Model Based Design of Distributed Real Time Embedded Systems

Sherif Abdelwahed

March 23, 2007

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee
New Demands on DRE Systems

The Past

- Standalone Embedded Systems
  - Domain specific components
  - Resource constrained
  - Limited flexibility
  - Controlled operation environment

The Future

- Enterprise DRE systems
  - Tightly integrated SW & HW components
  - Network centric “systems of systems”
  - Stringent simultaneous QoS demands
    - e.g., latency, jitter, availability, etc.
  - Open environment, real-time interactions

This talk focuses on technologies for enhancing DRE system productivity and quality
Key Challenges for DRE Systems: Analysis

- Popular technologies and tools provide inadequate support for
  - Expressing design intent more clearly using domain concepts
  - Checking constraints and invariants
  - Specifying and analyzing dependencies
Promising Solution: Model Driven Development

- Develop, validate, and standardize generative software technologies that:
  1. Model
  2. Analyze
  3. Synthesize
  4. Provision

multiple layers of application components that require simultaneous verification and control of multiple QoS properties end-to-end

- Partial specialization is essential for inter-/intra-layer optimization & advanced product-line architectures

Goal is to enhance the reliability and quality of DRE systems by providing automated design and analysis tools for application developers
MDD Technology Evolution

Artifacts, Programming Languages, and Platforms

Model-Driven Engineering (MDE)

Domain-specific modeling languages
- ESML
- PICML
- Mathematica
- Metamodels

Domain-independent modeling languages
- State Charts
- Timed Automata
- Interaction Diagrams
- Activity Diagrams

Research goal is to validate model transformation, and automate safety and QoS analysis
Domain, Model, and DSL

- A domain describes a **bounded area of knowledge or interest**
  - It can be structured into various **subdomains**
- A metamodel is a formal representation of the **concepts** in that particular (sub-)domain, as well as their relationships
  - A metamodel is also called the **abstract syntax** (of a DSL)

- A formal model (or just "model") built by the DSL
  - Is an **instance of its metamodel** and respects the static semantics
  - Uses the **concrete syntax** of the DSL
  - and gets its **meaning** from the DSL's semantics

- A **Domain-Specific Language** (or DSL) comprises
  - The **metamodel** (the concepts it represents)
  - A **concrete syntax** to represent these concepts
  - As well as the **semantics** of the concepts
- A DSL is sometimes called a **domain-specific modeling language** (DSML)
Model Transformation

- **Model-to-Model** transformations can be used to produce a **formal model** from a given **design specifications**
- This is typically **based on a different metamodel** defining the transformation rules
- The transformation is based on a **general semantic domain** shared by the source and destination DSMLs

\[ L = < C, A, S, M_S, M_C> \]

- Abstract Syntax \( A \)
  - Modeling Concepts
  - Relations
  - Well formedness rules
- Syntactic Mapping \( M_C \)
- Concrete Syntax \( C \)
  - Notations for representing models
- Semantic Mapping \( M_S \)
- Semantic Domain \( S \)
  - Mathematical abstraction for specifying the meaning of models

Diagram:
- Transformation from DSML\(_1\) to DSML\(_j\)
- \( M_{DSML_i, DSML_j} \)
- DSML\(_i\) to DSML\(_j\) (MoC)
- DSML\(_k\) (GT)
- GME Meta

Notations for representing models
Well formedness rules
Mathematical abstraction for specifying the meaning of models
MDD Application to System Verification

- Mapping functional requirements to target platform
- Predictability of DRE systems
  - Develop a suitable abstraction of the system behavior
  - Verify real-time properties and requirements.
- Ensuring safe composition of system components
  - Early detection of deadlock and livelock states
- Scalable solutions for large-scale systems
The Verification Tool Chain

The developed framework employs the Graph Rewriting and Transformation (GReAT) tool, which utilizes graph transformations, and the UPPAAL model checker to verify the schedulability of event-driven real-time embedded systems.
Key System Characteristics

