Behavior Language for Embedded Systems with Software (BLESS) Language and Tools

Brian R Larson
brl@ksu.edu
The Laboratory for Specification, Analysis, and Transformation of Software (SAnToS)

February 14, 2019

Model-Based Engineering Challenges

AADL superbly models embedded system structure, interfaces, and non-functional properties for analysis.

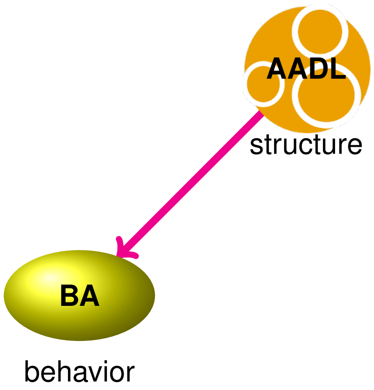
Can we ensure deployed systems (software) conform to their models, and function correctly?

AADL for System Structure

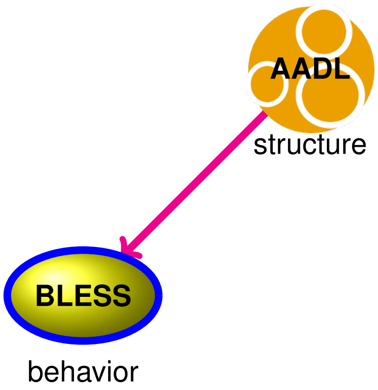
AADL

structure

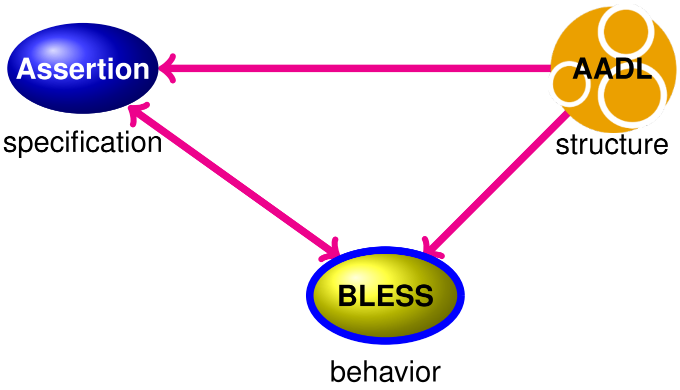
Add Behavior Annex



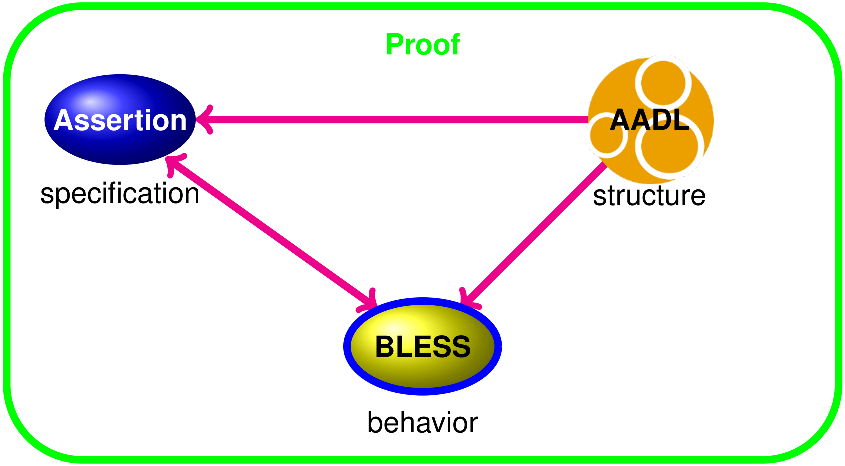
BLESS is Superset of BA



BLESS Assertion adds Declarative Specification



Formal Verification by Proof



Temporal Logic

BLESS uses first-order predicate calculus, extended by simple temporal operators to declaratively specify behavior. Such temporal logic formulas are called BLESS *assertions*.

`p@t` ≡ evaluate predicate `p` at time `t`.

```
<<VP: : --cause ventricular pace
  (n or p)@(now-lrl)  --last beat occurred LRL interval ago,
  and --not since then
    not (exists t:time  --there is no time
      in now-lrl,,now  --since then, ",," means open interval
      that (n or p)@t) >>  --with a beat
```

Quantification over time together with simple temporal operators makes BLESS assertions uniquely capable of expressing timing of embedded systems.

Behavior Specification

BLESS::Assertion properties specify
what is guaranteed about events issued by **out** ports,
what is assumed about events received by **in** ports, and
what is always true.

```
thread VVI
  features
    s: in event port; --signal from analog front-end
    p: out event port --pace ventricle
      {BLESS::Assertion => "<<VP()>>";};
    n: out event port --natural contraction
      {BLESS::Assertion => "<<(now=0) or VS()>>";};
    lrl: in data port ms;  --lower rate limit interval
    vrp: in data port ms;  --ventricular refractory period
  properties
    Dispatch_Protocol => Aperiodic;
    BLESS::Invariant => "<<LRL(now)>>";
end VVI;
```

State-Transition Machine

BLESS began as BA, adding assertions to express what is true about the system when a machine is in a particular state.

Actions performed during transitions can also be augmented with assertions.

