Formal Analysis of Event Driven Systems Leveraging AADL

Serban Gheorghe
Edgewater Computer Systems Inc
serban.gheorghe@edgewater.ca
Outline

• Domain Specific Proof Based Model Driven Development
• RTEdge™ Technical Influences and Foundations
• An AADL Run Time Microkernel based on RTEdge™
• Semantic Extensions to the AADL concept base
  • Achieving Real-Time Predictability
  • Extensions to Flows and the use of Flows in WCRT Analysis
• Behavior specification and conformance mechanisms
  • Formal Specifications and Compositional Verification
  • Extending AADL Port Groups to Protocol Based Port Groups
• Formal Checking of RTEdge™ Applications
The Premises and Goals of Edgewater’s Work

Build a Model Driven Software Development Toolset to assist in the construction of theoretical correct real-time applications

Clear Domain Focus

- Hard real-time software systems
- Mission critical software systems
- Real-time systems distributed over communication networks

RTEdge™: a Proof Based MDD software toolset

- The RTEdge approach is a combination of
  - *domain specific Model Driven Development (MDD)*
    - distributed critical software systems with real-time deadlines
  - *proof based engineering*
    - Real-time software applications engineering backed up by mathematical proof
    - Support for proving functional correctness through static analysis model checking
**RTEdge™ technical influences**

**UML-RT → UML 2.0 (OMG, 2003)**
- Bi-directional Ports, Structured Classes, tighter semantics of State Machines

**OOTiA report (NASA & FAA - 2004)**
- The challenges of using OO design in safety critical software

**AADL spec (SAE AS5506, 2004)**
- An excellent set of answers to the above report

- Survey of industry approaches to scheduling and the verification of temporal correctness
- Conclusions:
  - Temporal correctness is signed off as being correct in ad hoc ways, *no formal proof*
  - Priority Driven Processor Scheduling policies preferable to Clock Driven and Processor Sharing policies
    - Deadline Monotonic Scheduling (DMS – used by RTEdge) recommended as optimal among static priority allocation algorithms

**The mathematics of fix priority scheduling for hard real time systems with task precedence**
- Many false starts
- RTEdge Worst Case Response Time analysis algorithm is based on work published by Lehoczky, Harbour, Klein

**Assume-Guarantee Compositional Verification**
Critical Real-Time Applications Concerns

Design Concerns:
- Specify Real-Time Constraints
- Prove meeting Real-Time Constraints

Guaranteed Timeliness
- Specify Real-Time Constraints
- Prove meeting Real-Time Constraints

Guaranteed Functional Behavior
- Specify required behavior
- Prove meeting required behavior

RTEdge™ Modeling Subset

Performance Degradation
- CPU and Channel Scheduling Models

Environment Hazards
- Formal Model Checking

Design concerns:
- Specify Required Behavior
- Prove meeting Required Behavior

Behavioral Determinism
- Missing inputs
- Unexpected inputs
- Sequence errors

Copyright Edgewater Computer System Inc.  May 18, 2010
RTEdge™ Modeling Subset Design Choices

1. Component based
   1. A subset of AADL component modeling concepts, augmented with a UML2 subset (Protocols, FSM subset)
   2. Explicit Assume-Guarantee environment/component contract - PROTOCOLS
   3. Component type specification – CAPSULE INTERFACE
   4. Multiple possible conformant implementations
      - ATOMIC CAPSULES (AC) with FSM behavior
      - COMPOSITE CAPSULES
   5. Specification of hierarchical component architectures
      - decomposition/aggregation of component structure

2. Explicit real-time constraints specification, proof and enforcement
   - Specification and enforcement of time arrival constraints on System Input events
   - Specification and proof of deadline constraints on System Flows

3. Minimalistic Execution Semantics
   1. Concurrent AC FSMs with asynchronously message passing (EVENTS/EVENT DATA)
      - Run to completion semantics, preemptive priority based event dispatch
   2. Static Priority assignment to EVENTS
      - Deadline Monotonic Scheduling algorithm based on specified real-time event pair constraints
   3. Explicit data sharing and Ceiling Priority protocol

4. Component refinement by feature and behavior extensions
   - Component types and component implementations have separate inheritance trees

5. Minimal set of modeling constructs
   - Expressive enough to build templates for well known and widely used communications and synchronization mechanisms
AADL run-time based on a RTEdge™ microkernel

