A Resilient Real-Time Traffic Control System: Software Behavior Monitoring and Adaptation

Axel Krings
Ahmed Serageldin
Ahmed Abdel-Rahim

This research was supported by grant DTFH61-10-P-00123 from the Federal Highway Administration - US DoT

Integrating Clarus data into RT-App.

- Challenges
  - The Engineering Challenge
  - The Security Challenge
  - The Real-time Challenge
  - The Survivability Challenge (includes all “illities”)

- Apply the newest technology to a survivability architecture
  - Design Methodology based on Design for Survivability
Project Architecture

- A system operating in an unbounded environment
- Inheriting all problems from such environment

The big picture

- The problem:
  *Should we connect the control network to the Internet?*
Utilizing local sensor data to do what?
### Highly Critical (Essential) Clarus Data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>essPrecipSituation</td>
<td>Describes the weather situation in terms of precipitation, integer values indicate situation</td>
</tr>
<tr>
<td>essPrecipYesNo</td>
<td>Indicates whether or not moisture is detected by the sensor: (1) precip; (2) noPrecip; (3) error</td>
</tr>
<tr>
<td>essPrecipRate</td>
<td>The rainfall, or water equivalent of snow, rate</td>
</tr>
<tr>
<td>essRoadwaySnowpackDepth</td>
<td>The current depth of packed snow on the roadway surface</td>
</tr>
<tr>
<td>essAirTemperature</td>
<td>The dry-bulb temperature; instantaneous</td>
</tr>
<tr>
<td>essVisibilitySituation</td>
<td>integer value, describes the travel environment in terms of visibility</td>
</tr>
<tr>
<td>essVisibility</td>
<td>Surface visibility (distance)</td>
</tr>
<tr>
<td>essSurfaceStatus</td>
<td>integer value, a value indicating the pavement surface status</td>
</tr>
</tbody>
</table>

### Prototype

![Prototype Image]
What could possibly go wrong?

- What assumptions should one place on a system?
  - Anything is possible!
  - and it will happen!

- Malicious act will occur sooner or later

- It is hard or impossible to predict the behavior of an attack

Unique Opportunity

- What is unique about this project?
  - The application domain is part of a Critical Infrastructure
  
  - The project is just small enough to demonstrate a survivability architecture

  - The code is relatively small
  - The execution is relatively deterministic
  - The run-time support is relatively mature
What is Survivability

- Closely related Terms
  - Intrusion Tolerance
  - Resilience

- Relationship to
  - Fault-tolerance
  - Security

Design for Survivability

- When Systems become too complex
  - Design by Integration of Survivability mechanisms
  - Build-in *not* add-on
  - Design for Survivability has surfaced in different contexts
Software Architecture

Overview

![Software Architecture Diagram]

Design Methodology

Measurement-based design and operation

![Design Methodology Diagram]
Our view of a System

- Different “machines”
  - Operations
  - Functions
  - Modules
- Epoch
  - defined by transitions

Formal Model of Sys. Arch.

- Measurement-based design and operation

![Diagram showing Operation Monitoring, Profiling Model, Dependency Model, SW Sensor Model, and Contingency Management System]
Profiling Model

Profiles

- Frequency Spectrum (...and more)
  - count invocations
  - probability of invocation
  - defined for an epoch
  - defined for operations, functions and modules
  - does not say anything about dependencies!
Profiles

- **Module Profile**
  - \( \mathbf{p} = <p_1, p_2, ..., p_{|M|}> \)
  - where \( p_i \) is probability that \( m_i \) is executing

Profiles

- **Observed Profile**
  - \( \hat{\mathbf{p}} = (\hat{p}_1, \hat{p}_2, ..., \hat{p}_{|M|}) \), where \( \hat{p}_i = c_i/n \) is the fraction of system activity due to invocations of module \( m_i \) and \( c_i \) is the count of invocations of \( m_i \).

  - \( \hat{\mathbf{p}}^k \) denotes the \( k^{th} \) observed module profile, observed over \( n \) epochs
Profiles and Certification

- System behavior
  - Analyze the observed profiles
  - What is the threshold for “normal” behavior?
  - How do we detect deviation from thresholds for “normal” executions?
  - Set the threshold of “normal” to “certified”
  - Looks like anomaly detection in IDS, or?

