The MathWorks™

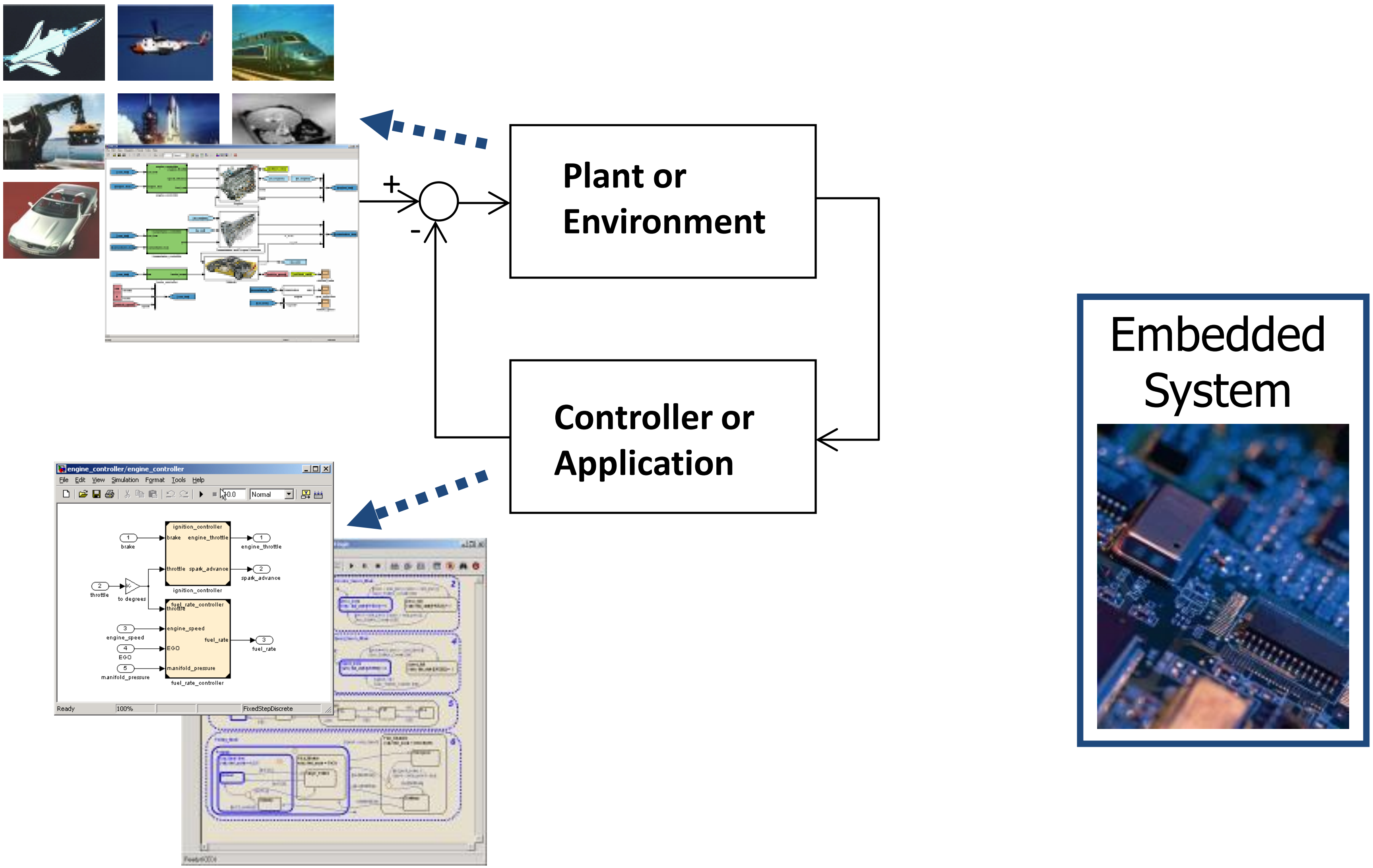
Software Synthesis for Control System Algorithms in Industrial Applications

