Use of Extended Higher-Order Petri Nets for the Design of Dynamic Embedded Systems

Franz J. Rammig

Acknowledgements

The following slides show some work of a larger team within our group. The main contributors are:

• Bernd Kleinjohann (Extended Pr/T Nets and SEA)
• Carsten Rust (Dynamicaly Modifiable Pr/T Nets)
• Friedhelm Stappert (Model Based WCET Analysis)
• Markus Hübel (Zero Overhead Pr/T Net Execution)
• Yang Wan Liu (Jini Based Execution Platform)

Parts of the presentation have been published elsewhere. Please contact our web pages for the respective references.
The Challenge

- Moore's law: IC complexity doubles every 18 months
- Nielsen's law: bandwidth doubles each 12 months
- Embedded SW: doubles each 10 months

Source: ST Microelectronics

Increasing SW-Intensity

The continuing increase of automotive electronics means significant increase of SW complexity

- more than 80% of functionality implemented by SW
- tendency rising

Premium class, 2000
Examples for dynamically modifiable Systems

• Adaptive Robot Control
  Dynamics due to autonomous modification of the robot control

Examples for dynamically modifiable Systems

• Adaptive Robot Control
  Dynamics due to autonomous modification of the robot control

• Dynamic Colonies of Robots
  Dynamics due to modification of hardware
Application Example

Examples for dynamically modifiable Systems

• Adaptive Robot Control
  
  Dynamics due to autonomous modification of the robot control

• Dynamic Colonies of Robots
  
  Dynamics due to modification of hardware

• “Traditional” Embedded Systems
  
  Dynamics due to failure of hardware
Safety Critical Systems

Basic requirement: disjoint „Fault Containment Regions“

Function A  Function B  Function N

Fault tolerant infrastructure (Byzantine protocols) on TTP or Flexray

ECU 1  ECU 2  ECU 3  ECU 4  ECU 5

clock 1  power 1  clock 3  power 4  clock 5  power 5

power 2

clock 2

Introduction

Design Methodology

Goal:

High Level Specifications

Modeling

Analysis & Partitioning

Extended Pr/T-Net

Synthesis

Target Implementations

Implementation
UML 2.0 Behavioral Description

Activity Diagrams (Example)
Basic Petri Net Model

Petri Net Model:
• Extended High-Level Petri nets

Extensions:
• Extended annotations of transitions
• Enable- and firing-delays
• Hierarchy with graphical representations

Source: Rust

Slide 14© Siemens und Paderborn University 2003 F.Rammig / C-LAB

Basic Petri Net Model

Petri Net Model:
• Extended High-Level Petri nets

Extensions:
• Extended annotations of transitions
• Enable- and firing-delays
• Hierarchy with graphical representations

Source: Rust

Slide 14© Siemens und Paderborn University 2003 F.Rammig / C-LAB
Introducing Timing

Time model
- Specification of intended timing
- Modeling of observed timing
- Discrete time with atomic unit of time (tick $\tau$)
- Distinction between activation and firing delay
- Timing intervals ([min., max.]) for both types of delays

Firing of a transition $t$

Timed Petri Nets

Example
- T2 has to be executed 2 ticks after T1 and it takes 2 ticks to fire T2
- Sometimes later T3 has to be executed and it takes 2 to 4 ticks to fire T3

Nondeterminism concerning timing
Hierarchical Petri Net Model

Source: Liu
Hierarchical Petri Net Model

CONSUME or ADD
READ
INHIBITOR
OVERWRITE

Source: Liu
Hierarchical Petri Net Model

CONSUME or ADD
READ
INHIBITOR
OVERWRITE

Source: Liu
Hierarchical places:
- Net embedded into hierarchical place remains active as long as the hierarchical place is marked
- Similar to hierarchy concept in StateCharts
- Implicit history (embedded net keeps marking when deactivated)

Hierarchical transitions
- Net embedded into hierarchical transition remains active as long as it is life
- Only then the hierarchical transition fires
- Semantics due to „Structured Nets“ (Cherkasova/Kotov)

Dynamic Modification of Petri Nets

Existing approaches:
- Activation / Deactivation of net elements (usually edges)
- Dynamic instantiation and deletion of predefined subnets

Our approach: Application of Petri net graph transformations at run time
- Annotation of transitions with graph transformation rules
- Inclusion of graph transformation rules into the transition firing rule

Formal definition: Coupling of a high-level Petri net model with elements of graph grammars
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

[Diagram of Petri Net with dynamic modifications]
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Source: Rust
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Source: Rust
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Source: Rust
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

[Diagram of a Petri Net with dynamic modification shown]

Source: Rust

Slide 41
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Source: Rust
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Source: Rust
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Source: Rust
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Source: Rust
Dynamic Modification of Petri Nets

Idea: Dynamic Modifications bound to Transitions

Informal Example:

Petri Net Transformation Rules

Production transforming a net L into a net R (L => R):

- Definition w.r.t. an interface net K
- Tuple of two functions ( l : K → L , r : K → R )