- Hard & soft real-time deadlines
  - ~20-40 Hz
- Low latency & jitter between boards
  - ~100 usecs
- Periodic & aperiodic processing
- Complex dependencies

Legacy Avionics Mission Computing Tasks

- Weapons targeting systems (WTS)
- Airframe & navigation (Nav)
- Sensor control (GPS, IFF, FLIR)
- Heads-up display (HUD)
- Auto-pilot (AP)

Benefits

- Seamless heterogeneity for inter-subsystem or off board communication
- Priority propagation models to preserve end-to-end QoS

System decoupling using the Publisher-Subscriber pattern
MDD Cycle for Verifying Bold Stroke Applications

Component-based Design

Service pattern and specification

Model Transformation

Composable Analytical Models
ESML Modeling of Bold Stroke applications

- ESML is a modeling language for component-based, event-driven systems
- It uses the publisher-subscriber communication pattern
- The models contain information about priorities, worst case execution times and deadlines for actions
The DRE Semantic Domain

Task $i$

- Preempted
- Error
- Executing
- Enabled

Scheduler

- Scheduler
- Schedule
- Select
- Enable

Channel $i,j$

- Send
- Idle
- Receive

Timer $k$

- Timer
- Period $PD_k$
- Clock $x_k$

Task $i$

- Start $?$
- $x_i = DL_i$
- $y_i = \beta$
- $x_i \leq DL_i$

Scheduler

- Schedule
- Select
- Enable
- $n$

Channel $i,j$

- Start $?$
- $x_c \leq \delta$
- $buffer_c$++

**Parameters**

- Clock $x_c$
- Delay $\delta$
- Variable $buffer_c$

**Conditions**

- $x_k = 0$
- $x_k \leq PD_k$
- $y_i = 0$
- $y_i \leq \beta$

**Variables**

- $w$
- $idle_i$
- $idle_c$

**Executions**

- $\delta$
- $buffer_c$

**Events**

- $run_i$
- $run_{n+1}$
- $start_i$
- $finished_i$
- $select$
- $enable$

**Time**

- $[\alpha, \beta]$
- Deadline $DL_i$

**Clocks**

- $x_k$
- $y_i$
- $w$
- $idle_i$

**Variables**

- $buffer_c$
- $idle_c$
- $idle_i$
- $x_k$

Model Transformation Rules

- Model Transformation is defined as a set of tests and rules in GReAT.
- BoldStroke and TA metamodels defined in GME are used to define the pattern matching in GReAT.
- A rule usually has multiple input and output ports.
  - For example, Actions correspond to TA and Events correspond to Channels.
- The rule shown here is that whenever an Action publishes an Event, the underlying TA should send an event to the corresponding Channel.
Avionics Application

Computations on different processors are driven by their respective timers

- Components do not necessarily execute with the timer’s rate
- Task priorities and sub-priorities can be arbitrary (user defined)

<table>
<thead>
<tr>
<th>Component</th>
<th>CPU</th>
<th>Sub-priority</th>
<th>WCET</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>CPU_1</td>
<td>VERY_HIGH</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>AIRFRAME</td>
<td>CPU_1</td>
<td>HIGH</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>PILOT WAYPOINTS</td>
<td>CPU_2</td>
<td>VERY_HIGH</td>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>ROUTES</td>
<td>CPU_2</td>
<td>VERY_LOW</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>NAVIGATOR NAVSTEERING</td>
<td>CPU_3</td>
<td>VERY_LOW</td>
<td>32</td>
<td>38</td>
</tr>
<tr>
<td>NAVATIVE POINTS</td>
<td>CPU_3</td>
<td>LOW</td>
<td>49</td>
<td>50</td>
</tr>
<tr>
<td>NAV_DISPLAY</td>
<td>CPU_4</td>
<td>MEDIUM</td>
<td>26</td>
<td>41</td>
</tr>
<tr>
<td>AF_MONITOR</td>
<td>CPU_4</td>
<td>HIGH</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>NAV_DISPLAY</td>
<td>CPU_4</td>
<td>MEDIUM</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>PILOT CONTROL</td>
<td>CPU_5</td>
<td>VERY_HIGH</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>TACTICAL STEERING</td>
<td>CPU_5</td>
<td>HIGH</td>
<td>68</td>
<td>60</td>
</tr>
</tbody>
</table>
Avionics example: TA Model
The system is schedulable if all tasks can be completed before their deadlines.

