EDGEWATER
COMPUTER SYSTEMS, INC

# RTEdge™ Platform Overview

Serban Gheorghe

Vice-President, Software Technology
Edgewater Computer Systems Inc.
serban.gheorghe@edgewater.ca

# Outline

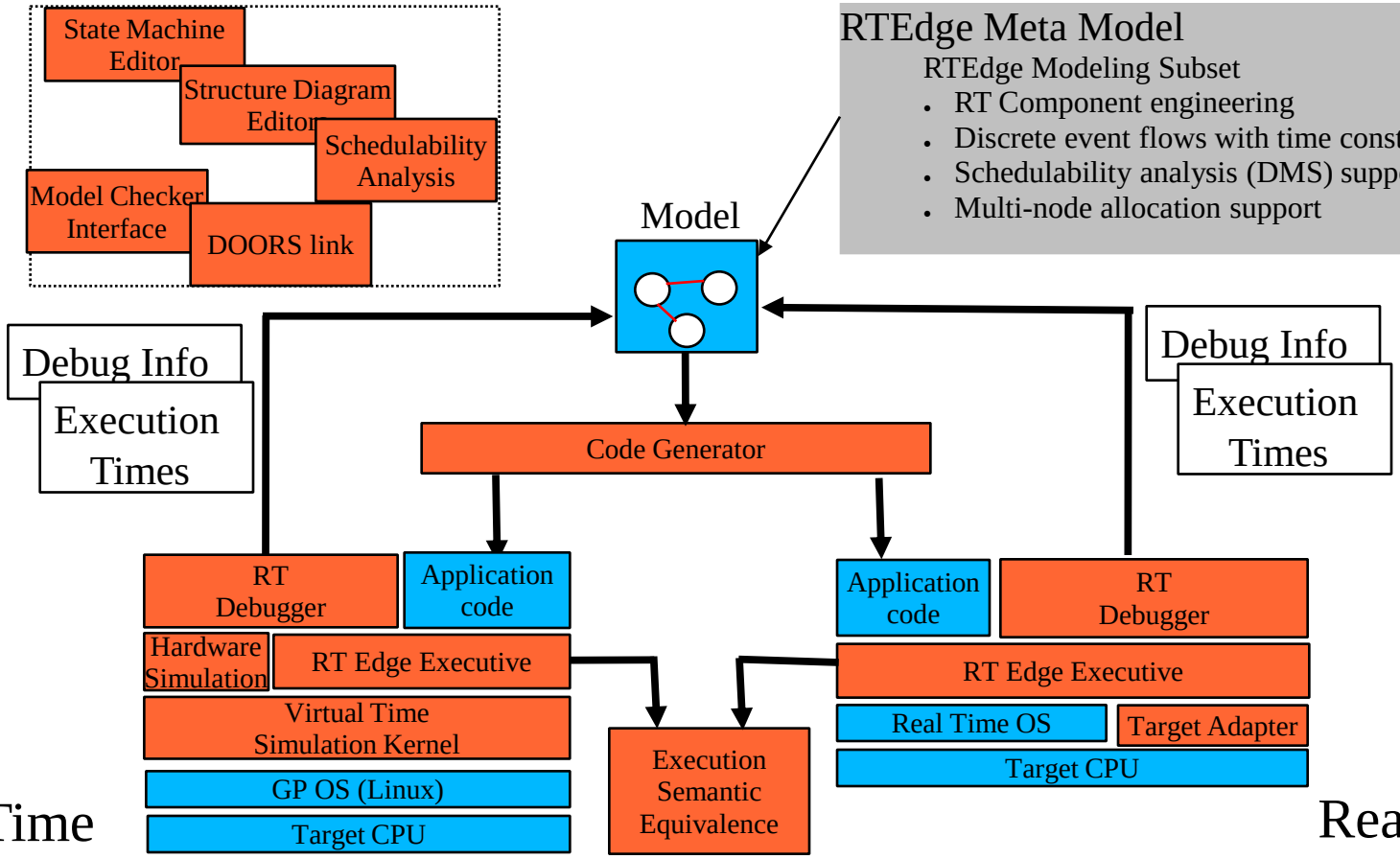
- What is RTEdge
- RTEdge domain focus:
  - The Critical Real-Time Problem Space
- RTEdge approach to proof based engineering
- Core Concepts of the RTEdge Modeling Subset
  - AADL, UML2 and MARTE roots

# What is RTEdge

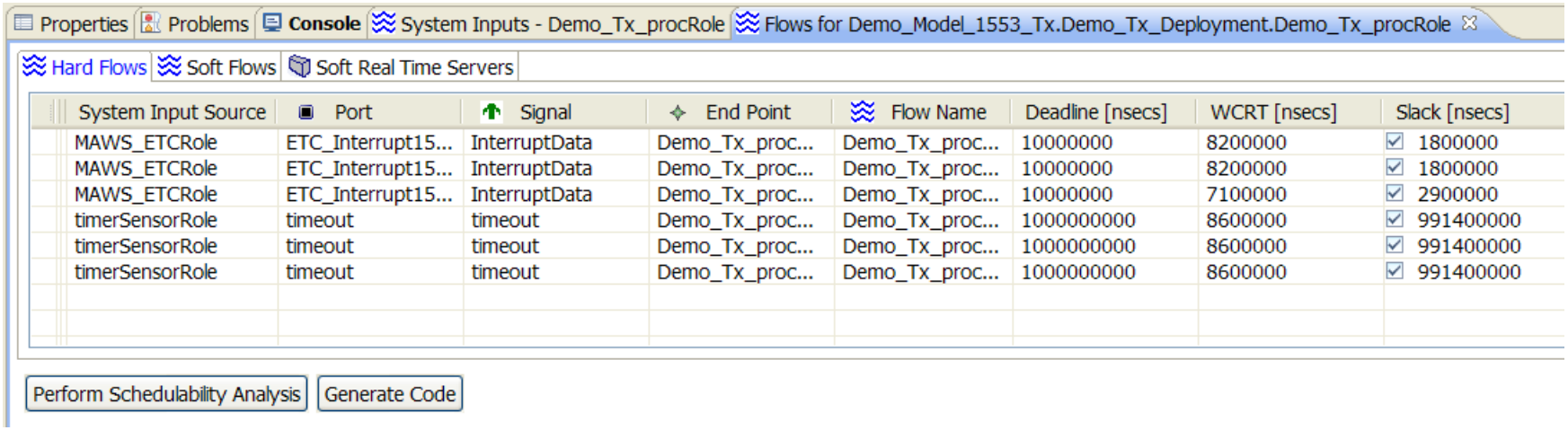
- A development platform for a sub-class of real-time embedded systems
  - Discrete event, reactive or command/control systems
  - Mix of hard real-time constraints, soft real-time constraints and best effort
    - Degrees of proof for timing constraints range from high assurance to no assurance
  - Distributed over multiple processors linked with communication channels
    - Multiple connected hardware resources, multiple connected schedulability domains
    - End-to-end deadline guarantees over multiple computing resources and communication resources
  - Safety critical or mission critical concerns
    - Requires high assurance correctness proof for a key subset of behaviors
- RTEdge™ is Edgewater's tools framework for proof based engineering of *real-time distributed embedded* safety/mission critical applications
- Release 1.1 Limited Availability is currently deployed in technology trials
- Release 1.2 General Availability is in final testing