Emmanuel Roy – The MathWorks

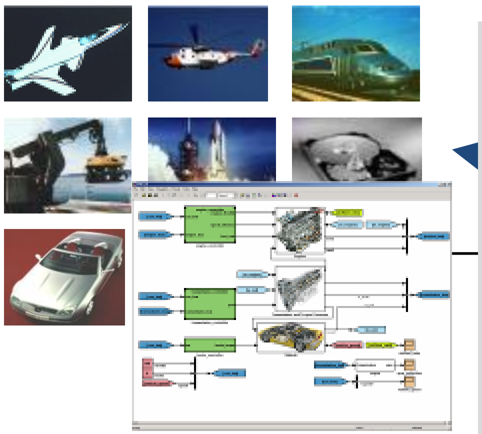
WorkShop on Software Synthesis

Friday, Oct. 16th, 2009, Grenoble, France

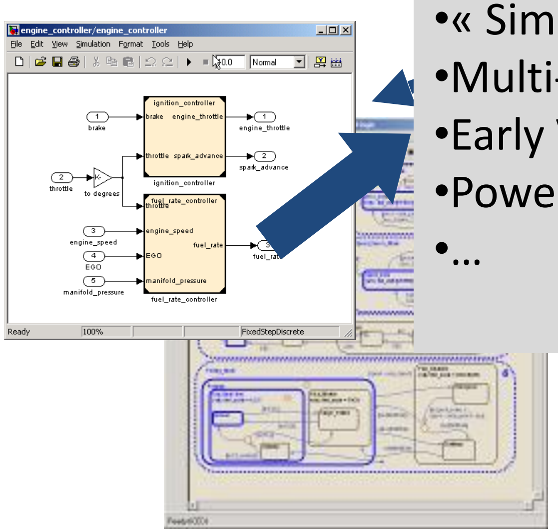
Model-Based Software Synthesis Overview

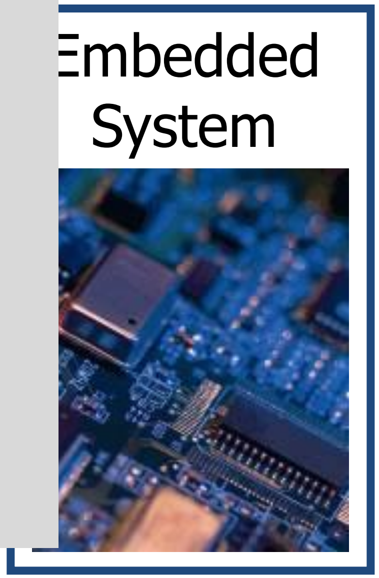


Model-Based Software Synthesis Overview

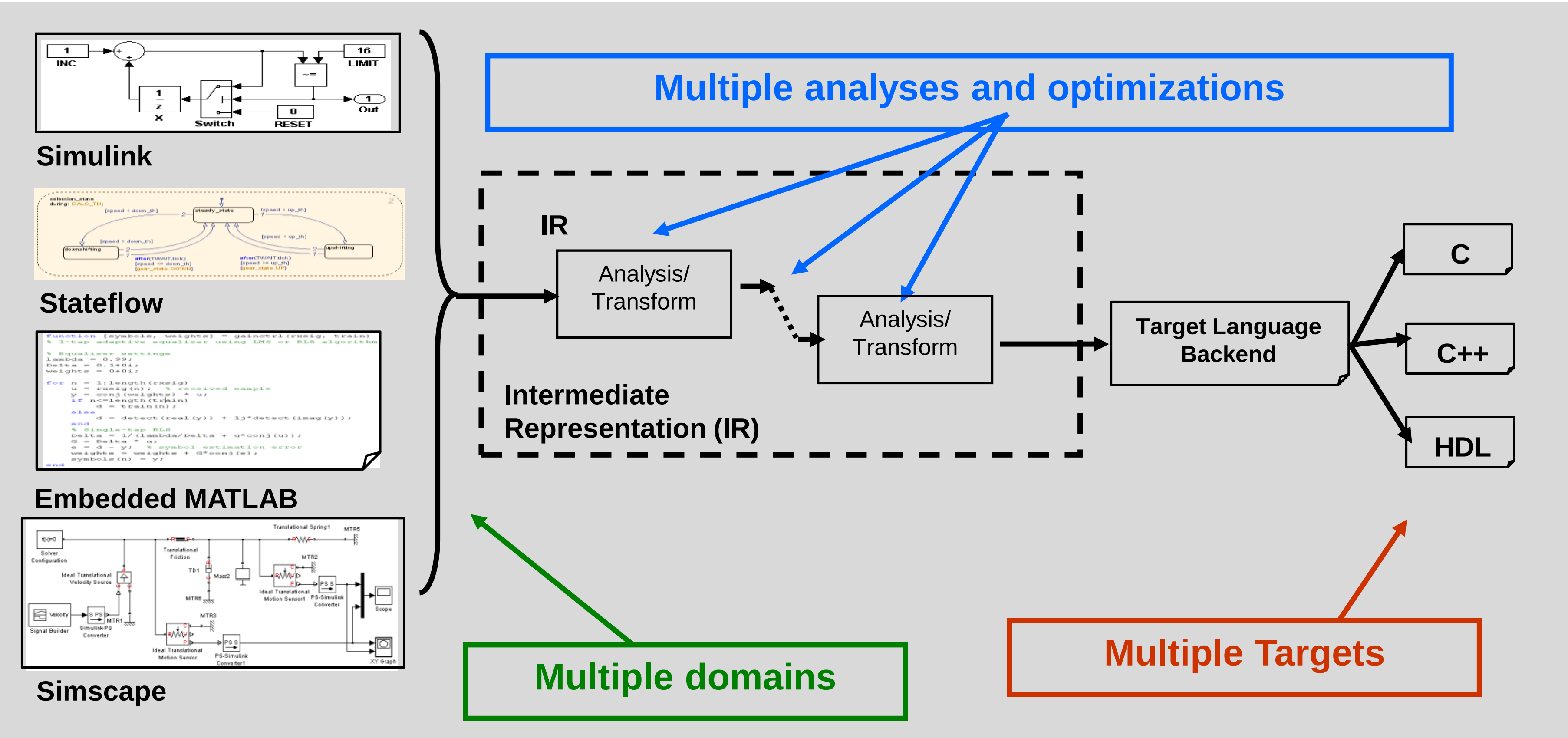


- High level of Abstraction:
  - Dynamic typing
  - Dynamic sizing
  - Type and rate propagation
  - Concise math representation
  - …
- Hierarchical Algorithm design
  - Complexity decomposition
  - Reuse
- « Simulable » (executable specification)
- Multi-Domain Modelling
- Early Verification&Validation
- Powerful visualization possibilities
- …

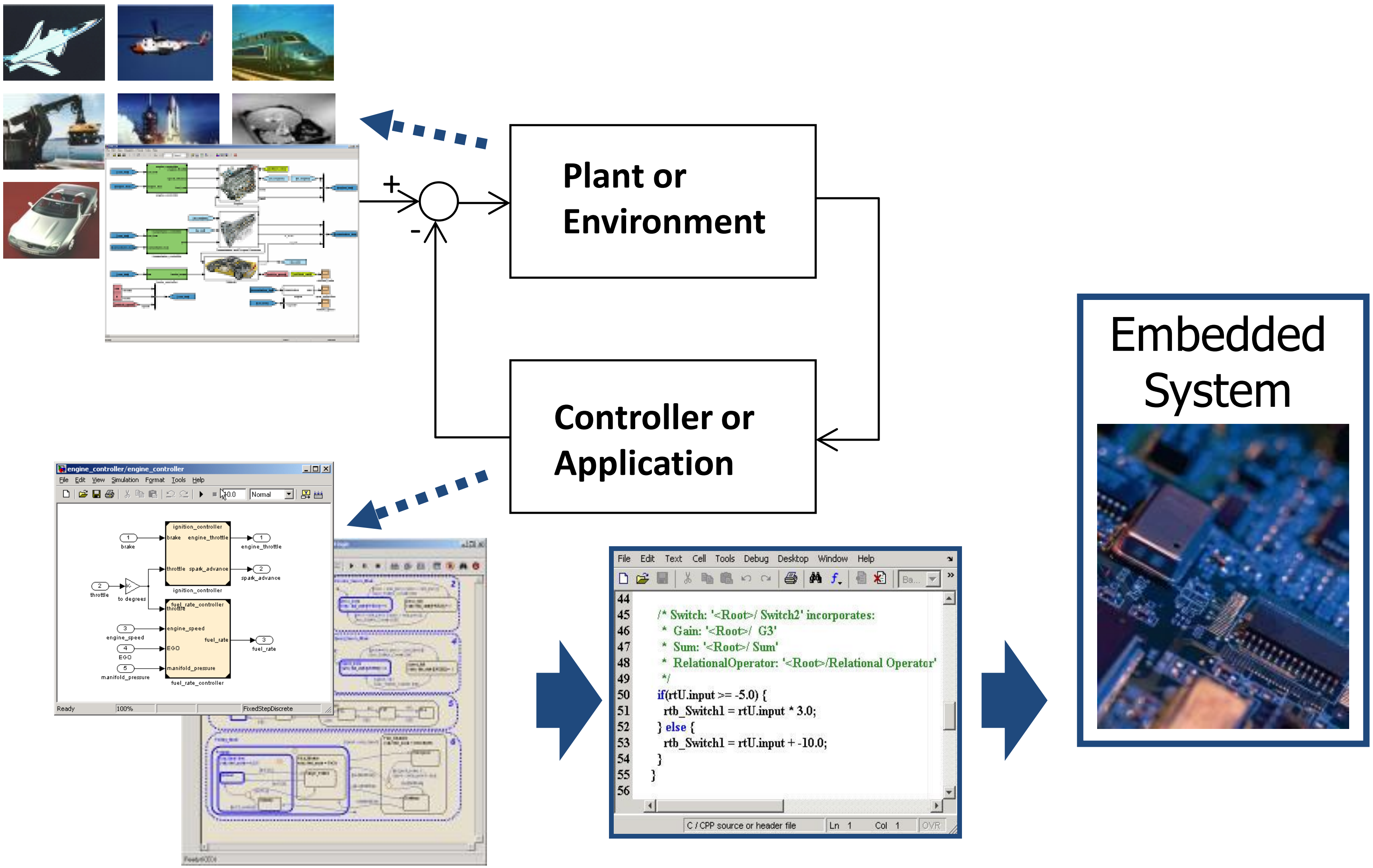




Model-Based Software Synthesis Overview



Model-Based Software Synthesis Overview

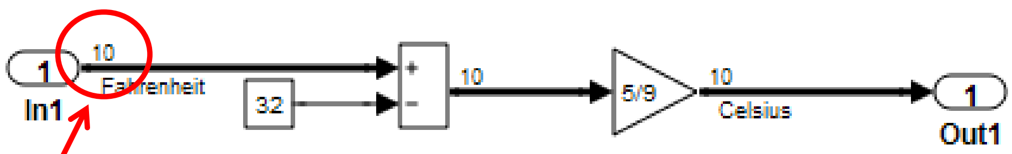


Model-Based Software Synthesis Key Benefits

- Separate algorithm from implementation
- Parameterizeable implementations
- Target independence
- Flexibility and reusability
- Multi (targeted) language support

Model-Based Software Synthesis Key Benefits

- The model expresses the algorithm concisely while hidding operation’s implementation details:

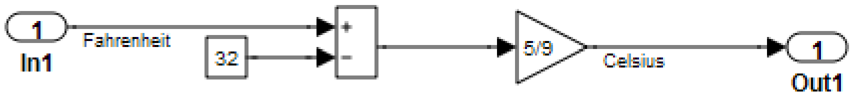


Implicit for loop

```
void fahrenheit2celsius(void) {
  int32_T i;
  for (i = 0; i < 10; i++) {
    Celsius[i] = (Fahrenheit[i] - 32.0) * 5.5555555555555558E-001;
  }
}
```

Model-Based Software Synthesis Key Benefits

- The model-based algorithm is independent of the integration platform:




```
void fahrenheit2celsius (void) {
   Celsius = (Fahrenheit - 32.0) * 5.5555555555555558E-001;
}
```

Global data Access

```
void fahrenheit2celsius (const real_T Fahrenheit, real_T* Celsius) {
  (*Celsius) = (Fahrenheit - 32.0) * 5.5555555555555558E-001;
}
```

