Appendix

A Implementation details of the modules

In this Appendix, we present details about how the different modules of the system were implemented.

Reinforcement Learning agent The agent was implemented in Python and the neural network model was trained using the PyTorch library. The model is a two-layer neural network with a hidden layer of 256 neurons and an output layer of 3 neurons, accepting an input vector of size 42. In our scenario, the possible output values for ($AtrialPace$(AP),$VentricularPace$(VP)) are $\{00,01,10\}$, hence the output layer has 3 neurons. We apply the ReLU activation function only to the hidden layer. The mean squared error was used as the loss function, and the Adam optimizer was employed. The initial value of epsilon was set to 0.5 during the training phase. The batch size used was 1000. The learning rate was set to 0.001 and the value of gamma (discount factor for Q learning) was set to 0.9.

Heart model We make use of the heart model presented in [1] which is physiologically-based computational heart model containing 33 nodes, modelled with Hybrid Automatas (HAs) that capture non-linear dynamics. A flag allows for dynamic reconfiguration of Sinoatrial (SA) autorhythmic failure, allowing for SA Block to be simulated. Additionally, we introduce variability in the pacing rate (heart rate variability) of the SA node by randomly altering the dominance between the sympathetic and parasympathetic nervous systems using the pacing rates. The heart rate variablity is calculated using the time domain measures SDANN, 24-hr SD, SD, RMSSD, as specified in the [2]. RMSSD is the abbreviation for the root mean square of successive differences between normal heartbeats. The value of RMSSD ranges between 20 to 120 in humans. We introduce the parameter *hrv* which represents the value by which the pacing rate will be changed.

The pacing rate is changed randomly every 3 to 4 heartbeats, with a probability of 0.67. Whenever this change occurs, there's a 0.33 probability that the dominant nervous system will also randomly shift. Additionally, the value of hrv is randomly altered to a value between 20 to 120 with a probability of 0.67 whenever the pacing rate changes.

Additionally, the timestamp at which the SA node will fail, as well as the duration of the ensuing failure, are also randomly determined.

Runtime Enforcement Our Runtime Enforcement (RE) component is based on the set of pacemaker requirements listed in Section 3.3. As we know

Event	Probability
Pacing rate change	0.67
Change in dominant nervous system	0.22
Change in *hrv* value	0.45

Table 1: Probabilities

that a pause will occur in Type-II SA block for exactly two or three times the preceding AS-AS interval, the agent should wait for two consecutive intrinsic atrial signals. Thus, the following two additional properties are considered (not in [4]) to avoid the Reinforcement Learning (RL) agent from pacing over the heart, i.e., allowing the heart to beat intrinsically if possible.

- An AP should be preceded by at least two consecutive Atrial Senses (ASs). An AP can appear when the Atrial Escape Interval (AEI) timer elapses irrespective of two preceding ASs.
- An AP should occur only after a duration greater than or equal to the interval of the preceding AS-AS plus threshold.

In addition to the seven properties P_1 through P_7 listed there, the enforcer also takes into account the above policies. We consider the intersection of all nine policies as our policy to be enforced and construct an enforcer using the approach discussed in Section 5. For synthesis of the enforcer, we adapt the easy-rte tool [3] which is an easy to use implementation that supports the bidirectional runtime enforcement framework in [4]. This tool accepts properties expressed as Discrete Timed Automata (DTAs) and written in a custom .erte format, and automatically converts them to executable C code.

How the components interact: One important note is that while the RL agent, and associated framework, is implemented in Python, the enforcer and heart components are implemented in C. To facilitate communication between these languages, we make use of a buffer-based communication system. The *input buffer* is written to by the C-based enforcer and data is subsequently read from the Python-based RL agent. For communication in the opposite direction, the *output buffer* carries the pacing signals from the RL agent which is used by the enforcer and heart models.