# RTEdge™ Platform Components

- Model Driven Real-Time System Development
- Eclipse based open platform

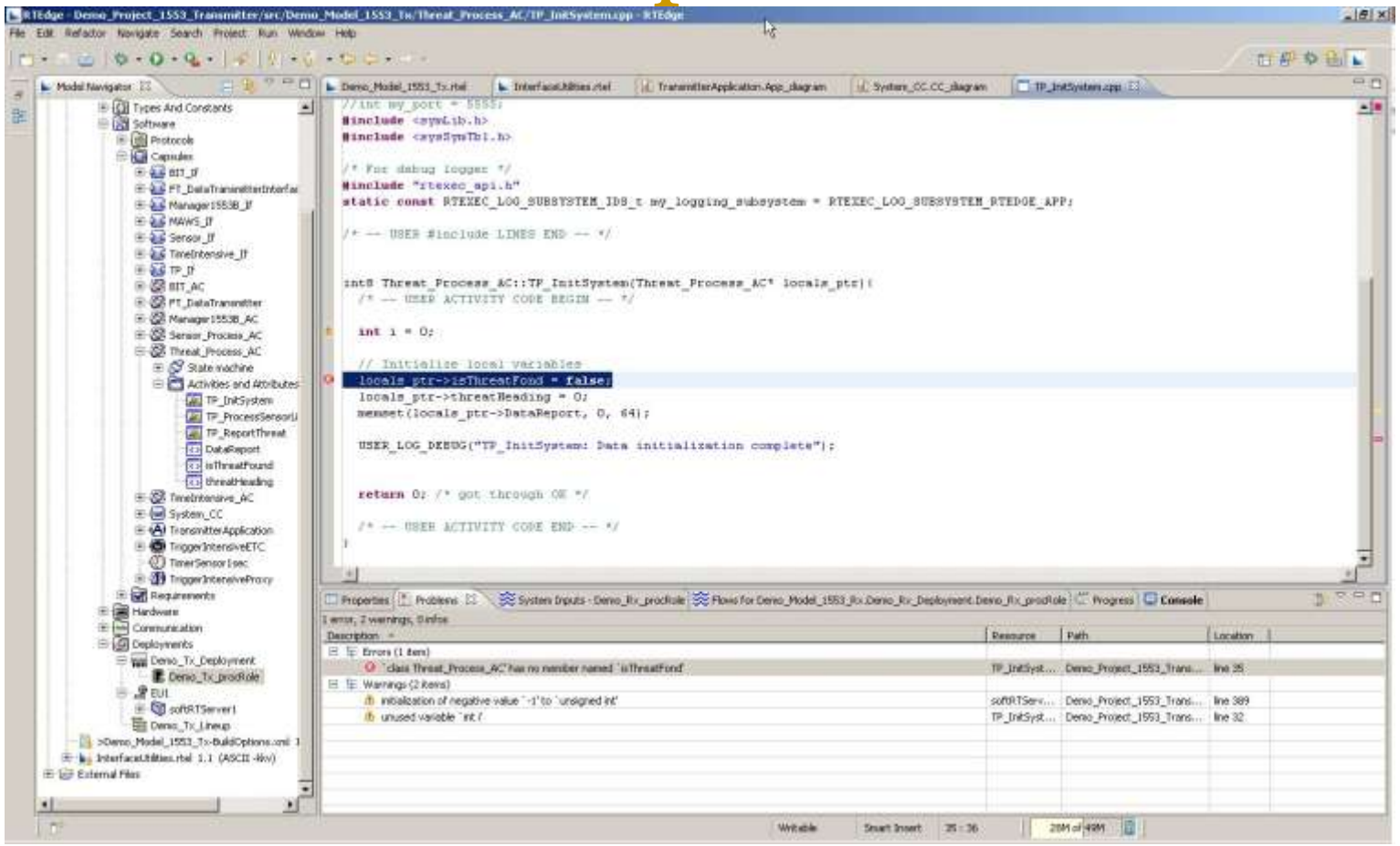


# Flow and Deadline Analysis



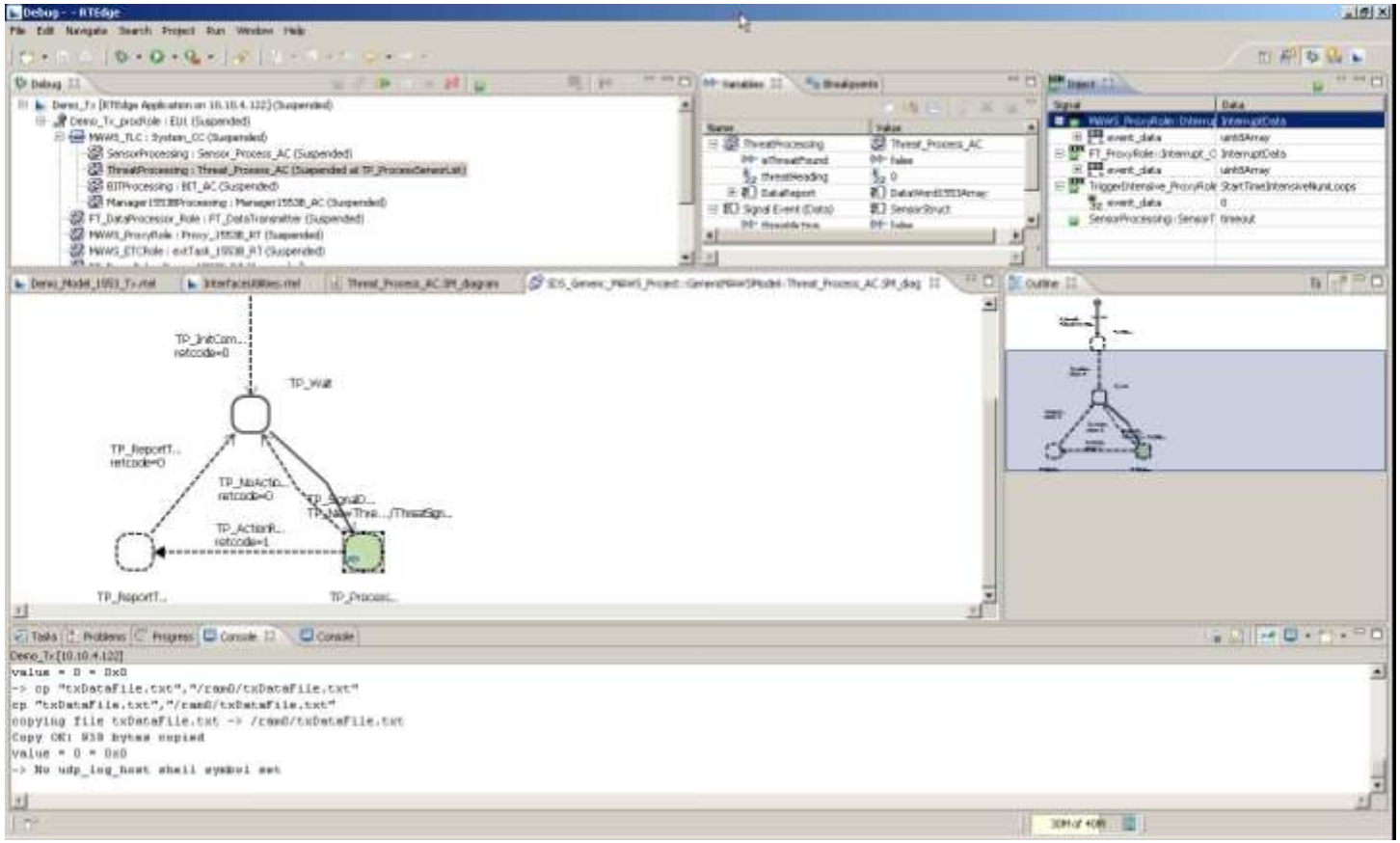

- Model analyzer computes all event flows through model and computes worst case response times ensuring deadlines are met
  - Priorities assigned based on Deadline Monotonic Analysis
  - Ceiling Priority Protocol for shared resource access
  - Global Worst Case Response time analysis