AADL Model

RTEdge Modeling Language Subset

- RTEdge Code generated from AADL
- Support AADL Thread Dispatch Modes
  - Periodic, Sporadic, Aperiodic, Timed and Hybrid
- Code Generated Dispatch Policy State Machines
- Support AADL inter-thread communication/synchronization semantics

RTEdge Core Execution Semantics

- Asynchronous Message Passing Executive
- Discrete Events only, no continuous data
- Events are time constrained
- Threads with FSM behavior, purely reactive
- Atomic Capsules ⇔ AADL Threads, features subset:
  - Event/Event Data Ports
  - Required/Provides Data Ports
  - FSM behavior, a subset of UML SM
  - No Subprogram access support
- Composite Capsules ⇔ AADL Thread Groups
  - Resolved to communicating Atomic Capsules
- Periodic Timer Service

C Code Generation

Bounded Overheads Executive

• Use RTEdge Flows Analysis and WCRT Analysis to bound and establish AADL run-time overheads
• Use RTEdge Formal Checking Mechanisms

Copyright Edgewater Computer System Inc. May 18, 2010
## RTEdge™ modeling elements comparison

<table>
<thead>
<tr>
<th>UML 2.0</th>
<th>AADL</th>
<th>RTEdge™</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System, Process</strong></td>
<td><strong>System, Process</strong></td>
<td><strong>Deployment Types</strong></td>
</tr>
<tr>
<td>Processor, Bus</td>
<td>Processor, Bus</td>
<td></td>
</tr>
<tr>
<td>Bidirectional Interfaces</td>
<td>Port Group Feature Types</td>
<td>Protocols</td>
</tr>
<tr>
<td>Ports</td>
<td>Port Groups</td>
<td>Ports</td>
</tr>
<tr>
<td>Data Ports</td>
<td>Data Ports</td>
<td></td>
</tr>
</tbody>
</table>

Copyright Edgewater Computer System Inc.    May 18, 2010
**RTEdge™ Type Layering**

- **Roles** (AC, CC, CI Type References)
- **Conformant Atomic Capsule (AC) Implementation**
- **Composite Capsules (templates)**
- **Temporal Assumptions (Internal Protocol Group)**
- **Temporal Assumptions (Protocol Group)**
- **Assertion Annotations**
- **Temporal Assumptions (Binary Protocol)**
- **Implementation Types**
- **Built-in Types**

**Protocols**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Res_request</td>
<td>Reservation _struct</td>
</tr>
<tr>
<td>res_ack</td>
<td>uint16</td>
</tr>
</tbody>
</table>

**Specification Types**

- `uint1...uint32, int1...int32, enums, structs, arrays`

Copyright Edgewater Computer System Inc. May 18, 2010
Achieving Predictable Timeliness

Current Development Practice

- Hard Real-time systems are **Time Triggered** and rely on **Time Division** of processor and data links capacity
  - Only synchronous periodic events
- **Cyclic executives and cyclic schedules**
  - Every task is treated the same (promotion of all tasks to Hard Real-time tasks)
  - Every task is assumed to require its full processing power on every cycle (worst case assumption)
  - Every task is interconnected to every other (through timing)

**RTEdge™ Execution Semantics Choices**

- Asynchronous events constrained in time (periodic, sporadic, bursty)
- Clocks are treated as periodic events
- Processor/channel allocation mathematical models available at design time
- Hard, Soft and Best Effort tasks can share safely processors and channels
- **Priority** based processor allocation
  - Reduces time coupling between tasks
  - Results in much higher resource utilization.
- Proof of achieving required timeliness (deadlines) is obtained before execution
From Periodic to Time-constrained Event Driven Systems

Real-Time Mission Applications are Event Driven, unlike Time Driven Control Applications

- Control Applications: Peak Rate ~ Sustained Rate, \( J << T \)
  - Objective: Meet Deadlines, Minimize Jitter

- Mission Applications: Peak Rate >> Sustained Rate, \( J >> T \) => high burstiness
  - Objectives: Meet Deadlines, Minimize Response Time, Maximize Resource Utilization

