A Graceful Degradation Framework for Distributed Embedded Systems

William Nace
Philip Koopman
RoSES Project

- Robust Self-configuring Embedded Systems (RoSES)
- Robustness gained with automatic graceful degradation
  - Must not require human intervention to specify or guide
- First shot → automated reconfiguration when fault detected

- Domain -- Distributed Embedded Systems
  - Distributed – functionality remains after most failures
  - Smart sensors – general compute capability
  - Most functionality is optimization

- Examples: elevators, autos, copiers, plant control, …
  - Not Internet toasters
Overview

- **System-level customization**
  - “Customize a system to maximize the functionality of given H/W”

- **3 step, iterative framework for algorithm**
  - Feature selection – find features to implement
  - S/W selection – determine which software components necessary
  - Allocation – fit the software to the hardware
The Framework: Mission Statement

“Customize a system so as to maximize the functionality of given H/W”

• Isn’t this just like:
  – Hardware-software co-design? I’m given H/W, not given functionality
  – Reconfigurable computing? I’m not using special purpose H/W, different time constraints
  – Load balancing? I select/allocate S/W for robustness/functionality, not performance

Example Scenario: Automotive

• Not run-time (yet)
  – Exploit ground states
• May require external assistance
  – Or fault tolerant subsystem (using standard techniques)
The Algorithm from 30,000 feet

Given: Hardware Spec
      Product Family Software Spec

1: Choose Features to implement

2: Choose S/WAdapters to form features

3: Allocate Adapters to hardware

Produce: Adapter ↔ PE mapping

Iterate on failure
HardwareSpecification

◆ List of Processing Elements (PEs)
  • Vector of available resources
    – CPU cycles, Flash, RAM, …
  • Operational sensors and actuators (attached to PEs)

◆ Network
  • Scalar resource (bandwidth)
  • Broadcast, real-time
  • Archetype: Control Area Network
Software Specification

- Adapter – RoSES terminology for a software object
- Configuration – collection of adapters
- Configurations form data flow graphs (DFGs)
  - Vertices are adapters and transducers
    - Adapter requirements specified as a vector, same terms as PEs
    - Transducers are merely sources/sinks of data (don’t consume PE resources)
  - Edges are communications
    - Labeled with dataflow requirement
  - Flow
    - from Sensor(s)
    - through 1+ adapters
    - to Actuator(s)
PFA Specification

- **Product Family Architecture (PFA)**
  - Structured view of all possible configurations

- **PFA graph – specifies capability of entire product/model line**
  - Merge DFGs for all configurations
    - Last year’s models
    - High to low end models
    - Etc….
  - PFA Graph is a supergraph of all configuration DFGs
PFA Graph

- Merge DFGs with choice elements
  - Data can flow from any one of the inputs
- Specialization: data element
  - Network message type
- Ex: Conveyor belt part identification

Diagram:
- Barcode Reader → Decoder
- Vision System → Pattern Recognition
- Decoder → Planner
- Pattern Recognition → Machine Control
- Planner → Sorter
- Machine Control → Rest of System
- Part Number
What’s a Feature?

◆ Required as a means to make optimization choices
  • I tried “Data path through PFA graph”
    – Rejected: Too restrictive

◆ “Use of a particular adapter”
  • Dependencies communicated via PFA graph
    – Other adapters automatically selected as “glue”

◆ Feature is given utility value

◆ Similar features require organization
  • Class-based Feature Model
Class-based Feature Model

- Collect similar features into *classes*

- Only one feature from each class useful
  - Do you want multiple transmission control algorithms running concurrently?

- Some classes may be *critical*
  - A valid feature set includes a feature from all critical classes

\[
Utility_{FeatureSet} = \sum_{c} \text{Utility}(Feature_c)
\]
Phase 1: Feature Selection

◆ Goal: Choose a *Feature Set*
  - Maximize utility
    - Try highest utility feature set first
      (In general case it won’t fit in H/W)

◆ Use a combinatorial algorithm
  - Start with highest utility feature from each feature class
  - Number of feature classes/features is small
Phase 2: Adapter Selection

Goal: Choose adapter set to implement the feature set

- Prune PFA graph of “dead” adapters
- Discover paths through features of interest
- Select paths based on what policies?
  - Heuristic analysis experiment
- Adapters from selected paths form Adapter Set
Phase 3: Adapter Allocation

Goal: Map adapter set to PEs, network

- Well-worn research area
- Bin-packing problem (NP complete)
  - Attack as list processing heuristic
    » Sort adapters into list (by what criteria?)
    » Start with largest, packing onto a PE (chosen how?)
    » Success if list gets emptied
A Pretty Picture of the Algorithm

GIVEN:
- PFA Graph
- Processing Elements
- Sensors
- Actuators
- Compute Nodes
- Network

Try Harder? → Failure → No → Report Failure

Phase 1: Feature Selection
Pick target feature set

Phase 2: Adapter Selection
Pick adapter set that fulfills feature set

Phase 3: Adapter Allocation
Pick PEs to hold each adapter

User Supplied Validity Check

ALGORITHM ADJUSTMENTS
- Speed vs. Quality tradeoff?
- Incremental?
- Failover safe?

DEPENDS ON FEATURE MODEL
- Select Features from critical classes
- May include non-critical features

PRUNE DEAD ADAPTERS
- Determine path alternatives
- Select adapters on chosen paths

FOREACH ADAPTER IN SORTED LIST:
- Determine set of available PEs
- Choose PE

PRODUCE:
Adapter allocation mapping
Backtracking/Failure Feedback

- When Phase n+1 fails, Phase n tries again
- What info does Phase n need to make intelligent choices for next attempt?
  - Type of allocation failure (network or PE overflow)
  - How far processing proceeded
  - State of algorithm at failure point
  - ...
- Phase 2 failures provide info for phase 1 to make larger moves in search space
Tar Pits I Want to Avoid Future Work

- **Universal Feature Model**
  - Composability of features, with complex interaction, without combinatorial explosion
  - Class based model is sufficiently expressive for most real systems

- **Validity Checks**
  - System dependent
  - User supplied validity check of resulting allocation

- **Scheduling**
  - Tough. Would obscure proposed work
  - Hack: oversize system and apply RMA, or treat as validity failure
Conclusions

◆ System-level customization
  • “Customize a system to maximize the functionality of given H/W”

◆ 3 step, iterative framework for algorithm
  • Feature selection – find features to implement
  • Adapter selection – determine which adapters necessary
  • Adapter allocation – fit the adapters to the hardware