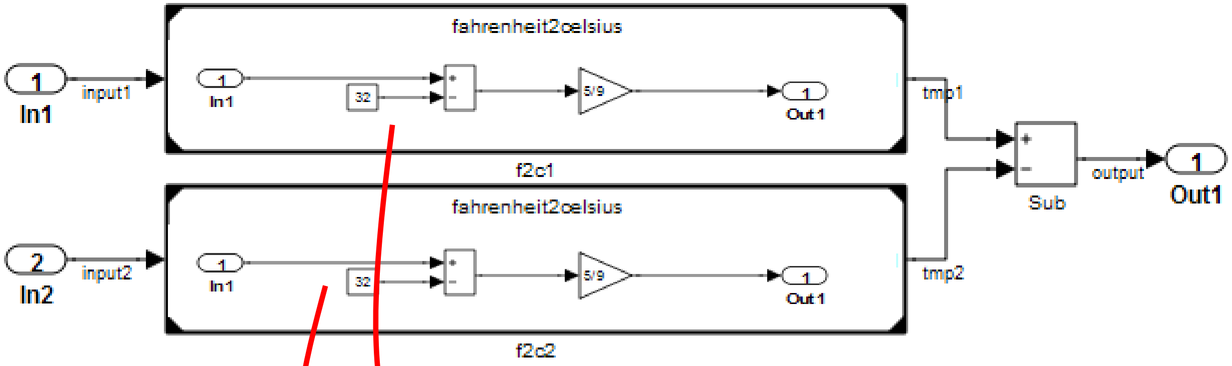
Function

AUTOSAR

```
void fahrenheit2celsius(void) {
   real_T rtb_Fahrenheit;
   Rte_Read_RPort_Fahrenheit(&rtb_Fahrenheit);
   Rte_Write_PPort_Celsius((rtb_Fahrenheit - 32.0)*5.5555555555555558E-001);
}
```

Model-Based Software Synthesis Key Benefits

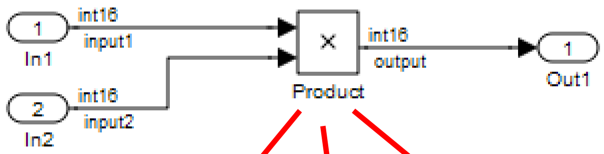
- Ensure high flexibility and reusability




```
void caller(void) {
  ...
  fahrenheit2celsius(&input1, &rtb_tmp1);
  fahrenheit2celsius(&input2, &rtb_tmp2);
  output = rtb_tmp1 - rtb_tmp2;
}
```

Model-Based Software Synthesis Key Benefits

- The model-based algorithm is independant of the specific target optimizations



Portable ANSI-C
(with overflow protection)

```
int32_T tmp;
tmp = (int32_T)input1 * (int32_T)input2;
if (tmp > 32767L) {
   output = MAX_int16_T;
} else if (tmp <= -32768L) {
   output = MIN_int16_T;
} else {
   output = (int16_T)tmp;
}
```

Compagny wide math operation

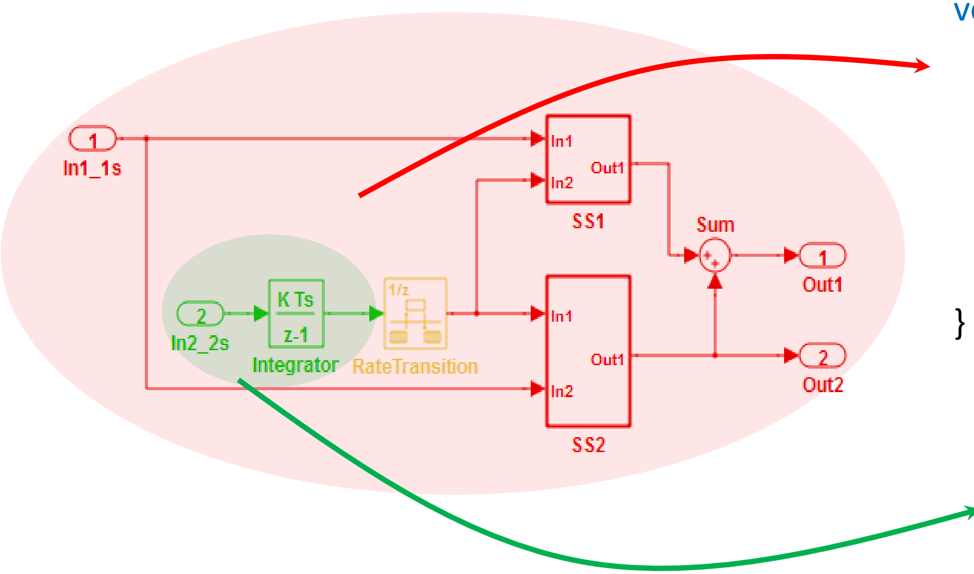
```
compagny_XXX_optimized_mul(&output, input1, input2);
```

Target specific optimized math operation

```
output = c28x_mul_s16_s16_s16_sat(input1, input2);
```

Model-Based Software Synthesis Key Benefits

- Automatic mapping of functional rates into software tasks (RM based scheduler)



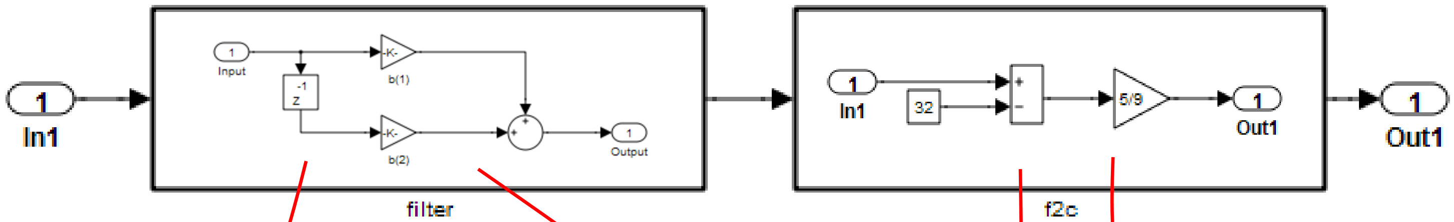

```
void task_fastest(void) {
   ...
   /* RateTransition: '<Root>/RateTransition' */
   if ((rtM->Timing.RateInteraction.TID0_1 == 1)) {
      rtB.RateTransition = rtDWork.RateTransition_Buffer0;
   }
   ...
}
```

```
void task_slowest(void) {
   ...
}
```

— Fastest rate
— Slowest rate

Model-Based Software Synthesis Key Benefits

- Multi-Language support



Verilog

```
...
  assign mul_temp = Input_rsvd * b_1_under_gainparam;
  assign b_1_out1 = ({{2{mul_temp[31]}}, mul_temp[31:0]}

  always @ (posedge clk or posedge reset)
    begin: BodyDelay2_process
      if (reset == 1'b1) begin
        BodyDelay2_out1 <= 0;
      end
      else begin
        if (enb_1_10_0 == 1'b1) b
          BodyDelay2_out1 <= Inpu
        end
      end
    end // BodyDelay2_process

...
```

VHDL

```
...
  mul_temp <= Input_signed * b_1_under_gainparam;
  b_1_out1 <= resize(shift_right(mul_temp(31) & mul_temp(31 DOWNTO 0)

  BodyDelay2_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      BodyDelay2_out1 <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
      IF enb_1_10_0 = '1' THEN
        BodyDelay2_out1 <= Input_signed;
      END IF;
    END IF;
  END PROCESS BodyDelay2_process;
...
```

