Integrating Run-time and Design-Time Assurance for Autonomy Operating System (AOS)

Michael Lowry, NASA Ames
Sanjai Rayadurgam, Univ. Minnesota
Johan Schumann, NASA Ames
Thomas Pressburger, NASA Ames
Michael Dalal, NASA Ames
Autonomy Operating System (AOS) Overview

- AOS is an open architecture for smart UAS flight software leading to Autonomy.
- The foundation is NASA Core Flight Software. Heritage of over 22 successful space missions, built for flight certification.
- Small Business Participation
- Integrated Artificial Intelligence with the goal of achieving acceptable pilot behavior independent of a remote ground crew
AOS Open Architecture

Apps

- Plexil Procedures
- Automated Reasoning
- Diagnostic/Prog Models

Autonomy Executive

Knowledge bus

Automated Reasoning
Speech NatLang
Plexil
Diagnostic Prognostic
Runtime Assurance
Detect & Avoid
cFS/cFE Middleware

- Publish-subscribe message bus
- Library of common flight software services
- OSAL: Cross-OS with abstractions
Integration with CFS

Artificial Intelligence Engines
- Plan Execution (PLEXIL)

CFS Applications
- Software bus interfacing
- Scheduler interface
- Event system support
- OS Abstraction Layer system calls

Core Flight Executive
- CoreFlight Systems 6.5 (Software Bus/Software Bus Network)
- 64 bit Intel OSAL, PSP
- Linux OS

Autonomy Apps
- Departure Procedures
- En-Route Procedures
- Lost Communication Procedure

Flight Hardware
- Autopilot Software
- Simulator-in-Loop

AOS Mission Computer
- Laptop Equivalent (Intel i7)

PLEXIL
MAV_BRIDGE
Health
Lost Communication Procedure

Route. (1) Last ATC clearance received; (2) If being radar vectored, by the direct route to fix specified in vector clearance (3) Route expected in a further clearance; or (4) In the absence of an assigned route or a route that ATC has advised may be expected in a further clearance by the route filed in the flight plan.

Altitude. At the HIGHEST of the following altitudes or flight levels FOR THE ROUTE SEGMENT BEING FLOWN: (1) The altitude in the last ATC clearance received; (2) The minimum altitude for IFR operations; or (3) The altitude or flight level ATC has advised may be expected in a further clearance.

Departure Clearance: RT 060 1 mile after depart, RV SJC, climb 3000, expect 5000 10 min after depart

[4] 007UAV : Received EFC for time 17 ; Alt = 5000
[4] 007UAV : RT 060
[5] 007UAV : Setting altitude: 3000
[8] ATC: Turn Heading 120
[9] 007UAV: Heading 120
[11] 007UAV : Beginning lost comm procedure...
[12] ATC: Fly direct Sunol
[12] 007UAV : Squawking 7600
[13] ATC: NORDO 007UAV
[13] 007UAV: Attempting to hail ATC
[13] 007UAV: Attempting to visually locate airports..
[14] 007UAV: Set waypoint SJC
[17] 007UAV: setting altitude to 5000, per EFC
[24] 007UAV: Reached waypoint: SJC
[24] 007UAV: Setting waypoint per flight plan: Sunol
PLEXIL Overview

- Language for encoding plans for automation
  - Expressive
    - Condition and event-driven logic
    - Concurrency
    - Iteration (repetition, loops)
  - Strong formal semantics with proven properties
- Technology for executing these plans on real or simulated systems
  - Executive (runs under Unix, Linux, embedded systems)
  - GUI tools for monitoring plan execution
Plexil Nodes

– Actions have conditions determining when they are executed and whether execution succeeded, describing what happens when they are executed.

– An empty node can contain only attributes and performs no action.

– An assignment node performs a local computation, whose value is assigned to a variable.

– A command node issues commands to the system being operated on.

– An update node provides information to the planning and decision-support interface.

– A library call node invoke nodes located in an external library.
Plexil Conditions

- A *start condition* specifies when the action should start execution.
- An *end condition* specifies when the action should finish its execution.
- A *repeat condition* specifies when the action should be made eligible for a repeat execution.
- A *skip condition* specifies when the action's execution should be bypassed altogether.
- A *precondition* is checked immediately after the start condition becomes true. If this check fails, the action will be aborted and have an outcome of failure. Preconditions are often used to verify that it is "safe" to execute the action.
Plexil Verification Paths
TPlex: Plexil Test Case Generation

