Refinement of Safety-Related Hazards into Verifiable Code Assertions

Ken Wong, University of British Columbia

Jeff Joyce, Raytheon Systems Canada Ltd.

Safety Verification

- System safety engineering
  - Hazard identification, analysis, and mitigation
- Safety verification of the source code
  - Providing evidence of hazard mitigation
- Safety Verification Conditions (SVCs)
  - Conditions mitigating or eliminating hazard

Tool-Based Code Verification

- E.g., SPARK Examiner
- **Input**: Annotated Program
  - *Program code* (SPARK Ada subset)
  - *Pre- and post-conditions* (SPARK annotations)
- **Output**: Verification Conditions (VCs)
  - Verify with help of SPARK proof tools

Safety Code Verification

System Hazard

Code Assertions

?

Semantic Gap

System Hazard

- end-user terminology
  - temperature, vessel
- abstract relationships
  - invalid temperature
- temporal relationships
  - “at some time T ...”

Source Code

- programmer’s lexicon
  - abstract classes
- concrete relationships
  - arithmetic, logical
- causal relationships
  - program execution

Transform Hazard

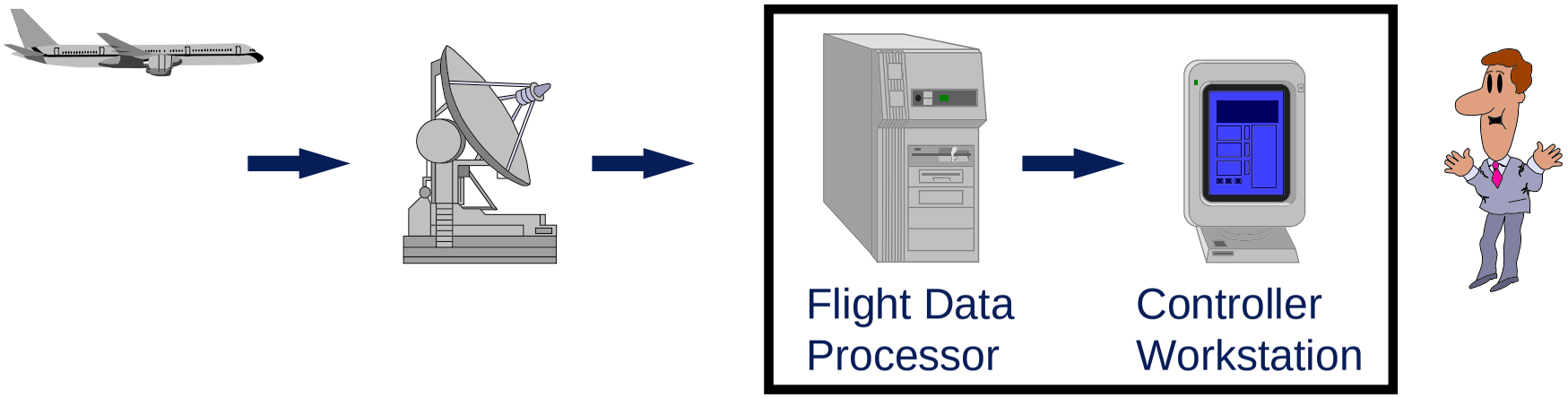
① **Hazard** into system-level SVCs

② System-level SVCs into code-level SVCs

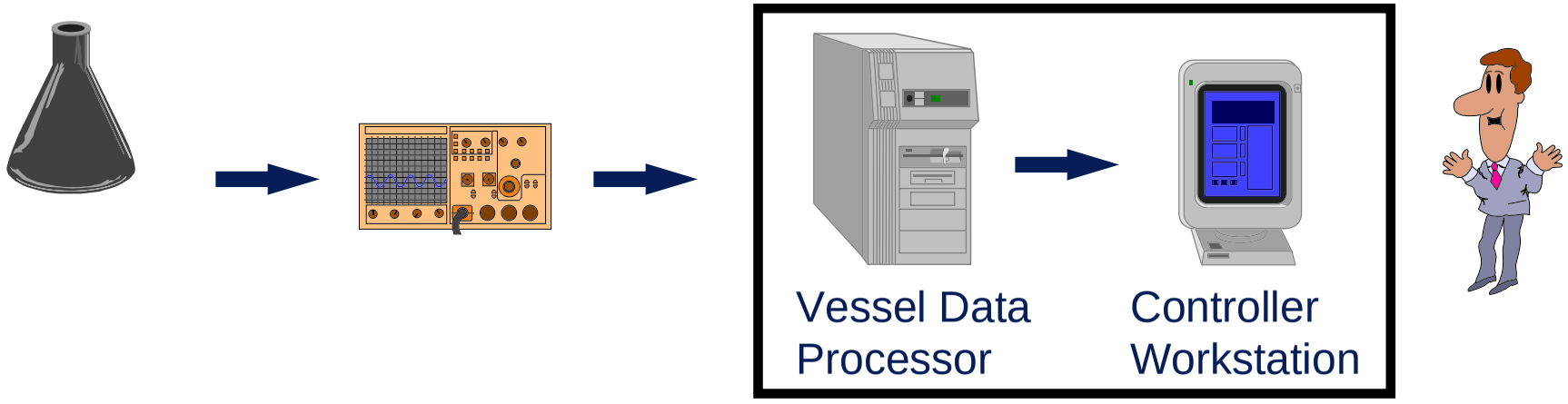
③ “Backwards” code-level SVCs into “forward” code-level SVCs