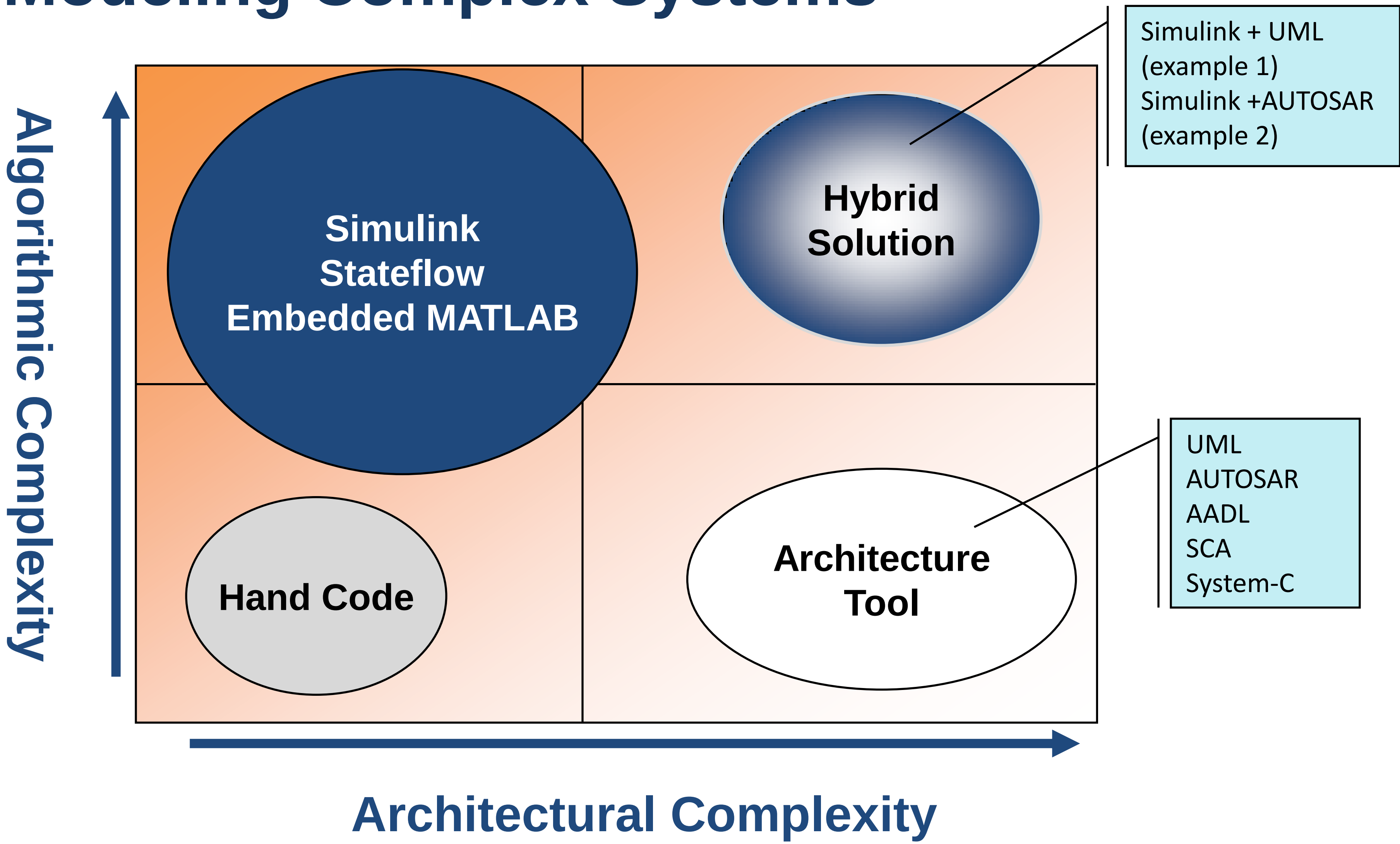
C

```
void f2c_step(void) {...}
```

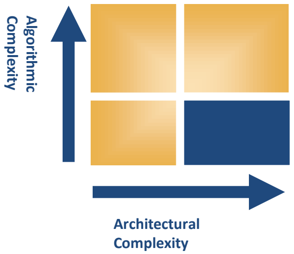
C++

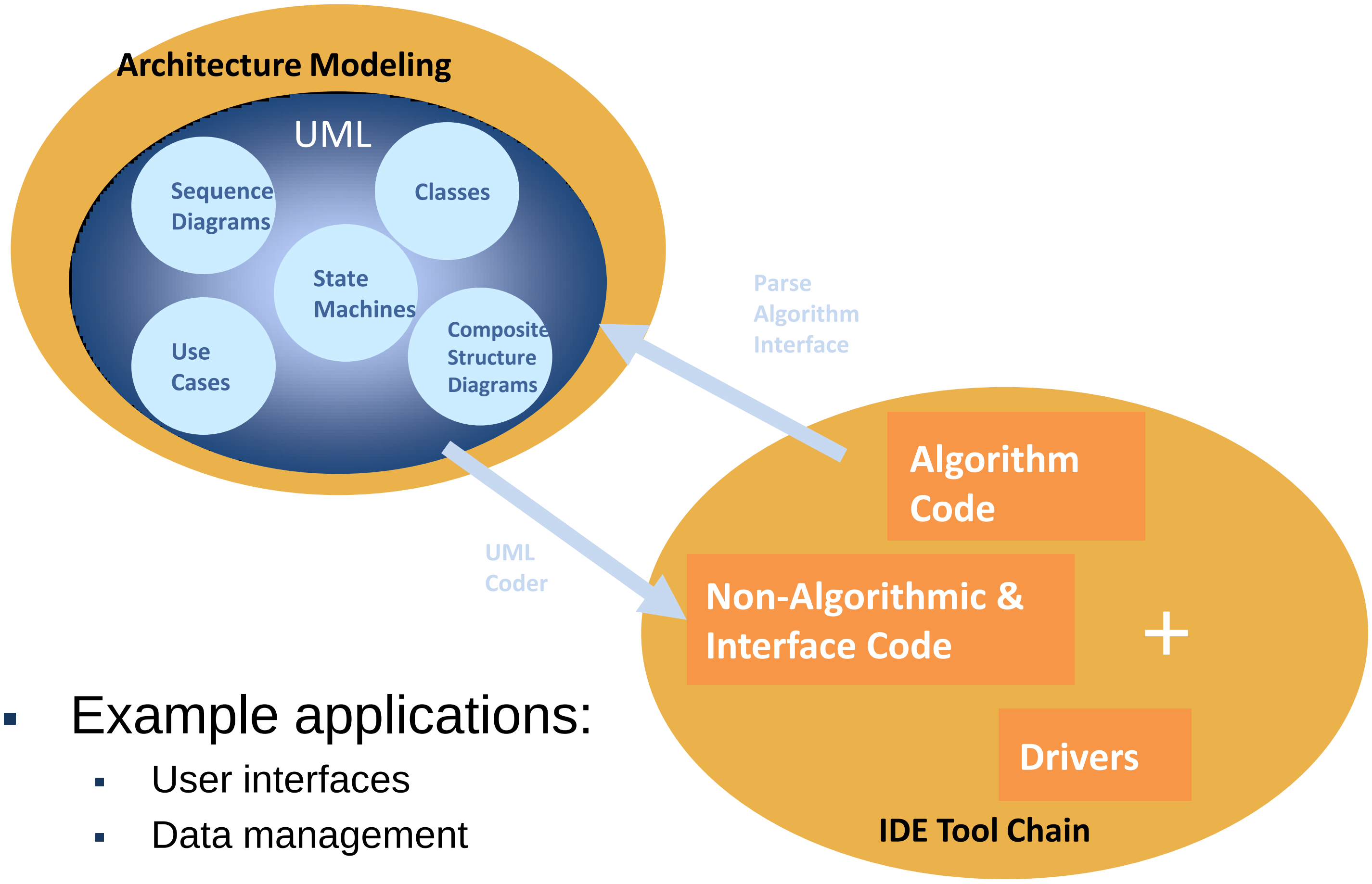
```
class f2c {
public:
    double compute(const double input);
    f2c();
    ~f2c();
};
```

