Proving Behavior Conforms to Formal Specification of DDD Mode Cardiac Pacing*

Brian R Larson, John Hatcliff

Kansas State University {brl,hatcliff}@ksu.edu

Abstract. In previous work, we introduced the Behavioral Language for Embedded Systems with Software (BLESS) as an annex sublanguage of the SAE standard Architecture Analysis and Design Language (AADL). BLESS provides mechanisms for specifying behaviors on component interfaces, defining AADL-runtime aware transition systems that capture the internal behavior of AADL components, and writing assertions to capture important state and event properties within the transition system notation. BLESS includes a verification-condition (VC) generation framework and an accompanying proof tool that enables engineers to prove VCs via proof scripts build from system axioms and rules from a user-customizable rule library. We illustrated how BLESS could be applied to single-chamber, VVI-mode cardiac pacing to formally specify behavior and prove its implementation conforms to the specification.
This paper describes BLESS applied to more complex dual-chamber DDD-mode pacing complicated by not just two chambers for sensing and pacing, but cross-chamber refractory period and tracking of ventricular paces following atrial senses, but not faster than a prescribed upper rate. We use BLESS Assertions to declaratively specify behavior, BLESS state-transition machines to define an implementation of DDD mode pacing, and assertions annotating the implementation to form a proof outline that is transformed into a complete formal proof using the BLESS proof tool, guided by human selection of tactics. This approach offers advantages over other formalizations of DDD pacing in that it provides a formal proof of compliance of an algorithm to declarative specifications for arbitrary configurations of the algorithm.

1 Introduction

1.1 Rigorous Development of Embedded Systems

Developing high-assurance embedded systems requires rigorous approaches to architecture specification, interface descriptions, hardware and software co-development, and the ability to specify and verify both functional and non-functional (*e.g.*, time-related) properties. The Architecture Analysis and Design Language

* Work supported in part by the US National Science Foundation (NSF) (#0932289, #1239543), the NSF US Food and Drug Administration Scholar-in-Residence Program (#1065887, #1238431)

(AADL) is a SAE International standard for design of embedded, cyber-physical systems. AADL supports design of both physical architecture (processors, memory, buses, devices), logical architecture (thread, processes, connections), and mapping of logical architecture onto physical architecture [3]. Recent industry consortia use of AADL [4] focuses on defining precise interface descriptions and exposing in architectural models important properties needed to perform component-wise and system-wide analysis of real-time scheduling and error propagation properties. Despite its ability to capture architectures, interface signatures, and system properties, AADL lacks (a) a formal behavioral interface specification language, (b) a formal semantics for component behavioral descriptions, and (c) tools for reasoning about the compliance of behaviors to interface specifications.

In previous work [9], we introduced the *Behavioral Language for Embedded Systems with Software* (BLESS)—a Behavioral Interface Specification Language (BISL) [5] and an associated proof environment for AADL. The AADL standard provides the notion of an annex sublanguage to support extensions to the modeling language, and BLESS uses this mechanism to introduce notations for (a) specifying behaviors on component interfaces, (b) defining AADL-runtime aware transition systems that capture the internal behavior of AADL components, and (c) writing assertions to capture important state and event properties within the transition system notation. BLESS annex subclauses can be inserted into AADL components transparently to other uses of the system architecture. Successful definition and tool engineering for formal reasoning frameworks like BLESS require a precisely defined formal semantics, and we have invested considerable effort in defining a such a semantics for BLESS[1]. Finally, BLESS includes a verification-condition (VC) generation framework and an accompanying proof tool that enables engineers to prove VCs via proof scripts build from system axioms and rules from a user-customizable rule library.

1.2 Cardiac Pacing – A Formal Methods Challenge

Community development of shared challenge problems can be an effective approach to focused advancement of research. The first author worked for five years at Boston Scientific, a leading pacemaker company, and was responsible for the release of a requirements document for a previous generation pacemaker [14]. This document, PACEMAKER System Specification, has been used in over 30 research papers within the formal methods community and provided the core set of examples of a recent book on embedded system modeling techniques [7]. This document is also the basis of the PACEMAKER Challenge sponsored by the Software Certification Consortium [14]. The PACEMAKER Challenge provides a subject for mock submission of applications for certification of safety-critical systems that rely on software. In particular, evidence from formal methods is encouraged to be part of the submission.

[1] Due to space constraints, the semantics cannot be presented here. We refer interested readers to [10] for the formal semantics and details of the VC generation process.

Figure 1 depicts a pacemaker (Pulse Generator) connected by pacing leads to the inside of a heart's right atrium and ventricle chambers. The pacemaker emits a short low voltage "pace" through a lead to cause cascade of cell contractions in the chamber associated with the lead, expelling blood to lungs or body. The same leads used to deliver paces can also sense intrinsic electrical activity of the heart. This allows the pacemaker to monitor the patient and inhibit electrical-pacing when intrinsic pacing voltage exceeds a lower limit (*i.e.*, when the heart beats well enough "on its own").

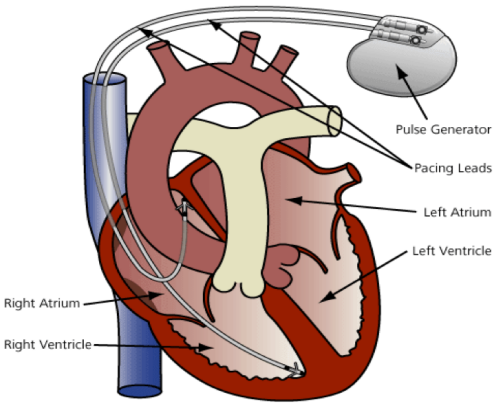

Fig. 1. Pacemaker Environment

1.3 Contributions of This Paper

There are industry-standard pacing modes that differ according to (a) which heart chambers (atrium, ventricle, or both) are paced and/or sensed and (b) how pacing behavior responds to senses. In [9], we used BLESS to formally render the requirements for the VVI pacing mode in a declarative specification, define an implementation using BLESS transition system notation, and carry out proofs showing the the implementation conformed to the formal specification. In VVI mode (Ventricle paced, Ventricle sensed, Inhibit a pace upon a sense), a single chamber (the ventricle) is paced and sensed. In this paper, consider the much more complex DDD mode. DDD paces both chambers, senses both chambers, and ventricular paces will track atrial senses. Upon either an atrial pace or sense, the ventricle will be paced if not sensed within a specified period of time. However, sensing electrical cardiac signals is complicated by ambient electrical noise (reduced by filtering), and inherent electrical noise due to "repolarization" following contraction, which is avoided by programable refractory periods during which cardiac signals are ignored.

A few other research efforts have also addressed DDD pacing. For example, both [2, 6] use model-checking techniques to specify and verify DDD safety properties. There are a number of trade-offs between techniques, and indeed the purpose of the PACEMAKER Challenge was to provide a well-defined space in which trade-offs could be explored and evaluated. The model-checking approaches used [2, 6] are much more automated than our approach, but due to the state-space explosion and other inherent limitations of model-checking, only a limited number of configurations (physician-determined parameter settings) could be addressed in the verification. The BLESS strategy, while requiring

manual steps in the proof tool, provides verification of a DDD implementation against a specification for *all* valid configurations of parameters.

The specific contributions of this paper are as follows:

- we illustrate how the primary requirements for DDD mode pacing can be rendered as formal BLESS temporal specifications,
- we provide a transition system implementation of DDD pacing, and
- we construct formal proofs in the BLESS tool that the transition system implementation satifies for the formal DDD mode specifications.

All artifacts to support this work including the formal BLESS specifications and BLESS proof scripts as well as the BLESS tool itself are available on the BLESS project web site [8].