Discrete Events Time Constraints Specification

<table>
<thead>
<tr>
<th>Strictly Periodic</th>
<th>Periodic with deadlines</th>
<th>Period, Jitter with deadlines</th>
<th>Bursty with deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {T, D=T} )</td>
<td>( {T, D&lt;T} )</td>
<td>( {T, J, D} )</td>
<td>( {T, B, I, D} )</td>
</tr>
<tr>
<td>( J &lt;&lt; T )</td>
<td></td>
<td></td>
<td>( J &gt;&gt; T )</td>
</tr>
</tbody>
</table>

More convenient form for bursty events
- \( B \)- Burst Size
- \( I \)- Integration Interval
Can be converted to \( (T,J,D) \)

RTEdge™ supports the construction of Time-constrained Event Driven Systems
Specifying Real-Time Contracts: Event Flows specification and Conformance

- **Event Specification Flow**: specifies a causal relationship between start and end Events
  - Key concepts:
    - A Flow uses processor and/or communications channel resources
    - PROCESSOR and COMMS resource scheduling are based on Event Flows
    - RTEdge automatically calculates implementation Flow paths and Worst-Case Response Times
  - User specifies real-time contracts in terms of
    - Time constraints: period / burst parameters and deadline
    - Flow Cost constraints: Processor/bus cost, optional input/output event ratio bounds

```
Adjust_flow  {Sensor_in {P=100ms}, Adjust_ctrl {D=75ms}}
```

Specified on Capsule Interfaces

Calculated on Capsule Implementations
Independent Inputs and Transactions

- Independent Inputs are Signal Event inputs to Application Ports
  - Not all Application inputs are independent
  - Some can be determined by Flow analysis of External Task FSM to be caused by previous Application outputs

- A Transaction is the transitive closure of all possible causally related events and triggered executions in contained AC FSM instances, caused by the arrival of an Independent Input Signal
  - All possible Flows paths sourced in an Independent Input and terminated on causally related observable RTEdge events
Bounding State Transition Cycles

Unbounded state transition loops can be formed

- Intra-AC, within the same AC FSM, when the flow of control from a Transient state leads back to a preceding Transient state
- Inter-AC loops, when ACs exchange repeatedly the same Signals and repeat the execution of the same transitions

Characterization of Loop Bounds is critical for Schedulability Analysis:

- Identification by analysis of unbounded Intra-AC and Inter-AC cycles
- Syntax for declaring an upper bound for how many times a Loop (and its Activities) will execute

Loop Heads are used for bounding Intra-AC FSM loops

- associated with Transient states
- automatically discovered and presented to the user
- user must enter maximum iteration counts for every Loop Head
Loop Constraint Flows

Loop Constraint Flows specify a Signal count ratio between two endpoints discovered by analysis as relevant in creating an infinite cycle.

Users can create Loop Constraint Flows on Composite Capsules and Applications.

Inter-AC unbounded loops appear after analysis as $1:\infty$ ratio. The user bounds the Loop for Schedulability Analysis by specifying an upper bound for the number of events created in the infinite cycle.

Grammar:

```
loopConstraintFlow : 'loopConstraintFlow' flowName ':' 'start' flowLimit ',' 'end' flowLimit;
```

Example:

```
Flow BoundLoop start AC1_R1.p1.sig1, end AC2_R2.p2.sig2;
loopConstraintFlow BoundLoop: start 1, end 3
There can occur 3 instances of AC2_R2.p2.sig2 for every 1 instance of AC1_R1.p1.sig1
```
Hard and Soft Real-time Engineering

RTEdge™ allows Flow allocation to hard and soft real-time domains

- Hard real-time ensures best CPU allocation to meet all Flow deadlines (based on Deadline Monotonic Analysis)
- Soft real-time allocates percentage of CPU to Flows that do not have to complete within a deadline (using Weighted Fair Queuing)

Allows mix of critical and non-critical functions

- Packet routing (hard) can be combined with statistics collection (soft)

Soft Real-Time Server is a Hard Real-Time Transaction triggered by a Timer input

- Implemented as an RTEdge Atomic Capsule
Soft Real-Time Server AC FSM Policer

- Soft Real-Time with Elastic mode is equivalent to Background dispatch
- Non Elastic Soft-Real time servers are limited to their %CPU allocation
- This implementation shows how RTEdge can be used as a micro-kernel for implementing richer dispatch and synchronization protocols
Schedulability Analysis

Modified Lehoczky, Harbour Klein Algorithm for Fixed Priority Task Set with Precedence