Modeling Complex Systems



Projects with Low Algorithm Complexity and High Architectural Complexity

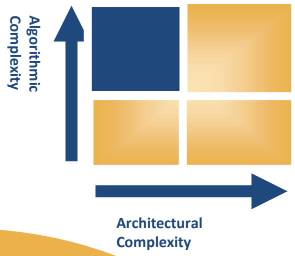


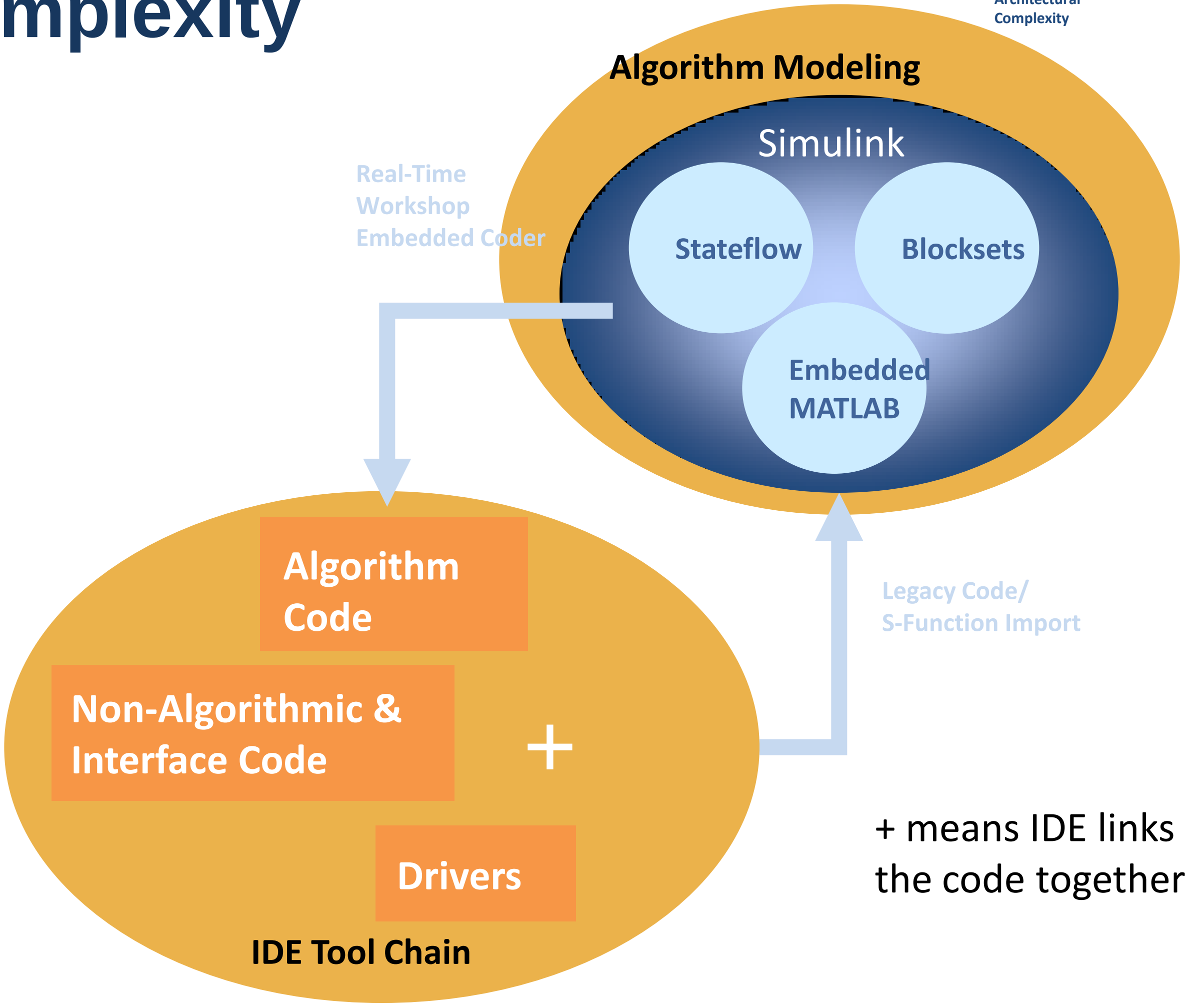


- Example applications:
  - User interfaces
  - Data management

+ means IDE links the code together

Projects with High Algorithm Complexity and Low/Moderate Architectural Complexity

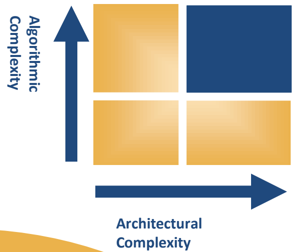


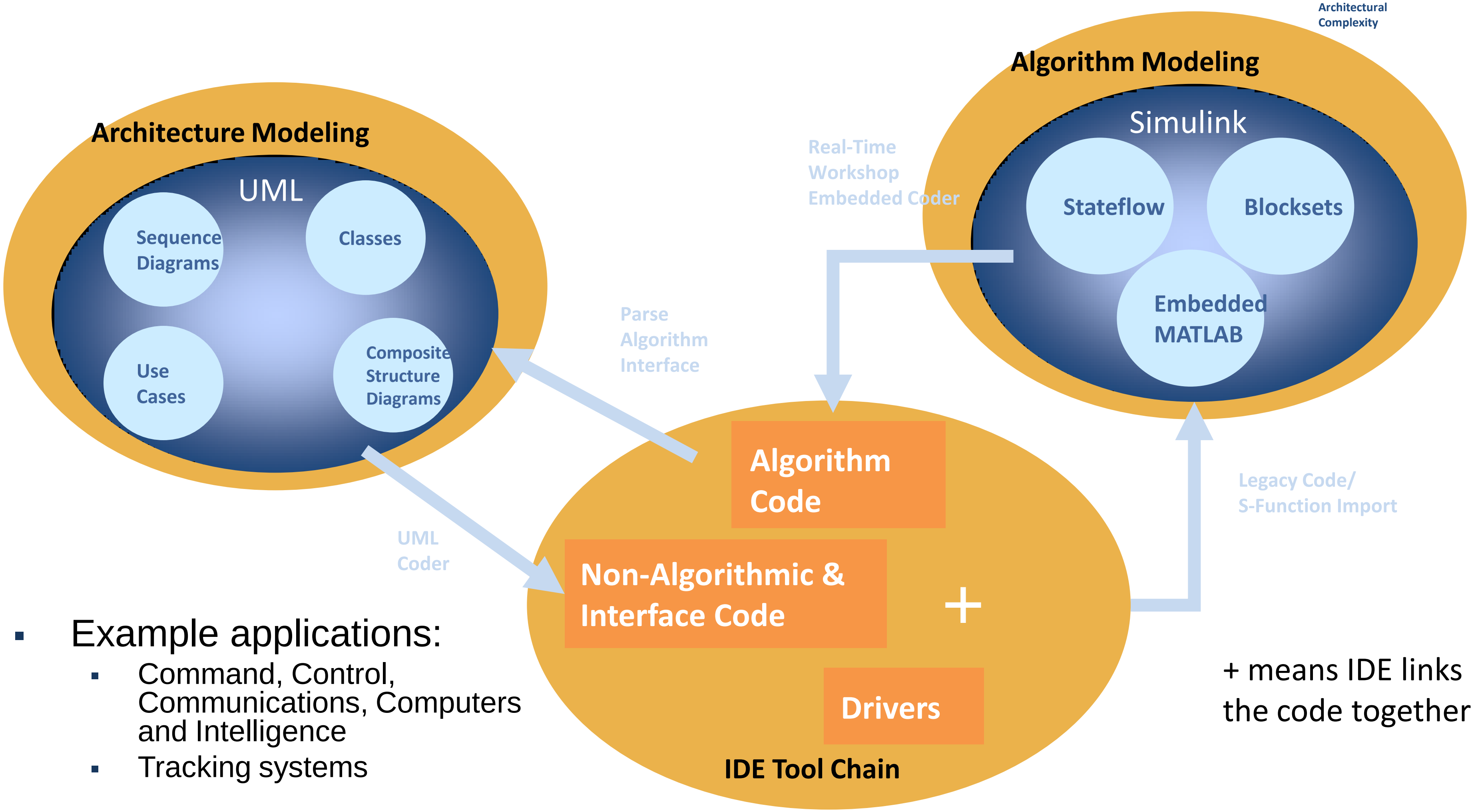


- Example applications:
  - Controllers (avionics, powertrain)
  - Signal processing

+ means IDE links the code together

Projects with High Algorithmic and Architectural Complexity





- Example applications:
  - Command, Control, Communications, Computers and Intelligence
  - Tracking systems

Example 1: Hybrid Simulink and UML Software Development

Goals

- Enable integration of Simulink components into a UML-defined software architecture
- Ensure that code generated from Simulink fits with framework generated from UML

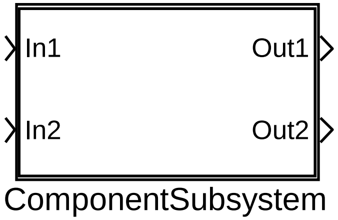
Role of Simulink

- Design and test an algorithmic component
- Generate C++ compatible production code