④ “Forward” code-level SVCs into pre- and post-conditions (e.g., **SPARK annotations**)

Air Traffic Management (ATM) System



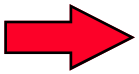
Chemical Factory Management (CFM) System



Chemical Factory Hazard

- Hazard
  - An “invalid” temperature is displayed
- Hazard causes
  - Displaying corrupt or stale temperature
  - But not, failure to display temperature
- “Backward” SVCs
  - *If* temperature displayed (output conditions)
    then update received (input conditions)

Step 1



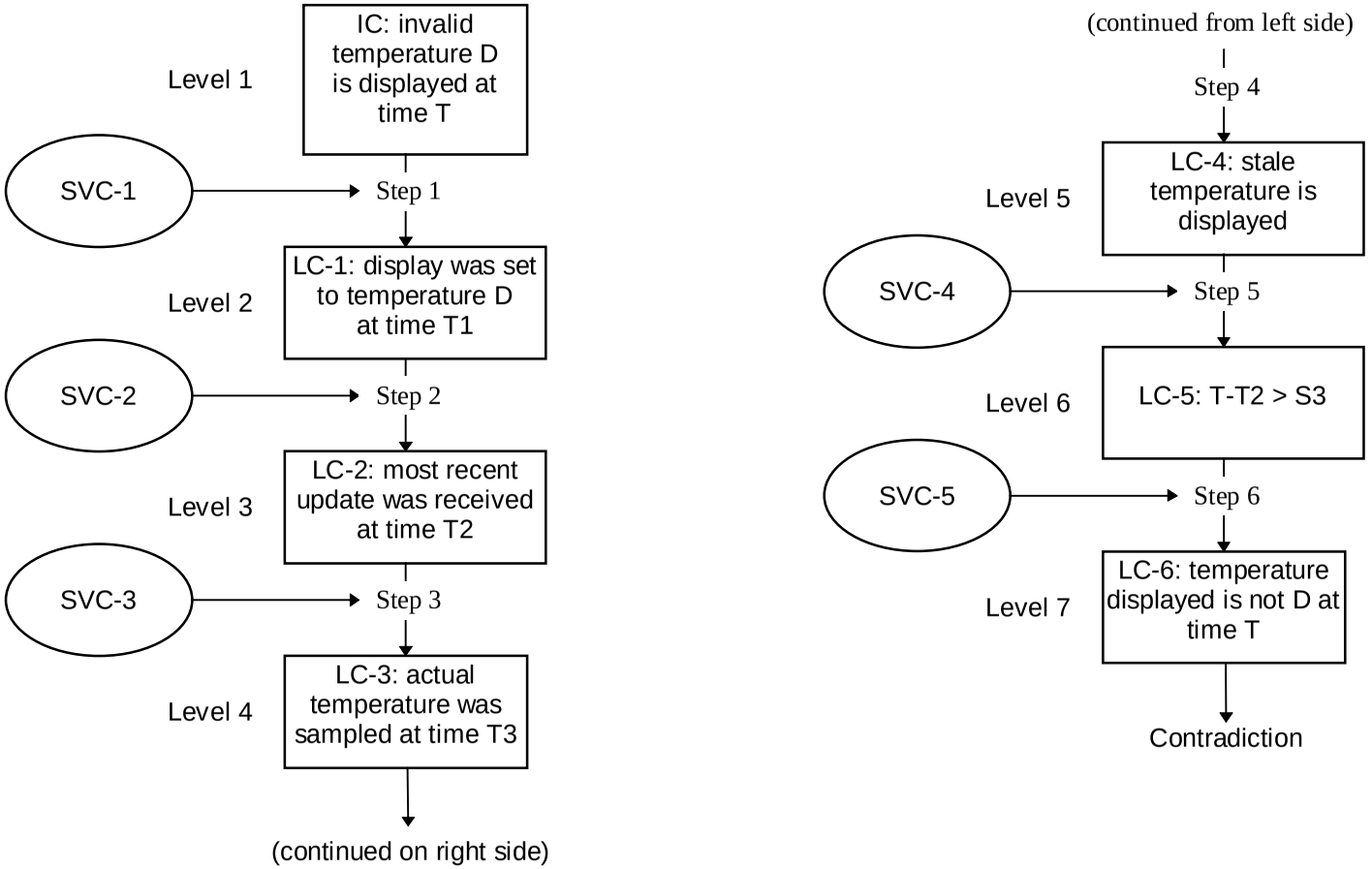
① **Hazard** into system-level SVCs

② System-level SVCs into code-level SVCs

③ “Backwards” code-level SVCs into “forward” code-level SVCs

④ “Forward” code-level SVCs into pre- and post-conditions (e.g., **SPARK annotations**)

Verification Tree Method



System-Level SVC

- “Black box” view of system
- Five system-level SVCs, e.g.,

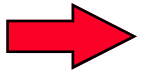
  For all vessels, v, displayed temperatures, d, and times, t,

  if the displayed temperature of vessel v is set to d at time t,

  then at some time no earlier than MSPT milliseconds before t the system received a report from the external sensor monitoring system that the temperature of vessel v is d.