Linear precedence sub-tasks of original algorithm supplemented with
- Conditional execution nodes
- Parallel (fork) nodes
- Bounded Cycles
to reflect RTEdge Transaction graphs

1. Static Priorities are assigned per event and state based on deadlines (DMS)
2. Priorities are statically adjusted based on resource usage (priority ceiling protocol)
3. Create task system from model (all RTEdge Transactions originated in Independent System Inputs)
   - task system consists of one periodic task per system input
   - each task is modeled as a graph of subtasks with cycles and conditional paths
   - ensure cycles are bounded by user-supplied bounds
4. calculate Worst Case Response Time (WCRT)
   - calculate utilization, give up if greater than one
   - for each task
     ‣ determine worst case phasing
     ‣ Categorize different classes of Interference from other tasks
     ‣ Determine maximum blocking from lower priority Tasks
     ‣ calculate worst case response times for each subtask
     ‣ compare WCRTs against deadlines
Assume-Guarantee Compositional Verification (1)

- A divide and conquer strategy for large state space systems formal verification
- Formal Model Checking does not scale up, works well for small state spaces
- System broken down into smaller state-space interacting sub-components
  - System sub-component behavior described by *Finite State Machines (FSM)* described by Label Transition Systems
  - Global System State Space is a parallel composition of interacting FSMs
1. Check if a Property of a sub-component in “guaranteed”, “assuming” a certain environment behavior
2. Extend the Property “guarantee” to parallel compositions of sub-components that are conformant with the “assumption”

Assume-Guarantee Compositional Verification (2)

Is property $P_1$ TRUE?

1. If $S = C_0 \ || \ C_1$ and
2. If environment assumption $A_0$ is TRUE $\Rightarrow C_1$
   property $P_1$ is TRUE
   $$<A_0> C_1 <P_1> $$ (Hoare Triples) (1)
3. Always $C_0$ interacts with $C_1$ conformant to
   assumption $A_0$
   $$<\text{TRUE}> C_0 <A_0> $$ (2)
   then
4. Property $P_1$ is TRUE for $S=C_0 \ || \ C_1$
   $$<\text{TRUE}> C_0 || C_1 <P_1> $$ (3)

$C_0, C_1$ behavior, Property $P_1$ and Assumption $A_0$ can be expressed as FSMs

$P_1$ might be an Assumption for interacting with a $C_2$ component

Generalization to multiple parallel components
Specifying Assumptions as 2-Roles Protocol Contracts

Assume-Guarantee Verification techniques use iterative Model Checking and counter-example learning to “discover” Assumptions between existing components.

Conjugated Role Port

Base Role Port

‘protocol’ A₀

Output/Required | Input/Provided
---|---
Event | Data | Event | Data
res_ack | res_id_uint16 | res_request | reservation_struct
res_info | reservation_struct | res_cancel |
| | | res_query | res_id_uint16

Assumptions on Signal Order and Data

The UML2 concept of Protocol can be used as a specification mechanism for capturing and refining interaction Assumptions when building Software Components.

Copyright Edgewater Computer System Inc. May 18, 2010
Capsule Interfaces

Capsule Interfaces are sets of Protocol Typed Ports and Provides/Requires Data Ports

A Subset of AADL Component Type features
- No Subprogram and Subprogram Group Access
- AADL Ports \( \Rightarrow \) Capsule Interface Ports typed by a Protocol with only one Event

Formality of the specification contract can be ratcheted up by adding order constraints
- No order imposed – Assumption is that all orders are possible
- Order among Events of a 2-role Protocol
- Order among Events of different Ports in the Capsule Interface Port grouping
Capsule Interface Order Assumptions

Formalisms for expressing partial Event Order Constraints
- Capsule Interface Flows
  - optional deadline constraint, flow cost, flow start/end event ratio bounds
- Regular expressions of Events
- Protocol State Machines and Environment Capsules

Consistency between partial Event Order Constraints can be verified by Model Checking

Conformant Implementation must satisfy all the partial Event Order Constraints declared

Refinement by “Extends” relationship must preserve parent’s Event Order Constraints

Copyright Edgewater Computer System Inc. May 18, 2010
Component Structure Hierarchy
 Specification and Refinement

- Pay back in test avoidance by precise specification and early model checking
- Use hierarchical structure to specify complex behavior and prove correctness
- Use of Protocols for top-down Components specification and bottom-up integration
Core Component – Atomic Capsule