2 DDD Thread Component

DDD mode paces both right atrium and right ventricle, senses contractions in both chambers, inhibits pacing when the intrinsic (*i.e.*, natural) heart rate is fast enough, and tracks atrial senses with ventricular paces. For hearts with damaged conduction from right atrium to right ventricle, tracking can mimic lost conduction. Tracking keeps the chambers synchronized.

The critical parameters of DDD pacing configured by a physician are *lower rate limit* (LRL) and *upper rate limit* (URL). LRL defines the minimum beats per minute (either intrinsic contractions or pace-induced contractions) that the pacemaker will guarantee. URL prohibits pacing too fast by putting an upper limit on the rate of paces. URL does *not* prevent an intrinsic heart rate faster than URL, such as during exercise; it only restricts ventricular paces tracking atrial senses. Tracking is controlled by the *AV delay* parameter, nominally 150 ms. If a ventricular contraction is not sensed within the AV delay following an atrial event (pace or sense) the ventricle will be paced, but not if pacing will exceed URL.

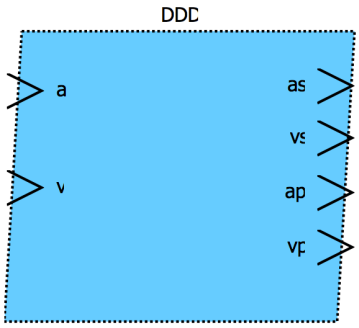

Fig. 2. AADL Thread Component

A pacemaker specified using AADL would include "front end" components implementing analog/digital voltage conversion and other signal processing steps. The pacing logic components receive signal events from the front end, indicating that intrinsic contractions of appropriate voltage levels have been sensed on the pacemaker leads. The logic for DDD mode is performed by a single AADL thread component, whose interface is shown in Figure 2. The thread has two in event ports for signals from the front end indicating either an atrial signal (`a`) or a ventricular signal (`v`).

Whether the signals are interpreted as "senses" of contractions or are disregarded makes DDD behavior challenging. Refractory periods during which heart signals are ignored reflect that following contraction, spurious electrical signals are generated which if interpreted as actual senses lead to incorrect, even hazardous, behavior.

```
thread DDD
 features
  a: in event port; --atrial signal
  v: in event port; --ventricular signal
  ap: out event port --pace atrium
   {BLESS::Assertion=>"<<AP(now)>>";};
  vp: out event port --pace ventricle, but not too soon
   {BLESS::Assertion=>"<<VP(now) and URL(now)>>";};
  as: out event port --non-refractory atrial sense
   {BLESS::Assertion=>"<<NRAS(now)>>";};
  vs: out event port --non-refractory ventricular sense
   {BLESS::Assertion=>"<<NRVS(now)>>";};
  . . .
end DDD;
```

Table 1. DDD Thread Type Defines Externally-Visible Ports

When an event on port `a` is deemed by the logic in the thread to be an atrial sense, an event is issued on port `as`. Similarly, when an event on port `v` is deemed to be a ventricular sense, an event is issued on port `vs`. These output sense events have no effect on the heart, but are logged. When the thread decides the atrium needs to be paced, it issues an event on port `ap` which causes the front end to pace the atrium with a short (0.4 ms) single-digit voltage pulse (2V). Similarly, pacing the ventricle issues an event on the `vp` port.

All the out event ports have `BLESS::Assertion` that define what must be true when an event is issued from the port. These Assertions[2] define the top-level correctness properties of the pacing mode. The Assertion definitions are parameterized on a time value, and their use below is instantiated to the current time (indicated by the keyword `now`). We first define a number of basic properties of DDD pacing in Section 3 and then use these properties to define the semantics for the top-level port Assertions in Section 4.

3 Formalization of PACEMAKER Properties

In this section, we systematically consider the requirements (shown in italics) of DDD mode pacing from the PACEMAKER System Specification [14] and illustrate how they are naturally rendered as BLESS formal temporal specifications. Although expressed as rates (beats or pulses per minute), programmed parameters are always used as time in milliseconds. A rate of 60 bpm corresponds to a 1000 ms interval.

3.1 Lower Rate Limit (LRL)

> *The Lower Rate Limit (LRL) is the number of generator pace pulses delivered per minute (atrium or ventricle) in the absence of sensed intrinsic activity.*

[2] Capital 'A' as a proper noun for assertions used in BLESS

One of the primary goals of DDD pacing is to ensure that there is either a natural heartbeat (instrinsic activity) or a pace within the LRL interval (L). Figure 3 shows DDD pacing in the absence of intrinsic activity. Ventricular paces follow atrial paces after a prescribed AV delay defined in section 3.4. The DDD algorithm needs to ensure that both atrial paces and ventricular paces occur within the LRL interval applied to activity in the corresponding chamber.

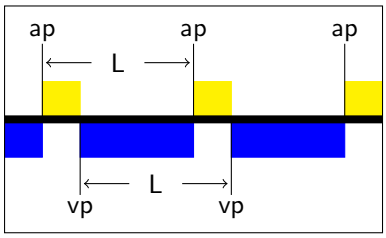

Fig. 3. DDD Pacing at Lower Rate Limit

Lower Rate Limit As Math: Let the lower rate limit interval be L, an intrinsic ventricular contraction sensed at time t be $\mathfrak{M}_t[\![n]\!]$, and a paced ventricular contraction caused at time t be $\mathfrak{M}_t[\![p]\!]$. Then for an arbitrary time x, the predicate $LRL(x)$ is defined as

$$LRL(x) \equiv \exists t \in [(x-L)..x] \mid (\mathfrak{M}_t[\![n]\!] \vee \mathfrak{M}_t[\![p]\!])$$

In this BLESS predicate `LRL` applied to parameter `x`, where `n` and `p` are names of ports, `n@t` (`p@t`) means an event occurred on port `n` (on port `p`). `<<LRL(now)>>`, where `now` refers to the present instant, expresses that a heartbeat, either sensed or paced, occurred within the last LRL interval. This is a fundamental effectiveness property of DDD pacing, and so this assertion will be declared as a thread invariant in Section 5.

```
<<LRL:x: exists t:T in x-L..x that (n@t or p@t) >>
```

Table 2. Lower Rate Limit Assertion

3.2 Ventricular Refractory Period (VRP)

The Ventricular Refractory Period shall be the programmed time interval following a ventricular event during which time ventricular senses shall not inhibit nor trigger pacing.

Electrical noise follows ventricular contraction; so ventricular signals are ignored immediately after heartbeats. Being unresponsive to signals is called *refractory*. A ventricular sense will be non-refractory when the most-recent beat, whether intrinsic or paced, was further in the past than the Ventricular Refractory Period (VRP). VRP is needed because hearts are electrically noisy following contraction, either paced, vp, or intrinsic, vs.

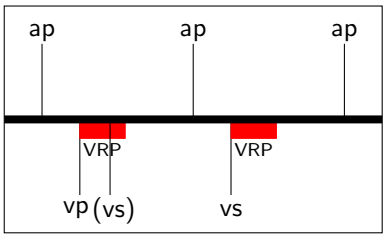

Fig. 4. Ventricular Refractory Period

Figure 4 shows a ventricular sense being ignored because it occurred during VRP following a ventricular pace. Parentheses indicate the ventricular sense is ignored.

