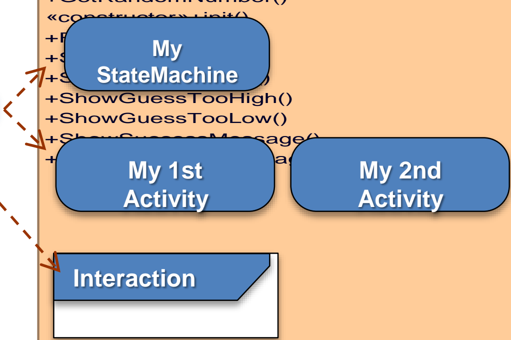
Classic Object Oriented



OO extended by Behavior concept

- 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++

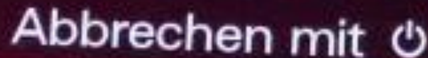




LieberLieber

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

LieberLieber

```
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

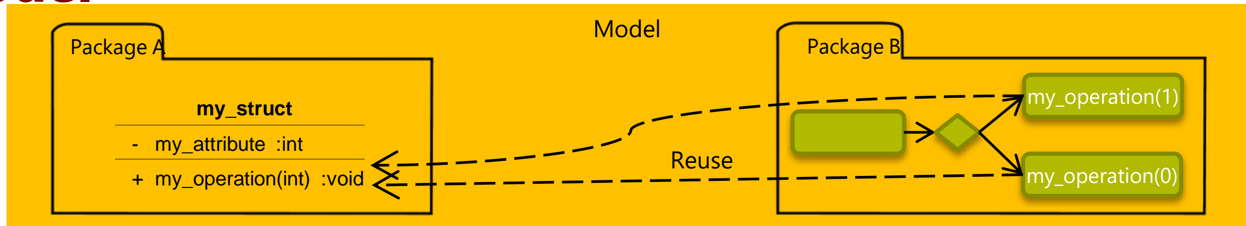
LieberLieber

Generate Code from behaviour Models

- Full (Behavior) round trip is a myth
- 2017 forward only
- Reverse for legacy
- Optional: synchronization of method/function content

LieberLieber

Reuse your source code, integrate it into your model



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

Generate new code
by Forward Engineering

existing source code

```
typedef my_struct my_struct;
struct my_struct
{
  int my_attribute;
};

void my_operation(int my_parameter);
```

Reuse

Generated source code

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

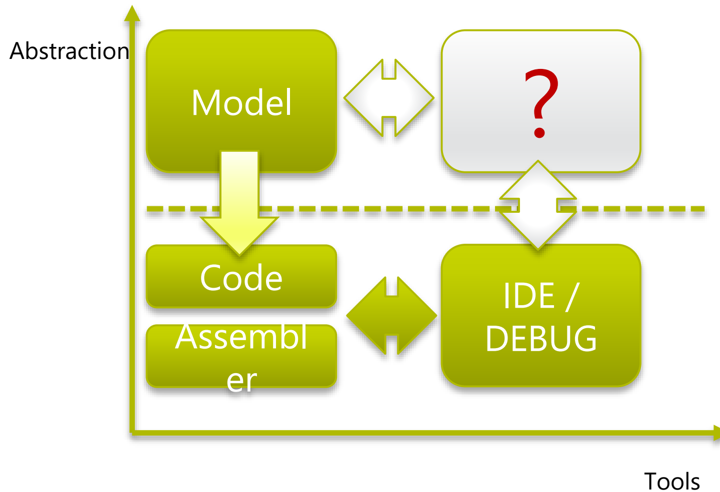
LieberLieber

Template/VS Programmed generator

- All modeling tools offer template based code generators – XTEXT, T4 and others
  - You need to learn some thing 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

LieberLieber

The missing Link – Debugging for Models



LieberLieber

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!

LieberLieber

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 /….

LieberLieber

Developer Machine

«No Magic»
Magic Draw:
Modeling Platform

«LieberLieber»
Embedded Enginner:
Debugger

«LieberLieber»
Embedded Engineer:

create

Your Model: MD project

generate

:Source Code

control

Embedded IDE

«Lauterbach»
TRACE32 Debugger

Compiler

compile

:Binary Code

{alternative variants}

«Lauterbach»
TRACE32 :
Debug Device

: USB

: JTAG

: Ethernet

: USB

: Ethernet

Embedded System (Target)

:Binary Code

: JTAG

download

LieberLieber

Millionenrad.h | Millionenrad.c | t_tors_t2.asm

```
215            default:
216              break;
217          }
218          break;
219        case Millionenrad_MainLogic_Standby:
220          switch(stm->Standby.activeSubState)
221          {
222            case Millionenrad_MainLogic_DecideNextIdleAnimation:
223              /* DecideNextIdleAnimation -> EntryPoint */
224              stm->Standby.activeSubState = Millionenrad_MainLogic_IdleAnimations;
225              stm->IdleAnimations.startTime = FSM_getTime();
226              if((me->animationCycle % 5) == 0)
227              {
```

- miLLionenrad
  - miLLionenrad
  - Implementation
    - miLLionenrad
    - ADC
    - Accelerometer
    - App
    - Buzzer
    - LightStrip
    - Millionenrad
      - Millionenrad
      - DecideWinState
      - InitSystem
      - MainLogic

MainLogic | Standby

Initialization: + Init() :entry

Initial

Main: + UpdateAccelerometer() :do

GewinnAnzeige

LeiderNein

Standby

History

RotationSpeedExit

[fabs(me->acc.rotationSpeed) > 0.25f]

Drehung

[me->animationSwitchable]

WaitVariableTime

Target Connected

Breakpoints

Name	Id
Millionenrad.c:220	C:\T32\MILLIONENRAD\GENE

Locals

Name	Value	Type
me.acc	tryin...	Acc...
me.acc.angle	0	tryi...
me.acc.AN...	no fi...	tryi...
angleHistory	2000...	floa...
me.acc.axis1	0	tryi...
me.acc.axis2	0	tryi...
me.acc.axis3	0	tryi...

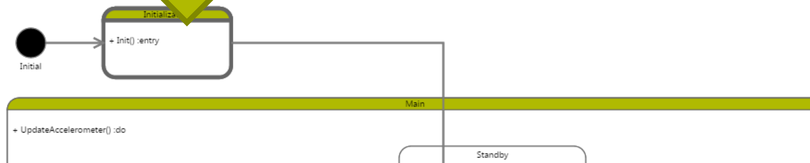
LieberLieber

Millionenrad.h | **Millionenrad.c**

```
57  bool Millionenrad_MainLogic(Millionenrad* const me, Millionenrad_MainLogic_STM* stm, Signals msg)
58  {
59    bool evConsumed = 0;
60    switch(stm->mainState.activeSubState)
61    {
62      case Millionenrad_MainLogic_Initialization:
63        evConsumed = 1;
64        /* Initialization -> History */
65        stm->mainState.activeSubState = Millionenrad_MainLogic_Main;
66        Millionenrad_MainLogic_EnterDeepHistory(me, stm, &(stm->Standby));
67        break;
68      case Millionenrad_MainLogic_Main:
69        /* do actions for state Main */
70        Accelerometer_Up[illegible]ateAxis(&me->acc, &me->adc);
71        /* end of do [illegible] for state Main */
72        switch(stm-[illegible]eSubState)
73        {
```

MainLogic

Selected state also visualized in C



LieberLieber

DEMO

LieberLieber

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

LieberLieber

Our Approach

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

LieberLieber

Outlook

- Technology can also be used to create moduls 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

LieberLieber

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

LieberLieber

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

LieberLieber

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

LieberLieber