Example:
Application of Transformation Rules

Application of a production \( p = (l : K \rightarrow L, r : K \rightarrow R) \) to a given Petri net \( N \) leading to a Petri net \( M \):

\[
\begin{align*}
L & \quad l \quad K \quad r \quad R \\
N & \quad n \quad c \quad m \quad M \\
\end{align*}
\]

Example:

Source: Rust
Application of Transformation Rules

Example:

- Hierarchical Petri net graph $F = (P, T, \text{pre}, \text{post}, s)$
- Annotations
  - $A_P : P \rightarrow S$ (Place-Types, sorts of a signature)
  - $A_I : T \rightarrow L_d \times P$ (Variable sums at transition in edges)
  - $A_G : T \rightarrow L_{op} (\text{bool})$ (Transition Guards)
  - $A_O : T \rightarrow L_m \times P$ (Term sums at transition out edges)
  - $A_T : T \rightarrow R$ (Transformation rules)
  - $A_P : P \rightarrow L_{const}$ (Initial place markings)

Transformation rule: $r = (p, v)$ where
- $p = (l : K \rightarrow L, r : K \rightarrow R)$ is a production
- $v$ is a hierarchical node (the scope of $r$)
**Dynamically Modifiable High-Level Petri Nets**

- Hierarchical Petri net graph $F = (P, T, \text{pre}, \text{post}, s)$
- Annotations
  - $A_P : P \rightarrow S$
  - $A_I : T \rightarrow L_d \times P$
  - $A_G : T \rightarrow L_{op}(\text{bool})$
  - $A_O : T \rightarrow L_m \times P$
  - $A_T : T \rightarrow R$
  - $A_P : P \rightarrow L_{\text{const}}$

Transformation rule: $r = (p, v)$ where
- $p = (l : K \rightarrow L, r : K \rightarrow R)$ is a production
- $v$ is a hierarchical node (the scope of $r$)

**Semantics of Dynamically Modifiable High-Level Petri Nets**

- Petri net configuration: $C = (N, M)$ where
  - $N$ is a Petri net
  - $M$ is a marking of $N$
- Transition step: $S = (t, D, B)$ where
  - $t$ is a transition
  - $D$ is a direct derivation using the production of $t$
  - $B$ is a consistent substitution for the variables of $t$
- Incremental effects of a step $S = (t, D, B)$ in $C = (N, M)$

  $E_{N^-}(S)$ \{ Functions realizing $D$ by removing nodes \}

  $E_{N^+}(S)$ \{ $(E_{N^-}(S))$ and adding new ones $(E_{N^+}(S))$ \}

  $E_{M^-(S)}$ \{ Token removed from input places of $t$ \}

  $E_{M^+(S)}$ \{ Token added to output places of $t$ \}
Semantics of Dynamically Modifiable High-Level Petri Nets: Firing rule (1)

Given:
- a transition $t$ with the transformation rule $A_t = (p, v)$ where $p = (l : K \rightarrow L, r : K \rightarrow R)$
- a transition step: $S = (t, D, B)$
- a configuration $C_1 = (N_1, M_1)$

$S$ is enabled in $C_1$ if:
- $t$ is an element of $N_1$
- $v$ is a hierarchical node of $N_1$
- $v$ instantiates $N$ (see the diagram on the right)
- $E_{M^t}(S)$ is included in $M_1$

Direct derivation $D$:

Semantics of Dynamically Modifiable High-Level Petri Nets: Firing rule (2)

Given:
- a transition $t$ with the transformation rule $A_t = (p, v)$ where $p = (l : K \rightarrow L, r : K \rightarrow R)$
- a transition step: $S = (t, D, B)$
- a configuration $C_1 = (N_1, M_1)$

If $S$ is enabled in $C_1$ it may fire leading to $C_2 = (N_2, M_2)$ where:
- $N_2$ results from the direct derivation $D$
- $M_2$ results from subtraction of $E_{M^t}(S)$ from $M_1$ and addition of $E_{M^t}(S)$

Source: Rust
**Design Flow**

- **Modeling**
  - Petri Net Simulation
  - HL-Petri Net
- **Analysis & Partitioning**
  - Partitioning
- **Simulation of Execution**
  - Partitioned HL-Petri Net
- **Synthesis**
- **Implementation**
  - Processor
  - Processor
  - Processor
  - Sensor
  - Actuator

**Simulation: SEA Environment**

- Offline Simulation and Animation
- Modeling of Heterogeneous Systems
- Predicate/Transition Nets
- Abstract Graphical Representation for Interfaces
- User Defined Graphical Libraries (Components)
- User Defined Interactions (Panels)
- Interactive Animation
Simulation in SEA

SEA: System Engineering & Animation

7 Segment Counter built from Library
Introduction

Hierarchy

Self-Modification

Simulation

Middleware

RTOS Integration

Dynamics via self-modification of the control

Dynamics via modification of the underlying hardware

Source: Liu
Architecture of Execution Platform (Jini)