# Build Perspective




- Automatic code generation and compiler ensures code matches the model

# Debug Perspective



- Integrated debugger exposes runtime application and reduces test time

# RTEdge tools framework design goals

- Temporal correctness - static proof via mathematical analysis
- Functional correctness – degrees of formal specification and static compliance analysis
- Achieve **substantially higher** Processor and Bandwidth average **utilization** than current synchronous design approaches
  - Preemptive Priority processor scheduling policy
  - Priority queues based access to communication channels
- Support for Edgewater E1553™ data bus, future support for other communication technologies
- Component centric, processor independent approach, results in portable applications
  - Systematic separation of mandated temporal constraints from constraints introduced by allocation to target platforms
- Core requirements for the RTEdge platform driven and validated by the USAF Aeronautical Systems Centre

# Outline

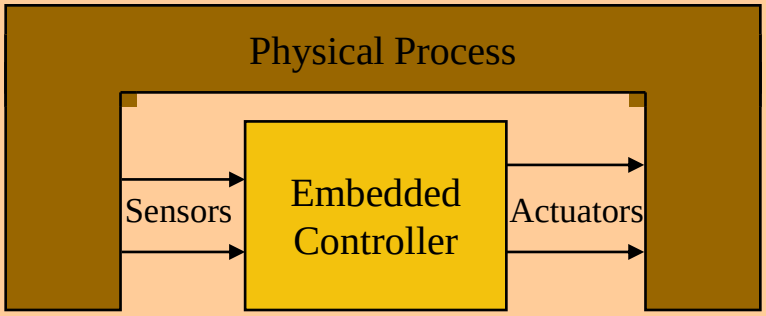
- What is RTEdge
- RTEdge domain focus:
  - The Critical Real-Time Problem Space
- RTEdge approach to proof based engineering
- Core Concepts of the RTEdge Modeling Subset
  - AADL, UML2 roots and relationship to MARTE

# The Evolution of the Real Time Problem Space

1978

…from control loops…

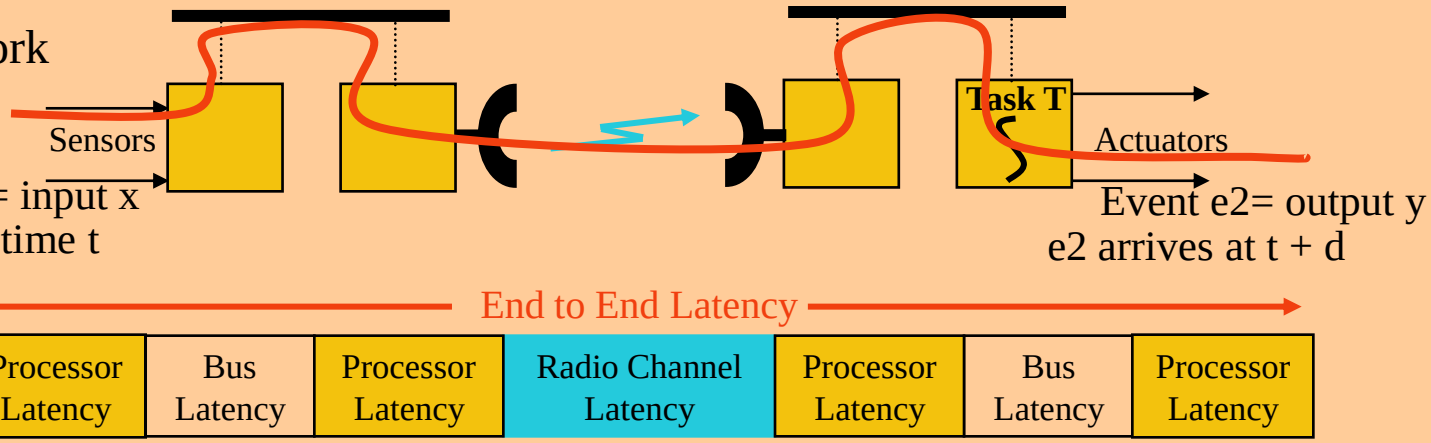
Real Time Embedded Controller




2008

…to real-time network applications…

Distributed Real Time Applications



# Current design paradigm limitations

1. Critical Real-Time Avionics Systems design is currently based on *total temporal determinism*
   - Main problems:
     - Modern mission applications are Event Driven, unlike traditional Time Driven control loops
     - ***Time division resource allocation*** IS NOT WORK CONSERVING
       - leads to ***low average utilization*** of processing capacity and channel bandwidth
     - Systems do not scale up
2. Barriers to proving temporal and computational behavior with high assurance stem from
   - Informal Specification of required behavior
   - The use of multiple languages and formalisms for design and implementation

## Pushing through the limits:

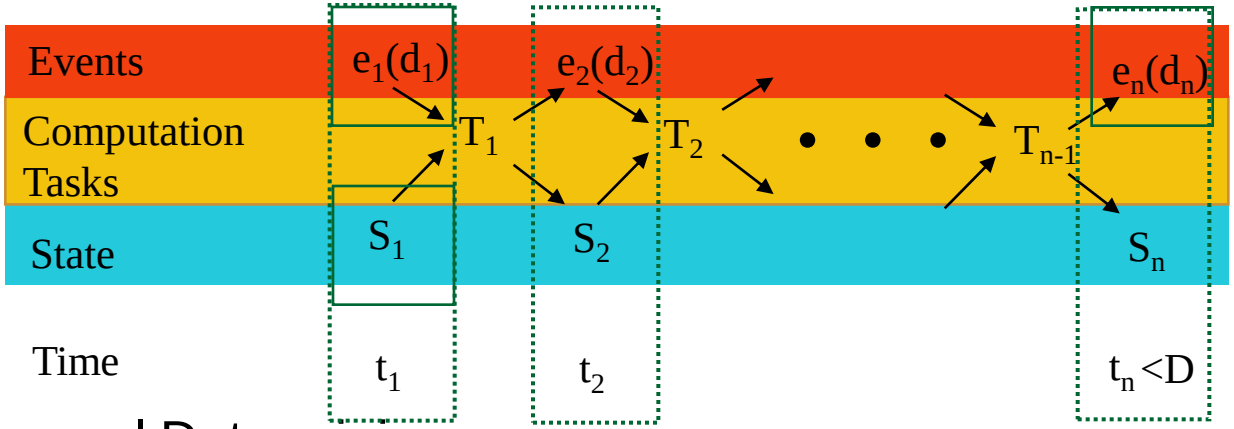
- Evolution to Critical Real-Time System design based on **priority based scheduling** with **mathematically guaranteed deadlines**
  - See 2005 FAA Schedulability Study (document **DOT/FAA/AR-05/27**)
- Evolution to a **Component** centric design practice based on formal **Specification Contracts**
  - Enable high assurance, formal design time proof of meeting certain functional and non-functional properties deemed to have a safety impact

## Other projects in the modeling space with similar goals:

- UML Omega Profile
- COTRE – TOPCASED
- ASSERT (European Space Agency)

# Determinism – foundation of Safety Critical Systems

- Deterministic Computation
  - $e_1(d_1)$ an **input** event $e_1$carrying data $d_1$
  - presented when the system is in the **global state** $S_1$
  - will result always in the same **output** event $e_n(d_n)$
  - and the same sequence of intermediary events, computation steps and states (i.e. **behavior**)




- Temporal Determinism
  - Total
    - all events at pre-defined times
  - Selective
    - Distinguished events constrained in time

The essence of Safety Critical Systems design is to SPECIFY without ambiguity and PROVE with high assurance Computational and Temporal behavior

EDGEWATER COMPUTER SYSTEMS, INC

# Outline

- What is RTEdge
- RTEdge domain focus:
  - The Critical Real-Time Problem Space
- RTEdge approach to proof based engineering
- Core Concepts of the RTEdge Modeling Subset
  - AADL, UML2 roots and relationship to MARTE

