Your partner when introducing and using modern software development tools

Klaus Wachsmuth
Dr. Peter Dencker
Aonix - Worldwide Presence

- In business since 1980
- HQ in San Diego
- In Software TOP 500
- 300+ employees
- Offices in Germany, France, Sweden, UK and US
- Represented in 28 countries
- A Gores Technology Group (GTG) company
Aonix GmbH - Close to You

with Consulting, Sales, Support and Training about our Products

Region D A CH
including Eastern Europe

Offices in Karlsruhe, München and Düsseldorf
25 Employees
Aonix Product Families

- **Software through Pictures (StP)**
  Family of modeling tools
  - /UML for object oriented analysis and design
  - /ACD template driven code generation
  - /SE for structured analysis and design

- **ObjectAda**
  Software development environment for Ada 95

- **Raven**
  Certifiable runtime kernel for safety critical real-time applications

- **TeleUSE**
  Generator for graphical user interfaces based on Motif
Ada-Development

• ObjectAda
  – Ada 95 development environment for PCs and Unix
  – ObjectAda Windows and ObjectAda Real-Time

• Real-Time/Raven:
  – implements Ravenscar Profile (RP)
  – Checks the RP properties during compilation
  – Certification documents for highest criticality (DO-178B) with warranty available
  – supporting partitioning
Raven Certification

- December 2001:
  Pratt & Whitney certification was achieved at software Level-A of RTCA's DO-178B for the PW6000 commercial jet engine

- June 2002:
  Pratt & Whitney has selected Aonix ObjectAda®/Raven™ for its next generation military jet engine for JSF
From UML Design Pattern to Safety Critical Software

- Introduction
- Elements of Design Pattern
- Model elements from the Ravenscar Profile
- Automatic code generation
- Example
- Conclusion
UML Introduction

- UML language with powerful graphical expressiveness
  concentration on
  - class diagrams
  - state diagrams

- Semantical interpretation freedom
  - Commercial Software
  - Real-time Software

- Concentration on Real-time Systems
Real-time System - Characteristics

• Natural parallelism in design structures

• Historical solutions: sequentialized fixed time frames

• Typical example
  – cyclic readout of a sensor value
  – put into a buffer
  – then processed by controllers and visualized on displays
Real-time System - Items

- Active items - active classes
  (state automata)
  => own control flow
  => Thread, Ada Task

- Passive items
  (Ex: buffer)
  => no independent control flow
  => Module, Ada Package

- How are they modelled with UML?
UML Stereotype

- Defines meta-property
- Used to classify UML items
- Constraints, Tagged Values for the refinement of the meta-properties
Cyclic Class Characteristics

- Stereotype = "CyclicTask"
  - has its own control flow
  - runs endless
  - Priority
  - Periodicity
Buffer Characteristics

- Stereotype = „resource control“
  - no independant control flow
  - implicit put method
  - implicit get method
  - synchronisation of methods

- Tagged Value: Element type
- Tagged Value: Number_of_Elements
Simple Design Pattern

communication between parallel activities
Details: Tagged Values

- **Class Pressure**
  - priority = 1
  - period = milliseconds(100)

- **Class PressureControl**
  - priority = 2
  - period = milliseconds(200)

- **Class PressureBuffer**
  - ItemType = Pressure_Type
  - No_Of_Elements = 100
Ravenscar Profile

• Industrial standard for safety critical real-time systems with Ada
• Idea: structuring an application with a set of tasks
  – cyclic
  – sporadic
  – cooperating
• communicating through events and buffers
Active model elements

- **RepetitiveTask**
  - recurrent activity without fixed period
  - tagged values: priority, stacksize

- **CyclicTask**
  - like RepetitiveTask, but with fixed period
  - tagged values: priority, stacksize, period
Cyclic Task Stereotype with attributes