B Appendix

In this Appendix, we present the event mapping used in the implementation in Table 2. The state information required by the RL agent (the corrected input signals from the enforcer and the A-A and A-V intervals in the last 10 beats) is modeled in the implementation as illustrated in Table 3.

Let the event mapping of i^{th} atrial event be A_i and i^{th} ventricular event be V_i. Then the state is represented as follows. The time elapsed from the i^{th} atrial event be A_i be t_{A_i} and time elapsed from the ventricular event V_i be t_{V_i}.

(AS, VS, AP, VP)	Mapping
(0,0,0,1)	-1
(0,0,1,0)	-2
(0,1,0,0)	-3
(1,0,0,0)	-4

Table 2: Event mapping

State
A_1
t_{A_1}
V_1
$t_{A_1}+t_{V_1}$
$\vdots$
A_i
t_{A_i}
V_i
$t_{A_i}+t_{V_i}$
$\vdots$
A_{10}
$t_{A_{10}}$
V_{10}
$t_{A_{10}}+t_{V_{10}}$
$AS_{t_{A_{10}}+t_{V_{10}}}$
$VS_{t_{A_{10}}+t_{V_{10}}}$

Table 3: State information

C Discussion on the enforcement mechanism constraints, and enforceability conditions

In this Appendix, we provide some discussion on the enforcement mechanism constraints, and enforceability conditions from [4].

Remark 1 (Enforcement mechanism constraints). The enforcer should satisfy various constraints such as soundness and transparency.

- *Soundness:* means that for any input word $\sigma \in \Sigma^*$, the output of the enforcer $\Pi_1(E_{\varphi,\varphi_{op}}(\sigma))$ can be extended to a sequence that satisfies φ (i.e., $\exists \sigma' \in \Sigma^* : \Pi_1(E_{\varphi,\varphi_{op}}(\sigma) \cdot \sigma' \in \varphi)$.
- *Monotonicity*: expresses that the output of the enforcer for an extended input word σ' of an input word σ, extends the output produced by the enforcer for σ. The monotonicity constraint means that the enforcer cannot undo what is already released as output.
- *Instantaneity* expresses that for any given input sequence σ, the output of the enforcer $E_{\varphi,\varphi_{op}}(\sigma)$ should contain exactly the same number of events that are in σ (i.e., $|\sigma| = |E_{\varphi,\varphi_{op}}(\sigma)|$). This means that, the enforcer cannot delay, insert and suppress events. Whenever the enforcer receives a new input event, it must react instantaneously and produce an output event immediately.
- *Transparency* means that the enforcer will not unnecessarily edit any event. Any new input event (x,y) will be simply forwarded by the enforcer if what has been computed as output earlier by the enforcer followed by (x,y) can be extended to a sequence that satisfies φ in the future.
- *Causality* expresses that for every input event (x,y) the enforcer produces output event (x',y') where the enforcer first processes the input part x, to produce the transformed input x' according to property φ. It later reads and transforms output $y \in \Sigma_O$ (output of the controller/program after it is fed with x'), to produce the transformed output y'.

Remark 2 (Enforceability). We also recall the definition of enforceability from [4]. Consider a safety property $\varphi \subseteq \Sigma^*$. It is *enforceable* iff an enforcer $E_{\varphi,\varphi_{op}}$ for φ exists satisfying all the constraints (soundness, transparency, instantaneity, monotonicity, and causality). It is shown in [4] that not all regular properties defined as DTA are enforceable.