Ventricular Refractory Period As Math: Let the time of the most recent heartbeat be lb (for last-beat), $\mathfrak{M}_{lb}[\![n]\!]$ be an intrinsic heartbeat occurred at time lb, $\mathfrak{M}_{lb}[\![p]\!]$ be a paced heartbeat was caused at time lb, r be the Ventricular Refractory Period, and *now* be the present instant. Then ventricular senses will be non-refractory, NR, when the last beat occurred earlier than the Ventricular Refractory Period before time x is defined as

```
<<NR:x: -- not in Ventricular Refractory Period
  not exists t:T in [x-r..x] that (n@t or p@t)>>
```

Table 3. Ventricular Refractory Period Assertion

$$NR(x) \equiv \neg\exists t \in [(x-r)..x] \mid (\mathfrak{M}_t[\![n]\!] \vee \mathfrak{M}_t[\![p]\!])$$

3.3 Upper Rate Limit (URL)

The Upper Rate Limit (URL) is the maximum rate at which the paced ventricular rate will track sensed atrial events. The URL interval is the minimum time between a ventricular event and the next ventricular pace.

Tracking of ventricular paces following atrial senses is the last 'D' in DDD. The Upper Rate Limit (URL) prevents pacing above a prescribed rate when ventricular paces track atrial senses. Pacing too fast can injure a heart. URL is the fundamental safety property of DDD pacing. DDD must only issue ventricular paces when the most recent ventricular pace or sense occurred at least u milliseconds in the past, where u is the URL interval.

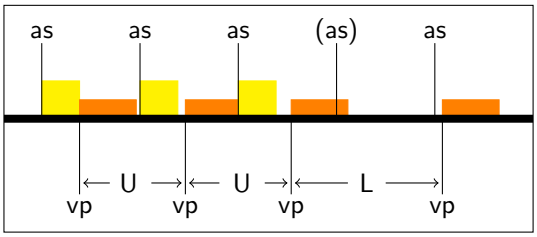

Fig. 5. Tracking Atrial Senses Faster Than Upper Rate Limit

Figure 5 shows an intrinsic atrial rate faster than URL. Normally, ventricular paces follow atrial senses after a specified AV delay (yellow rectangles, to the right of the lines marked 'as'), but when the atrial rate exceeds URL, ventricular paces fall further behind until an atrial sense occurs during a refractory period (orange rectangles, to the right of the lines marked 'vp'), so is ignored. The last ventricular pace does not track an atrial sense, but occurs so that the time between ventricular paces does not exceed the LRL interval.

```
<<URL:x: not (exists t:T in x-u,,x that (vs or vp)@t) >>
```

Table 4. Upper Rate Limit Assertion

Upper Rate Limit As Math: Let the upper rate limit interval be u, a non-refractory ventricular sense occurring at time t be $\mathfrak{M}_t[\![vs]\!]$, a ventricular pace caused at time t be $\mathfrak{M}_t[\![vp]\!]$, and $[(x-u),,x]$ be the open interval (excluding endpoints) between $x-u$ and x. Then the upper rate limit safety property is

$$URL(x) \equiv \neg\exists t \in [(x-u),,x] \mid (\mathfrak{M}_t[\![vs]\!] \vee \mathfrak{M}_t[\![vp]\!])$$

The DDD algorithm must only generate a ventricular pace (`vp!`, Section 4.2) when $URL(now)$ holds, *i.e.*, when no ventricular event (pace or sense) occurred in the previous URL interval.

3.4 Atrioventricular (AV) Delay

The AV delay shall be the programmable time period from an atrial event (either intrinsic or paced) to a ventricular pace. In atrial tracking modes, ventricular pacing shall occur in the absence of a sensed ventricular event within the programmed AV delay when the sensed atrial rate is between the programmed LRL and URL.

The Atrioventricular (AV) Delay is the maximum allowed time following an atrial sense or pace for a ventricular sense to be detected, before a ventricular pace is issued.

Figure 6 shows a heart beating normally, faster that URL, with ventricular senses following atrial senses sooner than the (sensed) AV delay (SAV). The AV delay may be prescribed to be shorter following atrial senses than atrial paces, because conduction of atrial senses can be quicker. This allows a cardiologist to have ventricular paces following atrial contractions to be the same whether paced or sensed.

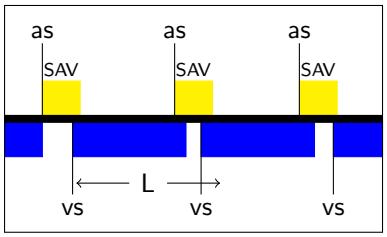

Fig. 6. Intrinsic Sensing faster than Lower Rate Limit

Paced AV Delay

A paced AV (PAV) delay shall occur when the AV delay is initiated by an atrial pace.

```
<<PAV:x: exists t:T in x-av..x that ap@t >>
```

Table 5. Paced AV-Delay Assertion

Paced AV delay (PAV), atrial pace followed by ventricular pace happens when pacing at LRL as shown in Figure 3 as yellow rectangles, to the right of the lines marked 'as'. An atrial pace will follow a ventricular event, sense or pace so that if no ventricular sense occurs before the expiration of PAV, a ventricular pace will be one LRL interval (blue rectangles, to the right of the lines marked 'vs') after the previous ventricular event. (Section 4.1 describes the function of this VA-interval.)

Paced AV Delay As Math: Let the AV delay interval be av, and an atrial pace occurring at time t be $\mathfrak{M}_t[\![ap]\!]$. Then the paced AV delay property is

$$PAV(x) \equiv \exists t \in [(x - av)..x] \mid \mathfrak{M}_t[\![ap]\!]$$

Sensed AV Delay

A sensed AV (SAV) delay shall occur when the AV delay is initiated by an atrial sense.

The sensed atrio-ventricular (SAV) delay causes "tracking": ventricular paces following atrial senses, but not faster than the Upper Rate Limit. Figure 7 shows intrinsic ventricular sense following an atrial sense sooner than SAV; a ventricular pace SAV after an atrial sense; and, another ventricular sense sooner than SAV.

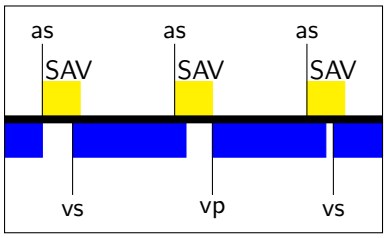

Fig. 7. Sensed Atrio-Ventricular (SAV) Delay

Sensed AV Delay As Math: Let the AV delay interval be av, the (negative) sensed AV offset be o, a non-refractory atrial sense occurring at time t be $\mathfrak{M}_t[\![as]\!]$, a non-refractory ventricular sense occurring at time t be $\mathfrak{M}_t[\![vs]\!]$, a ventricular pace occurring at time t be $\mathfrak{M}_t[\![vp]\!]$. Then the sensed AV delay property is

$$SAV(x) \equiv \exists t \in [x - (av + o)..x] \mid \mathfrak{M}_t[\![as]\!]$$

An atrial sense occurred in the previous sensed AV delay interval, or the time since the previous ventricular event is greater than URL interval

```
<<SAV:x: exists t:T in x-(av+o)..x that as@t >>
```

Table 6. Sensed AV-Delay Assertion

3.5 Post-Ventricular Atrial Refractory Period (PVARP)

The Post Ventricular Atrial Refractory Period shall be available in modes with ventricular pacing and atrial sensing. The Post Ventricular Atrial Refractory Period shall be the programmable time interval following a ventricular event when an atrial cardiac event shall not

1. *Inhibit an atrial pace.*
2. *Trigger a ventricular pace.*

The Post-Ventricular Atrial Refractory Period (PVARP) inhibits atrial sensing following ventricular events. Otherwise ventricular noise, or worse, reverse conduction, can create a bogus atrial sense, that is not an actual atrial contraction. Figure 8 shows an atrial sense that is ignored, (as), because it occurs in the PVARP.