Interpretation of Certified Behavior

- If profiles are beyond the certified threshold, we simply have not seen such behavior before!
- Could be benign or malicious reasons

- What is our response?
  - We could simply not allow the operation to continue and go to fail-safe state
Profile Vector

- Vector $\hat{p} = (\hat{p}_1, \hat{p}_2, ..., \hat{p}_{|M|})$
- notice log scale

![Graph showing typical observed profile of 4 costates (module IDs and frequencies on the axis)](image)

Fig. 5. Typical observed profile of 4 costates (module IDs and frequencies on the axis)

Profile Vector & Scalar

- Observe $h$ sequences of $n$ epochs
- Define a centroid $\overline{p} = (\overline{p}_1, \overline{p}_2, ..., \overline{p}_{|M|})$, where

\[
\overline{p}_i = \frac{1}{h} \sum_{j=1}^{h} \hat{p}_i^j
\]

and the distance of $\hat{p}^k$ from centroid $\overline{p}$ is given by

\[
d_k = \sum_{i=1}^{n} (\overline{p}_i - \hat{p}_i^k)^2
\]
Multitasking Model

Rabbit runs Dynamic C which support costatements

In a multitasking environment, more than one task (each representing a sequence of operations) can appear to execute in parallel. In reality, a single processor can only execute one instruction at a time. If an application has multiple tasks to perform, multitasking software can usually take advantage of natural delays in each task to increase the overall performance of the system. Each task can do some of its work while the other tasks are waiting for an event, or for something to do. In this way, the tasks execute almost in parallel.

There are two types of multitasking available for developing applications in Dynamic C: preemptive and cooperative. In a cooperative multitasking environment, each well-behaved task voluntarily gives up control when it is waiting, allowing other tasks to execute. Dynamic C has language extensions, costatements and cofunctions, to support cooperative multitasking.

Preemptive multitasking is supported by the slice statement, which allows a computation to be divided into small slices of a few milliseconds each, and by the µC/OS-II real-time kernel.

5.1 Cooperative Multitasking

In the absence of a preemptive multitasking kernel or operating system, a programmer given a real-time programming problem that involves running separate tasks on different time scales will often come up with a solution that can be described as a big loop driving state machines.

Figure 5.1  Big Loop

(Figure from Dynamic C Users Manual)

Figure 5.3 shows the execution thread through a costatement when a waitfor evaluates to true.

Figure 5.3  Execution thread when waitfor evaluates to true

The yield statement makes an unconditional exit from a costatement or a cofunction. Execution continues at the statement following yield the next time the costatement or cofunction is encountered by the execution thread.

Figure 5.4  Execution thread with yield statement

Dynamic C, costates and yield
(Figure from Dynamic C Users Manual)
Profiles considering costates

Definitions based on costate \( \alpha \):

\[
\hat{p}[\alpha], \hat{p}^k[\alpha], \overline{p}[\alpha] \text{ and } d_k[\alpha]
\]
**Multitasking Model**

- One knows which costate is executing
- Profiles of costates are not polluted with activity from other costates
- Result is lower degree of non-determinism of execution

**Certified Behavior**

- The distance of the observed costate profiles $\hat{p}^k[\alpha]$ from $p[\alpha]$ can be used so that departure beyond it indicates non-certified behavior of costate $\alpha$. Two threshold vectors:

$$\epsilon^{max}[\alpha] = (\epsilon_1^{max}[\alpha], ..., \epsilon_{|M|}^{max}[\alpha])$$  \hspace{1cm} (3)

$$\epsilon^{min}[\alpha] = (\epsilon_1^{min}[\alpha], ..., \epsilon_{|M|}^{min}[\alpha])$$  \hspace{1cm} (4)

where $\epsilon_i^{max}[\alpha]$ and $\epsilon_i^{max}[\alpha]$ are the upper and lower threshold values of $m_i$, representing a dual-bound threshold.
Certified Behavior