Role of a UML tool

- Define the architecture of the software system
- Capture the interface to Simulink generated code
- Generate non-algorithmic code

Mapping a Model to a “Component” Class



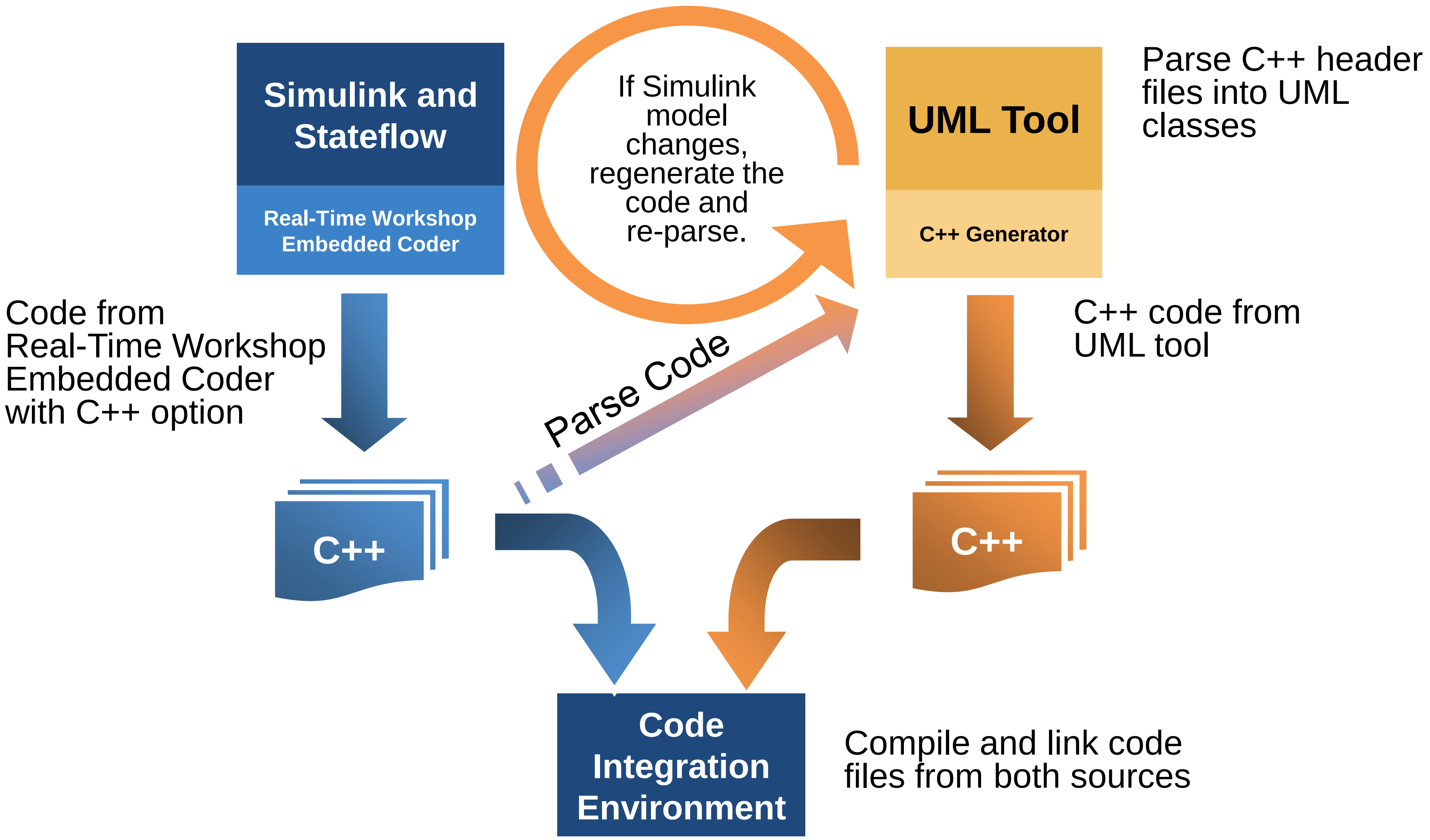

```
class ComponentSubsystem {
public:
  void initialize(void);
  void terminate(void);
  int16_T step_method(real_T &arg_In1,
    real_T *arg_In2, boolean_T *arg_Out2);

  ComponentSubsystem(void);

  ~ComponentSubsystem();
   RT_MODEL * getRTM(void);

protected:
   ExternalOutputs rtY;
private:
   RT_MODEL *rtM
};
```

```
ComponentSubsystem
---------------------------------
+initialize();
+terminate();
+step_method(in real_T arg_In1,
     in real_T arg_In2,
     out boolean_T arg_Out2) int16_T
+ComponentSubsystem();
+~ ComponentSubsystem();
+getRTM(): RT_MODEL
---------------------------------
#rtY: ExternalOutputs
-rtM: RT_MODEL
```

1. Build model (or atomic subsystem)
2. Generate code for model/subsystem*
3. Create a class for the subsystem
4. Relate operations to function members in code
5. Relate attributes to data members in code

* Uses automatic C++ encapsulation feature

Automated Code-Level Integration Approach

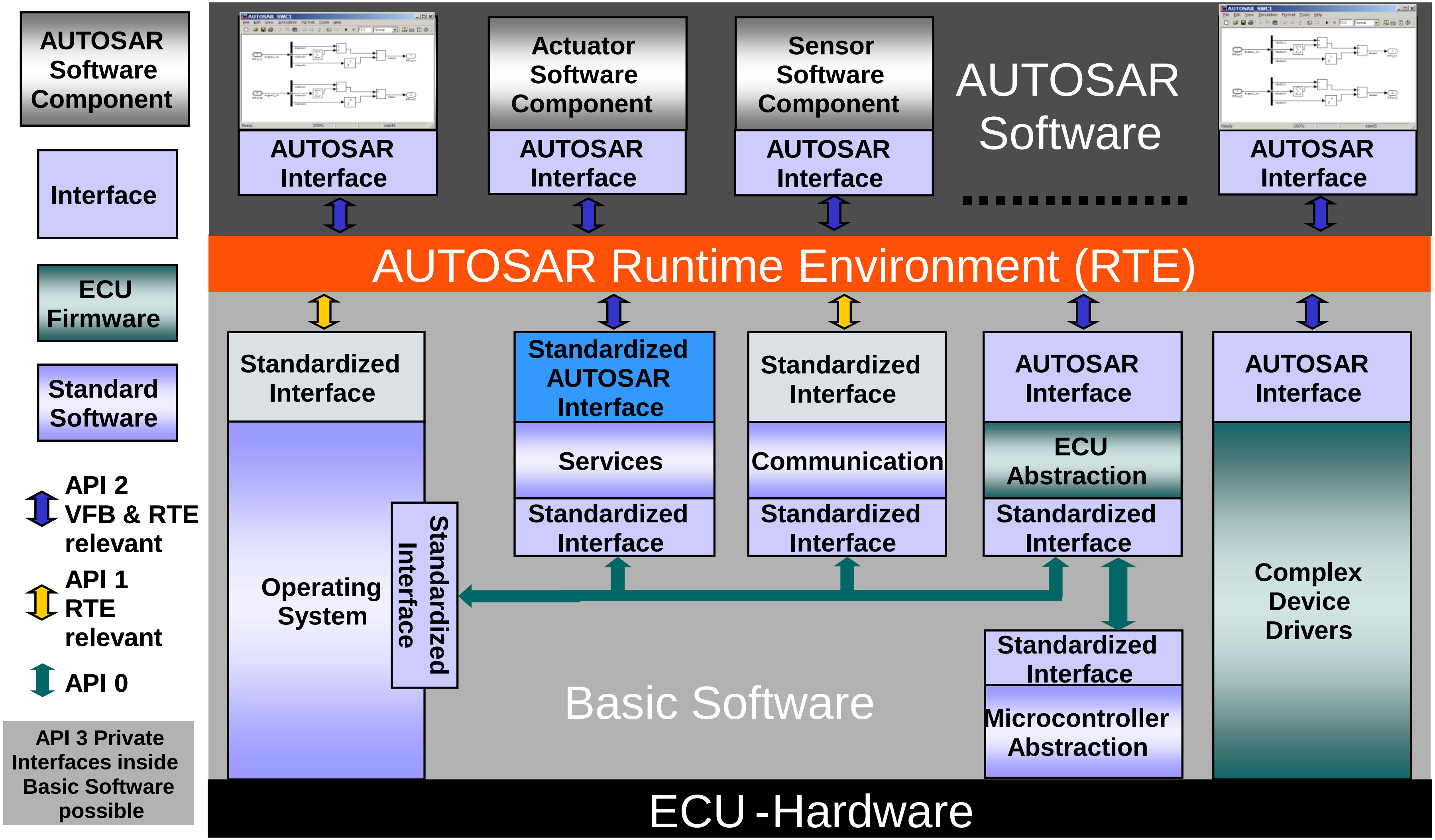


Example 2: Model-Level Integration with AUTOSAR (Automotive Open System Architecture)

