GUARDS Presentation

Kevin Dorow/Wesley Lawrence
CPT S 562
April 9, 2002
Presentation Outline

- GUARDS Introduction
- Design Rationale
- The Generic Architecture
- Interchannel Communication Network
- Interchannel Error Processing and Fault Treatment
- Conclusions
- Questions / Discussion
GUARDS Introduction

- Previous ultra-dependable real-time systems have been specialized to meet the requirements of one particular application
  - Costly, inflexible hardware-based solutions
  - Systems need to be in service for long periods (decades) and cannot be upgraded (sometimes obsolete before release!)
GUARDS Introduction

GUARDS Project
- Generic Upgradeable Architecture for Real-Time Dependable Systems
- Consortium of European companies and academic partners
- Primary application domains—nuclear subs, railway, and space
- Overall aim—significantly decrease life cycle cost of embedded systems
- 3 Pronged approach for reducing cost of validation and certification
  - Design for validation (minimum set of critical components)
  - Reuse of already validated components
  - Software components of different criticalities
Presentation Outline

- GUARDS Introduction
- **Design Rationale**
- The Generic Architecture
- Interchannel Communication Network
- Interchannel Error Processing and Fault Treatment
- Conclusions
- Questions / Discussion
Design Rationale

- Key nonfunctional requirements
  - Railway
    - Fail-safe control system
    - Extremely low catastrophic failure rates for subsystems
    - Physically segregate vital and non-vital subsystems
Design Rationale

- Key nonfunctional requirements (cont.)
  - Nuclear Submarine
    - Secondary protection functions (for rare accidents)
    - Separate redundant elements by several meters
    - Use of unmodified COTS operating systems is mandatory (to avoid obsolescence)
Design Rationale

- Key nonfunctional requirements
  - Space
    - Phases where tolerance of several faults may be required
    - Phases where redundant elements may be powered down to save power
    - Ability to support software components of different integrity levels
Design Rationale

- Fault Classes
  - Tolerate permanent and temporary physical faults (both internal and external origin)
  - Provide tolerance or confinement of software design faults
  - Permanent internal physical faults are handled by physical redundancy
  - Permanent external physical faults handled by geographic separation of redundant systems
Design Rationale

- **Fault Classes (cont.)**
  - Temporary external physical faults (transients) are handled by recovery mechanisms (recover corrupted processes)
  - Temporary internal physical faults (intermittents) are treated as either permanent or transient faults according to their rate of occurrence
Design Rationale

- Design Faults
  - Many can be handled like intermittents (if they have diverse activation conditions)
  - Design faults that are activated systematically can only be tolerated by design diversification
    - GUARDS does not do this, but does allow for it in the future
    - Have created mechanisms to prevent design faults in non-critical components from affecting critical ones
Design Rationale

- Design Faults (cont.)
  - Diversification to handle design faults in COTS operating systems
  - Activation diversification to handle design faults in replicated hardware and replicated applications
Design Rationale

- Real-Time
  - Stay tuned for the second half of the show (Wes has all the info ready for you!)
Presentation Outline

- GUARDS Introduction
- Design Rationale
- The Generic Architecture
- Interchannel Communication Network
- Interchannel Error Processing and Fault Treatment
- Conclusions
- Questions / Discussion
The Generic Architecture
The Generic Architecture

- Can be configured into a wide variety of instances
- Favors the use of COTS hardware and software
- Application transparent fault tolerance implemented by software (middleware 😊!)
The Generic Architecture

- Three dimensions of fault containment
  - Integrity levels (design fault containment regions)
  - Lanes (secondary physical fault containment regions)
  - Channels (primary physical fault containment regions)
The Generic Architecture

- A particular instance is defined by the dimensional parameters \( \{C, L, I\} \), a reconfiguration strategy, and the appropriate selection of generic hardware and software GUARDS components

- GUARDS Software Components
  - Interchannel communications
  - Output Data consolidation
The Generic Architecture