Source: Liu
Szenario: Check-out of a Processor

Introduction
Hierarchy
Self-Modification
Simulation
Middleware
RTOS Integration

PetriNet Client
PetriNet Client
PetriNet Client
Lookup-Service
Allocation Service
PN-Engine
PN-Engine
PN-Engine
PN-Engine
Processor
Processor
Processor
Network
Sensor
Actuator

Source: Liu

© Siemens und Paderborn University 2003 F.Rammig / C-LAB
Szenario: Check-out of a Processor

Execution Environment

PetriNet Client  PetriNet Client  Lookup-Service
Allocation Service

PN-Engine  PN-Engine  PN-Engine  PN-Engine
Processor  Processor  Processor  Processor

Network

Sensor  Actuator

Timed Pr/T Nets
Hierarchy
Self-Modification
Simulation
Middleware
RTOS Integration

Execution Environment

PetriNet Client  PetriNet Client  Lookup-Service
Allocation Service

PN-Engine  PN-Engine  PN-Engine  PN-Engine
Processor  Processor  Processor  Processor

Network

Sensor  Actuator

Source: Liu
Other Scenarios

- New service
- Client looks for service
- Client looks for service with sufficient capacity using allocation service
- Client hands over its Petri Net to other engine
- New Petri Net
- Move Petri Net from one processor to another one
- Check-out processor
- Delete Petri Net
- Execute Petri Net

Open Question: How to Implement the PN-Engine
Traditional Pr/T-Net Simulation

Process

Net Simulator

Source: Hübel

OS Supported Pr/T-Net Execution

Integration

Pr/T-Net Engine

Analysis

Customization

Limitation of Token
Place is used by single Transition
etc.

Use of Static Array in the Mailbox
No Synchronization required
etc.

Source: Hübel
Simulation vs Execution

Pr/T-Net

Operating System

Sub-Net Simulation

Transition Queue

Net Task

Token Buffer

Token Buffer

Token Buffer

Token Buffer

Token Buffer

Driver

Mailbox

Mailbox

Fireable (ready)

Firing (run)

Disabled (blocked)

Task Schedule

Source: Hübel

Slide 81

From Simulation to Execution

Goal:

- Deeper Integration into OS
- Exploitation of Similarities between
  - Token Buffer ⇔ Mailbox
  - Transition ⇔ Thread
  - Transition Queue ⇔ Task Scheduler’s Ready Queue
- Moving of Simulation Management into OS:
  - Simulator ⇔ Scheduler

Simulation ⇒ Execution

Source: Hübel

Slide 82
Pr/T-Net Implementation

- Net Modeling
- Partitioning
- Mapping
- Synthesis
- Integration of special services for Pr/T-Nets
- OS Execution

Pr/T-Net

Partitioning

Synthesis

OS Execution

Introduction
Hierarchy
Self-Modification
Simulation
Middleware
RTOS Integration

Source: Hübel
F. Rammig / C-LAB

Pr/T-Net Partitioning

- Exploitation of natural parallelism of Pr/T-Net
- Trade-Off: Minimize edges between partitions

Sub-Net A
Sub-Net B
Sub-Net C

Output-Place
Input-Place

Special I/O-Places required for external Token Propagation!
Parallel Pr/T-Net Simulation

Net Control Task
- Task Creation
- Sub-Net Initialization
- Termination Control

Parallel Pr/T-Net Execution

First step:
- create minimal Sub-Nets of depth 1
- only Transition with Input-Conflicts are in the same partition
- instead of simulation only a simpler management is required
- after receiving a token only one cycle of one transition can take place
Zero-Overhead Parallel Pr/T-Net Execution

Second step:
• additionally use Token Mailboxes for Places

Results

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Zero-Overhead Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>transition activation</td>
<td>transition activation</td>
</tr>
<tr>
<td>token flow</td>
<td>memory copy</td>
</tr>
<tr>
<td>synchronization</td>
<td>Simulator</td>
</tr>
<tr>
<td>real</td>
<td>Simulator on top of OS</td>
</tr>
</tbody>
</table>
**WCET Analysis**

*Here:* Model = Petri Net

- Provide additional information about (im)possible net behaviour

---

**WCET Analysis: Model Level**

Diagram showing the process from Petri Net to WCET Tool, including Code Generation, Annotated Source Code, Behaviour Information, Reachability Analysis, Reachability Graph, Behavioural Analysis, User Input, and finally WCET.
WCET Analysis

Introduction
- Hierarchy
- Self-Modification
- Simulation
- Middleware
- RTOS Integration

WCET Analysis

High-Level Analysis
- Program Source
- Compiler
- Object Code
- Simulator
- Timing Graph
- ILP Constrains
- Convergence
- Flow Facts
- Flow Analysis

Calculation
- or
- ILP based
- Path based
- Pipeline Analysis

Source: Stappert
P. Ramin / C-LAB

Conclusion

• Highly dynamic Embedded Systems need novel design approaches
• Extended PNs serve as ideal modeling means
• Timing, hierarchy and self-modification are essential
• Efficient run-time platform needed:
  • Jini-like services to support self-modification
  • Direct integration into RTOS for efficiency

Source: Stappert
P. Ramin / C-LAB
The End...

Thank you for your patience!!

Any questions??