A Prototype for a Real-Time Weather Responsive System

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
Clarus...

- Utilizing local sensor data to do what?

![Image of Clarus System]
## 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
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

Liu & Trivedi 2004
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
A Measurement-based Design and Evaluation Methodology for Embedded Control Systems

A. Krings, V. Balogun, S. Alshomrani, A. Abdel-Rahim, M. Dixon

Computer Science Department
Civil Engineering
University of Idaho
University of Idaho
Moscow, ID 83844-1010
Moscow, ID 83844-1022

ABSTRACT

A measurement-based design and evaluation methodology for embedded control systems is presented that features better control of non-deterministic executions through reduction of non-determinism, certified behavior of executions, real-time monitoring of operations, functionalities, and modules, which allows adaptation to non-certified behavior.

Using principles of Design for Survivability, the software system is broken down into costatements with low degree of non-determinism in their executions, thereby allowing a significant increase in the accuracy of runtime profiling. This in turn results in better detection of deviation from certified executions. The formal model to achieve these goals is introduced and the effectiveness is demonstrated in a real control application.

1. INTRODUCTION

As the components controlling our critical infrastructures are increasingly relying on networked computing systems, this connectivity also becomes the focal point for security and survivability considerations. It is thus more important than ever to include security and survivability starting at the specification and design stage, rather than in an addition fashion. Design for Survivability incorporates this philosophy.

1.1 Application

The application for which the design and evaluation methodology was developed is a traffic control application, however, the general principles described in the paper are application-independent and hold for a wide range of control applications. Our application considers traffic signals as part of a networked Intelligent Transportation System. A database server called Clarus which resulted from the Clarus Initial Program collects real-time sensor data about weather conditions of a large number of sensor stations maintained by most states in the US. The data from the Clarus server is now accessed and used by traffic controllers in order to adjust the timing of traffic lights to compensate for the impact of adverse local weather conditions in order to improve safety and thus reduce accidents.

The system consists of the traffic controller everything found in a modern traffic light-controlled intersection and a Rabbit-based system referred to as “Rabbit” in the rest of the paper that accesses the Clarus system or a Local Clarus Server which is a local mirror used for redundancy and scalability purposes via the Internet. The software architecture is depicted in Figure 1, where shaded areas refer to external hardware interfaces. The system connects to the LCS or Clarus using the network interface to the Internet. In regular intervals, emgmk local sensor data is typically updated every u to pu minutes, the Clarus data is read and converted by the Rabbit, the desired sensor data is extracted, and specific algorithms are used to compute the yellow timing from the critical extracted parameters. The traffic controller is then updated. All this is monitored by the Operation Monitoring and Configuration Management System.

Figure 1: Software Architecture Overview

1.2 Contributions
Design Methodology

- Measurement-based design and operation
Our view of a System

- Different “machines”
  - Operations
  - Functions
  - Modules

- Epoch
  - defined by transitions
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

\[ \mathbf{p} = \langle p_1, p_2, \ldots, p_{|M|} \rangle \]

where \( p_i \) is probability that \( m_i \) is executing
Profiles

[Red square] Observed Profile

\( \hat{p} = \left( \hat{p}_1, \hat{p}_2, \ldots, \hat{p}_{|M|} \right) \), 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 \( m_i \).

\( \hat{p}^k \) 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?
Profiles and Certification

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

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 $\overline{p} = (\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_{|M|})$, 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

$$d_k = \sum_{i=1}^{n} (\overline{p}_i - \hat{p}_i^k)^2$$
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
Dynamic C, *costates* and *yield*
(Figure from Dynamic C Users Manual)

---

Figure 5.3 shows the execution thread through a costatement when a *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
The costatement is initially active and will automatically execute the first time it is encountered in the execution thread. The costatement becomes inactive after it completes (or aborts). The costatement can be made inactive by CoPause().

If state is absent, a named costatement is initialized in a paused init_on condition. This means that the costatement will not execute until CoBegin() or CoResume() is executed. It will then execute once and become inactive again. Unnamed costatements are always_on. You cannot specify init_on without specifying a costatement name.

5.3.3 Control Statements

This section describes the control statements identified by the keywords: waitfor, yield and abort.

waitfor (expression);

The keyword waitfor indicates a special waitfor statement and not a function call. Each time waitfor is executed, expression is evaluated. If true (non-zero), execution proceeds to the next statement; otherwise a jump is made to the closing brace of the costatement or cofunction, with the statement pointer continuing to point to the waitfor statement. Any valid C function that returns a value can be used in a waitfor statement.

Figure 5.2 shows the execution thread through a costatement when a waitfor evaluates to false. The diagram on the left side shows which statements are executed the first time through the costatement. The diagram on the right shows that when the execution thread again reaches the costatement the only statement executed is the waitfor. As long as the waitfor continues to evaluate to false, it will be the only statement executed within the costatement.

Figure 5.2  Execution thread when waitfor evaluates to false

yield

The yield statement makes an unconditional exit from a costatement or 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

Execution thread when waitfor evaluates to true
Profile Scalar

- Definitions based on costate $\alpha$:

\[
\hat{p}[\alpha], \hat{p}^k[\alpha], \bar{p}[\alpha] \text{ and } d_k[\alpha]
\]
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

![Graph showing profiles of 4 costates](image)

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

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

\[
\epsilon^{max}[\alpha] = (\epsilon_1^{max}[\alpha], ..., \epsilon_M^{max}[\alpha])
\]

(3)

\[
\epsilon^{min}[\alpha] = (\epsilon_1^{min}[\alpha], ..., \epsilon_M^{min}[\alpha])
\]

(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

\[
\epsilon^{\min}[\alpha] \leq \hat{\mathbf{p}}^k[\alpha] \leq \epsilon^{\max}[\alpha]
\]

(5)

i.e., if \( \epsilon^{\min}_i[\alpha] \leq \hat{p}^k_i[\alpha] \leq \epsilon^{\max}_i[\alpha] \) for every \( 1 \leq i \leq |M| \).
A prototype has been built based on a Rabbit 5700 running Clarus version 10.5.4, which has been instrumented to monitor and 4) Utilities. As expected, costate 2 with its centroid indicates the module ID and the y-axis shows the frequencies below a subset of only 56 relevant modules was used.