Schedulability test is translated to deadlock checking.

Additional properties can also be checked as dependencies and dense time information are captured in the network of timed automata.
Key Challenges for DRE Systems: Design

Devising execution architectures, concurrency models, communication styles, and adaptation mechanisms that ensure multi-dimensional QoS and reliability of new/reusable components

- Popular technologies and tools provide inadequate support for
  - Identifying and reducing performance and robustness risks early in DRE system lifecycles
  - Satisfying multiple (often conflicting) QoS demands
    - e.g., secure, real-time, reliable
  - Satisfying QoS demands in face of fluctuating/insufficient resources
    - e.g., mobile ad-hoc networks
Promising Solution: Self-managing System

A system that senses its operating environment, models its behavior in that environment, and takes action to change the environment or its behavior. An autonomic self-managing system has the properties of self-configuration, self-healing, self-optimization and self-protection.

Self-managing systems deliver:

**Increased Responsiveness**
- Adapt to dynamically changing environments

**Robustness and Resiliency**
- Discover, diagnose, and act to prevent disruptions

**Operational Efficiency**
- Tune resources and balance inputs to maximize use of resources

**Secure Information and Resources**
- Anticipate, detect, identify, and protect against attacks

Source: IBM Autonomic Computing white paper
Model-based Self-management

Self Managing Systems

Operational Models and Specifications

Adaptive Model-based Design

Integrated Development Infrastructure

Hybrid bond graph model

Simulink model

Hierarchical control structure

Generic modeling environment
Control-based Framework for Self-managing Systems

An explicit internal model that captures system behavior

Filters estimate future environment parameters using past values

A practical DRE system composed of several hardware and software components

Actions optimizing system behavior are derived over a limited prediction horizon

Update the system model based on input/output measurements

Detect changes in the system parameters and operational settings

Model-based control

System Model \((M)\)

Performance Optimizer

Online Monitor

Model-learning module

Model Learning Structure

System Model \((M)\)

No change detected

Model change detected

System State \((x)\)

System response \((r)\)

Environment Input \((\lambda)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.

A practical DRE system composed of several hardware and software components

System response \((r)\)

A practical DRE system composed of several hardware and software components

System State \((x)\)

Updated parameters

Control \((d)\)

Detect changes in the system parameters and operational settings

Environment Estimation Filter

Updated parameters

Control Input

Control feedback

State feedback

System Specs and Const.
System Modeling

- Many practical systems exhibit a hybrid behavior comprising both discrete-event and continuous dynamics.
- In addition, control or tuning options must be chosen from a finite set at any given time.
- The dynamics of such systems can be captured formally as class of hybrid systems with finite control set; switching hybrid systems.

The dynamics of a switching hybrid system is represented by:

\[ x(k+1) = f(x(k), u(k), \omega(k)) \]

Where \( x(k) \) is the state at time \( k \), \( u(k) \) is the control input chosen from a finite set \( U \), and \( \omega(k) \) is the environment input.

Environment inputs are typically estimated at time \( k \) based on previous values.

Operation requirement (specification) is represented as a set point or utility function.

\[ \dot{q}(k+1) = q(k) + \left( \dot{\lambda}(k) - \frac{\phi(k)}{\hat{c}(k)} \right) T(k) \]

\[ \dot{r}(k+1) = (1 + q(k)) \frac{\hat{c}(k)}{\phi(k)} \]

\[ \psi(k+1) = \phi^2(k) \]
Limited Lookahead Control

Selection of the next step is based on a map that defines how close the current state is to $X_s$.
Controller constructs a tree of all future states up to certain depth.
A path that minimizes the distance to $X_s$ is traced back to current state and the initial step is selected.