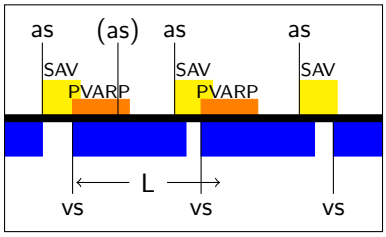

Fig. 8. Atrial Sense in Post-Ventricular Atrial Refractory Period

Post-Ventricular Atrial Refractory Period As Math: Let the post-ventricular, atrial refractory period be $pvarp$, an ventricular pace occurring at time t be $\mathfrak{M}_t[\![vp]\!]$ and a non-refractory ventricular sense occurring at time t be $\mathfrak{M}_t[\![vs]\!]$. Then the PVARP property is defined by a predicate that will hold if a ventricular event occurs during PVARP.

$$PVARP(x) \equiv \exists t \in [(x - pvarp), , x] \mid (\mathfrak{M}_t[\![vs]\!] \vee \mathfrak{M}_t[\![vp]\!])$$

4 DDD Port Assertions

```
<<PVARP:x: exists t:T in x-pvarp,,x that (vs or vp)@t >>
```

Table 7. Post-Ventricular Atrial Refractory Period Assertion

AADL allows properties to be attached to almost every architectural entity. `BLESS::Assertion` properties attached to ports specify what is assumed or guaranteed about data and events arriving or issued from ports. (Connections between components create assume-guarantee contracts that can be proved, but this example uses only a single component.) The ports of thread component DDD in Section 2 have `BLESS::Assertion` attached to out event ports that must be true at the instant an event is issued.

4.1 Pace Atrium

For DDD, `ap!` (put an event out port `ap`) means the most recent ventricular sense (`vs`) or pace (`vp`) occurred VA-interval (`va_interval`) time previously, and there has not been an atrial sense since the time-of-previous-suspension (`tops`). The VA-interval is depicted in Figure 3 as a blue rectangle, with the AV delay as a yellow rectangle. Always $AV + VA = L$.

```
ap: out event port --pace atrium
  {BLESS::Assertion=>"<<AP(now)>>";};
 . . .
 <<AP:x: (vp or vs)@(x-va_interval) and not
  (exists tv:T in (x-va_interval),,x that (vs or vp)@tv)
  and not (exists ta:T in tops,,now that a@ta ) >>
 . . .
va_interval : constant T := L-av
```

Table 8. Atrial Pace Assertion

4.2 Pace Ventricle

The ventricle is paced (`vp!`) to enforce the Lower Rate Limit (Section 3.1), or to track atrial senses, but not faster than the Upper Rate Limit (Section 3.3).

```
vp: out event port --pace ventricle, but not too soon
 {BLESS::Assertion=>"<<VP(now) and URL(now)>>";};
 . . .
 <<VP:x: PACE_ON_LRL(theTime) or PACE_ON_SAV_DELAY(x) >>
 <<PACE_ON_LRL:x: (vp or vs)@(x-L)
  and not (exists t:T in (x-L),,x that (vs or vp)@t) >>
 <<PACE_ON_SAV_DELAY:x: as@(x-(av+o))
  and not (exists tu:T in x-u,,x that (vs or vp)@tu )>>
```

Table 9. Ventricular Pace Assertion

4.3 Non-Refractory Atrial Sense

An atrial sense is non-refractory (`as!`) when not in Post-Ventricular Atrial Refractory Period (3.5), or either sensed or paced Atrioventricular Delay (3.4).

```
as: out event port --non-refractory atrial sense
  {BLESS::Assertion=>"<<AS(now)>>";};
 . . .
<<AS:x: a@x and not PVARP(x) and not (PAV(x) or SAV(x))>>
```

Table 10. Atrial Sense Assertion

4.4 Non-Refractory Ventricular Sense

A ventricular sense is non-refractory (`vs!`) when not in the Ventricular Refractory Period (3.2).

```
vs: out event port --non-refractory ventricular sense
  {BLESS::Assertion=>"<<VS(now)>>";};
. . .
<<VS:x: v@x and NR(x) >>
```

Table 11. Ventricular Sense Assertion

5 DDD State Machine

State machines in BLESS have variables, states, transitions, Assertions, and an invariant.

Variable values persist between dispatches. They may be accompanied by an Assertion that defines their meaning, and therefore use. Table 12 declares variable `last_vp_or_vs` with Assertion `LAST_VP_OR_VS` to be the time of most recent ventricular pace or sense.

```
variables
 last_vp_or_vs : T
  <<LAST_VP_OR_VS:x: (vp@last_vp_or_vs or vs@last_vp_or_vs) and
   not (exists t:T in last_vp_or_vs,,x that (vs or vp)@t) >>;
```

Table 12. Variable Declaration

Threads may also have invariants akin to loop invariants. Thread invariants must hold at both dispatch and suspension, much like loop invariants must be true before and after each iteration of a loop.

`LRL` is the fundamental effectiveness property so must always be true. (`URL`, the fundamental safety property must be true when a ventricular pace is commanded). `SAV` holds after an atrial sense, before ventricular pace or sense. `PAV` holds after an atrial pace, before ventricular pace or sense.

```
invariant
 <<INV: : LRL(now) and LAST_VP_OR_VS(now)>>
```

Table 13. Invariant Declaration

Execution begins in the `initial` state and ends in a `final` state. Entering a `complete` state suspends execution until dispatch. States without a label are 'execute' states, which must be transitory.

```
states
 sav : complete state --an atrial sense occurred
   <<SAV(now) and LRL(now) and V(now) >>;
 sav_check_vrp : state --V-sense in sensed AV delay
   <<v@now and SAV(now) and LRL(now) and V(now) >>;
```

Table 14. State Declarations

Transitions have one or more source states, a transition condition, a destination state, and possibly an action. Transitions leaving complete states must have dispatch conditions, evaluated by the AADL runtime system and may not refer to values of variables. Transitions leaving execute states have boolean transition conditions which may use variables.

```
transitions
 T1_PACE_AFTER_LRL: --fundamental lower-rate pacing
  va,sav,pav -[on dispatch timeout (vp vs) L ms]->va
   { . . . actions . . . };
 T2_VS_AFTER_AS: --ventricular sense during SAV delay
  sav -[on dispatch v]-> sav_check_vrp;
 T3_VS_AFTER_AS_IN_VRP: --v-sense during VRP
  sav_check_vrp -[last_vp_or_vs >= (now-r)]-> sav{};
 T4_VS_AFTER_AS_AFTER_VRP: --v-sense after VRP
  sav_check_vrp -[last_vp_or_vs < (now-r]-> va
   {<<VS(now) and LAST_AS(now) and LAST_AP(now)>>
   vs! <<vs@now and LAST_AS(now) and LAST_AP(now)>>
   & last_vp_or_vs:=now <<last_vp_or_vs=now>>};
```

Table 15. Transition Declarations

6 Transforming Proof Outline into Proof

Annotating states and actions with Assertions forms a proof outline. Verification conditions (a.k.a. proof obligations) are generated, one for each complete or execute state, one for each transition. Tactics are then applied to reduce the proof obligations to normal form to discharge them as axioms.