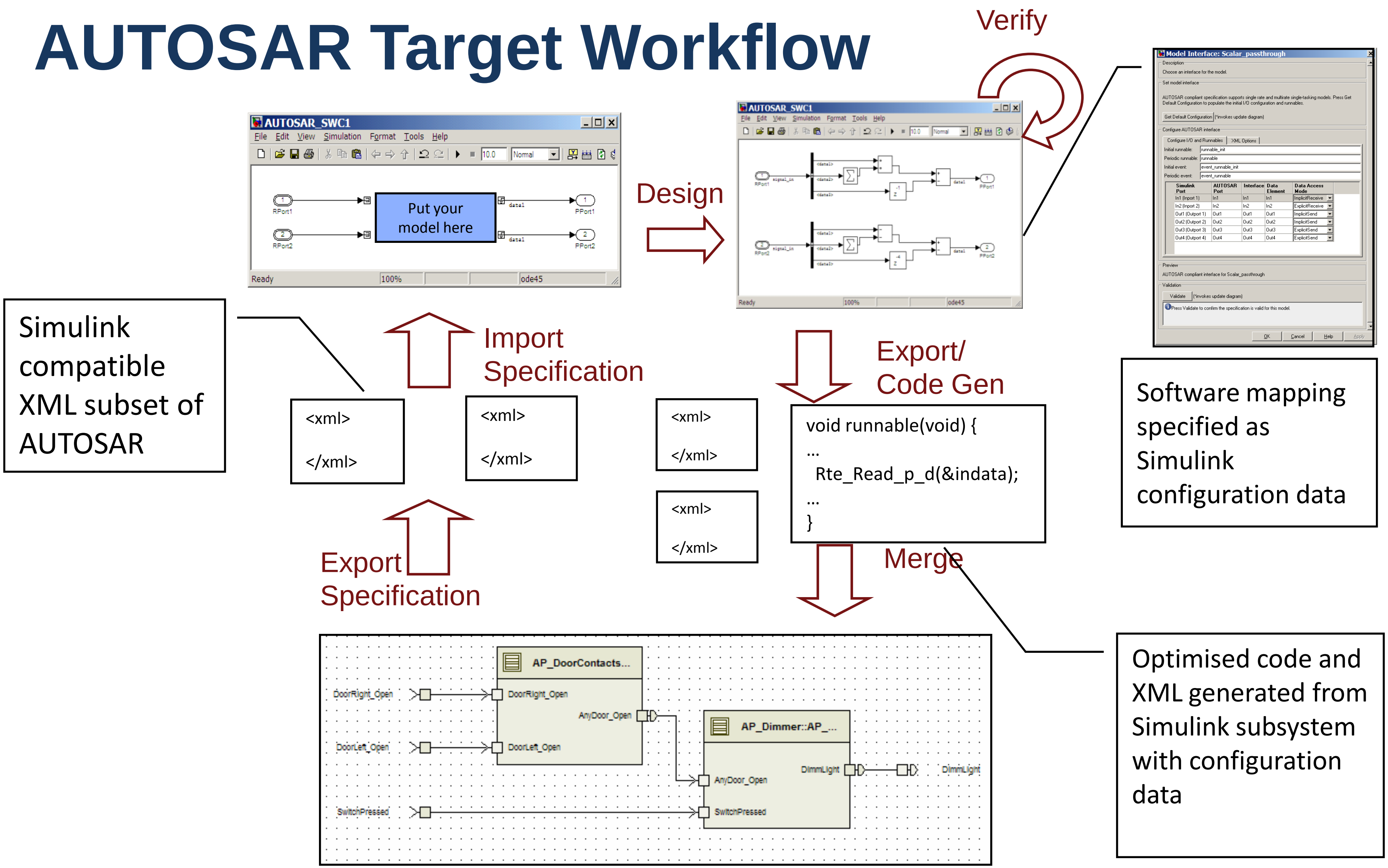
- AUTOSAR Goals
  - Implement and standardize on a single platform as an OEM-wide "Standard Core" solution
  - Enable OEMs/suppliers to focus on added value
- AUTOSAR Key Technologies*
  - Basic Software
    - ➢ Software architecture including a complete basic (environmental) software stack for an ECU as an integration platform for hardware-independent software applications
  - Methods of Software Integration
    - ➢ Exchange formats (templates) to enable a seamless configuration process of the basic software stack and the integration of application software in ECUs
  - Functional API
    - ➢ Specification of functional interfaces as a standard for application software modules