BLESS defines formal semantics for every construct, adding a type system to be a programming language.

```
variables
  last_beat : time
  --the last pace or non-refractory sense occurred at last_beat
  <<LAST: :(n or p)@last_beat>>;

states
  power_on : initial state  --powered-up,
    <<now=0>>;  --start with "sense"
  pace : complete state
      --a ventricular pace has occurred in the
      --previous LRL-interval milliseconds
    <<PACE(now)>>;
. . .
  check_pace_vrp : state
      --execute state to check if s is in vrp after pace
    <<s@now and PACE(now)>>;
. . .
. . .
  T3_PACE_LRL_AFTER_VP: --pace when LRL times out
  pace -[on dispatch timeout (n or p) lrl ms]-> pace
    { <<VP()>>
    p! <<p@now>>   --cause pace when LRL times out
    & last_beat:=now <<last_beat=now>>};

  T4_VS_AFTER_VP: --sense after pace=>check if in VRP
  pace -[on dispatch s]-> check_pace_vrp{};

  T5_VS_AFTER_VP_IN_VRP:  -- s in VRP,  go back to "pace" state
  check_pace_vrp -[(now-last_beat)<vrp]-> pace{};

  T6_VS_AFTER_VP_IS_NR: --s after VRP,
    --go to "sense" state, send n!, reset timeouts
  check_pace_vrp -[(now-last_beat)>=vrp]-> sense
    {  <<VS()>>
    n! <<n@now>>  --send n! to reset timeouts
    & last_beat:=now <<last_beat=now>>};
. . .
```

Xtext Editor

The BLESS editor plugin to OSATE was created with Xtext to seamlessly add syntax coloring, grammar checking while typing, and error markers for BLESS annex subclauses, and Assertion annex libraries.

Proof Assistant

The BLESS proof assistant plugin to OSATE generates proofs[1] that formally verifies that behavior implementations meet behavior specifications.

[1] with human guidance

Not a Theorem Prover!

The BLESS proof assistant is not a theorem prover (Coq, PVS, Isabelle/HOL). Theorem provers are powerful, general, and hard to use.

Nor is it a SMT-solver, model checker, symbolic executer, or static analyzer.

The BLESS proof assistant was made to be used by practicing engineers in industry.

Because it's different, there's some learning curve,

Transform Proof Outlines to Deductive Proofs

BLESS assertions attached to states, and interspersed though actions performed when transitions occur form a *proof outline*.

BLESS state machines are verified to uphold their specifications by transforming their proof outlines into deductive proofs.[2]

The last theorem in the proof says all verification conditions have been met.

[2]sequences of theorems, each of which is given or axiomatic, or derived from prior theorems in the sequence by sound inference rules

Thread Verification Conditions

- each complete state's assertion implies the thread invariant
- each execution state's assertion implies the disjunction of outgoing transition conditions
- each transition's action, begun having the source state's assertion and transition condition will terminate with the destination state's assertion[3]

[3]for transitions without actions, the conjunction of source state's assertions and transition condition implies destination state's assertion

Thread VC Examples

```
[serial 1004]: vvi_mode::VVI.i
P [70] <<PACE(now)>>
S [57]->
Q [57] <<LRL(now)>>
  What for: <<M(pace)>> -> <<I>> from invariant I when complete state pace has Assertion
   <<M(pace)>> in its definition.
```

```
[serial 1007]: vvi_mode::VVI.i
P [77] <<s@now and PACE(now)>>
S [77]->
Q [77] <<((now-last_beat) < vrp) or ((now-last_beat) >= vrp)>>
  What for: Serban's Theorem:  disjunction of execute conditions leaving execution state
   check_pace_vrp, <<M(check_pace_vrp)>> -> <<e1 or e2 or . . . en>>
```

```
[serial 1013]: vvi_mode::VVI.i
P [77] <<s@now and PACE(now) and ((now-last_beat) >= vrp)>>
S [108]<<VS()>>
n!
<<n@now>>
&
last_beat := now
<<last_beat = now>>
Q [74] <<SENSE(now)>>
  What for:  <<M(check_pace_vrp) and x>> A <<M(sense)>> for
   T6_VS_AFTER_VP_IS_NR:check_pace_vrp-[x]->sense{A};
```

Composition Verification Conditions

- out port's assertion implies connected in port's assertion (assume-guarantee)
- conjunction of subcomponents' invariants implies containing component's invariant

BLESS Generates C++

Kansas State University worked with Adventium Labs on Intrinsically-Secure, Open, and Safe Cyber-physically Enabled Life-critical Essential Services (ISOSCELES) for the Department of Homeland Security to develop a platform for secure medical devices.

Proof-of-concept C++ code auto-generated from BLESS using AADL runtime services implemented for ISOSCELES is included in Todd Carpenter's demonstration this afternoon.

BLESS Generates Slang

Alternatively, proof-of-concept Slang[4] can be generated from BLESS.

This enables BLESS to take advantage of the Slang development and simulation environment and its translation backends.

[4]KSU-developed dialect of Scala

AADL+BLESS unifies specification, programming, and verification with architecture (SSoT).

BLESS treats programs, specifications, and executions as mathematical objects; deductive proofs argue that *every* execution conforms to specification.

BLESS was created to be used by practicing engineers.

BLESS verification artifacts (correctness proofs) can be read, understood, and checked.

BLESS generates proof-of-concept executable code through two, different, compilation tool chains.

Together with architecture-centric virtual integration, BLESS may reduce costs and duration of development as tests confirm correctness rather than finding errors.

Please stop by my table to see dozens of threads with BLESS behaviors and proofs, and try BLESS yourself.