6.1 Verification Condition for Complete states

A complete state's Assertion must imply its thread invariant. The following verification condition for complete state `sav` asks whether its `BLESS::Assertion` property implies the thread invariant.

```
P [250] <<SAV(now) and LRL(now) and V(now)>>
S [212]->
Q [212] <<LRL(now) and LAST_VP_OR_VS(now)>>
```

Table 16. Complete State Verification Condition

6.2 Verification Condition for Execute States

Execute states must be transitory, so there must always be an enabled, outgoing transition leaving the execute state. The following verification condition for an execute state `sav_check_vrp` asks whether its `BLESS::Assertion` property implies the disjunction of outgoing transition conditions.

```
P [252] <<v@now and SAV(now) and LRL(now) and V(now)>>
S [252]->
Q [252] <<(last_vp_or_vs >= (now-r))
  or (last_vp_or_vs < (now-r))>>
```

Table 17. Execute State Verification Condition

6.3 Verification Condition for Execute Transitions With Actions

Including an action in an execute transition becomes a Hoare triple `<<P>> S <<Q>>` where the precondition is the Assertion of the source state, and the post-conditions is the Assertion of the destination state. Verification condition for `T4_VS_AFTER_AS_AFTER_VRP` is shown in Table 18.

```
P [252] <<v@now and SAV(now) and LRL(now) and V(now) and
  (last_vp_or_vs < (now-r))>>
S [336]<<VS(now) and LAST_AS(now) and LAST_AP(now)>>
vs! <<vs@now and LAST_AS(now) and LAST_AP(now)>>
& last_vp_or_vs := now <<last_vp_or_vs = now>>
Q [260] <<LRL(now) and V(now)>>
```

Table 18. Execute Transition with Action, Verification Condition

6.4 Verification Condition for Dispatch Transitions

Transitions leaving complete states have dispatch conditions (`on dispatch`). Such dispatch transitions, without actions, have verification conditions that the source state's `BLESS::Assertion` property, and the dispatch condition, must imply the destination state's `BLESS::Assertion` property, with one complication–none of the other dispatch conditions leaving the same state have occurred since the time of previous suspension (`tops`).

The verification condition of transition `T2_VS_AFTER_AS` is shown in Table 19. Terms for source state's `BLESS::Assertion` property, and the dispatch condition (event arrival at `v` port) are straightforward. That no transition from `sav` occurred since `tops` becomes complex.

```
P [326] <<SAV(now) and LRL(now) and V(now) and v@now
 and not (exists u:T in tops,,now that
  ((vp or vs)@(u-L)
  and not (exists t:T in u-L,,u that (vp or vs)@t )) )
 and not (exists u:T in tops,,now that v@u )
 and not (exists u:T in tops,,now that (as@(u-(av+o))
 and not (exists t:T in u-(av+o),,u that as@t )) )
 and not (exists u:T in tops,,now that stop@u )>>
S [326]->
Q [252] <<v@now and SAV(now) and LRL(now) and V(now)>>
```

Table 19. Dispatch Transition Verification Condition

Transition `T2_VS_AFTER_AS` has source state `sav`. Transitions `T1_PACE_AFTER_LRL`, `T5a_EXPIRED_SENSED_AV_DELAY`, and `T17_STOP` also have source state `sav`. Comments have been added to the verification condition generated for `T2_VS_AFTER_AS` have been added for guidance.

6.5 Solving Proof Obligations

The BLESS proof engine transforms behavior annotated with Assertions into proof with human help choosing proof tactics. Tactics may either be applied manually, or assembled into a script.

All the verification conditions are created and held in a buffer. The first verification condition (a.k.a proof obligation) is moved into the active set of proof obligations. Tactics are applied to the active set until all proof obligations are solved. The next proof obligation is then moved into the active set. When all the verification conditions have been solved, a formal proof is constructed.

Fortunately, the tactics have common patterns, and can be combined into scripts.

6.6 Generated Proof

Due to space limitation the full program, script, and generated proof for DDD can be found at `bless.santoslab.org`.

7 Related Work

Much work has been done on formal methods for reasoning about behavioral descriptions in high-level modeling languages such as UML. While model checking applied to notations such as Statecharts and other state machine notations has been well-studied (*e.g.*, [12]), such approaches primarily focus on verifying temporal properties or simple assertions, instead of the strong functional properties expressible in BLESS, or properties that relate directly to timing and scheduling periods in the run-time environment.

For previous work on verification of the behavioral aspects of AADL models, a translation of a subset of AADL BA into an extension of Petri nets is given by [1], while Ölveczky *et al.*translate AADL BA into Real-Time Maude [13]. These strategies employ model checking and term rewriting, respectively, to

verify simple assertions and temporal properties. Thus, they achieve a much higher degree of automation, but treat less expressive specification languages.

Also mentioned in the introduction was the important class of specification language named Behavioral Interface Specification Languages. Notable members of this class include the Java Modeling Language and Spec# (a BISL for C#). Most BISLs are enriched contract-based languages intended for the specification and verification of single-threaded software systems.

Mery and Singh [11], formally model all operational modes of a pacemaker system using Event-B. They apply an incremental refinement approach to a basic abstract model of the system and add more functional and timing properties. They use the ProB tool to validate their models in different situations such as absence of input pulses.

Z. Jiang, et al. [6] consider a number of interesting aspects of DDD pacing (including mode transitions) using behavioral models in UPPAAL (a mature timed automata model checker). Monitoring automate along with timed CTL formulas are used to formally render pacing requirements. The system is closed by composing the pacemaker model with highly non-deterministic models that aim to provide approach coverage of patterns of heart signals. To avoid state space explosion, model checking is applied to only system with a single set of parameter values. Instead of checking Assertions for URL and LRL over duration of operation, they check timing properties of their model over a single cardiac cycle. There are interesting tradeoffs between the approaches. Regarding specifications, one might argue that the BLESS Assertion language provides more direct, readable, and expressive approach to capturing the type of requirements inherent in pacing compared to a mixture of CTL and monitoring automata. Clearly, the model checking approach is much more automatic, but the model checking approach requires one to close the system with an automata simulating a heart rather than relying on the assume guarantee nature of interface contracts in BLESS. Moreover, due to challenges associated with state-space explosion, applying model-checking requires limiting the algorithms to specific sets of configuration parameters.

8 Conclusion and Future Work

The integration of BLESS interface and transition system behavior specifications with a standardized architecture description language provides a foundation for rigorous reasoning about component interfaces and behaviors in the development of embedded systems. We have illustrated how natural language requirements for DDD-mode cardiac pacing can be naturally rendered as formal temporal specifications in BLESS, how the implementation of DDD pacing can be expressed as a BLESS transition system, and how a proof outline can be transformed into a complete formal proof that the implementation conforms to formal specifications. We believe the BLESS provides interesting tradeoffs compared with other formal methods such as model checking, theorem proving, or static analysis that may be particularly valuable when high-confidence, rich specifications,

and high coverage behaviors are required. We are working to prove correctness of a pacemaker model including all of the features described in [14] including rate response, atrial tachycardia response, PVARP extension, hysteresis pacing, and AV squeeze.

References

1. Bernard Berthomieu, Jean-Paul Bodeveix, Christelle Chaudet, Silvano Zilio, Mamoun Filali, and François Vernadat. Formal verification of AADL specifications in the Topcased environment. In *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*, pages 207–221, 2009.
2. Sara Bessling and Michaela Huhn. Formal safety analysis and verification in the model driven development of a pacemaker product line. In *Proceedings of Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2012)*, pages 133–144, 2012.
3. Peter Feiler and David Gluch. *Model-based Engineering with AADL: An Introduction to the SAE Architecture Analysis and Design Language*. Addison-Wesley, 2013.
4. Peter H. Feiler, Jorgen Hansson, Dionisio de Niz, and Lutz Wrage. System architecture virtual integration: An industrial case study. Technical Report CMU/SEI-2009-TR-017, Software Engineering Institute, November 2009.
5. John Hatcliff, Gary T. Leavens, K. Rustan M. Leino, Peter Müller, and Matthew Parkinson. Behavioral interface specification languages. *ACM Comput. Surv.*, 44(3):16:1–16:58, 2012.
6. Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *Proceedings of the 2012 Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012)*, volume 7214 of *Lecture Notes in Computer Science*, pages 188–203. Springer Berlin Heidelberg, 2012.
7. Fabrice Kordon, Jerome Hugues, Agusti Canals, and Alain Dohet. *Embedded Systems: Analysis and Modeling with SysML, UML and AADL*. Wiley-ISTE, 2013.
8. Brian Larson. BLESS website. |bless.santoslab.org—.
9. Brian Larson, Patrice Chalin, and John Hatcliff. Bless: Formal specification and verification of behaviors for embedded systems with software. In *Proceedings of the 2013 NASA Formal Methods Conference*, volume 7871 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2013.
10. Brian R. Larson. *Behavior Language for Embedded Systems with Software Annex Sublanguage for AADL*, 2012. Available at [15].
11. Dominique Méry and Neeraj Kumar Singh. Formal specification of medical systems by proof-based refinement. *ACM Trans. Embed. Comput. Syst.*, 12(1):15:1–15:25, January 2013.
12. Erich Mikk, Yassine Lakhnech, Michael Siegel, and Gerard J. Holzmann. Implementing Statecharts in PROMELA/SPIN. In *Proceedings of the Workshop on Industrial Strength Formal Specification Techniques (WIFT)*, Washington, DC, USA, 1998. IEEE Computer Society.
13. Peter Csaba Ölveczky, Artur Boronat, and José Meseguer. Formal semantics and analysis of behavioral AADL models in Real-Time Maude. In *FORTE'10*, pages 47–62, 2010.
14. Boston Scientific. Pacemaker system specification. sqrl.mcmaster.ca/pacemaker.htm, 2007.
15. SAnToS TR 2012-12-01 web site. `info.santoslab.org/research/aadl/bless`.