Given a switching hybrid system with state space $X$ and control input $U$. The control problem for a set point $X_s$ is to:
- Drive the system from any state in $X$ to $X_s$ in finite time using inputs from $U$.
- Maintain the system in $X_s$

In the case of utility specification, the controller aims to reach the optimal state that maximizes the system utility.

Minimize $\sum_{i=k+1}^{k+N} J(x(i), u(i))$
Subject to $\dot{x}(i) = f(x(i), u(i), \hat{\omega}(i))$, $H(x(i)) \leq 0$, $u(i) \in U(x(i))$
Application: Signal Detection System

- **Objective:** identify relevant data from incoming signals
- Signals are received at a time-varying rate
- Detection accuracy and computation time depend on the signal size
- The controller must minimize the latency while maximizing accuracy

**System dynamics**
\[
\hat{q}_i(t+1) = q_i(t) + (\hat{\lambda}_i(t)c_i(r_i(t)) - v_i(t))
\]
\[
\hat{a}_i(t+1) = f_i(a_i(t), r_i(t), v_i(t))
\]

**Objective function**
\[
\max_t [w_1\hat{a}(t) - w_2\hat{q}(t)]
\]

- Experiment results show that the controller adapts quickly to signal arrivals.
- In the absence of such adaptation, the queue may overflow, missing potentially interesting signals.
QoS Guarantee: Stability Analysis Approach

**System Dynamics**
- Single-Mode Discrete-Time
  \[ x(k + 1) = f(x(k), u(k), \omega(k)) \]
- One-step online control policy
  \[ u^*(x) \in \arg \min_{u \in U} \| f(x, u) - x_s \| \]

**Objective**
For a domain \( D \) and an initial state \( x_s \in D \), decide if there is a neighborhood \( B(r, x_s) \subseteq D \) of \( x_s \) such that:
- \( B(r, x_s) \) is finitely reachable from any point in \( D \)
- The system remains in \( B(x_s) \) under the online control law

**Technical Results**

*To find \( B(x_s) \)*

\[ \text{find } r := \max_{x \in Q} \min_{u \in U} \| f(x, u) - x_s \| \text{ (NLP)} \]

where \( Q := \bigcup_{u \in U} \{ x \in \mathbb{R}^n | \| f(x, u) - x_s \| < \| x - x_s \| \} \)

**Theorem:** \( B(r, x_s) \) is the minimal containable region of \( x_s \)

To determine finite reachability

**Theorem:** \( \partial B(r, x_s) \subseteq Q \Rightarrow B(r, x_s) \) is finitely reachable from \( x \in \mathbb{R}^n \)
Tool Development

The tool supports system modeling, environment forecasting, parameter estimation, and automatic generation of adaptive control code.

GME-based visual modeling and control configuration environment

Prototype tool supporting model analysis and adaptive control configuration
Hierarchical Control Structure

- A global controller manages inter-component interaction and enforces global requirements.
- Abstract representation of the components is used for high-level control decisions.
- Global control actions are given as additional constraints for local controllers.
- Local controllers work then to optimize the performance of individual components.
**Objective:** Manage the power consumed by a heterogeneous computing cluster while satisfying QoS requirements.

- A multi-level control structure is developed where high-level limited lookahead controllers manage interactions between lower-level controllers.
- The cluster is logically partitioned into modules where each module comprises multiple processors.
- A three-level control hierarchy makes dynamic load balancing decisions.
Simulations using World Cup `98 workload traces show that the architecture is scalable and adapts quickly to time-varying workload patterns.

A module comprising four heterogeneous computers is simulated with a desired response time of 4 sec. for incoming HTTP requests.

Multiple modules can be composed to form larger clusters.

The average control overhead for a cluster of sixteen computers is 2.5 sec.
Looking Ahead: Performance Management and Verification

- System verification is conducted initially offline to ensure design correctness.
- System is verified again (incrementally) when the system or its real-time requirement is changed.
- The outcome of the online verification is used to update the control objectives and constraints to ensure the system is maintained at the required performance level.
Looking Ahead: Task Allocation in DRE Systems

Resource configuration graph

Task allocation map

Task configuration graph