- GUARDS Software Components (cont.)
  - Fault tolerance and integrity management
    - Software implemented in middleware (distributed set of system components)
    - Is itself fault tolerant (replication and distribution) with respect to faults that affect channels independently (physical faults)
    - Design faults are not addressed in this layer
The Generic Architecture

- **Integrity Dimension**
  - Provides containment with respect to software design faults
    - Protect critical components from the propagation of errors due to residual design faults in less critical components
  - Each application object is given a particular integrity level
    - Based on how much it can be trusted
The Generic Architecture

- **Integrity Dimension (cont.)**
  - Protection is achieved by enforcing an integrity policy to mediate the communication between objects of different levels
    - Prohibit flows from low to high levels
    - Data must inherit the level of integrity from object where it came from
    - Validation objects are provided to apply FT to information flows and output reliable info
      - Can upgrade the integrity level of data
The Generic Architecture

Integrity Dimension (cont.)

- Spatial and temporal isolation are used to ensure that integrity policy is not bypassed.
- It must be assumed for the most critical components and a core set of basic components that there are no design faults present (or they can be tolerated by some other means—diversification).
The Generic Architecture

Lane Dimension

- Multiple processors are used to define secondary physical fault containment regions
  - Can be used to improve the diagnosis of faults within a channel
  - Also scope for improving coverage for design faults (using intrachannel diversification)
  - Can be used to improve the availability of a channel
The Generic Architecture

- **Channel Dimension**
  - Provide primary fault containment regions
  - Employs active replication
    - Requires determinism and total ordering between channels
  - Error processing based on comparison / voting
The Generic Architecture

Channel Dimension (cont.)

- Number of channels in an instance is not fixed
  - 2 channels—improved safety or reliability
  - 3 channels—Triple Modular Redundancy
    - Enables most faults in one channel to be masked
  - 4 channels
    - Enables masking of arbitrary faults
    - Allows for one channel to be isolated and still guarantee TMR
Presentation Outline

- GUARDS Introduction
- Design Rationale
- The Generic Architecture
- **Interchannel Communication Network**
- Interchannel Error Processing and Fault Treatment
- Conclusions
- Questions / Discussion
Interchannel Communication Network

- Two essential functions
  - Provides a global clock to all channels
  - Allows channels to achieve interactive consistency on non-replicated data

- Architecture
  - ICN-manager for each channel and unidirectional serial links to interconnect the ICN-managers
Interchannel Communication Network

- Clock Synchronization
  - ICN-managers are fully connected nodes
  - Each node has a physical clock and computes a global logical clock time through and FT synchronization algorithm
  - Converging-Averaging solution in GUARDS
    - Susceptible to Byzantine clock in a 3 channel (or less) configuration
Interchannel Communication Network

- Interactive Consistency
  - Exchanging private data between channels and agreeing upon a common value in the presence of arbitrary faults
  - GUARDS uses a ZA algorithm for its interactive consistency protocol
Presentation Outline

- GUARDS Introduction
- Design Rationale
- The Generic Architecture
- Interchannel Communication Network
- Interchannel Error Processing and Fault Treatment
- Conclusions
- Questions / Discussion
Error recovery is achieved primarily by error compensation

- Bad state contains enough redundancy to transform its value to error-free
- Redundancy comes from active replication over C channels
- Error processing relies on N-modular redundancy to detect disagreeing channels and to mask errors
Interchannel Error Processing and Fault Treatment

- Diagnosis (Failure Detection)
  - Error reports collected at various locations in the system
  - Error filtering done using an $\alpha$-count mechanism (distributed)
    - Provides a mechanism to set tradeoff between safety and reliability (threshold)
    - Problems can occur if near-coincident faults occur on multiple channels
Interchannel Error Processing and Fault Treatment

- Once a channel has been diagnosed as bad...
  - It is isolated and reset
  - Self-test is carried out
    - If pass then channel is reintegrated
    - If fail then channel is switched off (and possibly repaired)
Interchannel Error Processing and Fault Treatment