A Appendix

As a convenience for the iFM reviewers, we provide in this appendix the complete BLESS Assertions and transition system implementation for DDD pacing. This information, along with the BLESS proofs of implementation conformance to Assertions specifying DDD requirements, can be found at `bless.santoslab.org`.

```
--DDD.aadl
--simple dual-chamber pacemaker, DDD mode

package ddd_mode
public
  with BLESS_Types, --AADL standard, predeclared property set for time
   BLESS,  --predeclared BLESS properties
   PP;  --imported property set defining pacing settings that control
        --the behavior of the device

thread DDD
  features
  a: in event port; --atrial sense
  ap: out event port  --pace atrium
    {BLESS::Assertion=>"<<AP(now)>>";};
  vp: out event port  --pace ventricle, but not too soon
    {BLESS::Assertion=>"<<VP(now) and URL(now)>>";};
  as: out event port --non-refractory atrial sense
    {BLESS::Assertion=>"<<AS(now)>>";};
  vs: out event port  --non-refractory ventricular sense
    {BLESS::Assertion=>"<<VS(now)>>";};
  v: in event port; --ventricular sense
  properties
    Dispatch_Protocol => Aperiodic;
annex BLESS
{**
assert

--lower rate limit, both A and V chambers
--5.1 Lower Rate Limit (LRL)
--  The Lower Rate Limit (LRL) is the number of generator pace pulses
--  delivered per minute (atrium or ventricle) in the absence of
--  ? Sensed intrinsic activity.
--  ? Sensor-controlled pacing at a higher rate.
--  The LRL is affected in the following ways:
--    1. When Rate Hysteresis is disabled, the LRL shall define the
--       longest allowable pacing interval.
--    2. In DXX or VXX modes, the LRL interval starts at a ventricular
--       sensed or paced event
<<LRL:theTime :
    -- there has been a V-pace or a non-refractory V-sense
    exists t:BLESS_Types::Time
      -- within the previous LRL interval
      in (theTime-PP::Lower_Rate_Limit_Interval)..theTime
      -- in which a heartbeat was sensed, or caused by pacing
      that (vs or vp)@t  >>

--upper rate limit, only ventricle
--5.2 Upper Rate Limit (URL)
--  The Upper Rate Limit (URL) is the maximum rate at which the paced
--  ventricular rate will track sensed atrial events.
--  The URL interval is the minimum time between a ventricular event
--  and the next ventricular pace.
<<URL:theTime: --a V-pace never happens too soon after either V-sense or V-pace
  -- applied to vp: out event port, no pace (out event) too soon, URL is true when
  -- event is sent
  not -- there must not be V-sense or -pace not in previous URL open-interval ,
```

```
   (exists tu:BLESS_Types::Time
      in (theTime-PP::Upper_Rate_Limit_Interval),,theTime
      that (vs or vp)@tu) >>

--atrial-ventricular delay
--5.3 Atrial-Ventricular (AV) Delay
--  The AV delay shall be the programmable time period from an
--  atrial event (either intrinsic or paced) to a ventricular pace.
-- In atrial tracking modes, ventricular pacing shall occur in the
--  absence of a sensed ventricular event within the programmed AV delay
--  when the sensed atrial rate is between the programmed LRL and URL.
--  AV delay shall either be
--    1. Fixed (absolute time)
--    2. Dynamic

--paced AV delay
--5.3.1 Paced AV Delay
--  A paced AV (PAV) delay shall occur when the AV delay is initiated
--  by an atrial pace.
<<PAV:theTime:  --there has been A-pace during the fixed AV delay
  exists ts:BLESS_Types::Time
       --in the previous fixed AV delay milliseconds,
      in theTime-PP::Fixed_AV_Delay..theTime
      that ap@ts >>

--sensed AV delay
--5.3.2 Sensed AV Delay
--  A sensed AV (SAV) delay shall occur when the AV delay is initiated
--  by an atrial sense.
--Sensed AV Delay Offset is always negative, then added to the
--Fixed AV Delay to determine Sensed AV Delay
<<SAV:theTime:
  (exists ta:BLESS_Types::Time
       --in the previous fixed AV delay milliseconds,
      in theTime-PP::Sensed_AV_Delay..theTime
      that as@ta)
  or  --a previous V-event prevents pacing above URL
  (exists tu:BLESS_Types::Time
    in theTime-PP::Upper_Rate_Limit_Interval..theTime
    that (vs or vp)@tu)  >>

--vetricular refractory period:
--5.4.1 Ventricular Refractory Period (VRP)
--  The Ventricular Refractory Period shall be the programmed time
--  interval following a ventricular event during which time
--  ventricular senses shall not inhibit nor trigger pacing.
<<VRP:theTime:
  exists tv:BLESS_Types::Time
  in theTime-PP::Ventricular_Refractory_Period,,theTime
  that (vs or vp)@tv  >>

--atrial refractory period
--5.4.2 Atrial Refractory Period (ARP)
--  For single chamber atrial modes, the Atrial Refractory Period (ARP)
--  shall be the programmed time interval following an atrial event
--  during which time atrial events shall not inhibit nor trigger pacing.
<<ARP:theTime:
  exists tar:BLESS_Types::Time
    in theTime-PP::Atrial_Refractory_Period,,theTime
    that (as or ap)@tar>>

--post-ventricular atrial refractory period
--5.4.3 Post Ventricular Atrial Refractory Period (PVARP)
--  The Post Ventricular Atrial Refractory Period shall be available
--  in modes with ventricular pacing and atrial sensing. The Post
--  Ventricular Atrial Refractory Period shall be the programmable
--  time interval following a ventricular event when an atrial
--  cardiac event shall not
```

```
--    1. Inhibit an atrial pace.
--    2. Trigger a ventricular pace.
<<PVARP:theTime:
  exists tv:BLESS_Types::Time
    in theTime-PP::PVARP,,theTime
    that (vs or vp)@tv >>

--non-refractory ventricular sense occurs, asserted by port vs
--  there is a ventricular sense, and the ventricular refractory period
--  has expired since previous vs or vp
<<VS:theTime: v@theTime and not VRP(theTime) >>

--non-refractory atrial sense occurs, asserted by port as
--  there is an atrial sense,
--  and the atrial refractory period has expired since previous as or ap
--  and the post-ventricular atrial refractory period has expired
--  since previous vs or vp
<<AS:theTime:
  a@theTime
  and not ARP(theTime)
  and not PVARP(theTime) >>

--ventricular pace
--  for DDD, vp! means either
--  pace_now occurs
--  or  either vp or vs occurred LRL interval ago and not since,
--  or ap occurred paced AV delay ago (coincident with pace at LRL), or
--  or as occurred sensed AV delay ago
--    and no ventricular sense or pace
--    occurred in the previous upper rate limit interval
<<VP:theTime:
  PACE_ON_LRL(theTime)  -- (vp or vs) occurred LRL interval ago
    or  --as occurred sensed AV delay ago, but not too fast
    PACE_ON_SAV_DELAY(theTime) >>

<<PACE_ON_LRL:theTime:    --no intrinsic activity, pace at LRL
  (vp or vs)@(theTime-PP::Lower_Rate_Limit_Interval)
      and --and not since
      not (exists t:BLESS_Types::Time
        in (theTime-PP::Lower_Rate_Limit_Interval),,theTime
        --with a non-refractory ventricular sense or pace
        that (vs or vp)@t) >>

<<PACE_ON_SAV_DELAY:theTime:  --track atrial sense
    as@(theTime-PP::Sensed_AV_Delay)
      and  --there have been no ventricular events in the upper-rate interval
      not (exists tu:BLESS_Types::Time
        in theTime-PP::Upper_Rate_Limit_Interval,,theTime
        that (vs or vp)@tu )>>

--atrial pace
--  for DDD, ap! means
--  the previous vs or vp occured VA interval perviously
--    (VA interval = LRL interval - PAV delay)
--  and there have been no ventricular senses since then
--  nor atrial pace or sense since LRL
<<AP:theTime:  --time out no activity since last ventricular sense or pace
    (vp or vs)@(theTime-va_interval)
      and --and not since
      not (exists tv:BLESS_Types::Time
        in (theTime-va_interval),,theTime
        --with a ventricular or atrial sense
        that (vs or vp)@tv)
      and  --not atrial-sensed since tops
      not (exists ta:BLESS_Types::Time
        in tops,,now
        that a@ta ) >>
```

```
--invariant of internal variables
<<V:theTime: LAST_VP_OR_VS(theTime) and LAST_AS(theTime)
   and LAST_AP(theTime)>>