Fig. 5 shows an actual observed profile which encapsulates the data transmission from the Clarus systems. The x-axis in logarithmic scale. The figure depicts profile frequencies are shown in Fig. 6. The average constitutes the dual-bound threshold vectors. For the figures shown the centroid frequencies count is shown as log-scale. For example, if we look tests.

Monitoring and 4) Utilities. As expected, costate 2 with its centroid indicates the module ID and the y-axis shows the frequencies below a subset of only 56 relevant modules was used.

Fig. 5 shows an actual observed profile which encapsulates the data transmission from the Clarus systems. The x-axis in logarithmic scale. The figure depicts profile frequencies are shown in Fig. 6. The average constitutes the dual-bound threshold vectors. For the figures shown the centroid frequencies count is shown as log-scale. For example, if we look tests.

Monitoring and 4) Utilities. As expected, costate 2 with its centroid indicates the module ID and the y-axis shows the frequencies below a subset of only 56 relevant modules was used.

Fig. 5 shows an actual observed profile which encapsulates the data transmission from the Clarus systems. The x-axis in logarithmic scale. The figure depicts profile frequencies are shown in Fig. 6. The average constitutes the dual-bound threshold vectors. For the figures shown the centroid frequencies count is shown as log-scale. For example, if we look tests.

Monitoring and 4) Utilities. As expected, costate 2 with its centroid indicates the module ID and the y-axis shows the frequencies below a subset of only 56 relevant modules was used.

Fig. 5 shows an actual observed profile which encapsulates the data transmission from the Clarus systems. The x-axis in logarithmic scale. The figure depicts profile frequencies are shown in Fig. 6. The average constitutes the dual-bound threshold vectors. For the figures shown the centroid frequencies count is shown as log-scale. For example, if we look tests.

Monitoring and 4) Utilities. As expected, costate 2 with its centroid indicates the module ID and the y-axis shows the frequencies below a subset of only 56 relevant modules was used.
Centroid

The figure depicts the profile of centroid frequencies with module IDs and frequencies on the x-axis and y-axis, respectively. The module IDs range from 30 to 56. The graph shows the distribution of frequencies for each module.

As indicated before, threshold vectors are generated by the system. As expected, costate 2 with its specific values indicated arose for costate 2. For this purpose, a dual-bound threshold function was established, which defines the nominal space as a subset of only 56 relevant modules. The latter is established by checking if the observed profiles are within a dual-bound threshold space that defines the nominal execution. The system can take real-time weather dependent data and modify traffic signal timing within safety standard.

Current efforts are to test the effectiveness of the system in a learning mode over time, thereby observing the system in a learning mode over time, thereby letting observed executions affect what is considered a normal operation.

The centroid and dual-bound threshold vectors (module IDs and frequencies on the axis) are shown in Fig. 6. The average constitutes the module ID and the y-axis shows the frequencies in logarithmic scale. The figure depicts profile profiles for costate 2. For this purpose, the centroid appears rather tight. However, the frequency count is shown as log-scale. For example, if we look at the execution was considered nominal.

As indicated before, threshold vectors are generated by the system. As expected, costate 2 with its specific values indicated arose. For this purpose, a dual-bound threshold function was established, which defines the nominal space as a subset of only 56 relevant modules. The latter is established by checking if the observed profiles are within a dual-bound threshold space that defines the nominal execution. The system can take real-time weather dependent data and modify traffic signal timing within safety standard.

Current efforts are to test the effectiveness of the system in a learning mode over time, thereby observing the system in a learning mode over time, thereby letting observed executions affect what is considered a normal operation.

As indicated before, threshold vectors are generated by the system. As expected, costate 2 with its specific values indicated arose. For this purpose, a dual-bound threshold function was established, which defines the nominal space as a subset of only 56 relevant modules. The latter is established by checking if the observed profiles are within a dual-bound threshold space that defines the nominal execution. The system can take real-time weather dependent data and modify traffic signal timing within safety standard.
Dependencies

- Relationship between Operations, Functionalities, and Modules

Mappings in \((O \times F \times M)\)
Dependencies

- Relationship within and between Operations, Functionalities, and Modules

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

3.2 Monitor Analysis

3.3 Receive data from Clarius

3.4 Receive data from LCS

3.5 Analyze Clarius data

3.6 Adjust controller

3.7 Monitor adaptive reconfiguration

3.8 Time synchronization

3.9 Support routines

Figure 3: Costates and Operations
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

- What is more effective: *vector* or *scalar* profiling/certification?
- Mature in setting thresholds. Need more experimentation to get more experience
- Looking forward to include 2012 Winter Data in real time
- Dealing with realities of Internet access in Intersection
Conclusions

- Prototype is up and running
  - 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

- Future potential: apply concept to other applications, e.g., Dedicated Short Range Communication (DSRC) apps
Questions
Design for Analyzability

- Not a new concept
- e.g., Series-Parallel RBD
  - Not all systems are Series-Parallel!
Fault Models: The world in which we live/operate

- All Faults
  - Malicious
    - Asymmetric
      - Transmissive Asymmetric
      - Strictly Omissive Asymmetric
    - Symmetric
      - Omissive Symmetric
      - Transmissive Symmetric
  - Benign
    - Benign