# Programming Languages "safety subset" approach

- A strategy to improve the Functional and Temporal determinism of programs written in General Purpose Languages (GPLs)
  - Choose a "safety" subset of the language
    - Spark83, Spark95, Ravenscar (ADA)
    - Misra C, Misra C++
    - JSF/C++
  - Build Static Analysis tools to enforce subset rules and prove properties
    - Spark Examiner, LDRA, etc…

- The "safety subset" strategy pioneered by Spark ADA is very much relevant in a MDD context:
  1. Constrain the state space complexity upfront, at design time
     - Assertions, use of simple types, range types, behavior expressed as FSM, constrained inputs
  2. Eliminate possibility of expanding or changing the state space at run time
     - Eliminate dynamic instantiation, support "mode changes" for controlled state space modification
  3. Eliminate language constructs that could impact the validity of static state space exploration
     - Eliminate implementation dependent, ambiguous or non-deterministic constructs, aliasing, pointers, side effects etc..
  4. Introduce **Component Interfaces** annotated with **Specification Contracts**
     - Component Interface types are separate from Component Implementation types
     - Techniques for checking Implementations compliance to the Component Interface Specification are provided
     - Defined composition rules for Composite Structured Classifiers, re-using Specification contracts of the Parts
  5. Define the formal syntax and semantics of the subset

# The RTEdge MDD "Correct by Construction" Approach

- ***Minimal subset*** of modeling constructs with precise execution semantics based on AADL and UML2 => RTEdge Modeling Subset
  - Defined such that is Static Analyzable
  - Includes RT Component Interface Specification Contracts

- Correct by Constructions means
  - Predict by Static Analysis at design time
    - Static Analysis performed
      - incrementally while building the model or
      - Global on demand or before execution
    - No modeling constructs that can result in unpredictable run time behavior – Computational determinism
    - Processor and Communication Channels Schedulability Analtsis - Temporal determinism
  - Verify and Enforce design assumptions at execution time

- Same modeling subset used consistently for the whole development cycle to produce
  - Specification models with increasing degrees of formality
  - High Level Design models with simulation capabilities
  - Integration, Build and Execution platform for Implementation Models
  - From simple Real Time Components to a System of Systems

The Model is the Specification
The Model is the Design
The Model is the Implementation

EDGEWATER COMPUTER SYSTEMS, INC

# Outline

- What is RTEdge
- RTEdge domain focus:
  - The Critical Real-Time Problem Space
- RTEdge approach to proof based engineering
- Core Concepts of the RTEdge Modeling Subset
  - AADL, UML2 roots and relationship to MARTE

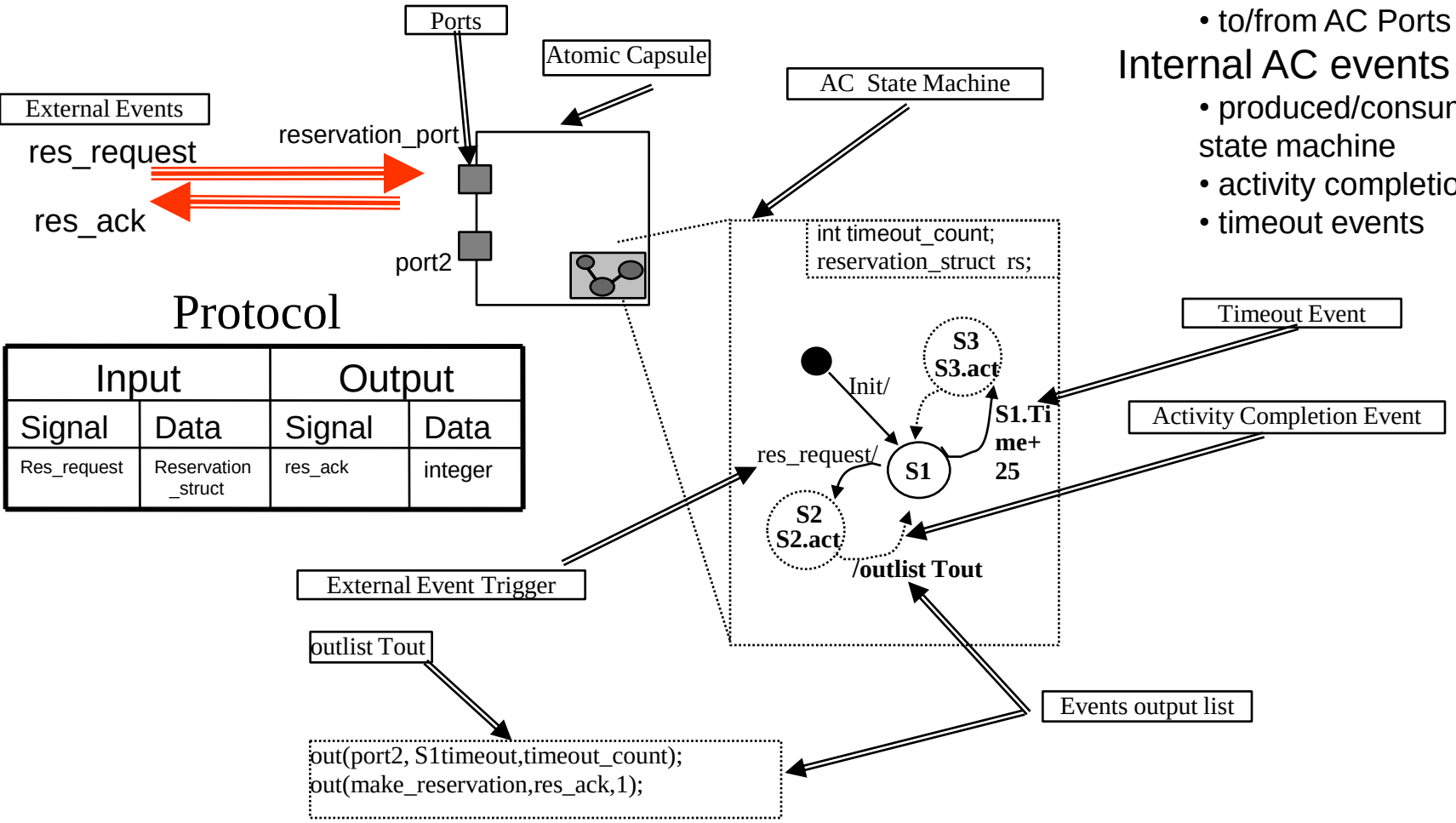
# RTEdge™ Modeling Subset Core Concepts

1. RT Component centric modeling
   - Atomic units of concurrency with State Machine behavior
2. RT Components composition
   - Structured Classifiers
3. Component Interface and Component Implementation type hierarchies
   - The “implements” relationship
   - Component Interface Inheritance
   - Component Implementation Inheritance
   - Component Interface Specification Contracts
4. Unified Processor and Channel Resource Scheduling
   - Discrete event Flows
   - Capsule Interface Flow specification
   - Capsule Implementation Flows calculation
   - Schedulability Analysis

# RTEdge basic component

- ***Atomic Capsule (AC)* – the basic Active component**
  - corresponds to an AADL thread
    - with behavior described by FSMs
    - with variables declared from a set of elementary types (Misra C types)
    - with *run to completion* State actions expressed as a Misra C function
  - Corresponds exactly to a MARTE RtUnit with
    - isDynamic = False - all owned SchedulableResources needed for execution
- **Only asynchronous messages (Signals) supported through AC Ports**
  - Ports described by groupings of Signals called *Protocols*
    - Corresponds to MARTE <<SignalSpecification>> interfaces
    - … and to AADL bidirectional event data ports
    - An optional Protocol State Machine can be associated
- **FSMs are a restricted subset of UML2 state machines**
  - No concurrent (orthogonal) regions, no history states
  - Clear separation of states and trigger events into
    - **Stable States** – synchronization points, no state action, accepting only external signals as triggers
    - **Transient States** – have an entry state action, create an internal action completion event with an enum return code
- **Support operational mode changes, one FSM per mode, same variables**
- **Explicit data sharing between Atomic Capsules through Data Ports**
  - Provide/Require access to public AC data
- **Support Clock objects and broadcast clock ports**