--interactive limit that LRL<URL
<<AXIOM_LRLi_gt_URLi_LIMIT:theTime:
  (theTime-PP::Lower_Rate_Limit_Interval)
  <(theTime-PP::Upper_Rate_Limit_Interval) >>

--time of previous atrial sense was no earlier than the
--  time-of-previous-suspension
<<AXIOM_LAST_AS_LE_TOPS: :last_as<=tops>>

--a is more recent than ap
<<AXIOM_LAST_AS_LT_LAST_AP: :last_as<last_ap>>
--ap is more recent than a
<<AXIOM_LAST_AP_LT_LAST_AS: :last_ap<last_as>>


invariant    --to prove this is always true at complete states,
                --only externally-observed behavior
<<INV: :
  LRL(now)   --lower rate limit property holds now, always
  and
  LAST_VP_OR_VS(now)  --have V-paced or V-sensed, waiting for A-sense
  >>

variables
  last_vp_or_vs : BLESS_Types::Time  --time of last ventricular pace or sense
    <<LAST_VP_OR_VS:theTime:
       --Assertions of state variables must also be invariant
      (vp@last_vp_or_vs or vs@last_vp_or_vs) and
      --a V-event occurred at last_vp_or_vs time
      not (exists t:BLESS_Types::Time  --and not since
       in last_vp_or_vs,,theTime   --note open interval
       that (vs or vp)@t) >>;
  last_as : BLESS_Types::Time   --time of last atrial sense
    <<LAST_AS:theTime: as@last_as and
      --a non-refractory atrial sense occurred at last_as
      not (exists t:BLESS_Types::Time  --and not since
       in last_as,,theTime
       that as@t) >>;
  last_ap : BLESS_Types::Time   --time of last atrial pace
    <<LAST_AP:theTime: ap@last_ap and --an atrial pace occurred at last_ap
      not (exists t:BLESS_Types::Time  --and not since
       in last_ap,,theTime
       that ap@t) >>;
  va_interval : constant BLESS_Types::Time  --ventrucular-atrial interval
    := PP::Lower_Rate_Limit_Interval-PP::Fixed_AV_Delay
       --atrial paces following vs or vp
  <<AXIOM_VA_INTERVAL: : va_interval =
     (PP::Lower_Rate_Limit_Interval-PP::Fixed_AV_Delay) >>;

states
  start : initial state
    --first state, start pacing as if no sense or pace in previous LRL interval
    <<PACE_ON_LRL(now) and AS(now) and AP(now)
    and  --no previous ventricular senses or paces
      not (exists t:BLESS_Types::Time
        in now-PP::Upper_Rate_Limit_Interval,,now
        that (vs or vp)@t )
    and  --no atrial senses or paces since time-or-previous-suspension
      not (exists t2:BLESS_Types::Time
        in tops,,now
        that (as or ap)@t2 )>>; --first instant of operation
  off : final state;
  sav : complete state  --an atrial sense occurred
    <<SAV(now) and LRL(now) and V(now) >>;
```

```
    --  in the previous Sensed AV Delay
  sav_check_vrp : state  --ventricular sense during sensed AV delay
    <<v@now and SAV(now) and LRL(now) and V(now) >>;  --was it in VRP?
  sav_check_url : state  --SAV delay timeout, check for URL before tracking
    <<as@(now-PP::Sensed_AV_Delay) and SAV(now) and LRL(now) and V(now) >>;
  pav : complete state  --an atrial pace occured
    <<PAV(now) and LRL(now)  and V(now)>>;
  --  in the previous Paced AV Delay
  pav_check_vrp : state  --ventricular sense during paced AV delay
    <<v@now and PAV(now) and LRL(now) and V(now) >>;  --was it in VRP?
  va : complete state   --a V pace or sense occured in the previous VA interval
    <<LRL(now) and V(now) >>;
  check_atrial_refractories : state   --check if atrial sense is in PVARP or ARP
    <<a@now and LRL(now) and V(now) and
        AXIOM_LAST_AS_LT_LAST_AP()>>;
  check_vrp : state
     --check if ventricular sense is premature ventricular contraction
    <<v@now and LRL(now) and V(now) >>;

transitions
  TO_GO:  --start pacing immediately
  start -[ ]-> va
    {<<PACE_ON_LRL(now) and AS(now) and AP(now)>> --first instant of operation
    {  --beginning of existetial lattice quantificcation
      <<PACE_ON_LRL(now)>>
      {
        <<PACE_ON_LRL(now) and AXIOM_LRLi_gt_URLi_LIMIT(now)>>
      vp!
        <<vp@now>>
      &
      last_vp_or_vs:=now
        <<last_vp_or_vs=now>>
      }
       <<LAST_VP_OR_VS(now) and LRL(now)>>
    &
      <<AS(now)>>
      {
        <<AS(now)>>
      as!  --fake, harmless event needed to establish LAST_AS
        <<as@now>>
      &
       last_as:=now
        <<last_as=now>>
      }
      <<LAST_AS(now)>>
    &
      <<AP(now)>>
      {
        <<AP(now)>>
      ap!  --the pacing circuitry delivers only v-pace
            --when it gets both ap! and vp!
        <<ap@now>>
      &
       last_ap:=now
        <<last_ap=now>>
      }
      <<LAST_AP(now)>>
    }  --end of existential lattice quantification
    <<LAST_VP_OR_VS(now) and LRL(now) and LAST_AS(now) and LAST_AP(now)>>
    };

  T1_PACE_AFTER_LRL:  --fundamental lower-rate pacing; keeps patients "pink"
  va,sav,pav -[on dispatch timeout (vp vs) PP::Lower_Rate_Limit_Interval ms]->va
      { <<(vp or vs)@(now-PP::Lower_Rate_Limit_Interval)
          and not (exists t:BLESS_Types::Time
           in (now-PP::Lower_Rate_Limit_Interval),,now
           that (vp or vs)@t) and LAST_AP(now)
         and LAST_AS(now) and AXIOM_LAST_AS_LE_TOPS()
```

```
          and AXIOM_LRLi_gt_URLi_LIMIT(now)>>
       vp!
         <<vp@now and not (exists t:BLESS_Types::Time
          in (now-PP::Lower_Rate_Limit_Interval),,now
          that (vp or vs)@t)
         and (vp or vs)@(now-PP::Lower_Rate_Limit_Interval)
         and LAST_AS(now) and LAST_AP(now) and AXIOM_LRLi_gt_URLi_LIMIT(now)>>
       ;  --note seqential composition
       last_vp_or_vs:=now
       <<vp@last_vp_or_vs and vp@now and AXIOM_LAST_AS_LE_TOPS()
         and (last_vp_or_vs=now) and LAST_AS(now) and LAST_AP(now)>>};

  T2_VS_AFTER_AS:    --ventricular sense during SAV delay
  sav -[on dispatch v]-> sav_check_vrp{};  --go check whether in VRP

  T3_VS_AFTER_AS_IN_VRP:  --ventricular sense during VRP, go back to sav
  sav_check_vrp -[last_vp_or_vs >= (now-PP::Ventricular_Refractory_Period)]-> sav{};

  T4_VS_AFTER_AS_AFTER_VRP:  --ventricular sense after VRP, wait for atrial sense
  sav_check_vrp -[last_vp_or_vs < (now-PP::Ventricular_Refractory_Period)]-> va
    {<<VS(now) and LAST_AS(now) and LAST_AP(now)>>
    vs!
     <<vs@now and LAST_AS(now) and LAST_AP(now) and AXIOM_LAST_AS_LE_TOPS()>>
    &
    last_vp_or_vs:=now <<last_vp_or_vs=now>>};

  T5a_EXPIRED_SENSED_AV_DELAY:  --V-pace after sensed AV delay
    sav -[on dispatch timeout (as) PP::Sensed_AV_Delay ms]-> sav_check_url{};

  T5b_TRACKED_VP_TOO_SOON_AFTER_PREVIOUS_VS_OR_VP:
   --tracked VP too close to previous vp or vs
   sav_check_url -[last_vp_or_vs > (now-PP::Upper_Rate_Limit_Interval)]-> va{};
     --go back and wait for another AS to track, or for LRL to time-out forcing VP

  T5c_TRACKED_VP_AFTER_AS:  --v-pace after a-sense, if not exceeding URL
   sav_check_url -[last_vp_or_vs <= (now-PP::Upper_Rate_Limit_Interval)]-> va
     {<<PACE_ON_SAV_DELAY(now) and V(now)
       and (last_vp_or_vs <= (now-PP::Upper_Rate_Limit_Interval))>>
    vp!  --ventricular pace tracking atrial sense
     <<vp@now and V(now) and AXIOM_LRLi_gt_URLi_LIMIT(now)>>
    &
    last_vp_or_vs:=now <<last_vp_or_vs=now>>};


  T6_VS_AFTER_AP:  --ventricular sense after atrial pace
  pav -[on dispatch v]-> pav_check_vrp{};  --go check whether in VRP

  T7_VS_AFTER_AP_IN_VRP:  --vs too soon after previous vs or vp, go back to pav
  pav_check_vrp -[last_vp_or_vs >= (now-PP::Ventricular_Refractory_Period)]-> pav{};

  T8_VS_AFTER_AP_AFTER_VRP:  --ventricular sense after VRP, wait for atrial sense
  pav_check_vrp -[ last_vp_or_vs < (now-PP::Ventricular_Refractory_Period)]-> va
    { <<VS(now) and V(now)>>
      {
        <<VS(now) and LAST_AS(now) and LAST_AP(now)>>
      vs!
        <<vs@now and LAST_AS(now) and LAST_AP(now) and
             AXIOM_LRLi_gt_URLi_LIMIT(now)>>
      &
      last_vp_or_vs:=now
        <<last_vp_or_vs=now>>
      }
      <<vs@now and (last_vp_or_vs=now) and
        LAST_AS(now) and LAST_AP(now) and LAST_VP_OR_VS(now)>> };

  T10_AS_AFTER_VS_OR_VP:   --a sense after v sense or v pace
  va -[on dispatch a]-> check_atrial_refractories{};  --is a too early?