*Source: Helmut Fennel, OOP 2007

AUTOSAR Architecture

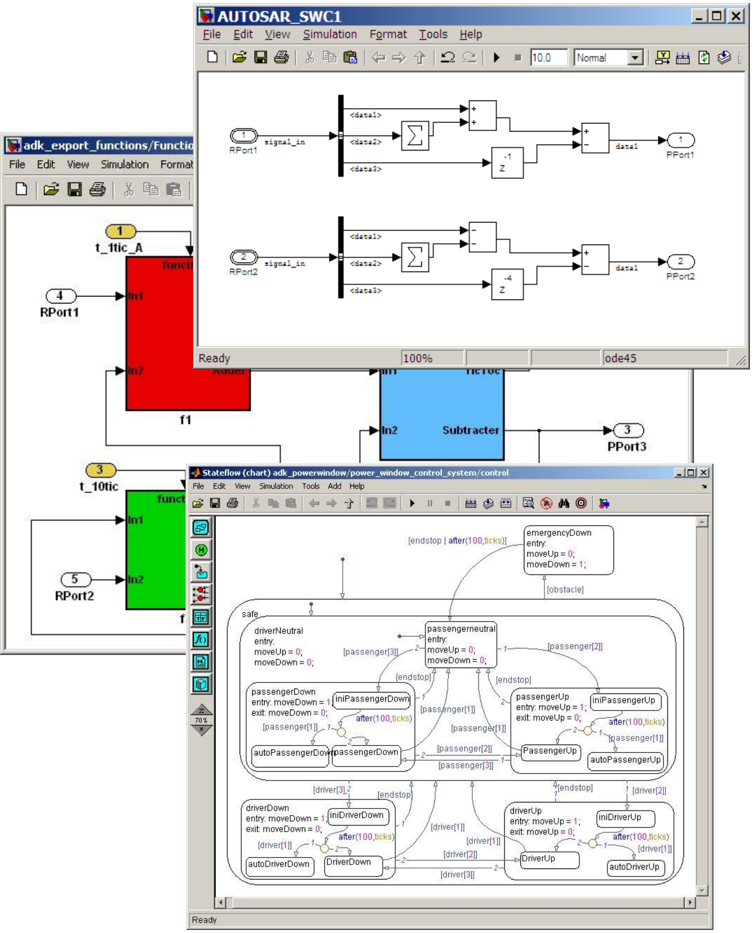


AUTOSAR Target Workflow

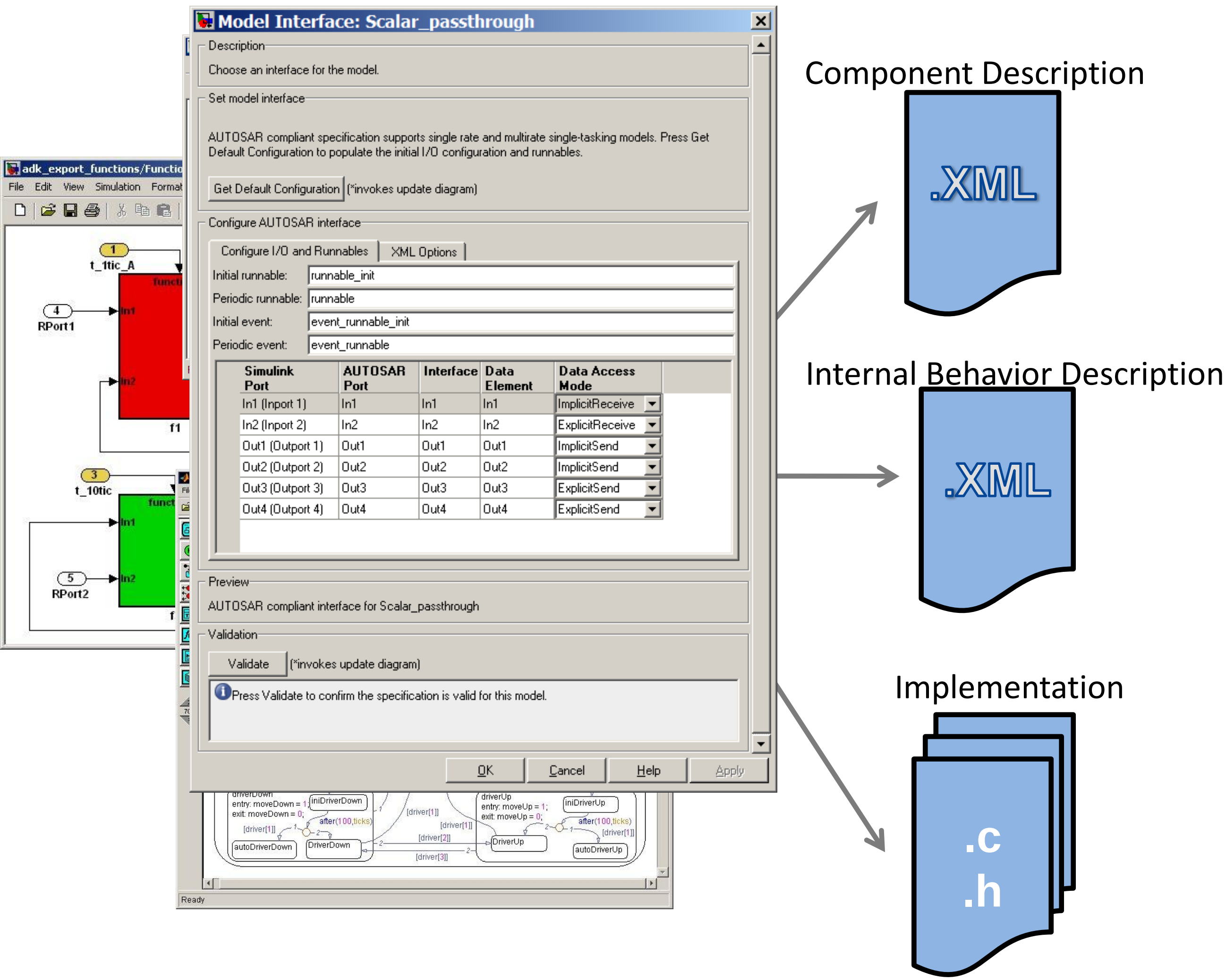


AUTOSAR System Authoring Tool

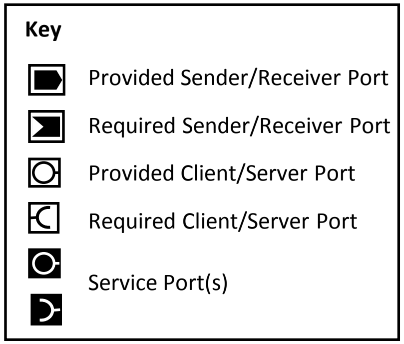
Generated Software/Description

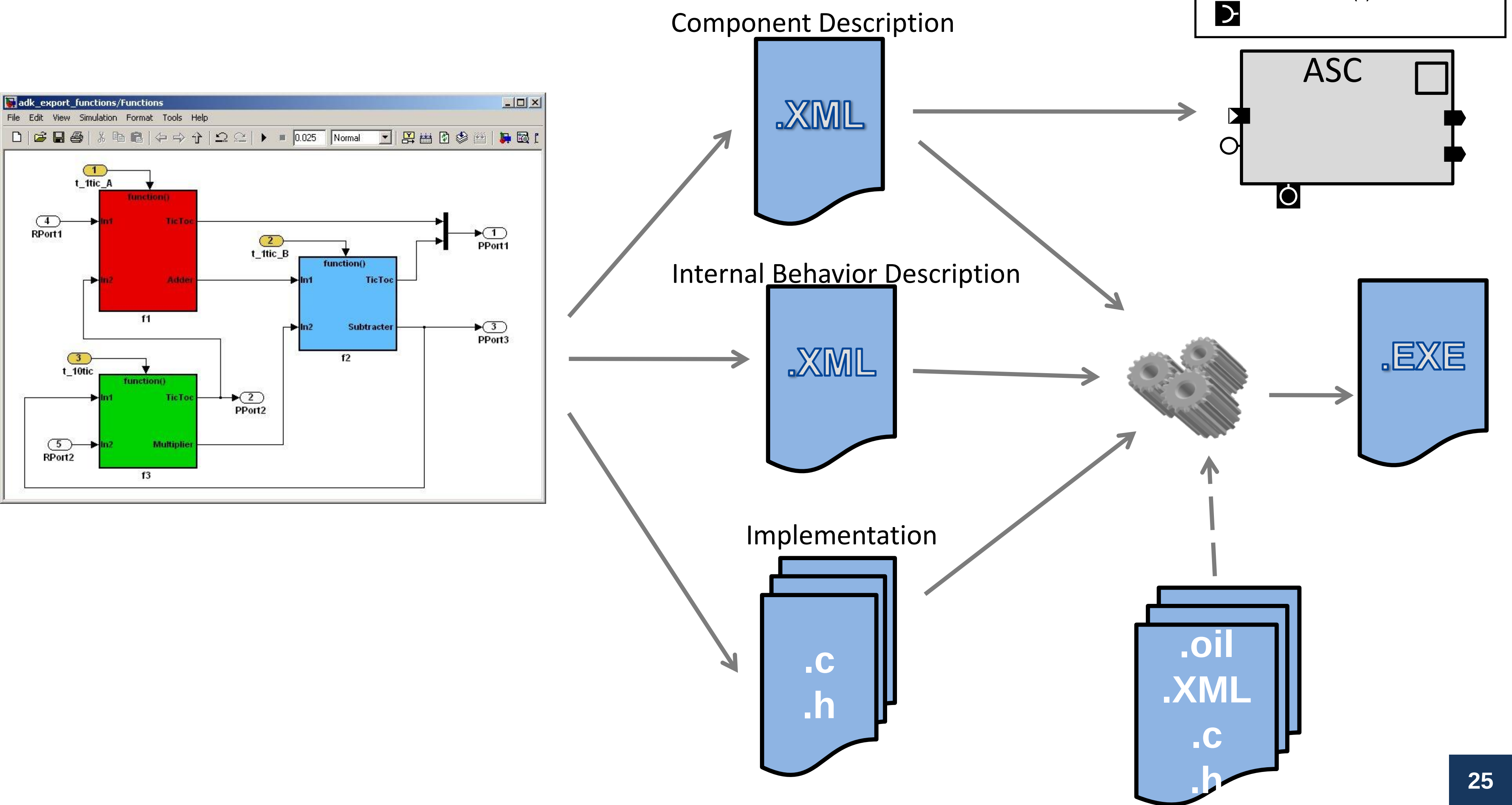


Generated Software/Description



Generated Software/Description





Conclusion

- Simulink is the established environment for algorithm development
  - Rich design and verification environment
  - Route to production code
- Software architectures are becoming more complex
  - A number of languages are emerging for the description of software architectures
  - Need to present algorithmic models as components for integration
  - Three main activities
    - Characterize an algorithmic model for optimal use in software
    - Adapt an algorithmic model to the demands of the application
    - Publish the adapted model for use in a given platform
- Challenges in supporting collaboration between software engineer and algorithm designer
  - Providing a framework within Simulink that allows the software engineer to prepare an algorithm for deployment into a software architecture
  - Maintaining a consistent representation of the algorithm in both environments
  - Generating the required artefacts to support integration with the software engineer's design flow