AMF: An assumptions management framework using AADL

Ajay Tirumala
tirumala@uiuc.edu
Outline

• Background and motivation
• Case studies
• Assumptions classification
• AMF Design
• Handling a dynamic set of components [new]
Background: Building real-time systems

Critical real-time systems are built using a) COTS components b) components from diverse development groups

Simple Car Control Systems

Control Systems in satellites
COTS Adoption and Concurrent Development

**Significant Benefits**

- Distributed expertise
- Uniform process for building different types of systems
- Faster turn around time
- Software / Hardware Component reuse.

**Problems (Real-time domain)**

- Gives rise to black-box software interfaces.
  - Acoustic sensor returns a 16-bit value when data is available.
  - `event dataReady(uint_16t data);`
- Software components make assumptions which are not reflected in the software interface
  - E.g: Max sensing delay < 10ms, sensing jitter < 1ms, acoustic intensity units = {Db}, saturation value for readings <= 1023, granularity of readings <= 0.1 Db.
Real-world example (Avionics)

• Ariane 5 reused some software developed for Ariane 4
• Ariane 4 made the following assumption
  – “The horizontal velocity component will not overflow a 16-bit variable”
• This was true for Ariane 4, but not for Ariane 5.
  – This triggered self-destruction roughly 40 seconds after the launch

Assumption was known before hand but was not recorded in a verifiable (machine-checkable) format
Real-world example (Automotive sector)

- Fatality due to a software controlled airbag deactivation.
- In presence of child seat
  - Airbag was deactivated by the primary controller.
- Under certain combination of environmental conditions
  - Primary controller gives the control to a simpler backup controller
- Backup controller has simple logic
  - Deploy all airbags on impact
- On impact, deploys airbag and causes fatality
  - Unaware of the environmental factor - child-seat presence OR
  - Has the capability of only deploying all or nothing.

Again, the environmental assumptions were known beforehand, but there was a disconnect between primary and backup airbag system design.
Outline

• Background and motivation
• Case studies
• Assumptions classification
• AMF Design
• Handling a dynamic set of components
Case-study hypotheses

- **Hypothesis I**: “Significant portion (>= 50%) of software defects in released products in RT-systems are related to inconsistent assumptions”
  - This hypothesis is related to the necessity of the assumption management framework.

- **Hypothesis II**: “Majority of the assumptions that result in defects can be *encoded* in a machine checkable format”
  - Thus can warn of defects in advance or prevent mismatched assumptions in end-products.
  - This hypothesis is related to the feasibility of such a framework.
Projects selection for case studies

• Criteria
  – Should be representative projects in the domain in which we intend to generalize the results
    • Should have close interactions with the operating environment
  – Should provide adequate data for validating the hypothesis.

• Projects selected
  – TinyOS [for Hypothesis 1&2]
    • an operating system for embedded devices
    • public repository of defect list available
  – Iperf [for Hypothesis 1&2]
    • an internet end-to-end bandwidth measurement tool
    • public repository of user-problems faced [mailing list available]
  – Inverted Pendulum Control System [for Hypothesis 2]
    • A system that balances an inverted pendulum using feedback control
Case study results for Hypothesis 1

<table>
<thead>
<tr>
<th></th>
<th>Iperf</th>
<th>TinyOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total # of defects</td>
<td>38</td>
<td>24</td>
</tr>
<tr>
<td>considered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related to inconsistent</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>assumptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related but not entirely</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>due to inconsistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>assumptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithmic errors</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>(loop – off by 1, bit-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shifts, bad state on</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exit, memory management)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Defect classification: Iperf and TinyOS

- **Not related to inconsistent assumptions**
- **Related but not entirely due to inconsistent assumptions**
- **Due to inconsistent assumptions**
Outline

• Background and motivation
• Case studies
• Assumptions classification
• AMF Design
• Handling a dynamic set of components
[Traditional] Software interface

- Traditional programming language interfaces (E.g. C, Java, CORBA, nesC etc)
- Used to exchange actual data
- Mostly concerns the syntax of data exchange.
  - E.g. Interface has one 16 bit unsigned integer.