Every observed profile that is in the region between the two vectors is assumed nominal. Thus we certify a profile \( \hat{\mathbf{p}}^k[\alpha] \) to be a *nominal profile* if

\[
\epsilon_{\text{min}}[\alpha] \leq \hat{\mathbf{p}}^k[\alpha] \leq \epsilon_{\text{max}}[\alpha]
\]
i.e., if \( \epsilon_i^{\text{min}}[\alpha] \leq \hat{\mathbf{p}}_i^k[\alpha] \leq \epsilon_i^{\text{max}}[\alpha] \) for every \( 1 \leq i \leq |M| \).
Centroid

Synchronized Profiling

So far we assumed that there is only one single behavior. However, there could be multiple.

Considering $h$ sequences of $n$ epochs each, we define a centroid of sets $\mathbf{P} = (\overline{P_1}, \overline{P_2}, ..., \overline{P_{|M|}})$, where

$$ \overline{P_r} = \overline{P_r} \cup p_i, \quad 1 \leq r \leq |M| \quad p_i = \frac{1}{h} \sum_{j=1}^{h} \hat{p}_i^j $$

(2)

for each behavior $i$. Thus $\overline{P}$ is a $|M|$-dimensional structure of sets, and again using the above financial metaphor, each element represents the “$h$-day moving average” of a specific set of stocks (module), where a day is measured as $n$ epochs, and again we want to track the past in order to establish “nominal”, i.e., expected, behavior from a set of behaviors.
Dependency-based Model

Inter-dependencies

- Relationship between Operations, Functionalities, and Modules

Mappings in \((O \times F \times M)\)
Intra-dependencies

- Relationship within Operations, Functionalities, and Modules

\[ G^O = (O, \prec^O) \]
\[ G^F = (F, \prec^F) \]
\[ G^M = (M, \prec^M) \]

Intra-dependencies

- In our current system we simplify to
Operations & Costates

**Figure 3: Costates and Operations**

1. Get Clarus data
2. Receive data from LCS
3. Receive data from Clarus
4. Analyze Clarus data
5. Adjust controller
6. Monitor analysis
7. Monitor adaptive reconfiguration
8. Time synchronization
9. Support routines

Sensor-based Model
Sensor-based Model

- Not every behavior can be extracted from profiles or dependencies.

- Specific data sensors are needed to observe specific data values or trigger exceptions.

Exception Triggers

- Exception trigger array

  - identify and profile exceptions, e.g., file does not exist, specific sensor data is not longer available.

  - any error condition can be viewed as an exception trigger
Data Sensors

- Observation of specific numeric values for analysis
- Example: the adjustment to the yellow timing
- What happens when someone changes to yellow time to zero? Is that possible?
System Module State Machine

System Operations State Machine

Operations:
0 : Initialize Program
1 : Runtime Timing Module
2 : Get Weather Data
3 : Update Controller
Application Control Costatement

![Flowchart of Application Control Costatement]

Exception Triggers

![Bar chart showing exception triggers]

- precipType: 20
- PrecipYesNo: 47
- PrecipRate: 0
- Visibility: 47
- SurfaceStatus: 5
- SurfaceTemperature: 0
Yellow adjustment in % over winter months
Profiles of key modules and two nominal behaviors

Profiles of module m23 with behavior set size equal 1
Profiles of module m23 with behavior set size equal 2

Current Status

- Contingency Management Description:
  
  A. Serageldin, A. Krings, and A. Abdel-Rahim, “A Survivable Critical Infrastructure Control Application”, 8th Annual Cyber Security and Information Intelligence Research Workshop, Oct. 30 Sept. 2 2012, ORNL
  

- Gaining Experience: prototype started running 24/7
  
  - Mature in setting thresholds.
  
  - Dealing with realities of Internet access in Intersection
Conclusions

- Prototype has been running over 1 year
  - uses real-time weather data to modify traffic signal timing within safety standard

Utilization of Design for Survivability

- Off-nominal executions detected (dual-bound thresholds)
- Violation of dependencies detected
- Contingency Management to Recover from anomalies