# RTEdge Atomic Capsule




## Protocol

| Input | | Output | |
|---|---|---|---|
| Signal | Data | Signal | Data |
| Res_request | Reservation _struct | res_ack | integer |

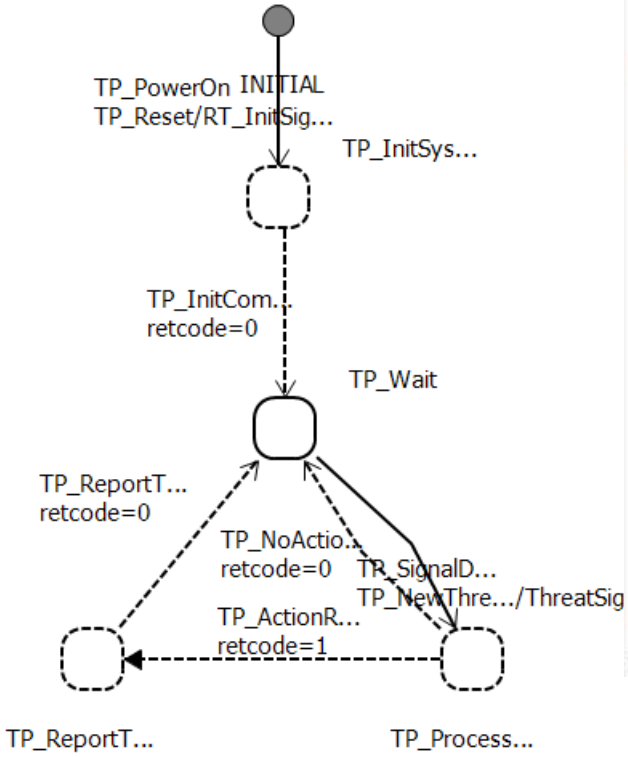
## External AC Events

- to/from AC Ports

## Internal AC events

- produced/consumed by the state machine
- activity completion events
- timeout events

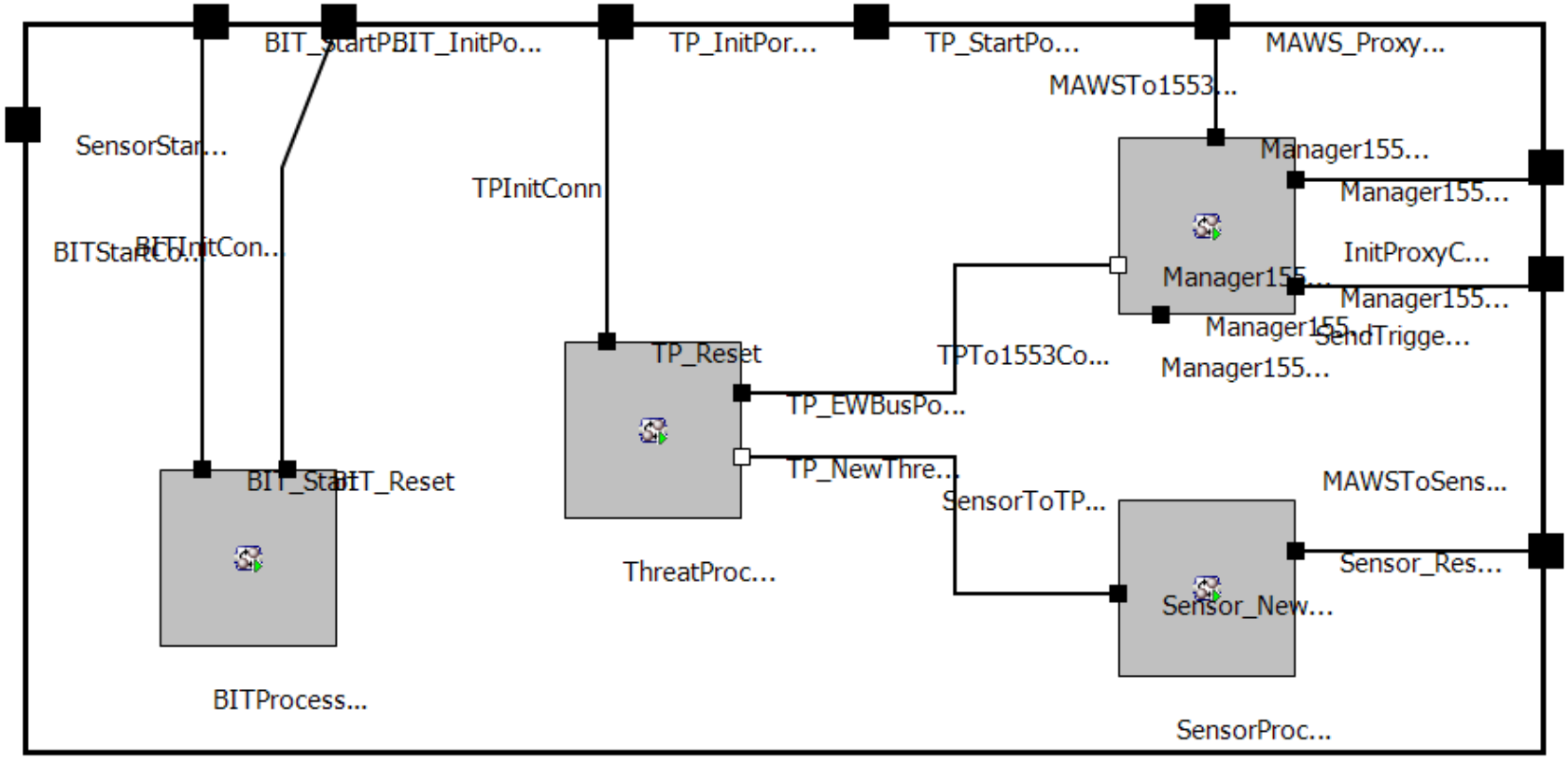
# Behaviour




```
int8 Threat_Process_AC::TP_ProcessSensorList(Threat_Process_AC* locals_ptr, SensorStruct event_data){
  /* -- USER ACTIVITY CODE BEGIN -- */

  /* Compare new value with old - if unchanged, do nothing otherwise action a report */
  if (locals_ptr->isThreatFound == event_data.threatActive)
  {
    return (0);
  }
  else
  {
    locals_ptr->isThreatFound = event_data.threatActive;
    locals_ptr->threatHeading = event_data.threatHeading;

    if (locals_ptr->isThreatFound)
    {
      printf("TP_ProcessSensorList - THREAT FOUND at %3.1d\n", locals_ptr->threatHeading);
    }
    else
    {
      printf("TP_ProcessSensorList - No threat detected\n");
    }
    // Action a report, either threat found or threat lost
    locals_ptr->DataReport[0] = event_data.threatActive;
    locals_ptr->DataReport[1] = event_data.threatHeading;

    return (1);
  }

  return(0);
  /* -- USER ACTIVITY CODE END -- */
}
```