UML Class

`<<Cyclic>>`

`CycTask`

`<<Tags>>`

StackSize = 2000
Priority = 28
Period = 50 ms

Source code

```
task CycTask is
  pragma Priority (28);
  pragma Storage_Size (2000);
end CycTask;

task body CycTask is
  Next_Time : Time;
  Period : Time_Span := Milliseconds(50);
begin
  Next_Time := Clock;
  loop
    delay until Next_Time;
    -- body
    Next_Time := Next_Time + Period;
  end loop;
end CycTask;
```

Generated Code (part)
More active model elements

• Transporter
  – like cyclic, but including
  – a Get-Association and
  – a Put-Association
  – priority, stacksize, period
  – Item_Type
Example: Transporter
More active model elements

- **SporadicTask**
  - waits for an event or interrupt
  - has an association to a class which represents the event
Example: Alarm
Code Generation

• Template based Code generator
  – simple mapping of patterns to code, selection via stereotypes
  – implementation of complicated pattern
  – semantic checking
  – easy modifiability
  => new patterns
Code generation

- Static semantic
  => class diagrams
- Dynamic semantic
  => State automata
- Patterns are language independant
- Ada provides convient language concepts
- Mapping to C, C++, Java and other languages possible
template CyclicTaskBody(MClass)
[OutputWiths([MClass])]
with Ada.Real_Time; use Ada.Real_Time; -- To get visibility to the "+" operator.
package body [MClass.name]_Pkg is
[genStateMachine([MClass])]
task body [MClass.name] is
     Next_Time : Ada.Real_Time.Time;
     Period    : constant Ada.Real_Time.Time_Span := [Period([MClass])];
[transporter_decl([MClass])]
begin
     Next_Time := Ada.Real_Time.Clock;
     loop
          delay until Next_Time;
[transporter_get([MClass])]
[genStateMachineCall([MClass])]
[mergeOut("UCOD:: "getUniqueId([Mclass,"User Def Code", "]")]
     Next_Time := Next_Time + Period;
     end loop;
end [MClass.name];
end [MClass.name]_Pkg;
end template
with PressureSensor_Pkg; with PressureBuffer_Pkg;
with Ada.Real_Time; use Ada.Real_Time;
package body Pressure_Pkg is

  task body Pressure is

  Next_Time : Ada.Real_Time.Time;
  Period    : constant Ada.Real_Time.Time_Span := Milliseconds(100);
  Item : Pressure_Type;

  begin
    Next_Time := Ada.Real_Time.Clock;
    loop
      delay until Next_Time;
      Item := PressureSensor_Pkg.Get; -- Raven Class Package
      PressureBuffer_Pkg.Put (Item); -- Raven Class Package
      --#ACD# M(UCOD:: 102:BOTTOM) User Defined Code
          -- Section for User Defined Code
      --#end ACD#
      Next_Time := Next_Time + Period;
    end loop;
  end Pressure;

end Pressure_Pkg;
Passive Element: RC

template ResourceControlSpec(MClass)
with System; -- for Priority value.
[end if]
package [MClass.name]_Pkg is
  protected [MClass.name] is
    function Get return [SharedDataType([MClass])];
    procedure Put(Item : [SharedDataType([MClass])]);
    [if (HasInterruptId([MClass]))]
      HandlerSpec([MClass])
    [end if]
    [ProtectedPriority([MClass])]
  private
    Shared_Data : [SharedDataType([MClass])];
  end [MClass.name];
end [MClass.name]_Pkg;
end template
package body PressureBuffer_Pkg is
  protected body PressureBuffer is
  function Get return Pressure_Type is
    begin
      return Shared_Data;
    end Get;
  procedure Put(Item : Pressure_Type) is
    begin
      Shared_Data := Item;
    end Put;
  end PressureBuffer;
end PressureBuffer_Pkg;
Example: Pressure Control
Conclusion

- Ravenscar profile patterns
  - language independant
  - easy mapping to Ada
- Easy composition of patterns
- High level of abstraction
- Mapping to target language realized thru template driven code generation

=> OMG Model Driven Architecture