Step 2

① **Hazard** into system-level SVCs

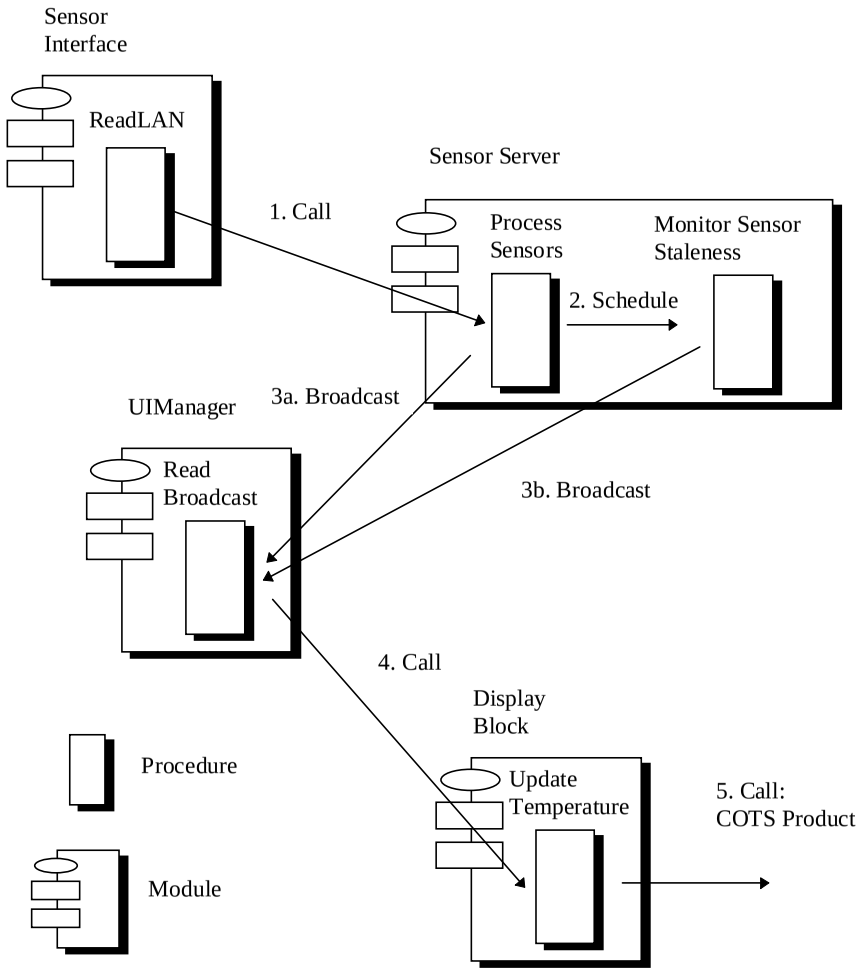


② System-level SVCs into code-level SVCs

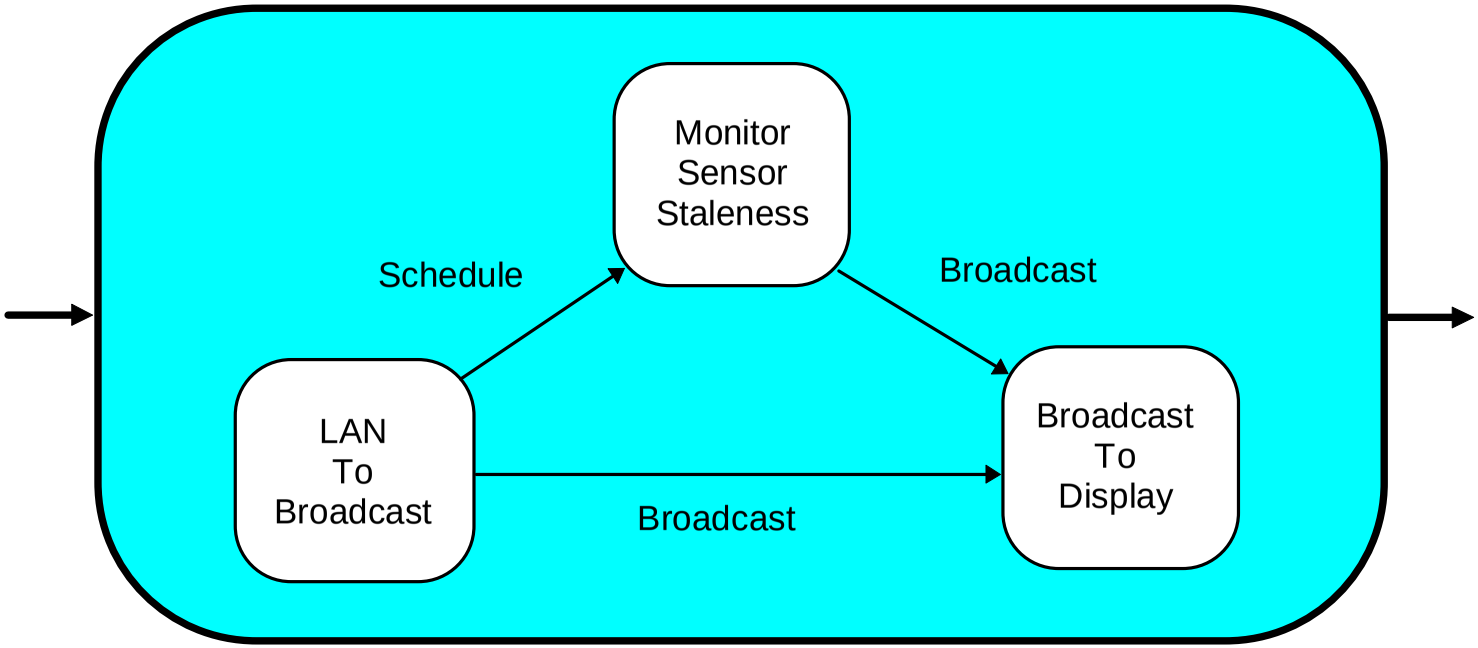
③ “Backwards” code-level SVCs into “forward” code-level SVCs

④ “Forward” code-level SVCs into pre- and post-conditions (e.g., **SPARK annotations**)

Hazard-Related Code Slice



Functional Blocks



LANToBroadcast Parameters

```
type LANMessage_SensorUpdate is
record
        InterpolatedState : LANMessage_InterpolationRange;
        TemperatureEstab : LANMessage_Temperature_T;
         SensorCodeEstab : LANMessage_SensorCode;
end record;


type Sensor_Object is
record
        SID : Sensor_SensorID;
        SensorOperation : Sensor_Operation;
        SensorQuality : Sensor_Quality;
        SensorTemperature : Sensor_Temperature;
end record;
```

Code-Level SVC

- LANToBroadcast Block

  For all Sensor Objects, s, in output
  BroadcastMessage, M,

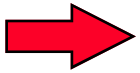
  if Sensor Object, s, contains the information
  SensorID, C, and PresentTemperature, D,

  then there exists a SensorUpdate, U, in input
  LANMessage, L, with the information
  SensorCodeEstab, C1, which is converted from
  SensorID, C, and TemperatureEstab, D1, which is
  converted from D.