External AC Events
• to/from AC Ports

Internal AC events
• produced/consumed by the state machine
• activity completion events
• timeout events (release 2.0)

Protocol

Data Access Points
• explicit access to shared data

External Events

Output
Input

AC Attributes (state variables)

Data Event Data Event

integer

res_ack

res_request

make_reservation

int timeout_count;

reservation_struct rs;

AC State Machine

S2

S2.act

RC=1

RC=2

S3

S3.act

S4

S4.act

outlist Tout

The total AC State Space is finite, encoded in the FSM, local data variables, shared data and incoming Signal Event data

Copyright Edgewater Computer System Inc. May 18, 2010
Atomic Capsules State Machines

- Subset of UML2 State Machines
- Three types of states:
  - Initial State
    - Initial starting point for all ACs
  - Stable States – equivalent to AADL2 BA Complete states
    - Synchronization points, awaiting dispatch of incoming trigger signal events, no execution on entry or exit
  - Transient States – equivalent to AADL2 BA Execute states
    - Associated with an Activity – equivalent of a AADL Subprogram
      - Activities are C functions (Misra C recommended)
      - Formal parameters code generated
        - Pointers to local AC attribute data, last message data and declared “required” shared data
      - Activities return a return code enum value
      - Outgoing transitions from Transient states are labeled and selected based on the returning return code value
        - One of the outgoing transition must be marked as “default”
      - A Loop Head counter can be declared on a Transient state to bound state cycles

- Only non-blocking Signal Data can be send to a local Port
  - The signal must appear as outgoing in the Protocol role assumed by the port
- Flow end marker on distinguished state
- Event causality can be statically determined by analysis and checked for conformance with Specified Flows on Capsule Interfaces
interface AC1_Interface {
    port AC1_ResetPort: base, protocol RT_InitProtocol;
    port DataToAc2Port: base, protocol BetweenACsProt;
    servicePort Timer1Port: protocol RT_PeriodicTimerProtocol, requires Timer1Service;
}

atomicCapsule AC1 implements AC1_Interface {
    attribute counter: type uint8;
    stateMachine (S0) {
        // States:
        stable S0;
        transient S1: activity InitData;
        stable WaitForTimer;
        transient IncrementCount: activity IncCount;
        // Transitions:
        transition T1: S0 -> S1, trigger AC1_ResetPort.RT_InitSignal;
        transition InitComplete: S1 -> WaitForTimer;
        transition GotTimer: WaitForTimer -> IncrementCount,
            trigger Timer1Port.RT_Timeout;
        transition DoneInc: IncrementCount ->
            WaitForTimer {
                out DataToAc2Port.dataSigBetweenAcs1,
                data counter;
            }
    };

    activity IncCount: (void) returns void, WCET (Lineup1 10 milliSec),
        file "IncCount.c";
    activity InitData: (void) returns void, WCET (Lineup1 5 milliSec),
        file "InitData.c";
};
Composite Capsule as Architecture Template
Top-down View

Component Architecture => specifying component structure
Integrating compliant sub-component implementations
Software System View in RTEdge™ 2.0

RT Components (Capsules)
Build, Execute, Debug
Build Options, Compiler and OS options (e.g., vxWorks 5.5.1)

Schedulability
Mathematical Models for Processor and Bus Latency

Physical Bus Role 1
Processor Role 1
(e.g., PPC, x86)

Hardware System

Processor Role 2

Processor Types

Flow Engineering:
System Event Flows with Time Constraints
• Period
• Deadline
• Jitter

Flow Latency Calculation

Ports and Protocols

Logical System

Task 1: RTEdge App
Task 2
Task 4

Flow Latency Calculation

Copyright Edgewater Computer System Inc. May 18, 2010
RTEdge™ Evolution

- AADL runtime microkernel OSATE integration
- Formal Checking Link to Promela/Spin
- IEEE1850 Property Specification Language adaptation
- Virtual time execution platform
- Scheduling and deployment support for multi-core processors
Promela/Spin Formal Checking Link

*RTEdge™ Formal Link* feature will allow proving statically, by model checking, various safety and liveness properties on RTEdge™ user-built models.