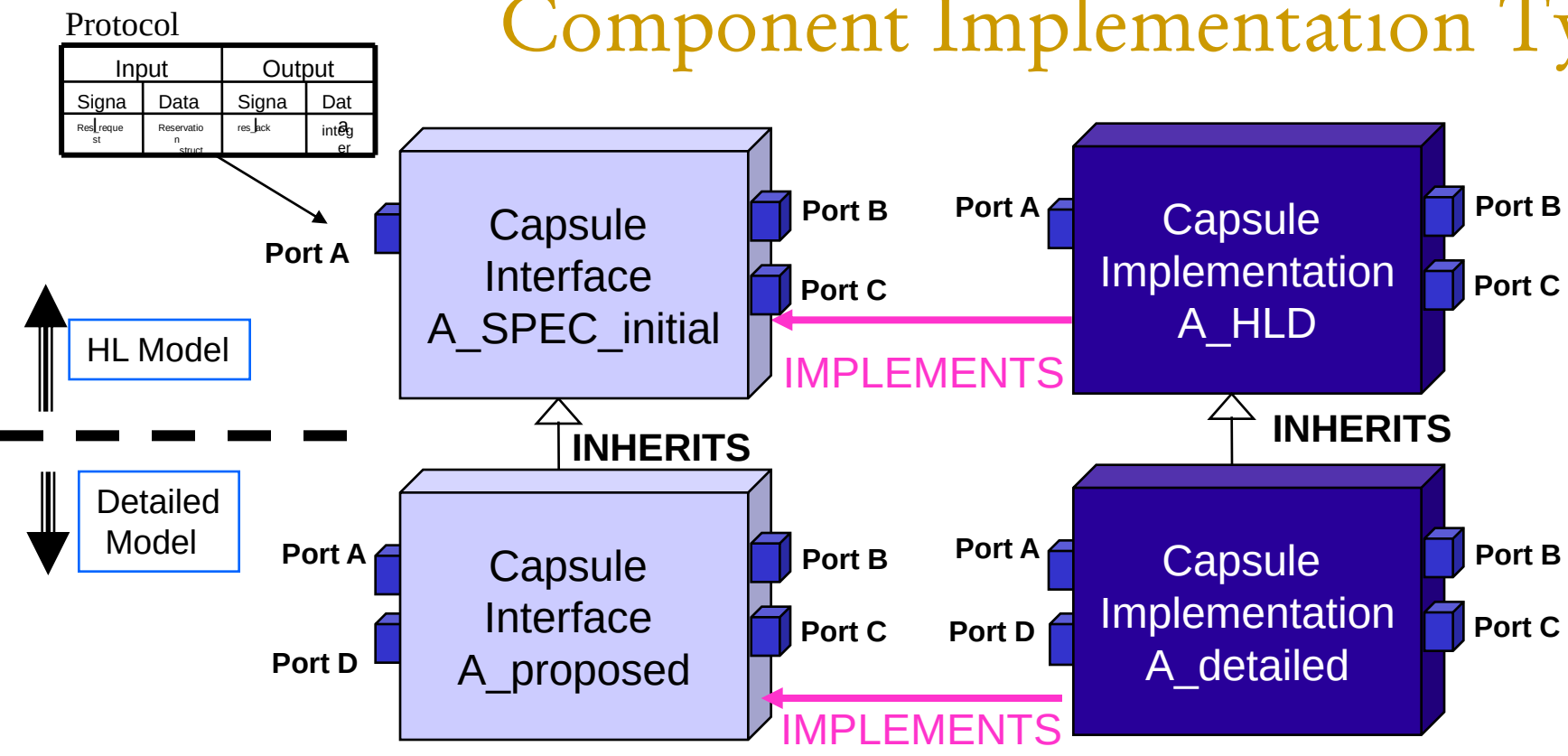
- Restricted state machines with embedded Misra C code for actions
- Syntactic access to all possible inter-AC interactions through messages or data sharing
- Can statically explore state space without crossing into the C language domain

# 2. RT Structured



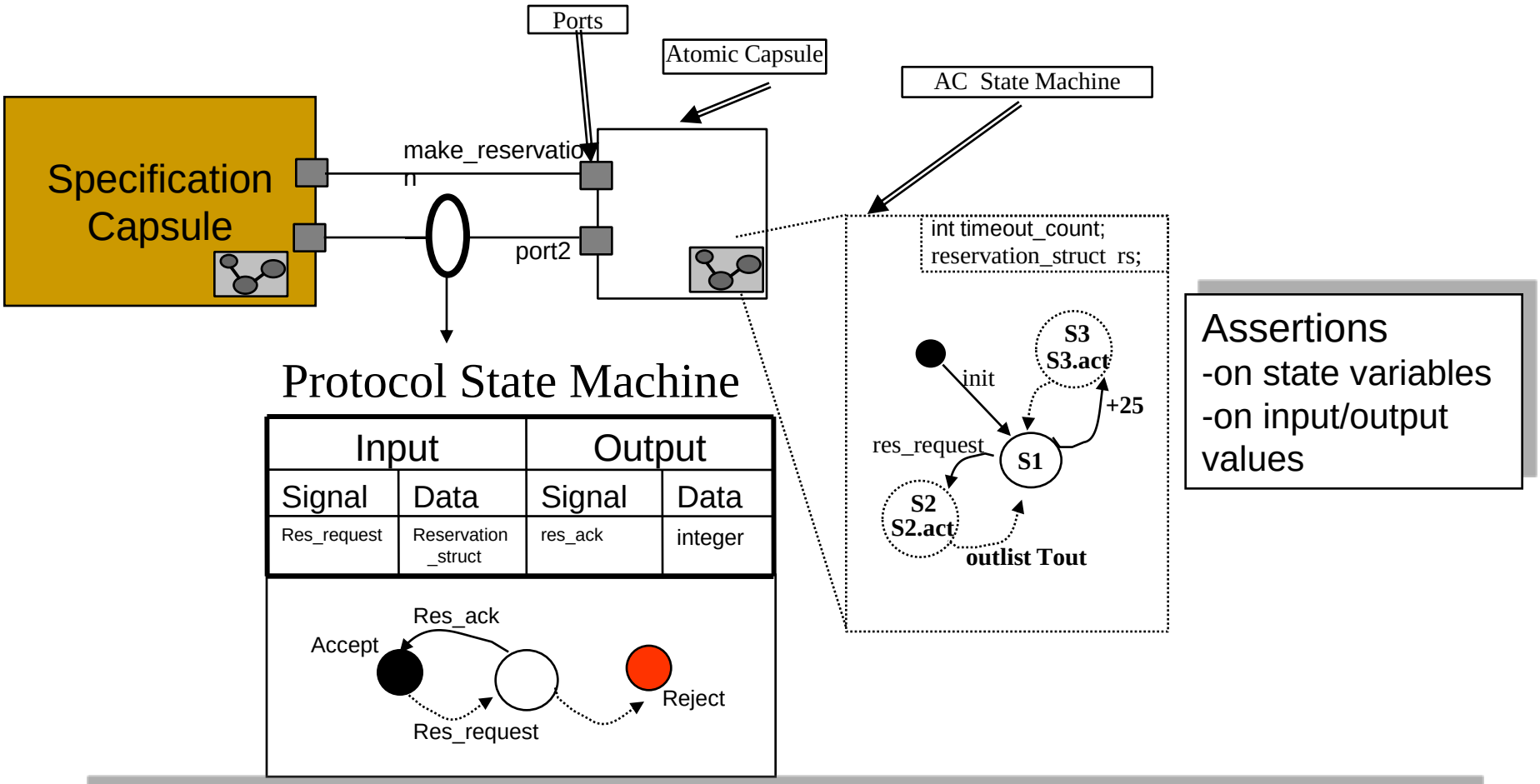

- RTEdge Composite Capsule
  - Contains Atomic Capsules or other Composite Capsule Parts
  - Connectors between compatible ports, behavior emergent from interactions
  - Corresponds to AADL thread group, UML2 Structured Classifier, MARTE Structured Component

# 3. Component Interface and Component Implementation Types




- RTEdge supports independent inheritance trees for Interfaces and Implementations
- Capsule Implementations must conform to Capsule Interface Contracts
- Capsule Interface Contracts can have increasing degree of formality
  - Flow declarations, Protocol State machines, Assertions