Step 3

① **Hazard** into system-level SVCs

② System-level SVCs into code-level SVCs



③ “Backwards” code-level SVCs into “forward” code-level SVCs

④ “Forward” code-level SVCs into pre- and post-conditions (e.g., **SPARK annotations**)

Pre- And Post-Condition Style

- "Backwards" SVC
  - *if* output conditions *then* input conditions
- Annotations are "forward"
  - *if* pre-condition on inputs
    then post-condition on outputs
- Take contrapositive of "Backwards" SVC
  - *if* NOT(input conditions)
    then NOT(output conditions)

“Forward” Code-Level SVC

- LANToBroadcast Block

  For all Sensor Objects, s, in output BroadcastMessage, M,

  if for all SensorUpdates, u, in input LANMessage, L, with the information SensorCodeEstab, C1, which is not converted from SensorID, C, or TemperatureEstab, D1, which is not converted from D,

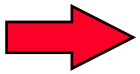
  then Sensor Object, s, does not contain the information SensorID, C, and PresentTemperature, D.

Step 4

① **Hazard** into system-level SVCs

② System-level SVCs into code-level SVCs

③ “Backwards” code-level SVCs into “forward” code-level SVCs



④ “Forward” code-level SVCs into pre- and post-conditions (e.g., **SPARK annotations**)

Verifiable Code Assertion

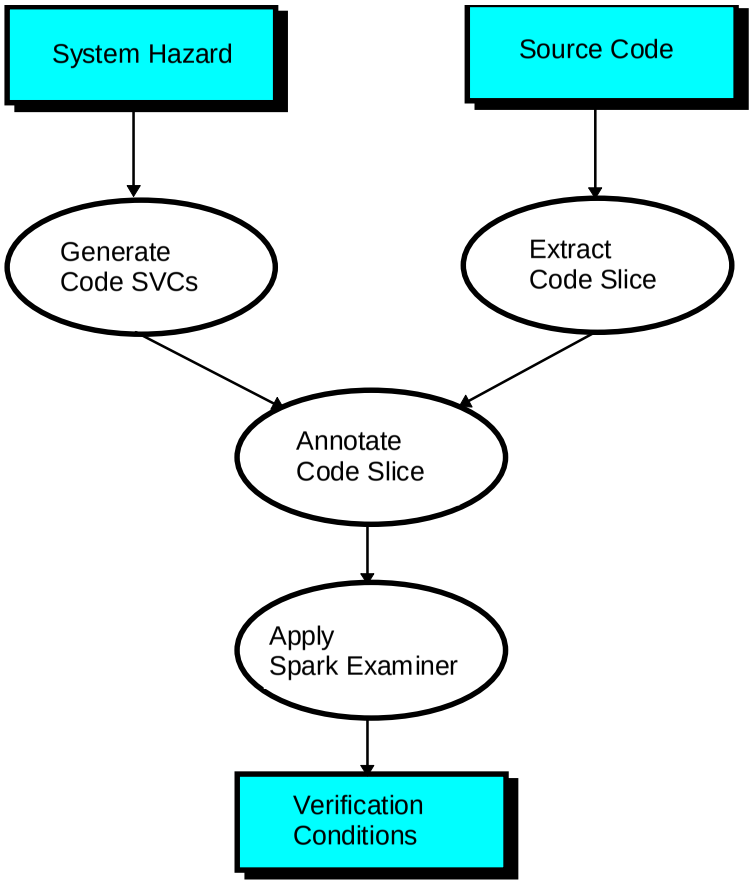
- SVC expressed with function ConvertAll in a SPARK annotation (i.e., post-condition),

```
--# post ConvertAll(Message, BroadcastMessage,
    LANMessage_SensorUpdateRange'First,
    LANMessage_SensorUpdateRange'Last);
```

- and the function is defined with a FDL rule (input for SPARK Proof tools),

```
ConvertAll(BM, LM, I, F) may_be_replaced_by ...
```

Safety Code Verification



Conclusions

- “Correctness verification” tool can be used in safety verification
- Involves systematic transformation of the hazard into machine-readable annotations
- Not all SVCs can be verified, e.g., timing or environmental SVCs
- Still must verify Examiner output, i.e., VCs

References

- http://www.cs.ubc.ca/formalWARE/safety.htm
- Generating Safety Verification Conditions Through Fault Tree Analysis and Rigorous Reasoning, ISSC’98
- Looking at Code With Your Safety Goggles On, Ada-Europe’98