- State Restoration
  - Channel reintegration
    - Resynchronize its clock
    - Resynchronize its state (SR process)
  - Minimum level of service needs to be provided during an SR process
  - GUARDS uses a distributed mechanism for performing SR
    - Each active channel is responsible for restoring part of the state in the recovering channel
Interchannel Error Processing and Fault Treatment

- Output Data Consolidation
  - Map replicated logical outputs of each channel onto actual physical outputs to the controlled process
  - Can vary greatly (application specific to the interfaces provided to the controlled process)
  - GUARDS provides no generic solution (except for networked and discrete outputs using voting)
Presentation Outline

- GUARDS Introduction
- Design Rationale
- The Generic Architecture
- Interchannel Communication Network
- Interchannel Error Processing and Fault Treatment
- Conclusions
- Questions / Discussion
Conclusions

- Interesting paper (I am experimental kind of guy...)
- This paper brought a lot of what we have learned into a practical application (nice to see how it fits together)
- I would like to know how far they have gotten (and how successful they have been)
- What about lessons learned?
Presentation Outline

- GUARDS Introduction
- Design Rationale
- The Generic Architecture
- Interchannel Communication Network
- Interchannel Error Processing and Fault Treatment
- Conclusions
- Questions / Discussion
Questions / Discussion

- Questions anyone??
- Thanks!!
Real-Time Scheduling in the GUARDS Architecture
GUARDS Elements

- Applicability to diverse embedded application domains
- Commercial off-the-shelf components
- Fault-tolerance
- Real-time computation and scheduling
- Development and validation tools
Outline of Presentation

- GUARDS Elements
- Real-Time Considerations
- Real-Time Models
- Scheduling
- Related Work
- Conclusions
Real-Time Considerations

- Key: Understanding timing properties of architecture.
  - intra-channel, inter-channel, and inter-channel network (ICN)

- Understand influence of configuration choices on timing. (replication, computational and scheduling model, etc.)

Figure 1: GUARDS Architecture
Real-Time Considerations

GUARDS Real-Time periodic transaction model

Three primary tasks

1) read sensor and send value to inter-channel network (ICN)
2) reads back from ICN manager values that have been received from all replicas, consolidates, processes, and produces result data
3) reads consolidated result from ICN manager and send to actuator
Outline of Presentation

- GUARDS Elements
- Real-Time Considerations
- Real-Time Models
- Scheduling
- Related Work
- Conclusions
Real-Time Models

- **Computational Model**
  - defines the form of concurrency and application program restrictions
    - Tasks, threads, asynchronous communication

- **Scheduling Model**
  - Defines the algorithm for allocating and ordering the system resources
    - Used with timing analysis method
GUARDS supports a range of real-time computational and scheduling models.

- Computational models
  - Time-triggered, event-triggered, and mixed
- Scheduling models
  - Cyclic, cooperative, preemptive

Models used with timing analysis to ensure real-time behavior of the application software is “timely”. 
Real-Time Models

GUARDS Models and Timing Analysis Techniques

<table>
<thead>
<tr>
<th></th>
<th>Computational model</th>
<th>Function release</th>
<th>Scheduling model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time-triggered</td>
<td>Periodic</td>
<td>Cyclic</td>
</tr>
<tr>
<td></td>
<td>Event-triggered</td>
<td>Sporadic</td>
<td>Co-operative</td>
</tr>
<tr>
<td></td>
<td>Mixed</td>
<td>Periodic &amp; sporadic</td>
<td>Pre-emptive</td>
</tr>
</tbody>
</table>

- **Cyclic**:
  - Timing analysis by construction
  - Response time analysis

- **Co-operative**:
  - Response time analysis

- **Pre-emptive**:
  - Rate monotonic analysis
  - Response time analysis
  - Response time analysis
Real-Time Models

- Cyclic and Cooperative scheduling models
  - Based on work of Wellings, Beus-Dukic, and Burns
  - Cyclic - traditional cyclic executive
  - Cooperative - application defined scheduler and prioritized tasks pass control between themselves