This is accomplished in four steps:

1. by optional annotation of the RTEdge™ model with assertions, constraints, behaviour assumptions and expected temporal properties
2. by generating Promela code from the annotated RTEdge™ model, such that the RTEdge™ user model is translated into an equivalent Promela model
3. by verifying safety and liveness properties through Spin verification
4. by interpreting Spin counter-examples as RTEdge™ model execution traces.
Formal Checking Link Architecture

- **SW System Spec**
  - SW System Spec
  - SW System Spec

- **RTEdge Tools**
  - RTEdge Model Editors
  - WCRT Mathematical Model
  - RTEdge Model Validator

- **RTEdge User Model**
  - Built-in Assumptions
  - Used defined Assumptions/Assertions
  - Order constraints: Flows, Other
  - Time constraints: Deadlines, WCETs

- **New RTEdge Tools**
  - RTEdge/SPIN Validator
    - Independent transactions
    - Deadlock, Livelock check

- **Code Generator, Build and Debug**

- **Source Language**
  - (C/C++)

- **Executables**

- **Target Debug**

- **Promela Model**

- **[Promela Language]**
  - Equivalent RTEdge User Model in Promela
  - Preserves the state space of the RTEdge Model

- **[Promela Language]**
  - RTEdge Property Claim

- **Spin to RTEdge Counter-example Parser**

- **Copyright Edgewater Computer System Inc.**
  - May 18, 2010
IEEE1850 Property Specification Language Adaptation

Define new annotations for RTEdge™ to express signal ordering based on a subset of IEEE1850 PSL Boolean Layer and Sequential Extended Regular Expressions (SERE) of Signals

Add *Type Annotations* to RTEdge™ types

Make these annotations an optional part of the RTEdge™ types specification:

- Regular Expressions of Signals on Protocols
- Regular Expressions of Signals on Capsule Interfaces
- Regular Expressions of Signals on Ports of Capsule Roles contained in a Composite Capsule type
- Atomic Capsule Types local variables assertions
- Atomic Capsules transient states behavioural action specifications

Prove Temporal properties for RTEdge™ Implementation Types expressed using an adaptation of the Temporal Layer of IEEE1850 Property Specification Language

Framework for Compositional Verification by using the RTEdge/Spin Formal Checking Link
The concept of Virtual Time Execution

- **Application Software**...
- **RTEdge™ Toolset**
- **Radar Processor Execution Costs**
- **Schedulability**

**System Inputs**
- Time Arrival
- Constraints

Generates application from user’s model.

**Virtual Time Run Time Exec**

**Desktop Processor & Operating System**

**RTEdge™ Run Time Exec**

**Radar Processor Hardware & Operating System**

**System Input Interface**
- System Input Time Arrival Constraints
- VT Processor Role

**Discrete Event Dispatcher**

**POST Event**

**NEXT Event**

**Time Services**

**Simulated System Input Interrupts**

Copyright Edgewater Computer System Inc. May 18, 2010
Multi-node Virtual Time Execution

Application Software...

RTEdge™ Toolset

Application Software...

RTEdge™ Virtual Time Run Time Exec

RTEdge™ Virtual Time Run Time Exec

System Input Interface

VT Processor Role

System Inputs Time Arrival Constraints

Discrete Event Dispatcher

Desktop Processor & Operating System

Radar Processor Execution Costs

Mission Processor Execution Costs

Simulated System Input Interrupts

Copyright Edgewater Computer System Inc. May 18, 2010
Summary of the RTEdge™ approach

Main RTEdge™ goal: substantial reduction of the effort and costs to build and verify hard real-time and safety critical distributed systems

Main RTEdge™ technical approach:

1. Definition of a domain specific modeling language subset of UML 2.0 and AADL
   - Minimalist approach: subset includes only domain specific, statically analyzable modeling constructs (composite structure, state machine, and flows)
     - “Safety subset” approach eliminates modeling constructs prone to induce temporal and functional non-determinism
       - Approach used previously in ADA (Spark83/95, Ravenscar), C (Misra) and C++ (JSF C++, Misra C++)

2. Include mathematical models for processor and communication resource allocation

3. Apply model driven development techniques that increase developer productivity

4. Multiple levels of formal specification with increased precision, link to model checkers

Benefits:

- Design time proof that real-time deadlines will be met
- Design time proof of certain critical functional properties
  - Provides high assurance correctness proof without exhaustive manual testing efforts
  - Provides automation in generating compliance tests