Software Interface
Data acq module: `event dataReady(uint16_t dataval);`
Sensor: `signal dataReady(200);`
Assumption Set

• Used to encode the assumptions and guarantees made by the software component
  – These assumptions and guarantees are not a part of the data exchanged using software interfaces

• Assumptions may or may not pertain to parameters in the software interface

• E.g:
  – Units of the intensity data (assumed to be decibel)
  – Data collector expects maximum sensing delay to be < 50 ms

<table>
<thead>
<tr>
<th>Acoustic intensity data units</th>
<th>= {Decibels}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum sensing delay</td>
<td>&lt;= 50ms</td>
</tr>
<tr>
<td>Max error in readings</td>
<td>&lt;= 10%</td>
</tr>
<tr>
<td>Max sensing jitter</td>
<td>&lt;= 10 ms</td>
</tr>
<tr>
<td>Jitter specification units</td>
<td>= {ms}</td>
</tr>
<tr>
<td>Saturation value for readings</td>
<td>&lt;= 100</td>
</tr>
<tr>
<td>Granularity of readings</td>
<td>&lt;= 0.1</td>
</tr>
<tr>
<td>Valid reading constraints</td>
<td>= func_valid()</td>
</tr>
</tbody>
</table>

**DATA COLLECTOR ASSUMPTIONS**

<table>
<thead>
<tr>
<th>Acoustic intensity data units</th>
<th>= Decibels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum sensing delay</td>
<td>= 10ms</td>
</tr>
<tr>
<td>Max error in readings</td>
<td>= 10%</td>
</tr>
<tr>
<td>Max sensing jitter</td>
<td>= 4 ms</td>
</tr>
<tr>
<td>Jitter specification units</td>
<td>= ms</td>
</tr>
<tr>
<td>Saturation value for readings</td>
<td>= 100 (Db)</td>
</tr>
<tr>
<td>Granularity of readings</td>
<td>= 0.1 (Db)</td>
</tr>
<tr>
<td>Valid reading constraints</td>
<td>= func_valid()</td>
</tr>
</tbody>
</table>

**SENSOR GUARANTEES**

**Definition:** Encodes sufficient information about the environment, and the assumptions made by the component that are not a part of the software interface. Further, these assumptions are encoded in a machine-checkable format.
Classification of assumptions: Dimension I

**Time-frame for assumption changes**

- **Static assumptions**
  - assumptions that change only when the software changes
- **System configuration assumptions.**
  - assumptions that change only when configuration of the system changes or the hardware changes
- **Dynamic assumptions.**
  - assumptions that may change along the mission or run of the system

**Necessity for this classification**

- **Cost** of checking assumptions
  - Not all assumptions need to be checked at all times
- **Manageability**
- **Enabling different tools** for different types of checks

<table>
<thead>
<tr>
<th>Acoustic intensity data units</th>
<th>= {Decibels}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum sensing delay</td>
<td>&lt; 50ms</td>
</tr>
<tr>
<td>Max error in readings</td>
<td>&lt;= 10%</td>
</tr>
<tr>
<td>Max sensing jitter</td>
<td>&lt;= 10ms</td>
</tr>
<tr>
<td>Jitter specification units</td>
<td>= {ms}</td>
</tr>
<tr>
<td>Saturation value for readings</td>
<td>&lt;= 100</td>
</tr>
<tr>
<td>Granularity of readings</td>
<td>&lt;= 0.1</td>
</tr>
<tr>
<td>Valid reading constraints</td>
<td>: func_valid()</td>
</tr>
</tbody>
</table>