Remark 3 (Condition for enforceability). Consider a property φ that is defined as DTA $\mathcal{A}_\varphi = (L, l_0, l_v, \Sigma, V, \Delta, F)$, with semantics $[[\mathcal{A}_\varphi]] = (Q, q_0, \Sigma, \rightarrow, Q_F, q_v)$. It is proved in [4] that a property φ is enforceable iff the following condition holds:

$$\forall q \in Q, q \notin q_v \Rightarrow \exists \sigma \in \Sigma^+ : q \xrightarrow{\sigma} q' \wedge q' \in Q_F. \qquad \textbf{(EnfCo)}$$

The condition for enforceability expresses that for a property φ defined as a DTA $\mathcal{A}_\varphi = (L, l_0, l_v, \Sigma, V, \Delta, F)$ with semantics $[[\mathcal{A}_\varphi]] = (Q, q_0, \Sigma, \rightarrow, Q_F, q_v)$, $E_{\varphi,\varphi_{op}}$ satisfying all the constraints (recalled in Remark 1) exists iff an accepting state is reachable from every non-violating state (i.e. from every state $q \notin q_v$) in 1 or more steps.

Remark 4 (Correctness). Given any property φ to be enforced defined as a DTA $\mathcal{A}_\varphi$ that satisfies Equation (**EnfCo**), the optimization policy φ_{op} the enforcement function $E_{\varphi,\varphi_{op}} : \Sigma^* \rightarrow (\Sigma, K)^*$ is an enforcer for φ. From the results proved in [4], we have for any σ, if we consider $\Pi_1(E_{\varphi,\varphi_{op}}(\sigma))$, it satisfies all the constraints such as soundness, transparency, monotonicity, instantaneity, and causality (See Remark 1).

Remark 5 (Enforcer for RL-based control).

- *Correctness* of the output given to the environment: The set of all the properties in the considered case-study (Section 3) are all enforceable, i.e., their respective DTA satisfies the condition for enforceability. Thus, in every cycle, the enforcer is guaranteed to produce an output that is correct/sound with respect to the given set of safety policies. In the proposed approach that output of the enforcer is given as the final output to the environment which is thus guaranteed to be safe.
- *Minimal interference:* In the proposed framework, the enforcer interferes and modifies the output of the agent only if it is unsafe.

D Electrical Signals

There are two main types of electrical signals that are of importance when discussing the heart system — Electrocardiograms (ECGs) and Electrograms (EGMs).

ECGs are the body-surface signals that are typically used by clinicians when diagnosing cardiac diseases. They consist of the typical shape which represents one complete heartbeat, made up from a series of subset waves, collectively known as *PQRST*, as shown in Healthy Heart image in Section 1. Firstly, the *P* wave denotes atrial depolarisation and aligns with their initial contraction. Subsequently, the *QRS* complex follows which represents the depolarization of the ventricles and the repolarization of the atria, corresponding to their contraction and relaxation respectively. Finally, the T wave capture the repolarization of the ventricles, aligning with their final relaxation.

Alternatively, EGMs are cardiac-surface signals which are detected on the physical cell walls of the heart, such as where pacemakers are attached. In this case, the signals are a lot more localised to the cells in their immediate surrounding, allowing for atrial and ventricular events to be separated into their own separate channels. In both cases, the sharp depolarization associated with contractions has a larger impact on the signal than the relatively slow repolarization, which are often not registered on an EGM.

References

1. Ai, W., Patel, N.D., Roop, P.S., Malik, A., Trew, M.L.: Cardiac electrical modeling for closed-loop validation of implantable devices. IEEE Tran. on Biomedical Engineering **67**, 536–544 (2020)
2. Cowan, M.J.: Measurement of heart rate variability. Western Journal of Nursing Research **17**, 32 – 48 (1995), `https://api.semanticscholar.org/CorpusID:21609769`
3. Pearce, H., Pinisetty, S., Roop, P.S., Kuo, M.M.Y., Ukil, A.: Smart i/o modules for mitigating cyber-physical attacks on industrial control systems. IEEE TII **16**(7), 4659–4669 (2020). https://doi.org/10.1109/TII.2019.2945520
4. Pinisetty, S., Roop, P.S., Smyth, S., Allen, N., Tripakis, S., von Hanxleden, R.: Runtime enforcement of cyber-physical systems. ACM TECS **16**, 1 – 25 (2017)