```

```
  T11_PVARP_EXPIRED:  --a sense not in PVARP, start sensed A-V delay
  check_atrial_refractories -[((now-PP::PVARP)>last_vp_or_vs)
        and ((now-PP::Atrial_Refractory_Period)>last_as)
        and ((now-PP::Atrial_Refractory_Period)>last_ap)]-> sav
    {<<AS(now) and LRL(now) and LAST_AP(now) and LAST_VP_OR_VS(now)>>
    as!
     <<as@now and LRL(now) and LAST_AP(now) and LAST_VP_OR_VS(now)
       and AXIOM_LRLi_gt_URLi_LIMIT(now) >>
    &
    last_as:=now
     <<(last_as=now)  and LAST_AS(now)>>};

  T12_STILL_IN_PVARP:  --a sense in PVARP, go back to va
  check_atrial_refractories -[((now-PP::PVARP)<=last_vp_or_vs)
      or ((now-PP::Atrial_Refractory_Period)<=last_as)
      or ((now-PP::Atrial_Refractory_Period)<=last_ap)]-> va{};

  T13_VS_AFTER_VS_OR_VP:  --v sense after v sense or v pace
  va -[on dispatch v]-> check_vrp{};

  T14_STILL_IN_VRP:  -- vs in VRP, go back to va
  check_vrp -[(now-PP::Ventricular_Refractory_Period)<last_vp_or_vs]-> va{};

  T15_VRP_EXPIRED:  --vs after VRP expired
  check_vrp -[(now-PP::Ventricular_Refractory_Period)>=last_vp_or_vs]-> va
    {<<VS(now) and LAST_VP_OR_VS(now) and LAST_AS(now) and LAST_AP(now)>>
    vs!
     <<vs@now and LAST_AS(now) and LAST_AP(now)
       -- and AXIOM_LRLi_gt_URLi_LIMIT(now)
     >>
    &
    last_vp_or_vs:=now
     <<(last_vp_or_vs=now) and LAST_VP_OR_VS(now)>>};

  T16_AP_AFTER_TIMEOUT_VS_OR_VP:
  --atrial pace if no atrial sense during V-A interval
  va -[on dispatch timeout (vp vs) va_interval ms]-> pav
    {{<<AP(now) and LAST_VP_OR_VS(now) and LAST_AS(now) and
       not (exists tv:BLESS_Types::Time
            in now-va_interval,,now
            that (vp or vs)@tv )>>
    ap!
     <<ap@now and LAST_VP_OR_VS(now) and LAST_AS(now) and
       not (exists tap:BLESS_Types::Time
            in now-va_interval,,now
            that (vp or vs)@tap )>>
    &
      <<LRL(now) >>
    last_ap:=now
      <<(last_ap=now) and LRL(now) >> }
    <<(last_ap=now) and LAST_AP(now) and LAST_AS(now) and LAST_VP_OR_VS(now) and
     not (exists tv:BLESS_Types::Time
          in now-va_interval,,now
          that (vp or vs)@tv )
     and LRL(now) >>};

  T17_STOP:  --turn off pacing
  sav,pav,va -[on dispatch stop]-> off{};

**};  --end of annex subclause

end DDD;

end ddd_mode;

--end of DDD.aadl
```