DATA COLLECTOR ASSUMPTIONS
Classification of assumptions: Dimension II

**Criticality of assumptions**

- Each component has a core-functionality
- **Critical assumptions (Class A)**
  - Violation of some assumptions cause the core component functionality to be compromised
- **Non-critical assumptions. (Class B-E)**
  - Violation of certain assumptions may cause performance degradation, core functionality holds
  - Graded by the user (Class B-E)

**Necessity for this classification**

- Certifying functionality correctness [on assumption violation]
- Can be used by dependency management tools
- Service gradations

<table>
<thead>
<tr>
<th>Acoustic intensity data units</th>
<th>= {Decibels}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum sensing delay</td>
<td>&lt; 50ms</td>
</tr>
<tr>
<td>Max error in readings</td>
<td>&lt;= 10%</td>
</tr>
<tr>
<td>Max sensing jitter</td>
<td>&lt;= 10ms</td>
</tr>
<tr>
<td>Jitter specification units</td>
<td>= {ms}</td>
</tr>
<tr>
<td>Saturation value for readings</td>
<td>&lt;= 100</td>
</tr>
<tr>
<td>Granularity of readings</td>
<td>&lt;= 0.1</td>
</tr>
<tr>
<td>Valid reading constraints</td>
<td>: func_valid()</td>
</tr>
</tbody>
</table>

DATA COLLECTOR ASSUMPTIONS
Composition of assumptions

- In the acoustic subsystem
  - Each component has a property *maximum delay*

- The acoustic sub-system is a part of the larger sub-system
  - E.g.: Vehicle classification system

- The integrator for the vehicle classification system
  - Is not interested in individual guarantees like that of data analyzer, acoustic sensor, etc.
  - Is interested in calculating the next *Max Delay* of the acoustic subsystem.

\[
\text{Max Delay} = \sum \text{Max Delay}_i
\]
Composition of assumptions - II

• Data collector makes quite a few assumptions that are satisfied by the sensor
  – These assumptions need not be exposed as a part of the larger sub-system.

Dimension III: Scope of assumptions

• **Private:** Any assumption (or guarantee) that has matching guarantee (or assn) and need not be exposed as a part of the larger sub-component.
  – *max_delay_1*, between the acoustic sensor and data collector

• **Public:** Any assumption (or guar) that needs to be exposed as a part of the sub-component.
  – *total_max_delay*: Sum of delays of components

• **Important:**
  – Public and private assumptions are only for easier manageability
  – Checks may still need to be performed for functionality and/or performance conformance on relevant changes.
Outline

• Background and motivation
• Case studies
• Assumptions classification
• AMF Design
• Handling a dynamic set of components
Assumption specification within AADL

```
system DataCollector
features -- ... Filter system features
annex assumptions {**
  componentAssumptions name= DataCollector {
    -- Dependent component DataCollector
    about Sensor {
      -- Assumptions
      assumes validReadings (int validReadingsID, int startLatency, int frequency) {
        validReadingsIDs > startLatency/frequency
      }
      {Criticality=CRITICAL_LEVEL_5}
      {ChangeInterval=STATIC}
      {Impact = VALUE_ERROR};

      -- Guarantees
      guarantees minFrequency {
        int frequency = 100;
      };;
    }
    about OtherComponents {
      ...
    };
  **}
```

Sensor

Makes assumption `validReadings`

Data Collector

Provides guarantee `minFrequency`
Model (EMF) based input for AMF
Composition of assumptions

- Complexity: $O(n)$ : $n$ is the number of assumptions for a complete composition.
- Can include domain specific composition rules.
- Use markers to point to the assumptions themselves.
Java code generation for assumptions

- Current implementation of the parser generates Java code to check the assumptions.
  - Any assumption that can be specified with predicate logic can be checked easily.
- Can interface with tools like JavaMOP for assumptions in temporal logic
  - JavaMOP generates Java code for testing temporal logic predicates.
XML persistent objects as a hierarchical database