1. Translate the input PLEXIL plan into the synchronous dataflow language Lustre.
2. Augment the Lustre program with properties that encode test coverage obligation.
3. Use the JKind model-checker to produce Lustre execution traces that are witnesses for the properties.
4. Translate back the Lustre execution traces to PLEXIL test script.
5. Check conformance between PLEXIL reference execution of the generated tests and Lustre traces.
Example Generated Test Case

```xml
<PLEXILScript>
  <InitialState>
    <State name="RadarVectored" type="bool">
      <Value>true</Value>
    </State>
    <State name="ExpectingLateralClearance" type="bool">
      <Value>true</Value>
    </State>
  </InitialState>
  <Result>COMMAND_SENT_TO_SYSTEM</Result>
  <Script>
    <Simultaneous>
      <CommandAck name="flyDirectToVectoredWaypoint" type="string">
        <Result>COMMAND_SENT_TO_SYSTEM</Result>
      </CommandAck>
      <State name="ClearanceAltitude" type="int">
        <Value>509</Value>
      </State>
    </Simultaneous>
  </Script>
</PLEXILScript>
```
Guiding TPIlex through Annotations

• Assume: constraints over the environment that enforce realistic test cases
• Expect: properties to be verified are formulated as expectations
• Desire: extends TPLEX auto-generated obligations for state and transition coverage with user-specified obligations
Formal Methods in Flight: Runtime Assurance

- Bounded computation for monitoring metric temporal logic
- Signal processing and Bayesian net reasoning
- Guaranteed real-time (microseconds on FPGA)
- Extends watch-dog processor to co-pilot cross-checks
- Synergistic with other Formal Methods capabilities
R2U2 Metric Temporal Logic

Metric Temporal Logic (MTL) reasons about bounded timelines:
- finite set of atomic propositions \{p, q\}
- Boolean connectives: \(\neg, \wedge, \vee, \text{and} \rightarrow\)
- temporal connectives with time bounds:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
<th>Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\square_{[2,6]}p)</td>
<td><strong>ALWAYS</strong>([2,6])</td>
<td>![Timeline 1]</td>
</tr>
<tr>
<td>(\Diamond_{[0,7]}p)</td>
<td><strong>EVENTUALLY</strong>([0,7])</td>
<td>![Timeline 2]</td>
</tr>
<tr>
<td>(p U_{[1,5]}q)</td>
<td><strong>UNTIL</strong>([1,5])</td>
<td>![Timeline 3]</td>
</tr>
</tbody>
</table>

Efficient implementation in FPGA [TACAS2014]
DHS Predator Crash Nogales, AZ 2006

- Large unmanned aerial vehicle (UAV) accident rates vary, but drones are 30 to 300 times more likely to crash than small civil aircrafts.
- DoD survey indicated 14% of failures involved lost communication links.
# Mode Confusion at GCS

<table>
<thead>
<tr>
<th>Condition Lever Control Position</th>
<th>PPO-1 (pilot) Result</th>
<th>PPO-2 (sensor operator) Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Opens fuel valve to engine</td>
<td>Increases camera iris opening</td>
</tr>
<tr>
<td>Middle</td>
<td>Closes fuel valve to engine (and shuts down engine)</td>
<td>Locks camera iris setting</td>
</tr>
<tr>
<td>Aft</td>
<td>Feathers prop</td>
<td>Decreases camera iris opening</td>
</tr>
</tbody>
</table>

![Diagram of levers and buttons]

![Diagram of levers and buttons]
Load Shedding Clobbers Recovery

• Mode confusion causes fuel cutoff
• Predator sheds electrical power
• Backup Iridium SatCom is shed
• Iridium is required for auto-restart
Design-Time Assurance (V&V)

- Symbolic execution to cover mission and environmental variations
- Attempt proofs (k-induction) for safety conditions
- Generate prime implicants for failed proofs
- Synthesize MTL monitors from prime implicants consisting of observable literals
Run-Time Assurance

• Cross-check that commanded actions have intended post-condition (e.g. climb -> gain altitude in 30 sec)

• Monitor for conditions that can’t be discharged during V&V

• Project fault propagation

• Sequence recovery procedures that block fault propagation

• Check preconditions for mitigations and recovery procedures
Automated Reasoning: Formal Methods in Flight

TPleX (Before Flight): symbolic model checking to prove autonomy procedures are correct. Residuals monitored by R2U2.

During Flight: Deducing Landing Sequence Order from ATC Utterances

10x SMT improvement in 4 years

1 second deductions faster than pilots
Flight Test (Feb 1, 2017, ARC)

Synopsis

• Integration of Runtime Assurance into AOS
• Automatic triggering and execution of emergency procedure after battery low failure
• Processing of MAVlink and Plexil data by Runtime with past-time temporal logic.
Automated Reasoning

- Deliberative navigation through SMT reasoning.
- Knowledge-base for assertions, implemented in SQL (student developed app)
Summary

• Formal methods for design-time and run-time assurance provide orthogonal defect mitigation

• Integration of design- and run-time feasible through symbolic calculation

• Next step: integration with recovery and mitigation procedures through onboard reasoning.