# Formal Capsule Specification Support




Assertions
-on state variables
-on input/output values

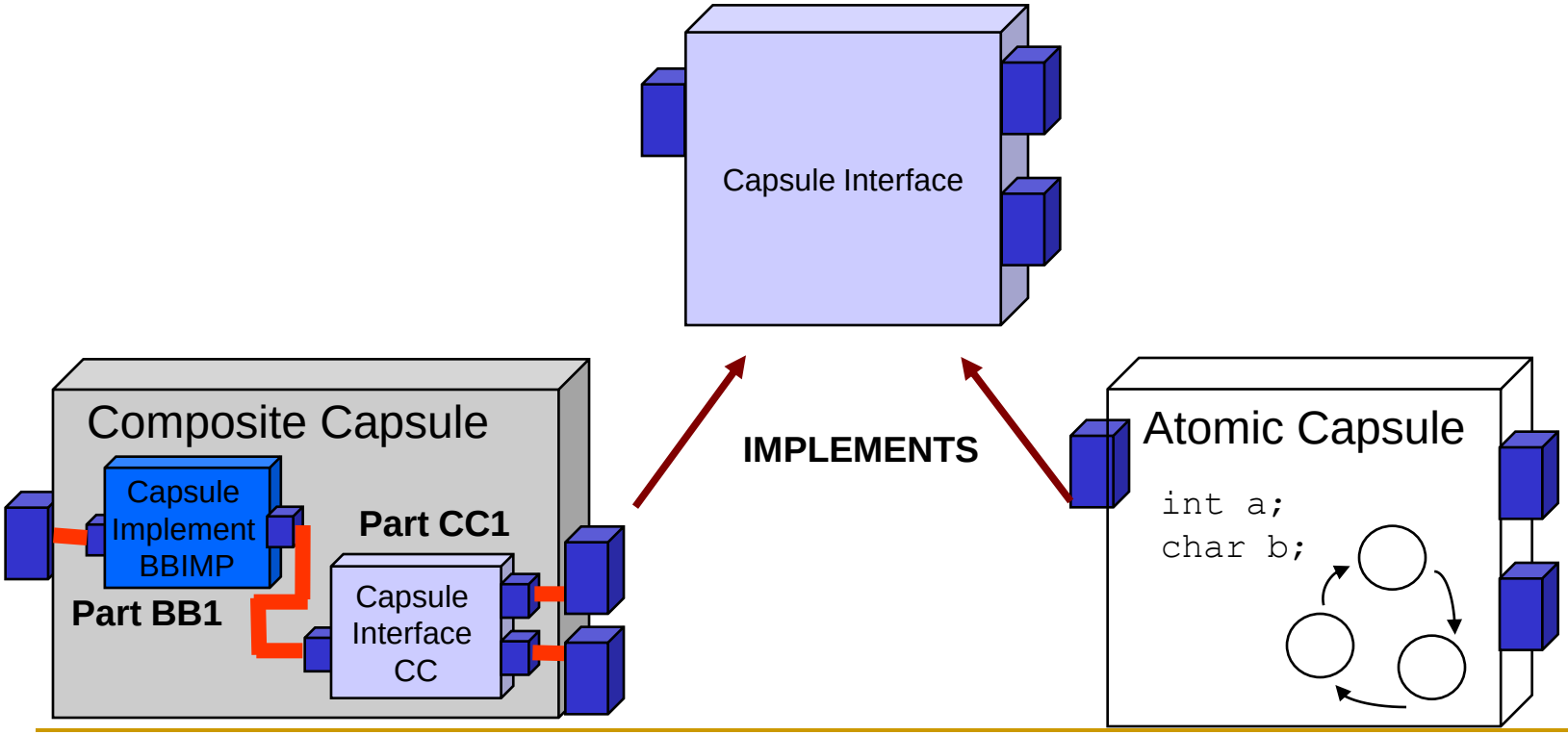
RTEdge support for Assertions, Protocol State Machines and Spec Capsules enables Automatic Test Generation and the link to formal Model Checkers

# Component Interface Specification Contracts

- Very high level of specification for RT Components interfaces
  - Functional Interface Contracts
    - Capsule Ports typed by Protocols (in-out signals, data)
    - Protocol State Machine (signal order on a port)
    - Data assertions (on signal data)
    - Specification Capsules (signal order between different ports, assertions on public data)
  - Temporal Interface Contracts
    - Flows – causal event relationship
      - intra capsule
      - Environmental
    - Flows with Deadlines

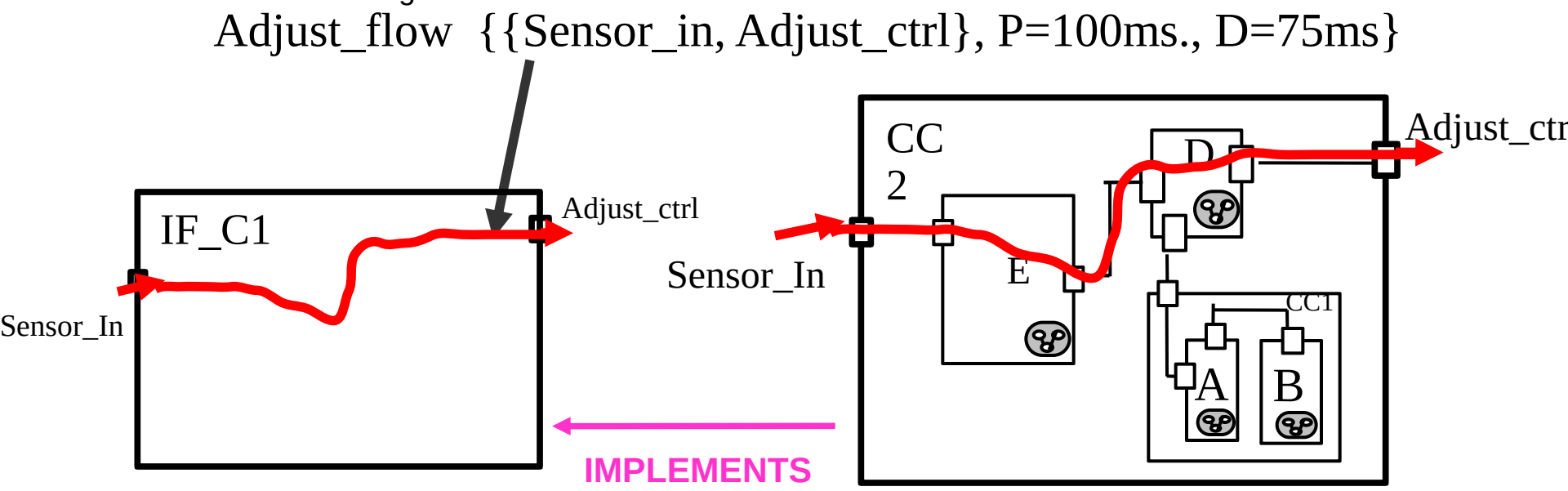
# Specifying Capsule Implementation Types

- **A further degree of refinement**
  - A **Capsule Implementation Type** is either specified as a **Composite Capsule Implementation Type** or an **Atomic Capsule**
  - **Composite Capsule Implementation Types** are used for specifying component structure
    - **Capsule Roles** – named references to Capsule Interface or Implementation Types
  - **Atomic Capsules** are used for defining behaviour