- Example of simple search queries.
  - Check status of all *critical* assumptions.
    - //assumptions/@criticality=‘CriticalLevel_5’
  - Get all assumptions for a particular component ‘x’
    - //assumptionSet/@dependentComponentName=‘x’.

```
//tinyos_req/@criticality="critical"
Assn x_1
Assn x_2,...
Assn x_n

//tinyos_req/@changeInterval="system_config"
Assn y_1...
Assn y_n
```
Assumptions validity

- Checks that assumptions can be made only between compatible AADL components
  - E.g.: *Data* component cannot make assumptions on the *Memory* component.
- Of course, flags all assumption violations.
Outline

• Background and motivation
• Case studies
• Assumptions classification
• AMF Design
• Handling a dynamic set of components
System with dynamic set of components

• We may have a system with components exporting libraries and services
  – The set of components that use these services or libraries are not known in advance – dynamic set of components

• Traditional RT systems abstract dynamism with admission control.
  – This concerns resource and timing requirements of the task
  – AMF has a similar admissions control based on validity of assumptions

• AMF allows specification of anonymous set of assumptions and guarantees.
  – Includes a set of rules to compose anonymous set of assumptions and guarantees
  – Basis for admission control
Car control testbed
Assumptions in dynamic architectures

- Components like Kernel which do not have the set of dependent components known before-hand
  - Kernel’s assumptions need to be satisfied by every component that uses the kernel
  - It can also provide a reflexive set of guarantees which other components can use, e.g.: OS it runs on.

- Basic composition rules
  - Reflexive assumptions need to be satisfied by every component that uses this component’s service
  - Reflexive guarantee can optionally be satisfied by a dependent component

```java
class Rectangle {
    language=JAVA;
    import edu.*;

    about _any_ {
        assumes inclinationReference (String inclinationReference) {
            (inclinationReference.equals(DEGREES))
        };
        Criticality=CRITICAL_LEVEL_5
        ChangeInterval=STATIC
        Impact=ValueError;
    };
}
```
Life-cycle for validating dynamic architectures

1. System’s state \{C_1, C_2, \ldots, C_n\}.
   - All/requisite assumptions validated.
   - \textit{Set the allowable cost for validating assumptions.}

2. Services of \(C_{n+1}\) are needed. \(C_{n+1}\) becomes available.

3. Find the set of assumptions that need to be validated – Find the cost.
   - Include impact of not satisfying an assumption.

4. All assumptions cannot be validated within the cost set.
   - Validation Fails.
   - Check if smaller set of assumptions can be satisfied with acceptable impact [LP/IP problem]
   - Handle impact of not including \(C_{n+1}\)

5. All assumptions can be validated within the cost set.
   - Validation Passes.
   - Update system state.
   - Update allowable cost for validating assumptions
   - \textbf{Perform Validation}

6. Update system state.
   - Update allowable cost for validating assumptions
Summary

• Invalid assumptions are root-cause of many defects in projects involving embedded systems
• AMF provides
  – A classification for assumption for easier manageability
  – A precise grammar and vocabulary for specifying assumptions
  – Automatic validation of a relevant subset of assumptions on changes or during system evolution
  – Composition of assumptions
  – Tight integration with AADL to enable assumptions specification during requirements and system design phase
• Status
  – Allows specification of assumptions in predicate logic
  – Allows automatic validation of static and system configuration assumptions.
  – Allows user defined library routines (java) to be used to specify guarantees that can be obtained only during run-time
  – Allows assumptions specifications for dynamic architectures
  – Impact and criticality fields can be used by dependency management tools.
• Current work
  – Temporal logic assertions ↔ Predicates for dynamic assumptions
  – Cost of checking dynamic assumptions
Thank you

E-mail: tirumala@uiuc.edu
Common annex utilities

• AMF allows predicate logic specification of assumptions in Java™ expression syntax
  – Presumably usable for many other annexes

• Search for annex objects that are defined in terms of EMF objects
  – Currently AMF specific. Can be made generic