- Pre-emptive scheduling
  - Based on work of Audsley et al, and Leung and Whitehead
  - Most challenging of models to implement
  - Uses standard pre-emptive priority scheme
Outline of Presentation

- GUARDS Elements
- Real-Time Considerations
- Real-Time Models
- Scheduling
- Related Work
- Conclusions
Scheduling

Task based scheduling assumptions

1) Application consists of fixed set of tasks.
2) All tasks are periodic with known periods or sporadic with defined minimum inter-arrival times.
3) All tasks have fixed worst-case execution times.
4) All tasks are allocated unique priorities.
5) All tasks are independent of each other. (impractical)

Tasks are scheduled using period, deadline, and response time.

Schedulability test

1) analytical approach to predict worst-case response time.
2) compare with task deadlines to determine if set is schedulable.
Scheduling Considerations

- Impractical for all applications to be independent, therefore allow message passing and shared memory.
  - Raises synchronization, mutual exclusion, blocking, deadlock issues.
  - Use atomic, non-blocking access to data objects using algorithms described by Simpson (1990) and ceiling priority pre-emptive protocols.
    - Block high priority task at most once during execution
    - Prevents deadlocks
    - Provides mutex
    - See (Simpson, 1990; Sha et al. 1990; and Rajkumar et al. 1994) for more details.

- Highest priority task response time equals its own computation time. Other tasks, suffer interference from higher priority tasks.
- Communication delays
- Replicated tasks for fault-tolerance
Response-time analysis for replicated tasks

- Some factors affecting response time of replicated tasks
  - Data Consistency
    - Readers and writers may be periodic or sporadic
    - No precedence constraint between reader and writer replicas
  - Agreement on sensor inputs
    - Jitter, bound jitter and agreement with interactive consistency
  - Output consolidation and actuation timing
    - Voting, actuator latency

- Solution to data consistency problem is to ensure that same internal data is read by each replica
  - Method is to analyze schedules and replica tasks response times to provide timing information for \textit{timestamping} data. Provides a common basis to enforce replica determinism.
Scheduling

Data Timestamp and Replicas
- Worst case time that all replicas must have written data.
- Reader compares its release time with data timestamp.

\[
\begin{align*}
\text{READER Release Time} & \geq \text{Data Timestamp} & = & \text{OK to Read Data} \\
\text{READER Release Time} & < \text{Data Timestamp} & = & \text{Not OK. Read Earlier Timestamped Data}
\end{align*}
\]
Scheduling

- **Timestamp**
  - Worst case time that all replicas must have written data.

\[
\text{Timestamp} = \text{release time of writer (periodic, sporadic, interrupt, etc.)} + \text{worst-case release jitter} + \text{worst-case response time of writer replicas} + \varepsilon \text{ (maximum clock drift between replicas)}
\]
Scheduling

- GUARDS timestamping approach has the properties:
  - Replicated data items have same timestamp.
  - Replicated writers complete writing data by timestamp time
  - Replicated readers read the most recent data that can be guaranteed to be read by all of them

- GUARDS approach trades-off fewer agreement communications (reduced overhead) versus early error detection.

- Finally, a schedule based on response time analysis is generated for the replicated tasks.
Outline of Presentation

- GUARDS Elements
- Real-Time Considerations
- Real-Time Models
- Scheduling
- Related Work
- Conclusions
Related Work

- GUARDS built upon previous efforts
  - SIFT - Software Implemented Fault Tolerance
    - SRI
  - MAFT - Multicomputer Architecture for Fault Tolerance
    - Allied Signal
  - FTPP - Fault Tolerant Parallel Processors
    - Draper Laboratories
  - Delta-4
    - European Consortium
Outline of Presentation

- GUARDS Elements
- Real-Time Considerations
- Real-Time Models
- Scheduling
- Related Work
- Conclusions
Conclusions

- Generic Real-Time FT architecture based on COTS
- Response-time analysis used as the primary timing analysis technique to support scheduling
- Timestamping approach used for achieving replica determinism
- Supports wide range of computational and scheduling models