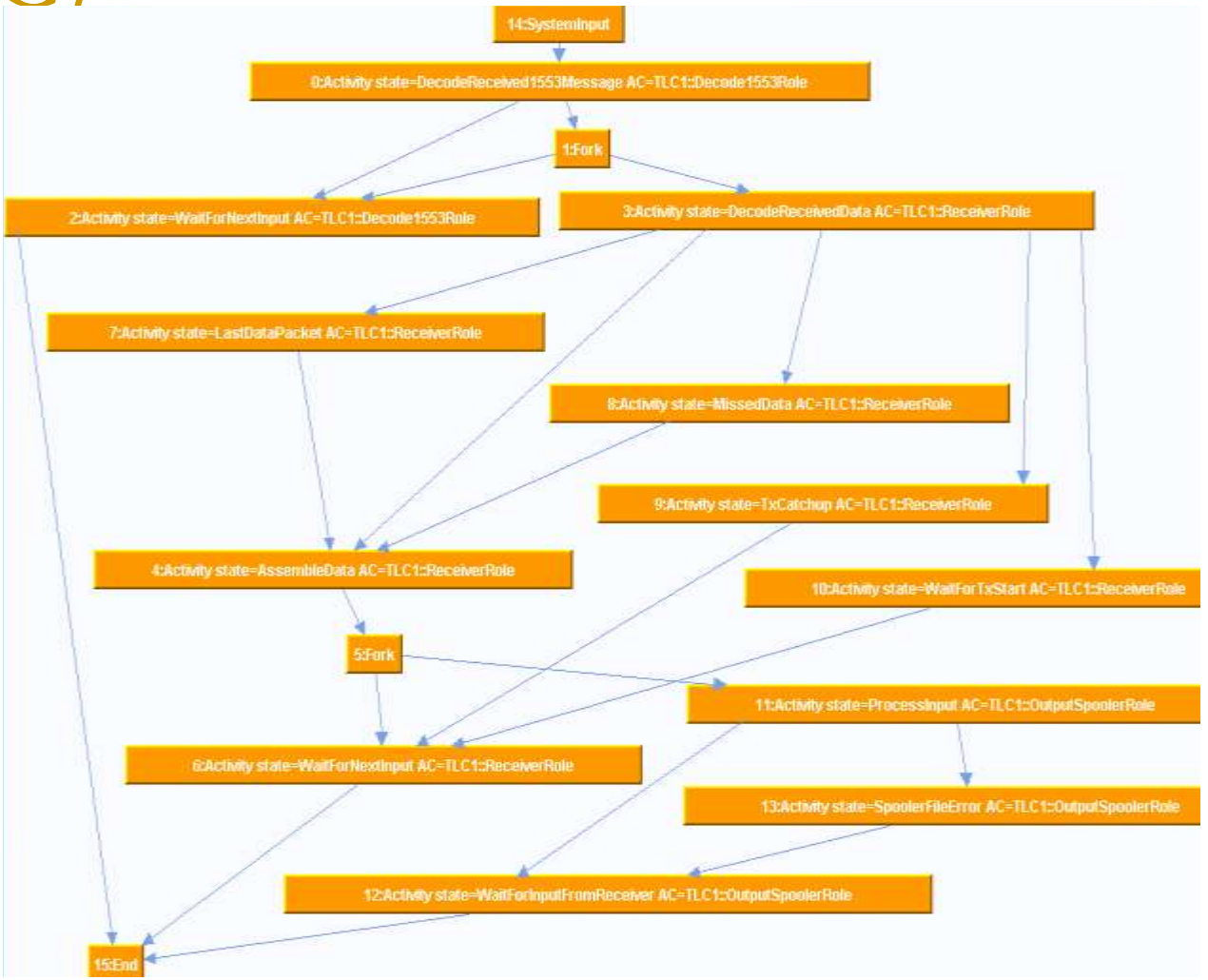
# 4. Component Interface and Implementation Flows

- **Specifies a causal relationship between an input event and an output or internal event**
  - A Flow uses processor and/or communications channel resources
  - Is the key RTEdge element for schedulability analysis
  - User specifies timing information for **Independent System Inputs**
    - period or burst signal specification (interarrival and integration period, optional jitter, phasing)
    - deadline for distinguished end of flow events
  - RTEdge automatically calculates flows and Worst-Case Response Times based on period, deadline and other flows contending for the same resources

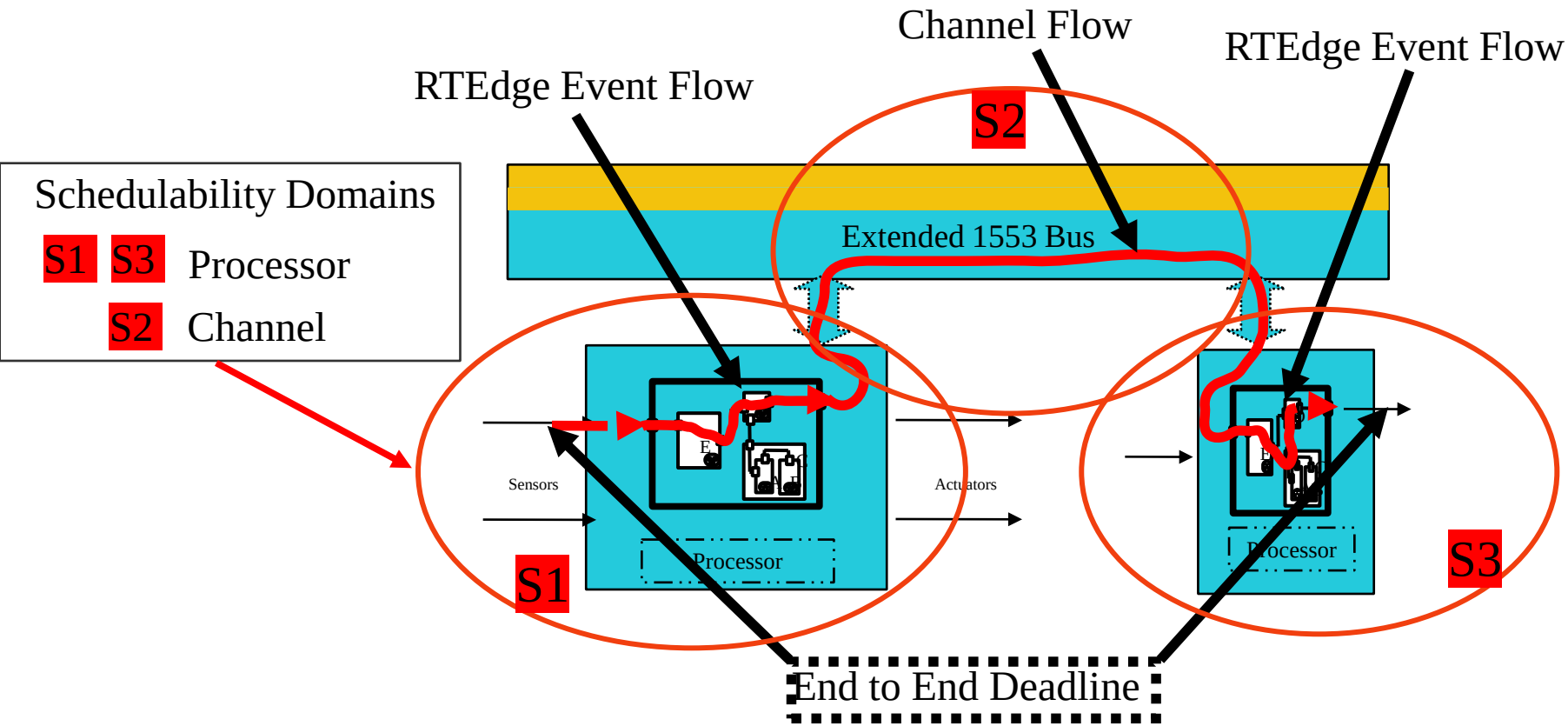


# Concurrent Control Flow Graphs (CCFG)

- Calculated Causality Graph rooted in a Independent System Input
- Used for Worst Case Response Analysis
- Can calculate a derived UML Activity Diagram



# End to End Flows and Schedulability Domains




- RTEdge calculates channels and processors schedulability based on Flow specifications, channel models and run time execution models
- Flows get their priorities re-mapped when they cross schedulability domains